The Pennsylvania State University
The Graduate School

A STUDY OF DRAM OPTIMIZATION TO BREAK THE MEMORY WALL

A Dissertation in
Computer Science and Engineering
by
Tao Zhang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2014

The dissertation of Tao Zhang was reviewed and approved* by the following:

Yuan Xie
Professor of Department of Computer Science and Engineering
Dissertation Advisor, Committee Chair

Mary Jane Irwin
Professor of Department of Computer Science and Engineering

Robert E. Noll Professor

Evan Pugh Professor

Vijaykrishnan Narayanan
Professor of Department of Computer Science and Engineering

Zhiwen Liu
Associate Professor of Department of Electrical Engineering

Raj Acharya
Professor of Department of Computer Science and Engineering
Department Head

*Signatures are on file in the Graduate School.

Abstract

The well-known "Memory Wall" has been raised in 1990s. At that time, the researchers noticed the diverging exponential increase in the performance of processor and main memory and thus claimed that the main memory would eventually become the bottleneck of the entire computing system. Furthermore, benefiting from the semiconductor process scaling, the number of transistors in a single chip keeps growing up. As a result, microprocessor designs enter multi-/many-core era and the instruction level parallelism (ILP) and thread level parallelism (TLP) have been extensively exploited. The improved parallelism requires the memory to provide low latency, high bandwidth and low power consumption. Unfortunately, as the de facto main memory technology, the evolution of DRAM is relatively slow due to the poor scalability and the extremely high sensitivity of design cost (cost per bit). To this end, we are hitting the "Memory Wall".

To break the memory wall, enormous research work has been proposed to optimize the DRAM architecture so that a better trade-off among performance, power, and design overhead can be achieved. Some of the previous proposals, however, are difficult to be implemented because of either the unaffordable design overhead or the unacceptable performance degradation. Moreover, certain new issues have shown up along with the evolution of DRAM. For example, the performance impact of refresh cannot be ignored anymore as it can significantly degrade the DRAM performance. The power consumption of DRAM is also critical as up to 40% power is consumed by the DRAM modules, which are massively populated in the data centers. Therefore, in the DRAM realm it still needs lots of research efforts to make sure DRAM can win the war against the "Memory Wall". This is the motivation of this dissertation.

In this dissertation, the author proposes several novel DRAM architectures, which aims at a better trade-off among DRAM performance, power, and design overhead. Both traditional DRAM technologies and the emerging 3D-stacked DRAMs are covered in this work. To relieve the refresh penalty in the commodity DRAM, a concurrent refresh aware memory, CREAM, is proposed. CREAM allows refresh and memory access to be served in parallel. In addition, Half-DRAM is proposed to achieve power reduction and performance improvement simultaneously. In Half-DRAM, a bank is redesigned so that it is able to activate only half of a row to reduce the activation and precharge power. In this way, Half-DRAM can easily eliminate the power constraints and thus improve the performance. Meanwhile, two half-rows can be accessed in parallel to further improve the performance once the sub-array level parallelism is deployed. Furthermore, a new precharge policy, Lazy Precharge, is introduced to minimize the precharge overhead. By leveraging the Lazy Precharge, multiple activations can share a precharge so that the number of precharge can be reduced significantly. Furthermore, In addition to the traditional

2D DRAM, a novel 3D WideIO architecture is proposed to increase the DRAM parallelism. To take advantage of the increasing wiring resource in the vertical dimension, a bank is further split into multiple sub-banks. Each sub-bank can provide the full cacheline. Finally, a 3D SoC chip has been taped out to demonstrate the feasibility of 3D-stacked DRAM. In the chip, a 3D-stacked DRAM directly stacks on a two-layer logic chip. The experimental results show that the proposed optimizations can either effectively improve DRAM performance or significantly reduce DRAM power with negligible area overhead.

Table of Contents

List of Figures

List of Tables

Acknowledgments

My sincerest gratitude is definitely to Dr. Yuan Xie, who is my research advisor in the past six years and is also my dissertation advisor. I would like to thank him to give me the chance to work in the great team with exciting research topics. In addition to his guidance, support, and encouragement in the research activities, he also helps me recognize my weaknesses and does his best to let me know how to be stronger. Moreover, he provides me enormous opportunities to meet and communicate with other leading researchers in the world. I can't imagine how I can graduate without his constant advising and guidance.

I also want to thank Dr. Mary Jane Irwin, Dr. Vijay Narayanan, and Dr. Zhiwen Liu. As the committee members, they give me constructive advices and helps throughout my research activities. Dr. Irwin teaches me how to design a partial product multiplier, which is finally implemented in a 3D chip. Dr. Vijay helps a lot on the 3D chip design. He even funds me a laptop that I use in the daily study. Dr. Zhiwen Liu shows me the possibility of silicon-compatible Y-shape optical splitter, which is also presented in one of my papers. It is my pleasure to work with them in my prior work. I am looking forward to the collaboration in my future research.

I would like to thank Dr. Xu Cheng, Dr. Dong Tong, Dr. Yi Feng, Dr. Kui Wang, and Dr. Junlin Lu from Peking University, Dr. Shau-Yin Tseng, Dr. Yi-Ta Wu from Industrial Technology Research Institute (Taiwan), Dr. Youn-Long Lin and Dr. Tingting Hwang from National Tsing Hua University, Dr. Eren Kursun from IBM Research, Dr. Jie Liu, Dr. Nissanka B. Priyantha and Dr. Aman Kansal from Microsoft Research, Dr. Heitor S. Ramos from Federal University of Alagoas, Dr. Ke Chen from Oracle Corporation, Dr. Frank Mueller and Dr. Paul D. Franzon from North Carolina State University, Dr. Jia Di from University of Arkansas. We have closely collaborated in various work and publications. I believe the great work relationship can continue in my new career.

Thanks a lot to the previous and present MDL members: Dr. Jin Ouyang, Cong Xu, Matthew Poremba, Jing Xie, Dr. Yibo Chen, Dr. Mike Debole, Qiaosha Zou, Dr. Lian Duan, Jishen Zhao, Hsiang-Yun Cheng, Dr. Xiangyu Dong, Dr. Xiaoxia Wu, Karthik Swaminathan, Dr. Yang Ding, Dr. Kevin Irick and Misun Park. In particular, I want to thank Dr. Guangyu Sun, who is one of my best friends and research partners for his help in my research and life. Also, I want to thank Dr. Raj Acharya (department head), Kathy Zimmerman, Cindy Milliron, Beth Kennedy, Annie Royer, and Karen Corl for their many helps on the administrative work. I am proud of working with them and I do remember it in my mind forever: WE ARE PENN STATE!

Finally, I give my deepest gratitude to my family. I know my parents always give silently but never ask for any return from me. I can feel their love forever. Also, I owe my wife Dan Zhu a lot because she does great sacrifice to support me.

Dedication

This dissertation is dedicated to my parents and my wife for their endless love, support and encouragement throughout my life.

Chapter 1

Introduction

In this chapter, we first briefly review the "Memory Wall" as the motivation of the work shown in the following chapters. We then summarize the various DRAM types as the response from the industry to the "Memory Wall". Afterwards, we show the DRAM hierarchy in which all our optimization schemes occur in this work. Finally, we depict the basic DRAM commands, including Activate (ACT), Read, Write, Precharge (PRE) and Refresh (REF), along with the introduction of command timing analysis.

1.1 The "Memory Wall"

The term "Memory Wall" can be traced back to 1994, when Dr. Wulf and Dr. McKee firstly raised the word in their short paper [9]. Their concern is that the main memory will finally become the bottleneck of the whole computing system due to the diverging exponential increase in the performance of process and main memory. Equation 1.1 is given in the paper to explain the cause of memory wall, which calculates the overall access time for a memory request. As shown, the access time is determined by the performance of the processor (T_{core})[1], the performance of main memory (T_{mem}) and the cache hit rate (P_{cache_hit}). Given the modern processor has high cache hit rate (>90%), the performance of CPU dominates the overall access time as long as T_{mem} is comparable to T_{core}.

$$T_{total} = T_{core} \times P_{cache\_hit} + T_{mem} \times (1 - P_{cache\_hit}) \text{ [9]} \tag{1.1}$$

Unfortunately, as the *de facto* main memory, DRAM technology turns out to be difficult to keep the pace with the processor scaling. By leveraging the "Moore's Law", processor performance can be doubled in every 18-24 months. Even though DRAM capacity can catch up the

[1]The paper assumes the processor and cache have the same performance. Also, the authors consider a uniform cache access (UCA) architecture.

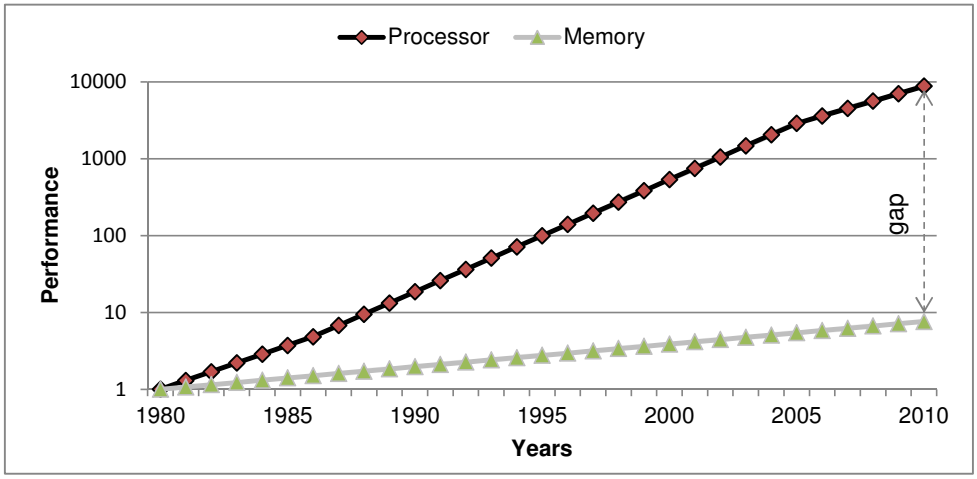


Figure 1.1: The performance divergence between processor and memory.

speed, DRAM performance can only be improved by 7% per year. Therefore, the performance gap between processor and memory increases exponentially, which is shown in Figure 1.1. As a consequence, the memory will eventually be dominant in the system performance and thus become the system bottleneck.

In fact, only the access latency is considered in the paper as the criteria of memory wall. In addition to the aforementioned latency issue, DRAM also needs to face the design challenges from the increasing bandwidth requirement, the reduction of power consumption, and the desire to further scale down the feature size. As a result, the memory wall is four-fold today: 1) **Latency**, 2) **Bandwidth**, 3) **Power**, and 4) **Scalability**. In this work, the first three issues are addressed with the corresponding optimization proposals. The goal is to make a better design trade-off among the latency, bandwidth and power. Furthermore, the design overhead is carefully analyzed to make sure the optimization schemes are practical.

1.2 The Response From The Industry

To win the war against "Memory Wall", the DRAM industry works hard to develop various DRAMs as the response to the increasing system requirements. Figure 1.2 illustrates the evolution of DRAM technologies. In particular, there are two DRAM families shown in the figure: DDRx and LPDDRx. Each family has advanced to the fourth generation (DDR4 [6] and LPDDR4 [10], respectively). DDRx is a high-performance DRAM that is usually used in PC, server and network processors. In contrast, LPDDRx is a low-power DRAM that is widely used in the mobile devices. Furthermore, some 3D-stacked DRAMs, such as Wide-IO [11], High Bandwidth Memory (HBM) [12] and Hybrid Memory Cube (HMC) [13], have been proposed in recent years. All these DRAMs are targeting at addressing one or multiple aforementioned challenges. The industry's solution can be summarized as follows.

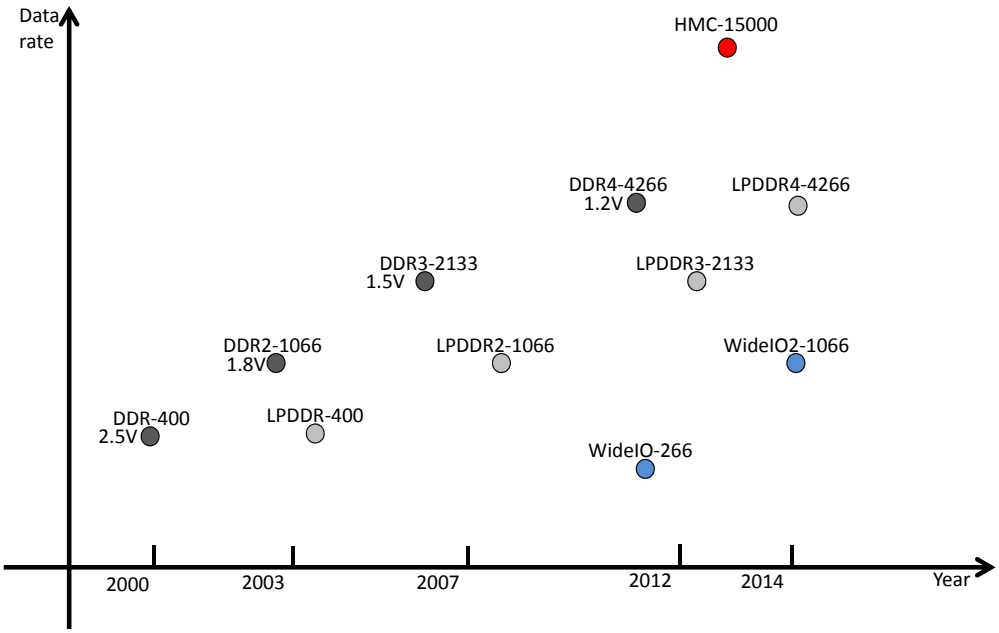


Figure 1.2: The evolution of DRAM technology. The operating frequency doubles in each generation of DDRx family. LPDDRx is introduced to reduce the power consumption. In addition, 3D-stacked DRAMs, including Wide-IO and HMC, are invented to provide high bandwidth with low power consumption.

- Latency reduction. Micron has shipped Reduced-Latency DRAM (RLDRAM) [14], which has smaller access latency. Similarly, Fujitsu has invented Fast-Cycle DRAM (FCRAM) [15] to obtain less access latency. The access latency of these two memories is improved due to the smaller load on the wordline in a smaller bank. In addition, a higher voltage is usually applied on the wordline to speed up the row opening[2].

- Bandwidth improvement. As the bandwidth is calculated as the product of data width and operating frequency (shown in Equation 1.2), DRAM vendors are striving to enhance both factors to improve the bandwidth. As shown in Figure 1.2, in each generation (from DDR [16] to DDR4), the data rate (2×frequency) is doubled to provide 2X bandwidth. HMC can even achieve up to 15Gbps data rate. On the other hand, Wide-IO and HBM have been proposed to increase the data width. Distinct from DDRx family in which the data is usually 64-bit wide, Wide-IO supports 512-bit data. HBM can even provide 1024-bit data via 8 channels.

$$\mathbf{B}_{DataBandwidth} = \mathbf{W}_{DataWidth} \times \mathbf{F}_{DataFrequency} \tag{1.2}$$

- Power reduction. The power reduction is achieved in two ways. First of all, the supply voltage drops in the DDRx family. As shown in Figure 1.2, the power supply that is 2.5V

[2]Please see Section 1.3 for the definition of these terms

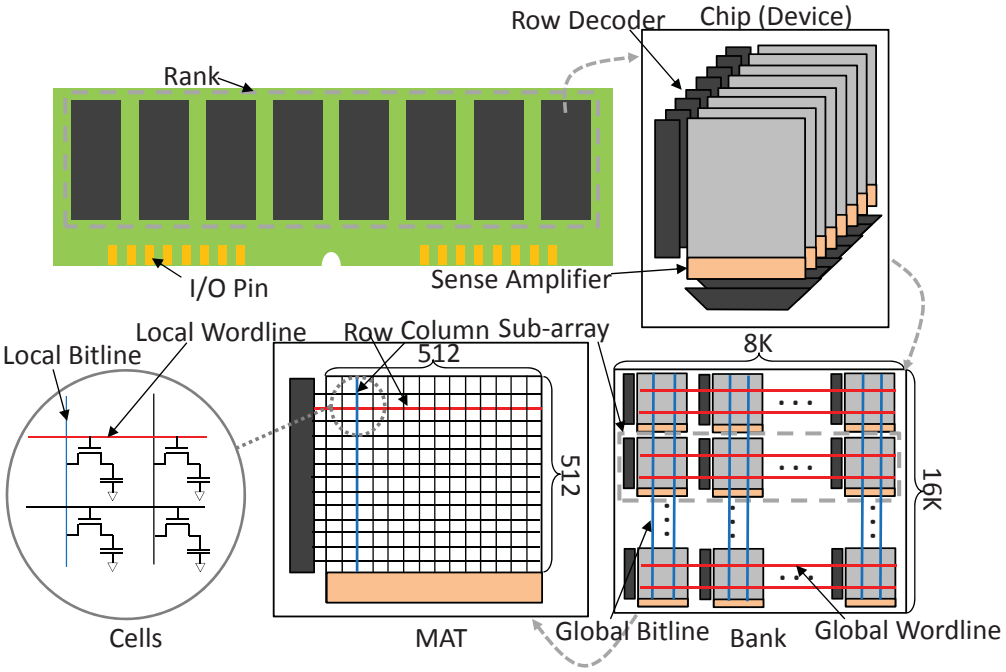


Figure 1.3: DRAM hierarchy – a 2Gb-8bank×16 example

in DDR shrinks to only 1.2V in DDR4. Both the dynamic and static power benefits from the supply voltage drop. Moreover, Low-power DDR DRAM (LPDDR) [17] (a.k.a. mobile DRAM) is introduced to further reduce the power. LPDDR has even smaller voltage than DDR. In LPDDR2 [18], the on-die termination (ODT) logic and delay-lock-loop (DLL) are removed to reduce the I/O and leakage power.

- Scalability. Taking advantage of semiconductor process scaling, DRAM process keeps scaling down accordingly. Recently, Samsung has announced to ship 20nm DDR3 soon [19]. 10nm DRAM is also on the roadmap and expected to be available in the near future. On the other hand, 3D ICs opens up another opportunity to sustain the scalability. By leveraging the Through-Silicon-Vias (TSVs), multiple DRAM dies can be stacked together and encapsulated in a single chip package. In this way, the 3D-stacked DRAM is able to increase the chip capacity, reduce the access latency and reduce power consumption at the same time.

1.3 The DRAM Hierarchy

Without loss of generality, the DRAM memory structure has a pyramid-like hierarchy, which consists of rank, chip, bank, sub-array, MAT, and cell, as shown in Figure 1.3 (from top to bottom). A rank is composed of multiple memory chips (a.k.a. device) that operate in lock-step to feed the data bus. Inside one chip, several banks are employed as cell arrays and can

be accessed independently. Therefore, bank-level parallelism is extensively studied to improve memory performance.

In fact, a bank can be further divided into many sub-arrays. All sub-arrays share the output of global row address decoder so that only one sub-array is allowed to be active at any time. In one sub-array, there are many MATs and each of them has its own row decoder and sense amplifier array as the local row buffer. Typically, a MAT is sized of 512×512 storage cells in row (wordline) and column (bitline) dimension. One storage cell is a **C**apacitor that is connected to an access **T**ransistor (so-called 1T1C structure). According to the different voltage in the capacitor, the cell holds logic '1' (V_{dd}) or '0' (0V). As the example shown in the figure, eight chips compose a rank to provide 64-bit data and each chip delivers 8-bit data (so-called ×8 chip). Every chip contains eight banks and each bank has 16K rows and 8K columns. Therefore, one bank has 32 ($=\frac{16,384}{512}$) sub-arrays and each sub-array consists of 16 ($=\frac{8,192}{512}$) MATs.

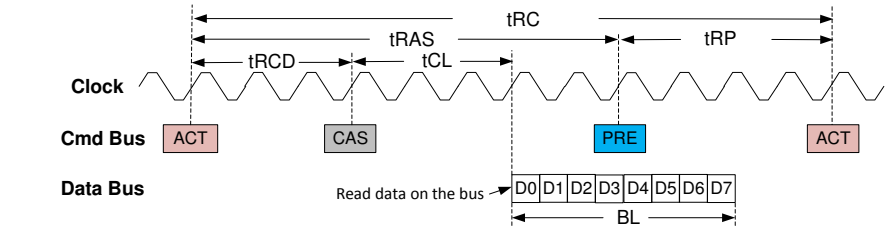
1.4 The Preliminary Of DRAM Commands

The commands a DRAM can accept can be classified into four types[3]: row activation `ACT` or `RAS` (Row Address Strobe), column read/write `CAS` (Column Address Strobe), precharge `PRE`, and refresh `REF`. Only `CAS` has data transfer between memory controller (MC) and memory modules while other three commands are only used for either access preparation (`ACT` and `PRE`) or data protection (`REF`). The purpose of the four commands is as follows.

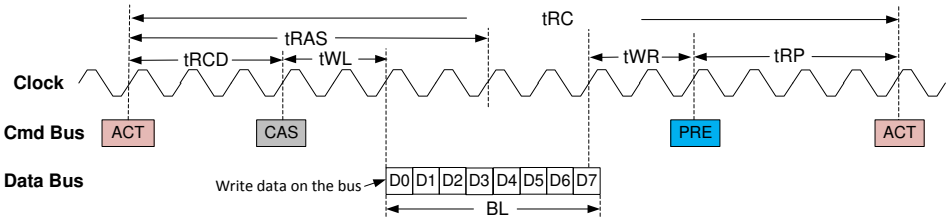
- `ACT`: An activation is to open a row in a bank. It enables the row address decoder to assert a wordline. The data will then be sensed and locked in the sense amplifier.
- `CAS`: A column read/write is to fetch a data segment from the sense amplifier. It also accounts for the data traverse on the internal bus and read/write buffer.
- `PRE`: A precharge is to close a row in a bank. It resets the sense amplifier and bitline to the idle state for the next activation.
- `REF`: A refresh is to protect the data. It simply reads out and writes back the data. As a result, it can be treated as a pair of activation and precharge.

Figure 1.4a illustrates the command flow for a read transaction. The flow can be simplified as a loop of "ACT-CAS-PRE-ACT". First of all, the target bank should be activated by receiving an `ACT` command. After the RAS-to-CAS delay (tRCD), the data has been sensed in the sense amplifier so that a `CAS` can be issued to fetch the data. After the read latency (tCL), the data shows up on the data bus. As double data rate (DDR) is widely applied in commodity DRAM, the data is sampled by MC at both rising and falling clock edge. In DDR3, the data burst length (BL) is 8, which means DRAM spends 4 clocks to burst out the data.

[3]In fact, more memory commands are available in JEDEC specification, such as implicit auto precharge, post-CAS, and commands for low-power mode. The ACT-CAS-PRE-ACT command order, however, is still applied. As a result, we only show these four basic commands in this work.



(a) DRAM Read



(b) DRAM Write

Figure 1.4: DRAM commands for read and write transaction. Note that a write can lock a bank for a longer time before the bank can be closed (precharged).

Note that the bank is unable to be closed immediately after `CAS`. Instead, the bank needs to spend time on restoring the data due to the destructive array access. The time period tRAS accounts for such constraint. Once the tRAS constraint is satisfied, a `PRE` can be issued to close the bank. As shown, there is also a minimum distance between the precharge and the next activation, which is denoted as tRP. As the bank is fully precharged, it can be reopened by the next activation. Therefore, the term *row cycle* tRC is the primary measurement of DRAM latency as it indicates how fast a bank can be reopened. As shown in Equation 1.3, the row cycle can be calculated as the sum of tRAS and tRP.

$$tRC_{read} = tRAS + tRP \tag{1.3}$$

$$tRC_{write} = tRCD + tWL + BL/2 + tWR + tRP \tag{1.4}$$

The command flow for a write transaction is shown in Figure 1.4b. Distinct from read, write has a shorter write latency tWL (tWL <tCL). Moreover, since DRAM has to change the value of the storage cell, it needs to wait for the write recovery latency tWR before the bank can be closed. Note that the write recovery latency starts right after the completeness of data burst on the bus. Therefore, the row cycle of a write transaction is different from that of read transaction. As shown in Equation 1.4, it is the sum of tRCD, tWL, BL/2, tWR and tRP.

1.5 The Structure Of The Dissertation

The rest of dissertation consists of six chapters. Chapter 2–Chapter 6 are the body of the dissertation. Chapter 7 concludes the dissertation with the summary of the proposals shown in this work and the discussion of future work. In general, the first five chapters follow the rule of "WWWH", which stands for "**W**hat is the problem", "**W**hy the problem is important", "**W**hat is the proposed solution", and "**How** much improvement the solution can achieve". Correspondingly, each chapter has the dedicated introduction (W), background/motivation (W), the proposal and design overhead (W), and experimental results (H).

Chapter 2 shows the in-depth details of CREAM, which is a concurrent refresh aware DRAM architecture. Sub-rank and sub-array level refresh are proposed to isolate the refresh from normal memory access. In this way, it is unnecessary to lock the whole memory during a refresh. Chapter 3 presents the Half-DRAM, in which the DRAM is redesigned so that only a half of row is activated. As the active row size shrinks, Half-DRAM can effectively reduce the activation power. In addition, it can also achieve performance improvement once the sub-array level parallelism is employed. Chapter 4 targets at the reduction of the precharge overhead and thus proposes a novel precharge policy, which is noted as "Lazy Precharge". Distinct from prior studies, Lazy Precharge can effectively reduce the number of precharge in that it allows multiple activations to share a common (lazy) precharge.

Chapter 5 and chapter 6 focus on the optimization in 3D-stacked memory. In chapter 5, 3D-SWIFT is proposed in which a 3D Wide-IO memory [11] is further split into multiple sub-banks. Stemming from the improved wiring capability in the third dimension, each sub-bank can provide a full cacheline. As a result, 3D-SWIFT can significantly reduce the activation power and thus break the TAW (two activation window) constraint for performance improvement. Chapter 6 demonstrates the feasibility of 3D IC and the on-chip memory stacking technology. The design flow of a 3D SoC is depicted in this chapter. The front-end and bank-end flow are described separately. In addition, a custom memory controller is designed to interact with the 3D-stacked DRAM.

Chapter 2

CREAM: The Relax Of Refresh Penalty

In recent years, DRAM density has been improved dramatically as DRAM technology evolves along with CMOS process scaling. Recently, 16Gb DRAM chip was defined in DDR4 specification [6]. The increasing density, especially the increasing row number, introduces significant refresh penalty because more rows are required to be refreshed at a time. As a result, it takes longer time and more power to complete a refresh. The larger refresh penalty is no longer trivial as it can result in negative impact on memory performance and power. For example, significant performance degradation has been observed [20]. Consequently, the DRAM system should be carefully designed to mitigate the refresh penalty. In this work, we propose a Concurrent-REfresh-Aware Memory (CREAM) architecture that allows memory access and refresh to be executed *concurrently* under the pre-defined power constraint. The contributions of this work can be summarized as follows:

- **Trade-off between power constraint and performance.** The capability of concurrent refresh in today's commodity DRAM is limited due to power (current) constraint. To relax the constraint, sub-rank-level refresh (SRLR) is proposed so that fewer banks are refreshed simultaneously to reduce the refresh power consumption. The saved power can in turn be used for memory accesses to improve performance.

- **Sub-array-level refresh.** With SRLR alone, the whole sub-rank is still locked during a refresh. To further improve the performance, a novel sub-array-level refresh (SALR) is proposed to increase the probability of memory concurrency between normal access and refresh. To our best knowledge, this is the first work to deploy SALR for the reduction of refresh penalty.

- **Concurrent refresh design.** By combining SRLR and SALR, CREAM can issue a memory access even when a refresh operation is ongoing, which means that the access and

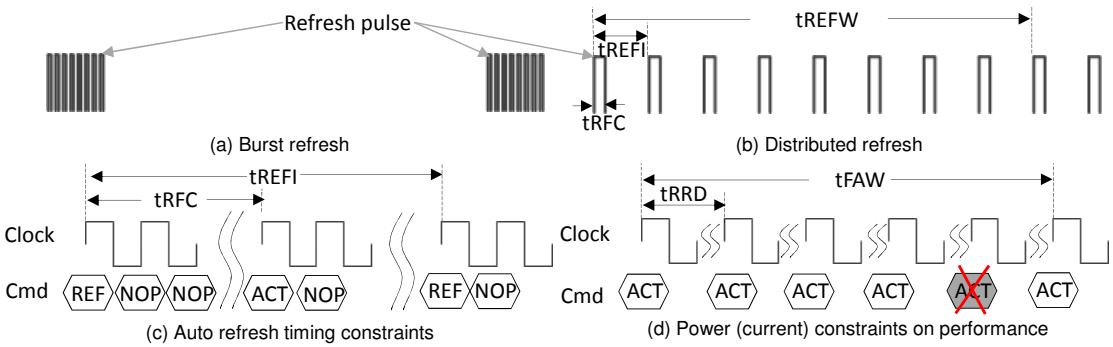


Figure 2.1: Refresh basics. (a) Burst refresh; (b) Distributed refresh; (c) Refresh command and related timing constraints; (d) impact of power constraint on performance.

refresh can be served in parallel. In this way, CREAM can effectively hide the refresh latency overhead and meanwhile improve the performance. Moreover, a quasi-ROR interface is designed to make CREAM compatible with JEDEC-DDR standard, with negligible hardware overhead.

- **Refresh-aware memory optimization.** Two mechanisms, sub-array round-robin and dynamic refresh scheduling, are devised for further performance improvement. Sub-array round-robin evenly distributes the fine-grained refresh in a bank to avoid conflict between memory access and refresh. In addition, the concurrent refresh can be scheduled dynamically according to the status of a sub-rank.

2.1 Background And Motivation

In this section, we first briefly review the refresh overhead in the conventional DRAM architecture, and then discuss the ineffectiveness of existing solutions to mitigate the refresh overhead, which motivates this work to come up with a more effective approach to re-architect the DRAM structure.

2.1.1 Preliminary Of DRAM Refresh

Due to the leakage current, the charge in each storage cell gradually dissipates and eventually the stored data can be lost as the state becomes unrecognizable. The lifetime of the data is noted as *retention time*. To prevent the data loss, refresh is required to recharge the storage cell within the retention time. In theory, a refresh scheme at any granularity is valid as long as the cell can be refreshed in time. The refresh in modern DRAM memory, however, is usually organized at rank level, which means *all chips in the rank and all banks in the chip are refreshed in lockstep*. No memory access is permitted to a rank when it undergoes a refresh. In other words, refresh and memory access are mutually exclusive to each other in the scale of rank.

Basically, two common refresh scheduling schemes have been developed to refresh the whole

memory system, which are shown in Figure 2.1. **Burst Refresh** requires the memory controller (MC) to send multiple successive refresh pulses in burst within a short period while **Distributed Refresh** mandates the MC to evenly send refresh pulses to DRAM chips periodically. Even though both methods can complete all refreshes before the data loss, burst refresh induces a relatively longer memory stall that degrades the performance. In addition, burst refresh consumes large power within a short time interval, which requires higher current density that is challenging for commodity DRAM. Therefore, distributed refresh is preferable for MC design due to the amortization of refresh overhead in terms of performance and power. In addition to these two schemes, there are two refresh modes in the commodity DRAM. **Auto Refresh** is one mode that MC initiates a refresh by sending a refresh command (`REF`) to DRAM chips. Alternatively, **Self Refresh** is employed as a low-power mode so that a DRAM chip can complete the refresh without the intervention from the MC. The refresh row address is generated by an internal row counter. Since only Auto Refresh is used during a memory access, we focus on this refresh mode in this work.

2.1.2 The Source Of Refresh Penalty

The primary refresh penalty is the mutual exclusion between refresh and memory access due to the sharing of pre-decoded row address in a bank. As a result, one bank can only serve one out of four memory operations at a time: activation (`RAS` or `ACT`), precharge (`PRE`), read/write (`CAS`), and refresh (`REF`). Consequently, all memory operations can only be issued in sequence, which we call *intrabank-zero-parallelism*. The intrabank-zero-parallelism requires memory operations to wait if a refresh is being processed. Moreover, since all banks are refreshed simultaneously, refresh further suppresses bank-level parallelism, which we denote as *interbank-zero-parallelism*. The combined effect of intrabank- and interbank-zero-parallelism mandates all memory accesses to be delayed until the refresh completes. Considering the refresh should be taken periodically, an application may experience a long, periodic delay, which can incur significant performance degradation.

In JEDEC-DDR standard [6, 21], three timing parameters are used to define the refresh characteristics of a DRAM chip. These timing parameters are illustrated in Figure 2.1(b).

- **tREFW** is the refresh window that refers to the cell retention time.
- **tREFI** is the time interval of two `REF` commands. Since distributed refresh is applied, the value of tREFI is determined by tREFW and the refresh count in a tREFW.
- **tRFC** is the refresh latency of one `REF` command, which is also known as refresh cycle.

Table 2.1 shows the values of these three parameters as the chip capacity ranges from 1Gb to 16Gb[1]. According to the operating temperature, tREFW can be either 64ms (Temp$\leq$85°C) or 32ms (85°C<Temp<95°C). From the table, the refresh count remains at 8,192 even though the

[1]The values of 16Gb DRAM has not been defined in JEDEC DDR4 specification. The data listed in the table is based on our projection according to the trend from 1Gb to 8Gb.

Table 2.1: Refresh related parameters with various DRAM capacities (based on Micron DDR3-1333 data sheet)

Capacity	**1Gb**	**2Gb**	**4Gb**	**8Gb**	**16Gb**
Row Num.	16K	32K	64K	128K	256K
Refresh Num.	8K	8K	8K	8K	8K
Rows/`REF`	2	4	8	16	32
tREFW(ms)	64/32	64/32	64/32	64/32	64/32
tREFI(µs)	7.8/3.9	7.8/3.9	7.8/3.9	7.8/3.9	7.8/3.9
tRFC(ns)	110	160	260	350	450
tRFC/tREFI	1.41% 2.82%	2.05% 4.11%	3.34% 6.68%	4.34% 8.68%	5.77% 11.54%
tRRD(ns)	6	6	6	6	6
tFAW(ns)	30	30	30	30	30
tRC(ns)	51	51	51	51	51
IDD5B(mA)	165	111	148	230	260
IDD0(mA)	65	41	47	85	95
Version	G[22]	K[7]	E[8]	D[23]	N/A

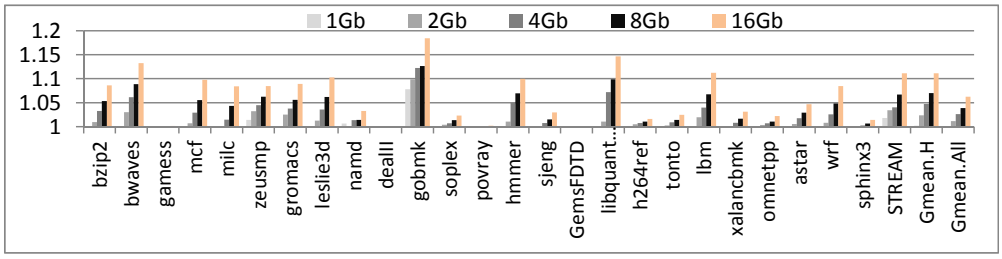
memory capacity increases. Therefore, tREFI sustains at 7.8µs (3.9µs) regardless of the DRAM evolution.

From Table 2.1, it is clear that the refresh cycle tRFC dramatically increases as the chip capacity keeps growing. Correspondingly, the refresh overhead that is defined as tRFC/tREFI becomes larger, which means the following memory accesses should wait for a longer time. The large performance degradation has been observed in [20] and we also see the similar result. Figure 2.2a illustrates the potential performance gain if there is no refresh in DRAM. The average speedup for all benchmarks and memory-intensive benchmarks is 6.3% and 11.2% with 3.9µs refresh rate, respectively. An interesting observation from Figure 2.2a is that some benchmarks, such as `gobmk` and `libquantum`, have larger performance drop (15.7% and 14.3%) than the maximum refresh overhead (11.54%), which indicates the refresh generates accumulative effect on an application and therefore aggravates the performance degradation.

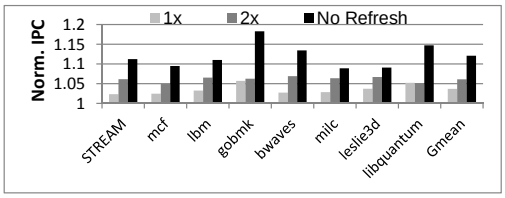
2.1.3 Power Constraint–The Secondary Cause

In addition to the aforementioned refresh penalty caused by intrabank- and interbank-zero-parallelism, power (current) constraint is the secondary factor that necessitates the mutual exclusion between refresh and memory access. Even though the refresh power constraint is not detailed in JEDEC standard and DRAM vendors' data sheet, we leverage the well-known power constraint on activation to help the explanation.

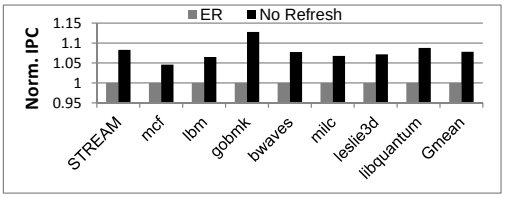
To ensure that the peak current consumption does not exceed the pre-defined threshold, a DRAM chip prohibits very frequent activations. Correspondingly, two timing parameters are used to limit the `ACT` frequency. tRRD is known as activation-to-activation delay, which determines the minimum interval between two successive `ACT`s. Moreover, the four-activation window constraint tFAW only allows four `ACT`s to be issued in any tFAW cycles. Once the MC issues an `ACT` to open a row, the next `ACT` can only be issued after tRRD cycles. Since tFAW is usually greater than



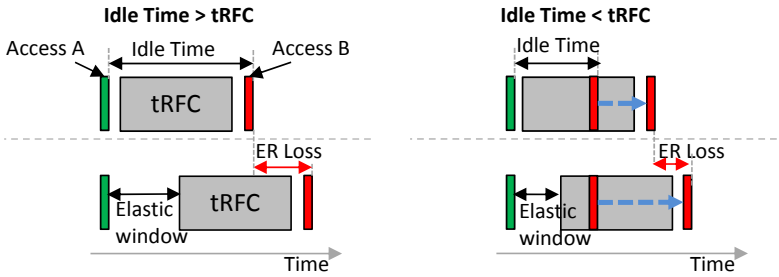
(a) Performance gain when no refresh is applied



(b) Ineffectiveness of fine-grained refresh defined in DDR4 [6]



(c) Ineffectiveness of elastic refresh in [20]



(d) The cause of ER's failure

Figure 2.2: The motivation experimental results. (a) performance gain without refresh as chip size ranges from 1Gb to 16Gb; (b) the ineffectiveness of fine-grained refresh in DDR4; both ×2 and ×4 modes are shown by the normalized IPC to the baseline ×1 mode; (c) the ineffectiveness of ER; (d) cause of ER's failure.

the sum of four tRRDs (see Table 2.1), the fifth `ACT` (highlighted in gray) can only be issued after tFAW cycles even if it satisfies tRRD constraint. Therefore, tFAW constrains the performance in a DRAM system that the *close-page* row buffer management policy is applied [24]. As CREAM aims at the server memory that close-page policy is commonly deployed, tFAW effectively limits CREAM's performance as well.

Since a refresh can be simply considered as a pair of activation and precharge (see Section 2.2), it should abide with the same power constraints too. From Table 2.1, a refresh consumes about $3X$ currents (IDD5B) than an activation (IDD0), which means it almost reaches the peak current. As a result, no activation can be issued during a refresh.

Table 2.2: Settings for fine-grained refresh in DDR4 [6]

	1×	2×	4×	8×
tRFC(ns)	350	260	160	75
tREFI(µs)	3.9	1.95	0.975	0.4875

2.1.4 The Ineffectiveness Of Existing Refresh Scheduling Methods

To mitigate the refresh penalty, the new DDR4 standard [6] proposes a fine-grained refresh scheme, in which refresh frequency can be two (2× mode) or four times (4× mode) higher than the original refresh rate (1× mode). Since more refreshes are involved, fewer rows need to be refreshed during a single refresh. In this way, the fine-grained refresh can reduce tRFC so that the waiting memory accesses can be served earlier to improve performance. Table 2.2 lists the timing changes when fine-grained refresh is applied[2]. We implement this fine-grained refresh in DRAMSim2 [26] and select some memory-intensive benchmarks from SPEC2006 suite to evaluate its effectiveness (see Section 2.4 for the detail of simulation platform). As shown in Figure 2.2b, on average 2× and 4× mode can only achieve 3.9% and 6.4% performance gain, respectively, which is consistent with JEDEC report [25]. Therefore, the proposed refresh scheme is ineffective to eliminate the refresh penalty, still leaving a big performance gap to be filled.

The main problem of such fine-grained refresh is that it only reduces the waiting time caused by intrabank- and interbank-zero-parallelism rather than eliminate either of them. As a result, all banks are still locked during the refresh even though the refresh cycle becomes smaller. Moreover, the higher refresh rate adversely affects the performance as well. Compared to the 1× mode where an application only needs to wait for one tRFC, the application running with the fine-grained refresh scheme can experience multiple refreshes in the same tREFI. The accumulative refresh penalty, which is calculated as N_{stall}×$tRFC_{2\times}$ ($tRFC_{4\times}$) where N_{stall} is the maximum stall number, could be larger than the single refresh cycle $tRFC_{1\times}$ and thus even induce performance degradation in 8× mode as shown in [25].

Another famous refresh scheduling policy has been proposed in [20], which is called *elastic refresh* (ER). ER leverages the 8-tREFI refresh scheduling flexibility defined in DDR standard to hide the refresh penalty. Different from prior work that defers a refresh until the memory is idle (DUE) [27], ER leaves an extra elastic window and expects that there are incoming memory accesses in the window. If so, ER further defers the refresh until either no access comes up during the window or it has deferred eight refreshes and thus hits the 9-tREFI refresh limitation. As the window width is critical to ER's performance, ER has capability to adjust the window width in-the-flight based on the average idle time and the number of deferred refreshes. We also implement this approach to measure the performance improvement. Figure 2.2c illustrates the results. Unfortunately, no result can be comparable to the ideal case where no refresh is applied. On average, there is 7.8% performance gap between ER and the ideal case. In particular, the potential improvement room can be as much as 12.1% (*gobmk*). Similar to the aforementioned fine-grained refresh, ER also renders the ineffectiveness to eliminate the refresh penalty.

[2] 8× mode is mentioned in [25] but not defined in [6].

The reason of ER's failure can be simply explained as follows. ER uses the average idle cycles as window width for the possible coming access. The average idle cycles calculated by arithmetic mean, however, means only half of accesses that have smaller interval than the idle cycles can benefit from the elastic idle time while the other half can't. Figure 2.2d shows two examples for the performance loss caused by ER. Firstly, let's assume 1) the elastic window starts after access A (green block) as memory becomes idle; and 2) access B (red block) is too far away from A to catch the window. As a result, as shown in the left part of Figure 2.2d, a deferred refresh is launched after the window. In this way, ER adversely destroys the potential performance gain since the idle time between A and B is actually long enough to hide a refresh. On the other hand, the right part gives another example where the idle time between A and B is shorter than tRFC. Consequently, B should wait for a while (blue dotted line). With ER, B needs to wait for a longer time because of the late launch of the refresh after the elastic window. As a result, even though ER can allow some memory requests to be served without stall, it may also add additional delay to other accesses, which in turn offsets its effectiveness.

Since the existing refresh policies still leave a big room for further performance improvement, CREAM is proposed as an essential solution to effectively reduce the refresh penalty. The goal of CREAM is to hide the refresh penalty and thus perform like a "non-refresh" DRAM.

2.2 The Proposed Memory Architecture – CREAM

As shown in Figure 2.3 (left), a basic refresh is a two-stage operation. First, the refresh row is opened to load the data from the cell to the sense amplifier (SA). The cell is then restored to logic '0' or '1'. Once the restoration completes, DRAM moves to the second stage, in which a pre-charge is applied to close the row by resetting the bitline and SA for the following memory accesses. Note that the column select signal CSL is turned off so that the refresh is invisible for the rest of the system. In this way, a refresh can be treated as a pair of activation and pre-charge. The two-stage refresh implies an important feature that CREAM leverages: **the range of data movement during a refresh is limited between a sub-array and the corresponding local SA so that it does not induce resource contention with other memory accesses outside the sub-array** (e.g., the contention on global row buffer, I/O gating logic, or I/O bus). Consequently, it provides an opportunity for a bank to serve a refresh and memory access in parallel as long as they can be completely isolated from each other.

2.2.1 Sub-Array-Level Refresh (SALR): Reduction Of Refresh Conflicts

According to Figure 2.3, one sub-array has a local (dedicated) bitline and local SA array so that it has the potential to complete the refresh without competing for the sharing resources with other sub-arrays. Therefore, we develop a technique called sub-array-level refresh (SALR) to eliminate the intrabank-zero-parallelism. Because of the sharing of row address logic, however,

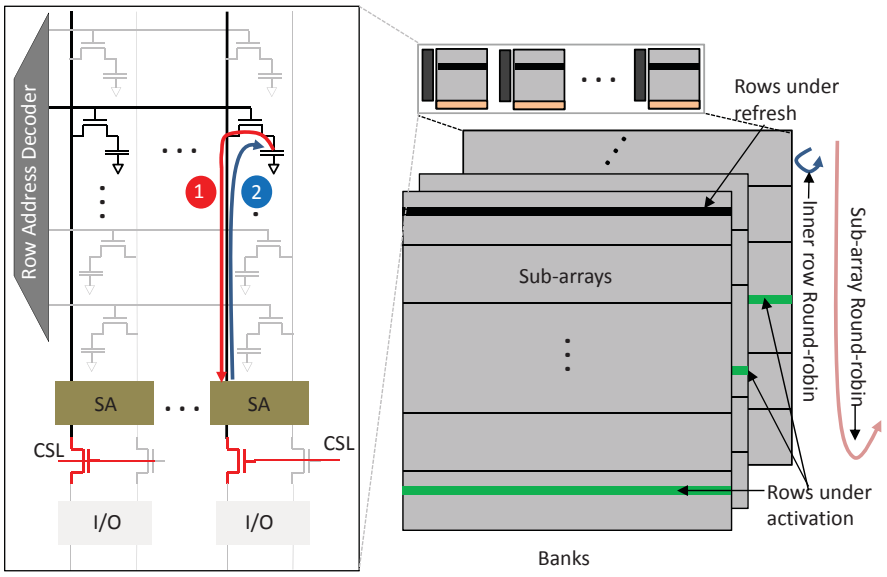


Figure 2.3: Enabler of concurrent refresh. Left: a refresh in a sub-array is a two-stage operation. The row is firstly open to recharge the cell and then the row is closed by a precharge. Right: a refresh and an activation can be performed concurrently in any banks as long as no sub-array conflict occurs

the sub-array-level refresh is not available in conventional DRAM design. To enable SALR, small modification is required and Section 3.2 will present the design detail.

SALR can significantly reduce refresh penalty due to the low probability of refresh conflict, which in turn indicates the high probability of concurrent refresh that helps hide refresh penalty. For any memory access, the probability of sub-array conflict can be presented as Equation 2.1, where N_{SA} is the number of sub-arrays in one bank. It is straightforward that the probability is inversely proportional to the number of sub-arrays if the memory access is uniformly distributed in the bank. Note that N_{SA} may not necessarily be equal to the real number of sub-arrays in a bank because multiple sub-arrays can group together as a single sub-array. For instance, N_{SA} can be only eight in a 4Gb bank that has 128 sub-arrays (64K rows with 512 rows per sub-array). According to Equation 2.1, the probability of a sub-array conflict is only 1.56% as 64 sub-arrays are employed. Thanks to the low probability, sub-array conflict can be effectively reduced to guarantee the high refresh concurrency.

$$P^{1bank}(N_{SA}) = \frac{1}{N_{SA}} \tag{2.1}$$

2.2.2 Sub-Rank-Level Refresh (SRLR): Consideration Of Power Constraint

The aforementioned fine-grained refresh does not take into account the limited current budget of DRAM. Since all memory operations must abide with the power constraint, no memory access can be issued if all banks are refreshed simultaneously, which in turn kills the memory concurrency.

Therefore, the only way to enable concurrent refresh is to reduce the current consumption by a refresh. Based on our observation from [28], **the total current consumed by a refresh is proportional to the number of banks that are refreshed simultaneously**[3]. As a result, it is straightforward that reducing the number of banks that are refreshed together can save power accordingly. CREAM employs this idea to develop a sub-rank-level refresh scheme (SRLR). The original rank with eight banks are further divided into two (SR2), four (SR4) or eight (SR8) sub-ranks. As a consequence, each sub-rank has four, two or one bank that are refreshed simultaneously. As fewer banks are involved in a refresh, CREAM makes use of the saved current to enable memory access in parallel. In this way, SRLR relaxes or even eliminate the interbank-zero-parallelism because it is no longer necessary to lock all banks during the refresh.

2.2.3 The Combination Of SALR And SRLR

The combination of SALR and SRLR provides CREAM high flexibility of fine-grained refresh due to the significant alleviation of intrabank- and interbank-zero-parallelism limitations. Once a sub-rank is being refreshed, memory accesses can be issued to any sub-ranks as long as there is no sub-array conflict. To obey the power constraint, power should be reassigned in the new architecture. In this work, we conservatively assume that halving bank number can save current for one activation. Based on the assumption, a refresh in SR2, SR4 and SR8 consumes the same current as three, two, and one normal `ACT`s[4]. As a result, these three refresh schemes have distinct capabilities to compete with `ACT` in a window, which is illustrated in Figure 2.4. For example, as a refresh in SR4 is identical to two `ACT`s in terms of current consumption, it reserves two slots for refresh and thus leaves another two `ACT` slots for the memory accesses, which is noted as 2-ACT. Similarly, 1-ACT and 3-ACT are available in SR1 and SR8, respectively.

Nonetheless, fewer banks per refresh require more refreshes in a tREFI. As shown in Figure 2.4, only two refreshes are required in SR2 while eight refreshes are executed in SR8. The more refreshes may induce performance drop as DDR4 does (see Section 2.1.4). Thanks to SALP, the probability of sub-array conflict in a sub-rank, which is shown as Equation 2.2, is low enough to effectively reduce the conflict. In Equation 2.2, N_{SR} is the number of sub-ranks and N_B is the number of banks in a rank. When there are eight sub-ranks and one bank has 64 sub-arrays, the probability of sub-array conflict is only 0.2%, which implies high possibility of concurrent refresh.

$$P^{subrank}(N_{SR}) = \frac{1}{N_{SR}} \times (1 - (1 - P^{1bank}(N_{SA}))^{\frac{N_B}{N_{SR}}}) \tag{2.2}$$

[3]Even though the relation between current consumption and bank number is given by partial-array self refresh (PASR), we believe the trend is also applicable to auto refresh in standard DDR SDRAM.

[4]Even though the reassignment is inaccurate, it is close to the truth. The sensitivity study is given in Section 2.4.3.2 by relaxing the power assignment.

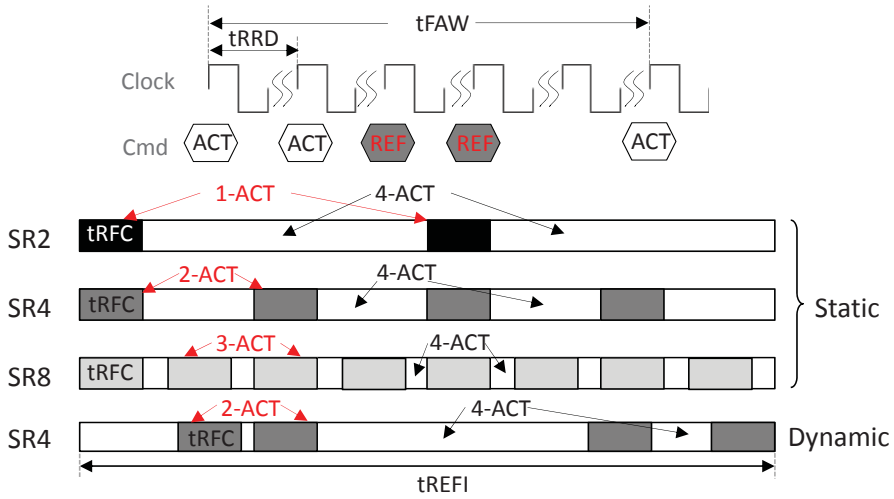


Figure 2.4: The different capabilities of concurrency in SR2, SR4 and SR8 refresh. A refresh in SR4 occupies two slots in the four-activation window based on current consumption. As a result, only two `ACTs` are available during a refresh. Similarly, one and three `ACTs` are available in SR2 and SR8, respectively.

2.2.4 Refresh-Aware Optimization

In addition to the aforementioned SRLR and SALR, we consider two refresh-aware optimizations to further improve the performance.

2.2.4.1 Sub-Array Round-Robin

The probability calculation given in Equation 2.1 and 2.2 assumes that the memory accesses are evenly distributed in a bank. In the reality, however, memory accesses are not evenly distributed. Instead, the spatial locality is more or less reflected in the access pattern. In addition, row bits are usually placed in the most significant bits (MSBs) in physical address mapping to augment either row buffer hit rate or bank-(rank-) level parallelism (or both). As a result, it is more likely memory accesses concentrate in a sub-array rather than scatters over the whole bank. To further avoid the sub-rank conflict, CREAM prioritizes the round-robin among sub-arrays over the inner row round-robin within a sub-array (see Figure 2.3). Once a refresh has been completed in a sub-array, next refresh will move to next sub-array rather than next row in the same sub-array. In this way, CREAM can control the sub-array conflict as a rare case.

2.2.4.2 Dynamic Refresh Scheduling

The first three fine-grained refreshes shown in Figure 2.4 adopt *static* refresh scheduling policy. The meaning of "static" is two-fold. First of all, all sub-ranks are refreshed in order (1-..-N_{SR}). In addition, the refresh is distributed evenly in a tREFI so that the time interval between two sub-rank refreshes can be calculated as tREFI/N_{SR}. The drawback of static refresh scheduling is similar to the baseline refresh scheme because MC initiates a refresh regardless of the memory status. When many memory accesses are queuing in the MC, static refresh may degrade the

performance as it wastes the precious ACT bandwidth. Alternatively, a *dynamic* refresh scheduling policy can be employed to address this issue.

According to the status of each sub-rank, the refresh can be executed out of order so that the idle sub-rank is refreshed at first. In this way, the dynamic refresh scheduling policy can further hide the refresh penalty at the sub-rank level. Since all sub-ranks must be refreshed within a tREFI, the sub-rank that is always busy will be forced to be refreshed at the end of a tREFI. In this work, MC simply checks the command queue to obtain the sub-rank status. Once the queue has few memory accesses, MC assumes the sub-rank is idle and starts refreshing it. Section 2.4 will evaluate the effectiveness of the dynamic refresh scheduling policy.

2.3 The Design Of CREAM

We present the design details of CREAM architecture in this section, with descriptions of protocol changes, hardware modification, and overhead analysis.

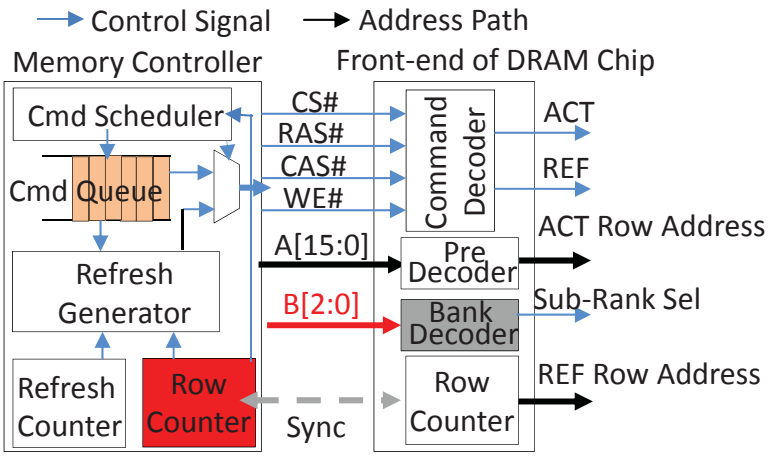
2.3.1 Protocol Change For Dynamic Scheduling And PAAR

To support dynamic scheduling, some modification should be made on the memory interface. As shown in Figure 2.5a, the original auto refresh that is also known as *CBR* (CAS#-Before-RAS# Refresh) should be changed to quasi-*ROR* (RAS#-Only Refresh). In CBR, all address signals, including bank ID (BA[2:0]), are ignored as "Don't Care" bits because the row address is generated by the internal row counter in DRAM chip. In contrast, original ROR requires the MC to maintain the row counter and explicitly send the refresh row address to DRAM chip. CREAM adopts the mix of ROR and CBR so that sub-rank ID is put on BA[2:0] along with REF command but the address bits are still "Don't Care" (so-called quasi-ROR), as illustrated in Figure 2.5a. DRAM chip reuses the bank decoder to decode the sub-rank ID and then the generated 'sub-rank sel' signal is delivered to the target sub-rank. The row address is still maintained by the internal row counter.

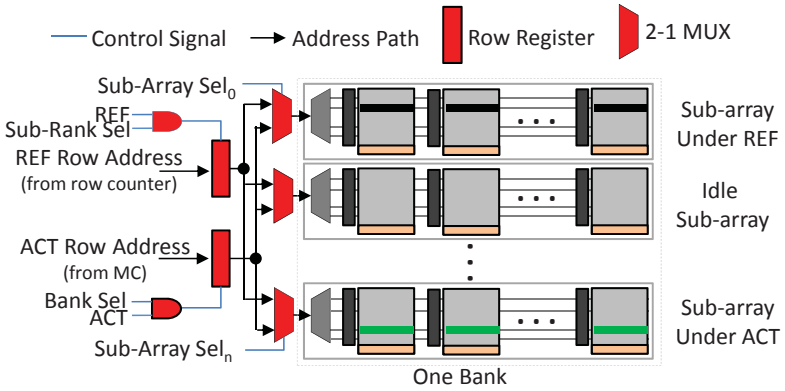
On the other hand, the MC of CREAM needs to know the current refresh row for the avoidance of sub-array conflict. Therefore, a replica of row counter must be deployed in MC and used by the command scheduler. Note that one rank has a dedicated row counter in MC. The row counter keeps counting even if some ranks are in self refresh mode. In this way, the synchronization (highlighted by dotted gray line) between these two row counters is established to make sure no refresh violation occurs[5].

In addition, this quasi-ROR interface can be easily extended to an ROR interface to support the partial array auto refresh (PAAR) (see Section 2.5). At this time, MC puts both bank ID and row address to tell a sub-rank which sub-array should be refreshed.

[5]A safer but more expensive synchronization scheme is to make the row counter inside DRAM chip visible to MC so that MC can access it immediately after the exit of self refresh, which is the same as MC accesses a mode register.



(a) Interface protocol modification



(b) Row logic modification

Figure 2.5: Hardware modifications to enable SALR and SRLR. All additional logics are highlighted in red. (a) Quasi-ROR is employed to put sub-rank ID on BA[2:0]; (b) Two registers are used to separate refresh row address and activation row address.

2.3.2 The Enabling Technology For SALR

Figure 2.5b illustrates the modification to conventional DRAM structure to enable SALR. Two row address registers are added to isolate the refreshing row address from the activation address. In addition, a multiplexer array is employed to select the row address for the purpose of either activation or refresh. Once the refresh command `REF` comes in, the dedicated register is enabled to store the refresh row address from the internal row counter. In addition, the corresponding multiplexer is set to select the output of the refresh register. The target sub-array uses the latched refresh row address to complete the refresh in tRFC cycles. During the refresh, the signal CSL (see Figure 2.3) is turned off so that the refresh is isolated from the rest system. As a result, any other sub-arrays can serve normal `ACT`, `CAS` and `PRE` by using the activation row register.

Note that the proposed architecture is much simpler than the one in SALP [4], where multiple registers are employed for the parallel activation. Instead, only one register is dedicated to latch

the row address for an `ACT` command, which means that **at anytime only one sub-array can be activated**. As a result, CREAM can completely reuse the legacy logics in conventional DRAM to serve an `ACT/CAS/PRE` under various timing constraints. Given a register costs much more area than a multiplexer, the simplicity also reduces the area overhead due to the use of less registers. In addition, CREAM has better scalability than SALP. The number of register in CREAM remains at two while the number increases in SALP, if more sub-arrays are parallelized.

2.3.3 Design Overhead Analysis

In this section, we evaluate the design overhead caused by CREAM. As the cost of DRAM is sensitive to the area, we firstly focus on the area overhead on the DRAM chip. Then, we discuss the reuse of pin to prove that CREAM does not induce any pin-out change.

2.3.3.1 Overhead On DRAM Chip

The area overhead on the DRAM chip should be carefully considered as DRAM cost is highly sensitive to the area and power. In this work, we implement the additional logics shown in Figure 2.5b by Verilog HDL and then synthesize the design by Synopsys Design Compiler [29] with TSMC 45nm low-power technology for the area and power analysis. For the comparison, the extra logics introduced in SALP are also synthesized. All implementations are applied to a 2Gb DRAM chip that has 32K rows (or 64 sub-arrays) and the address register is consistent with SALP, which is 40-bit wide.

As shown in Table 5.1, when both have eight sub-arrays (SA8), CREAM only consumes 786µm^2 area and 427µW power while SALP has to consume 2,325µm^2 area and 1,311µW power. On the other hand, as the number of sub-array increases, CREAM presents much better scalability than SALP as well. For example, the area overhead of SALP almost doubles as the sub-array numbers increases to 16. In contrast, CREAM that employs 64 sub-arrays (SA64) has moderate area and power increase due to the simple pass-transistor-based implementation of multiplexor array. Given a 50mm^2 DRAM die with 64 sub-arrays, the overall area overhead can be calculated by 1,975µm^2×8banks, which is only 0.016mm^2 (0.032%). Therefore, the area overhead on the DRAM chip is negligible.

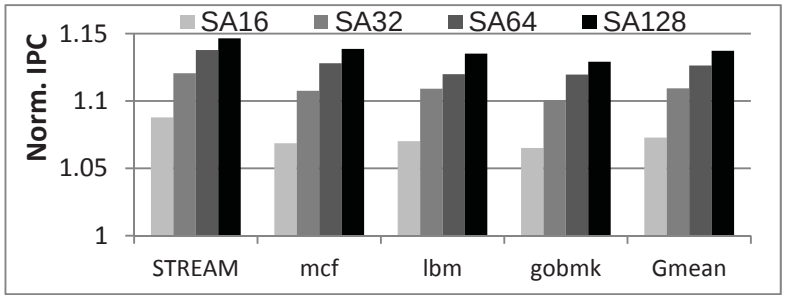
Furthermore, as summarized in [30, 31], it is more costly when the modification is applied to the bitline and sense amplifier while it is less costly when the change is conducted in the row logic and I/O. Considering SALP changes the bitline layout but CREAM only puts additional hardware around row logic and never touches the DRAM core, we believe the architecture of CREAM introduces much smaller hardware overhead than that of SALP.

2.3.3.2 Interface Protocol Change

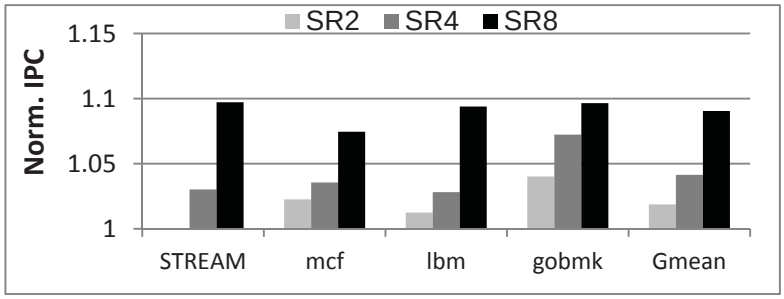
The only change at the interface is the reuse of the link bank ID (BA[2:0]) that was ignored before. No additional pin is introduced in CREAM. As a result, CREAM is completely compatible with the conventional JEDEC-DDR interface protocol. Furthermore, if the dynamic scheduling and

Table 2.3: Synthesis Result Comparison for a 2Gb DRAM Chip

	CREAM				**SALP** [4]	
	SA8	SA16	SA32	SA64	SA8	SA16
Power(µW)	427	528	613	**770**	**1,311**	2,609
Area(µm^2)	786	1,007	1,316	**1,975**	**2,325**	4,651



(a) Sub-array (normalized to SA8)



(b) Sub-rank (normalized to no sub-rank)

Figure 2.6: Design space exploration for sub-rank and sub-array number.

PAAR are not employed, no change is even needed to the interface. Also note that the new row counter in MC has fewer bits than the peer in DRAM chip because MC is only aware of sub-array ID. As the maximum number of sub-arrays in a single DRAM chip is 128, the row counter for one bank is a 7-bit register. Given a four-rank DRAM system, the overall overhead is a 40bit register (=4ranks×(7bit row counter + 3bit sub-rank ID)), which is also negligible. In contrast, Smart Refresh [32] needs 768KB storage for the row counters and RAIDR [33] requires at least 1.25KB storage for the bloom filter. As a result, CREAM is compelling in terms of area overhead.

2.4 Evaluation Results

In this work, gem5 [34] is used as our simulation platform. Instead of the time-consuming full-system (FS) simulation, system-call emulation (SE) mode is adopted for the sake of simulation speed. The well-known DRAMSim2 simulator [26] is integrated into gem5 and modified to implement the proposed concurrent refresh schemes. Table 5.2 shows the gem5 setup. The selected

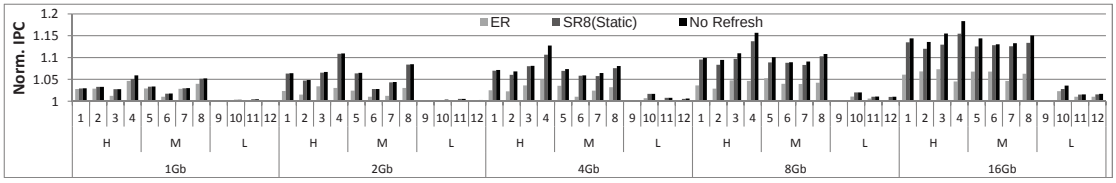


Figure 2.7: One-core simulation results with 4GB memory. All results are normalized to IR.

Table 2.4: Simulation Platform Configuration

Cores	1/4, ALPHA, out-of-order
CPU Clock Freq.	3 GHz
LDQ/STQ/ROB Size	32 / 32 / 128 entries
Issue/Commit Width	8 / 8
L1-D/L1-I Cache	32kB / 32kB 4-way 2-cycle latency
D-TLB/I-TLB Size	64 / 48 entries
L2 Cache	Shared, Snooping, 1MB, LRU 8-way, 10-cycle latency
Memory	JEDEC-DDR3, 4GB, 64bit I/O bus, 8 banks(×8), 666MHz(DDR-1333), tRCD-tCAS-tRP-tWR 10-10-10-10, address map: row-column-bank-rank,

SPEC2006 CPU benchmark with reference input size [35] and STREAM with all functions [36] are evaluated as multi-programmed testbench. We run all benchmarks for 500 million instructions to warm up the cache and then the following 100 million instructions for the statistics. The instructions-per-cycle (IPC) is used as the performance criteria through the evaluation. All timing parameters are excerpted from Micron's data sheet [22, 7, 8, 23] and the refresh-related parameters are listed in Table 2.1. As CREAM targets at the massive memory system used in server or datacenter where the operating temperature is usually greater than 85°C [37, 38], all simulations are run under extended temperature range, where tREFW=32ms and tREFI=3.9µs.

Two refresh scheduling schemes, **immediate refresh** (IR) and **elastic refresh** (ER) are implemented in DRAMSim2 as the references. IR immediately initiates a refresh command once the refresh counter is timed up. This is the baseline since it simulates the refresh scheduling that is widely used in modern MCs. As mentioned in Section 2.1.4, ER devised in [20] defers the refresh for an elastic window and expects some memory accesses will come in. The window width is dynamically tuned according to the calculated average idle cycle and number of deferred refreshes. In addition, the memory with **no refresh** is also evaluated as the ideal case.

According to the motivation results shown in Figure 2.2a, we classify the benchmarks into three categories and select four benchmarks as the representatives for each category, which are symbolized as *H*, *M*, and *L* for high (>10), medium ([1,10]), and low (<1) miss per kilo instructions (MPKI) of last level cache (LLC). Each benchmark is assigned a number for the simplicity as shown in Table 5.3. Both single-core and four-core simulations are done for the evaluation.

Table 2.5: Benchmark Classification and Model Settings

	#**Benchmarks(MPKI)**
H	[1]STREAM(34.96), [2]mcf(16.26), [3]lbm(31.92), [4]gobmk(38.35)
M	[5]bwaves(6.07), [6]milc(5.13), [7]leslie3d(4.30), [8]libquantum(6.95)
L	[9]gamess(0.02), [10]namd(0.12), [11]h264ref(0.35), [12]sphinx3(0.09)
Memory Models (4GB, DDR3-1333)	
1Gb	Micron MT41J128M8[22], 8B(bank)×8, 4-rank
2Gb	Micron MT41J256M8[7], 8B×8, 2-rank
4Gb	Micron MT41J512M8[8], 8B×8, 1-rank
8Gb	Micron[23] & JEDEC-DDR4[6], 8B×8, 1-rank
16Gb	JEDEC DDR4[6], 16B×8, 1-rank
Concurrent Refresh Symbols	
SAx	sub-array settings, x: 8/16/32/64/128
SRy	sub-rank settings, y: 2/4/8, SALR+SRLR
SRLRz	sub-rank settings, z: 2/4/8, SRLR only

We simply duplicate four copies of each benchmark for the four-core simulation. In addition, five 4GB memory models are simulated with various chip capacities. In particular, 1Gb (2/4Gb) memory is the ordinary Micron DDR3-1333 that has four (two/one) ranks and eight banks. 8Gb memory has 128K rows to simulate Micron TwinDie DDR3-1333. Finally, 16Gb memory refers to DDR4-1333 that has 128K rows and 16 banks[6]. On the other hand, concurrent refresh with different sub-array and sub-rank numbers are noted as **SA*x*** and **SR*y***, where x stands for the sub-array number that can be 8-128, and y is the sub-rank number that can be 2/4/8. Moreover, **SRLR*z*** stands for a memory system that only SRLR is applied. Table 5.3 gives the comprehensive information about the memory models and the symbol of concurrent refresh with different settings.

2.4.1 Design Space Exploration

Before the performance evaluation, we conduct a design space exploration to determine the optimal sub-rank and sub-array numbers. Only class H benchmarks are employed for the evaluation. We first sweep the sub-array number from 8 (SA8) to 128 (SA128) to find the optimal sub-array number. All results in Figure 2.6a are normalized to SA8. As shown, SA16 and SA32 can achieve 7.2% and 10.4% performance improvement on average than SA8. In addition, SA64 can further outperform SA32 with 3.1% performance gain. However, only 1% benefit is obtained as the sub-array number increases from 64 to 128. As a result, either SA32 or SA64 can be used as the optimal solution. Without special statement, SA64 is applied in all following experiments (SA32 is used in 1Gb simulation because it is the maximum number of sub-array).

With the optional sub-array number, the result of sub-rank exploration is given by Figure 2.6b. In general, SR8 has the highest speedup (9.7%) over the baseline. SR2 and SR4 have less than 5% speedup due to the insufficient memory concurrency. In particular, SR2 even incurs 1.7% performance drop in `STREAM`. The reason is that SR2 has two refreshes in a tREFI. The

[6]In fact, the data rate 1333MHz is not supported in [6]. We simply double the bank number to simulate an equivalent DDR4 module, such as DDR4-1600.

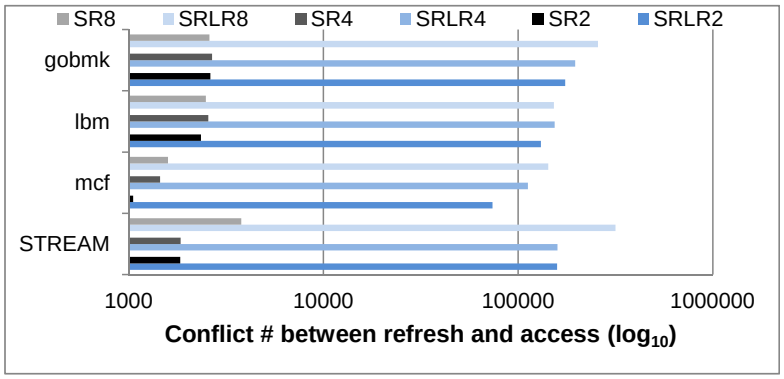


Figure 2.8: Refresh conflict number under SRLR-only and SR

benchmark experiences two refresh stalls that can offload the limited performance gain from 1-ACT concurrency.

2.4.2 Single-Core Simulation Results

Single-core simulation is done with the optional sub-rank and sub-array numbers given above to evaluate the effectiveness of concurrent refresh. Two messages can be taken according to the result presented in Figure 2.7: 1) the three categories render distinct sensitivity to refresh overhead. The refresh has significant impact on H and M while the L class that has less memory intensity is almost immune to the increasing refresh penalty; and 2) concurrent refresh (SR8) can be comparable to the ideal case (around 99% of No Refresh). On average, SR8 can outperform IR and ER by 9.7% (12.9%) and 6.6% (8.1%) in 8Gb (16Gb) memory system, respectively.

2.4.2.1 Proof Of Effectiveness Of SALR

To verify the effectiveness of SALR, we collect and compare the number of conflict between refresh and memory access in both SRLR-only and SR models. For the sake of accuracy, only those memory accesses that satisfy the power constraint are counted. Since SRLR-only model only has sub-rank-level parallelism, the value actually reflects the number of sub-rank-level conflict. In contrast, sub-array-level conflict is collected in SR models. As shown in Figure 2.8, SR model can achieve $9X$-$90X$ conflict reduction over SRLR-only model with only 0.1% sub-array conflict. An interesting observation is that SALR can effectively limit the conflict number with modest increase as sub-rank number increases from SR2 to SR8. Moreover, there is even a small decrease in `lbm`(3) and `gobmk`(4). Alternatively, the conflict in SRLR-only system dramatically increases when more sub-ranks are deployed. Consequently, all benchmarks have performance drops when the sub-rank numbers changes from two to eight. The reason for this phenomenon is similar to the DDR4 fine-grained refresh (see Section 2.1.4) that more refreshes are invoked when sub-rank number increases. As a result, it is possible for a benchmark to experience multiple stalls due to the more frequent refreshes. Therefore, SRLR-only model cannot work well and SALR is necessary for the success of concurrent refresh.

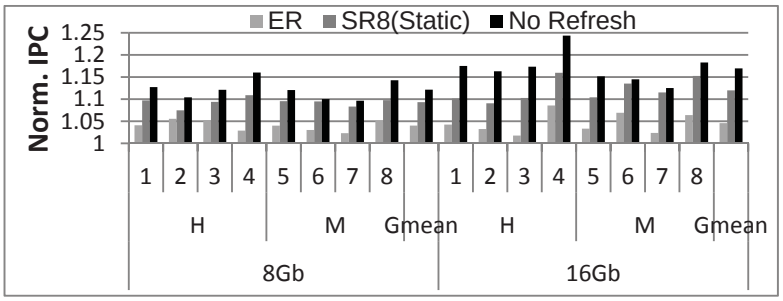


Figure 2.9: Four-core simulation with 8GB memory. All results are normalized to IR.

2.4.2.2 Effectiveness Of Dynamic Refresh Scheduling

Figure 2.10a presents the effectiveness of dynamic refresh scheduling scheme. The geometric mean value is shown in the figure. Once the dynamic scheduling is applied to SR4 (SR4(D)), it can obtain 4.3% performance improvement than the other SR4 model that employs static scheduling (SR4(S)). The result is even close to SR8(S). However, no further gain is observed in SR8(D). As shown in Figure 2.4, SR8 has eight refreshes so that it leaves little room to dynamically schedule these refreshes.

2.4.3 Sensitivity Study

In this section, the various core number and memory size are explored firstly, followed by the relax of power constraint. Finally, the impact of different address mapping schemes and smaller refresh cycles are evaluated to measure the performance of CREAM.

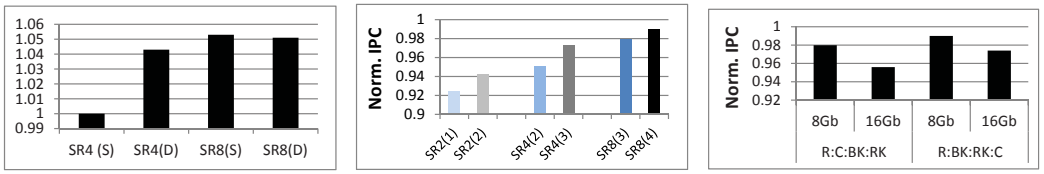
2.4.3.1 Core Number And Memory Size

To measure how much gain CREAM can accomplish in a multi-core system, four-core[7] simulations are run under 8GB memory. Only *H* and *M* class along with 8Gb and 16Gb models are used to mimic a heavy-workload environment. The benchmarks in *H* and *M* are simply duplicated for four copies and assigned to each core as multi-programmed simulation. From Figure 2.9, it is clear that CREAM can achieve more performance gain than that of ER. In general, CREAM has 9.3% and 11.9% improvement than IR while ER can only get 4% and 4.5% improvement, respectively. The gap between CREAM and No Refresh, however, becomes bigger (-2.1% for 8Gb and -4.3% for 16Gb) due to the increased memory intensity (MPKI is about four times larger).

2.4.3.2 Relax Of Power Constraint

The compromise between performance and power always exists in DRAM system. Even though the battle will continue, we could anticipate the current consumption for sub-rank refresh can be lower than our conservative settings. We relax the power constraint so that two, three, and four `ACT`s can be issued in SR2, SR4, and SR8, respectively. The last configuration that

[7]As gem5 SE mode needs large memory space in the host machine for the simulation, this is the maximum core number we can afford.



(a) Effectiveness of dynamic refresh (b) Impact of relaxing power constraint (c) Impact of address mapping schemes

Figure 2.10: Simulation results for sensitivity study. In (a), all results are normalized to SR4(S); In (b) and (c), all results are normalized to No Refresh.

enables concurrent refresh with normal four-activation window constraint is interesting. As long as no sub-array conflict occurs, the refresh can be completely hidden in the background without introducing any refresh penalty. We name it *background refresh* as the solution of concurrent refresh in the near future. Figure 2.10b illustrates the `ACT` number configuration (in the parenthesis) and simulation result with 8Gb model. The performance can be dramatically improved by relaxing the power constraint for all benchmarks. Specifically, the background refresh (SR8(4)) can almost perform the same as No Refresh (98.9%). From this study, we can learn that power constraint is crucial for the success of concurrent refresh. If the constraint can be relaxed, even SR4(3) can have comparable performance (97.3%) to the ideal case.

2.4.3.3 Address Mapping Policy

Figure 2.10c shows the result when CREAM adopts different address mapping policies. In this work, we assume the row bits are placed as MSBs to maximize the row-buffer hit rate, rank- and bank-level interleaving, which is commonly deployed in state-of-the-art MCs. The left part is the result of "R(row):C(column):BK(bank):RK(rank)" that is used as the default address mapping policy through this paper. The other two policies, "R:C:RK:BK" and "R:BK:RK:C" are also evaluated. "R:C:RK:BK" has very similar result as "R:C:BK:RK" so that the result is omitted. Alternatively, "R:BK:RK:C" has even better performance because it re-maps more memory accesses into the same bank, which helps the other banks has sufficient time to hide the refresh.

2.4.3.4 Smaller Refresh Cycle

All above simulations conservatively sustain tRFC even though SALR and SRLR are employed. In fact, similar to the fine-grained refresh in DDR4, tRFC in CREAM can become smaller if fewer sub-ranks are refreshed together. We rerun simulation with the smaller tRFC given in Table 2.2. However, the performance enhancement is trivial (result is omitted). Compared to the smaller refresh cycle, the concurrent refresh is more effective to elevate the performance. The reason is straightforward: if memory access can be served during a refresh, the smaller refresh cycle won't change a lot.

2.5 Discussion

So far, we have discussed CREAM and shown the stand-alone performance benefit. In fact, CREAM can cooperate with other techniques to help the performance. In this section, we give several examples to show the potential of CREAM.

2.5.1 Integration Of Partial Array Self Refresh (PASR)

PASR can be seamlessly integrated into CREAM to achieve power efficiency. In PASR, only sub-arrays that have "hot" data are refreshed while the data in rest sub-arrays will be lost. Beneficial from SALR, CREAM naturally supports PASR without extra design effort. In addition, CREAM can be easily extended to enable partial array auto refresh (PAAR) so that MC has the capability to issue selective refresh as well. Like PASR, PAAR is visible to OS for the decision of which part of array should be protected by refresh. Considering auto refresh consumes larger power than self refresh, we believe PAAR is more meaningful for the DRAM power efficiency as well as performance improvement. The decision of selective refresh is out of scope of this paper even though several studies have been done recently [39, 40].

2.5.2 Support For 3D-stacked DRAM

The emerging 3D-stacked DRAMs [13, 41, 42] have been reported to have severe thermal challenges due to the higher power density. The higher temperature in turn results in even larger refresh frequency (e.g., 8ms refresh as 95°C<Temp<115°C [41]), which aggravates the refresh overhead correspondingly. CREAM can be easily extended to 3D DRAM to hide the more frequent refresh and thus reduce the refresh penalty. Considering the high bandwidth requirement, we believe CREAM is a promising solution in 3D-stacked DRAM.

2.5.3 The Scalability For Future Refresh Technology

The simple row-by-row refresh, which is performed as a pair of `ACT` and `PRE`, cannot scale as the number of rows per refresh keeps increasing. Since tRC is the row cycle, the refresh delay caused by row-by-row refresh can be calculated as $N_{row} \times$tRC, which can be much larger than tRFC(see Table 2.1). Alternatively, multi-row parallel refresh could be implemented to make sure such many rows can be refreshed in a tRFC. The parallel refresh reduces the number of available sub-arrays that weakens the effectiveness of CREAM. However, as the result shown in Section 2.4, the performance can be guaranteed if 32 or 64 sub-arrays are available in CREAM.

2.6 Related Work

Enormous work has been done to hide the refresh overhead in DRAM. According to the merits of technology, we classify the prior work into four categories, which are discussed below.

2.6.1 Concurrent Refresh

Kirihata *et al.* [43] proposed a concurrent refresh scheme that can carry on the bank-level refresh, which is similar to our SRLR scheme. However, this concurrent refresh targets at embedded DRAM (eDRAM) with small capacity so that it does not consider the power constraint, which is critical to the standard DRAM. Also, as mentioned in Section 2.4.2, the bank-level refresh still incurs lots of refresh conflicts without sub-array-level refresh. Our proposed CREAM technique improves beyond their scheme to achieve more performance gain.

2.6.2 Refresh Deferring

To alleviate the increasing refresh penalty, some studies make use of the flexibility of refresh rescheduling in $8 \times tREFI$. The basic idea is straightforward, which defers refresh when memory is busy and hope the deferred refresh can be served later as memory becomes idle. Ipek *et al.* proposed a refresh scheme called DUE to simply defer a refresh until the memory is idle[27]. The drawback of DUE was observed by Stuecheli *et al.* [20]: it can delay the future memory accesses that come out during a refresh. As a result, elastic refresh (ER) was proposed to further defer a refresh for certain cycles even the memory is idle [20]. As mentioned in Section 2.1.4, ER can adversely introduce extra delay that limits the further performance improvement. Nevertheless, all refresh deferring scheduling methods have a common problem: the number of deferred refresh is limited to eight. Once the deferred refresh number hits the limitation, immediate refresh should be forced. As a consequence, it is ineffective for memory-intensive applications that can quickly accumulate deferred refreshes to reach the constraint.

2.6.3 Refresh Reduction

In addition to the above prior arts, several studies have been done to improve the power efficiency by reducing the refresh count. Prior work [39, 43] takes advantage of PASR (Partial Array Self-Refresh) [6, 21] to reduce self-refresh counts. By leveraging SALR, PASR can be seamlessly integrated into CREAM to improve the power efficiency. On the other hand, Smart Refresh [32] leverages the fact that a read/write is equivalent to a refresh due to the destructive array access. However, the size of the dedicated counter for each row, which is proportional to the total row number, can be too large to afford (e.g., 1.5MB in a 32GB system). In addition, the effectiveness of smart refresh varies for different memory access patterns.

Recently, RAIDR [33] devises a refresh scheduling policy that differentiates the retention time and correspondingly applies different refresh rate to cells that have different retention times. A bloom filter is deployed to simplify the refresh tracking. However, RAIDR incurs significant area overhead in MC since it requires at least 1.25KB storage to implement the bloom filters. In addition, it may have reliability issue due to the problem of variable retention time (VRT) [24], where the retention time of one cell can suddenly change due to the leakage current. Alternatively, the overhead of CREAM is negligible and all sub-arrays are still refreshed at a conservative rate.

2.6.4 Refresh Prediction

A hardware-based predictor is developed where the memory accesses are classified into three groups and the maximum overhead associated with the refresh group is calculated [44]. In contrast, a software-based predictor is designed to eliminate the conflict between the refresh generated by memory controller and the memory requests generated by CPU [45]. As the CPU deals with the refreshing, it has full knowledge to interleave these two tasks to avoid conflict. Such work is orthogonal to CREAM.

2.7 Summary

As mentioned, the refresh penalty is no longer negligible as DRAM capacity keeps growing. CREAM is proposed to mitigate the increasing refresh overhead. With the power constraint, CREAM leverages sub-rank-level refresh (SRLR) and sub-array-level refresh (SALR) to make better trade-off between performance and power. In addition, the optimization technologies, such as sub-array refresh round-robin and dynamic refresh scheduling, are designed to help the performance improvement. CREAM does not introduce additional pin-out and only incurs negligible hardware overhead. The experimental results show that CREAM can achieve as much as 12.9% and 7.1% performance gain over the conventional memory and the memory that uses elastic refresh scheduling policy, respectively. Moreover, CREAM can be simply extended to support PASR/PAAR, 3D-stacked memory, and future aggressive refresh technologies.

Chapter 3

Half-DRAM: The Rethinking Of Fine-grained Activation

The power consumption for DRAM memory can compose a significant percentage of the total power consumption in modern computing systems, especially for server and data center systems. For example, prior work has demonstrated that DRAM can consume significant amount of power (sometimes more than 25% of the total power in a datacenter) [46, 47, 48, 49, 50]. As a result, how to improve the power efficiency of DRAM is one of the major challenges in the memory architecture design. Furthermore, the performance improvement of DRAM for both traditional commodity DRAM [21] and emerging 3D-stacked DRAM [11] is limited due to the power constraints. For example, the well-known timing constraints tRRD and tFAW (tTAW in 3D wide-IO DRAM standard [11]) must be obeyed, which limits the activation frequency and thus diminishes the opportunity of potential performance gain from better memory parallelism.

The *activation and pre-charge* power is the major contributor to the DRAM power consumption. The activation and precharge power can be around 25% of the total DRAM power (as illustrated in Figure 2 and the detail analysis in Section 3.1.1). Consequently, fine-grained activation techniques [2, 3] have been proposed to reduce activation power[1]. These work, however, suffers from either large area overhead or significant performance degradation due to the severely suppressed bank bandwidth. As a result, they are impractical to be implemented in modern memory systems unless the fine-grained activation is re-designed to mitigate the overhead on either performance or area. In this work, we propose a novel DRAM architecture called *Half-DRAM* to address the design challenge of fine-grained activation. In Half-DRAM, only half of a row is activated such that the activation power is reduced substantially. Furthermore, as the fine-grained activation relaxes the power constraints, Half-DRAM can further improve the memory performance with sub-array level parallelism [4].

[1]Without specific comment, the term "activation power" represents both activation and precharge power consumption in the rest of paper.

In general, our contributions can be summarized as follows.

- We demonstrate that the prior work on fine-grained activation techniques can result in significant memory performance degradation without careful design. We show in-depth details of the bandwidth degradation of previous work, which directly motivates our work.
- We propose *Half-DRAM* as an effective solution, targeting at low activation power and high memory bandwidth. In Half-DRAM, the *1RD-2HFF* layout is designed to reorganize the cell arrays in order to enable half-bank level activation and thus reduce activation power without sacrificing memory bandwidth. To our best knowledge, this is also the first work to enable the fine-grain activation without performance overhead by reorganizing the DRAM.
- We further propose the *Half-DRAM-2Row* technique to enhance the performance by relaxing the power constraint in Half-DRAM. This technique integrates sub-array level parallelism [4] into Half-DRAM to exploit the half-bank level parallelism. The experiment results verify that Half-DRAM can achieve better performance with lower power consumption, which is promising for future memory systems.

3.1 Background And Motivation

In this section, we first review the breakdown of DRAM power consumption memory and the data overfetching in DRAM to help understand the innovative techniques in Half-DRAM. Then, we show the power inefficiency in DRAM and the impracticability of prior fine-grained activation techniques.

3.1.1 The DRAM Power Breakdown

To evaluate the DRAM power consumption, we leverage Micron's DDR3 power calculator [51] and run `STREAM` benchmark to collect the related statistics. Figure 3.1a shows the power breakdown with the simulation results[2]. In the simulation, aggressive fast-exit powerdown mode is adopted so that DRAM is powered down whenever the request queue is empty and the system is idle (The experiment setup is described in Section 5.4).

The power breakdown in Figure 3.1a can be clustered into 3 categories in Figure 3.1b: (1) *Background Power.* The background power takes up 31.6% of total power that accounts for the static power and refresh power, which consists of the powers from 'ACT_STBY' (active standby), 'PRE_STBY' (precharged standby), 'ACT_PDN' (active powerdown), 'PRE_PDN' (precharged powerdown) and 'REF' (refresh). (2) *RD/WR/Termination Power.*[3] About 43.5% of the power comes from RD/WR/Termination which represents the power for the data movement, including the powers from 'RD' (read burst), 'WR' (write burst), 'READ I/O' (read bus

[2]Due to space limitation, we do not explain how DRAM power is calculated. The reader can refer to Micron's technical note [52] for the detail.

[3]RD/WR power accounts for the power dissipation on the internal bus during the data transfer, which includes both read/write FIFO access and internal bus traverse. Alternatively, Termination power is generated due to the on-die termination and delay-locked loop (DLL) at chip interface.

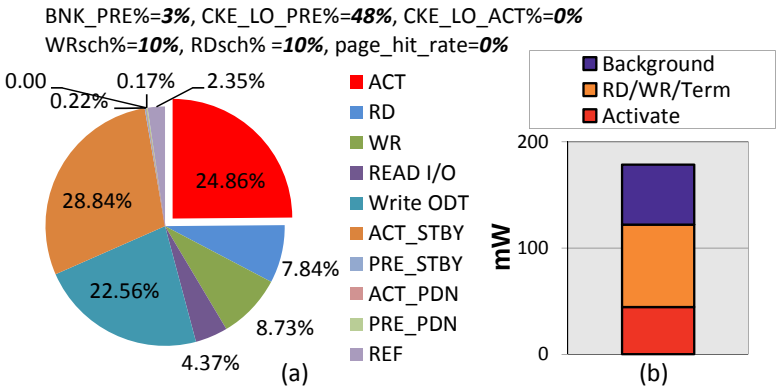


Figure 3.1: Power breakdown of STREAM

driver), and 'Write ODT' (write signal on-die termination). (3) *Activation Power*. This consists of 24.9% of the DRAM power. Many prior work has been proposed to reduce the *Background Power* [32, 53, 54, 39, 55] or *RD/WR/Termination Power* [56, 18], In this paper, our focus is on the minimization of the *Activation Power*.

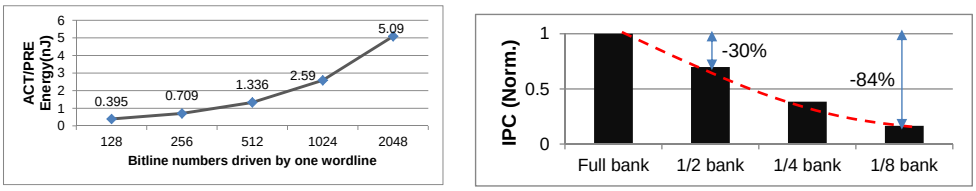
3.1.2 Row Overfetching And N-bit Prefetching.

To reduce the activation power, we first need to understand how the data are fetched in a DRAM memory. In JEDEC-DDR, there are two terms describing the internal data fetching: *Row Overfetching* and *n-bit Prefetching*. As shown in Figure 1.3, *Row Overfetching* mandates the entire row to be activated (black block) even though only a small portion of data are fetched at a time (red block). In our baseline model, a 16Kb row is activated while only 128b data are fetched per request. Ideally, row overfetching is supposed to be helpful for performance since it prevents repeated activations if multiple requests access the same row (so-called row buffer hit). It is supposed to also improve power efficiency since the activation power can be amortized over these accesses. Unfortunately, in modern CMP architectures the application interference from other cores randomizes the requests and thus lowers the row buffer hit rate [3, 24]. Moreover, memory controller should take into account the starvation and fairness issue among requests. Therefore, only a few accesses (<5) can be served for one activation [57]. The low row utilization incurs the power inefficiency because the row overfetching consumes lots of power from activation.

On the other hand, *n-bit Prefetching* is widely used in the DDRx DRAM family [21, 6], where $\boldsymbol{n}$ stands for the multiplier of the width of internal data bus over I/O bus. 2-/4-bit prefetching is implemented in DDR and DDR2 while 8-bit prefetching is in DDR3 and DDR4[4]. The n-bit prefetching is used to address the asymmetric bus frequencies between the I/O bus and the internal bus. As highlighted in red color in Figure 1.3, the data frequency on I/O bus is 1.6GHz (double data rate with 800MHz clock) while the internal bus only runs at 200MHz[5]

[4]Please do not confuse the term '8-bit' as the 8b data width used in many other literatures. In DRAM, 8-bit prefetching in fact determines the internal data fetching primitives and the corresponding burst length.

[5]The internal bus can run faster with narrower bus width. For simplicity, we assume all data is given in one transfer as the data bandwidth is constant.



(a) Energy Proportionality to Bitline Number (b) The Impact of Reduced Data Bandwidth

Figure 3.2: Motivation study. (a) The activation/precharge energy proportionality to the number of bitlines based on CACTI-3DD [1]; (b) The Impact of reduced data bandwidth in prior work [2, 3]. These are simulation results of the STREAM benchmark, and are normalized to the speedup of the baseline full bank activation.

with single data rate. Since data bandwidth is calculated as the product of data width and data frequency, 128b data is fetched at each time to provide the same bandwidth as the I/O bus requires. Accordingly, the data burst length is eight and only 16b data are sampled per clock edge within the bank.

3.1.3 The Opportunity For Activation Power Reduction

The Power Inefficiency in Overfetching. The first issue we seek to address is the power inefficiency caused by the row overfetching. To understand why and how the row overfetching introduces power inefficiency, CACTI-3DD [1] is deployed to further break down the power inside a DRAM chip. The result is shown in Table 3.1. It is obvious that the power is mainly consumed on the bitline (local bitline) and sense amplifier (local SA), where activation and precharge operations manifest. Moreover, **the activation power is proportional to the number of bitlines being activated during a memory access**. Figure 3.2a shows the change of activation energy as the number of bitlines driven by a single wordline increases, where the energy proportionality is clearly captured. For future memory chips with larger capacity and more bitlines, this power efficiency problem of row activation will obviously become even worse.

We also use data from industrial reports to further prove the necessity of fine-grained activation. This time, we compare the **pure** current dissipated by activation in two DRAM chips that have 16Kb and 8Kb row buffer sizes, respectively. According to the power calculator [52], the pure activation current is calculated by Equation 3.1, where IDD0 is the raw activation current during a row cycle tRC, and IDD2N and IDD3N are the static currents for the situation that all banks are idle or at least one bank is active, respectively. Table 3.2 shows the power parameters we collect from Micron DDR3 DRAM power datasheet [7]. With these parameters, the activation current consumed for opening a 16Kb row is 16mA and becomes 10mA for 8Kb row. As a result, 37.5% activation current is reduced by halving the row size, which demonstrates the effectiveness of fine-grained activation. Note that the power saving is less than the expected 50%, which stems from the larger static current IDD3N in a 16Kb row. If we hypothetically assume that 8Kb row has the same IDD3N as 16Kb has (37mA), then the current saving can be as much as 43.7% and

Table 3.1: DRAM Area and Power Breakdown by CACTI-3DD [1]

Area per Chip (mm^2)			
DRAM cell	14.677	sense amplifier	3.189
row predecoder	0.051	column decoder	0.002
local wordline driver			6.789
Total area			37.129
Energy per MAT (nJ)			
local bitline	1.336	local wordline	0.005
local SA	0.117	column select line	0.08
row decoder	0.004	column decoder	0.001

Table 3.2: Power Parameters based on Micron Datasheet [7]

Row Size	IDD0	IDD3N	IDD2N	tRAS	tRC
16,384	49mA	37mA	23mA	35ns	48.75ns
8,192	42mA	35mA	23mA	35ns	48.75ns

is close to 50%[6]. Based on these results, we are confident that halving the row size (or bitline number) can significantly reduce the activation power.

$$I_{ACT} = IDD0 - \frac{IDD3N \times tRAS + IDD2N \times (tRC - tRAS)}{tRC} \tag{3.1}$$

3.1.4 The Dilemma Of Fine-Grained Activation.

According to the analysis above, it is straightforward that activation power can be reduced by limiting the number of involved bitlines during a row activation. Previously, *fine-grained activation*[2] and *selective bitline activation*[3] have been proposed to take advantage of the intrinsic fine-granularity structure of DRAM so as to reduce the number of active bitlines. We denote these techniques as fine-grained activation.

Figure 3.3 illustrates the fine-grained memory architecture inside a DRAM bank. In fact, a bank can be further vertically divided into many sub-arrays as the figure shows. Each sub-array horizontally consists of multiple cell matrices (MATs), which are the atomic access units (square blocks in the figure) for a single memory operation. Each MAT has a local row decoder and a local sense amplifier array (a.k.a row buffer). Typically, a MAT has 512×512 storage cells in row (wordline) and column (bitline) dimensions. Given the 512×512 MAT size, there are totally 32 sub-arrays in one bank and 32 MATs per sub-array. Note that the prefetched data chunk comes from **all** 32 MATs within a sub-array and each MAT only contributes 4b data. There are dedicated helper flip-flops (HFFs) in each MAT to further latch the selected data and then relay them on the internal data bus [5].

Figure 3.4 further illustrates how the intrinsic fine-grained structure inside a DRAM bank is leveraged by fine-grained activation in the prior work [2, 3]. As shown in the figure, an activation

[6]The assumption here is close to the truth since both chip have the same capacity. To our knowledge, the more complicated row decoder in 8Kb DRAM consumes more power during activation. Therefore the power saving can never reach the ideal 50%.

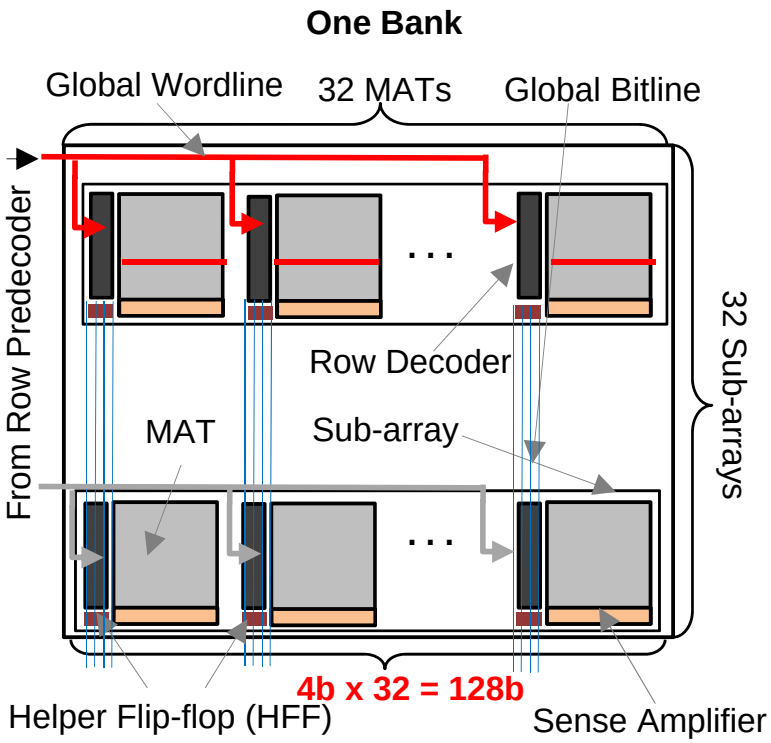


Figure 3.3: Zoom-in view of the fine-grained structure inside one DRAM bank. The bank has 32 sub-arrays and each sub-array contains 32 MATs. Global bitlines are shared by all sub-arrays. Every MAT can only provides 4b data to form 128b/bank data width.

decoder is deployed to determine which MATs should be activated within a sub-array. The posted-CAS (column address strobe) command is used so that the activation decoder can know in advance the MAT ID from the column address.

Even though the idea of fine-grained activation sounds plausible, naively implementing it without careful re-design inevitably incurs significant performance degradation as they fail to comply with the n-bit prefetching [2, 3]. As a simple example illustrated in Figure 3.4, the left two MATs of a sub-array are normally activated while the right two are still inactive. Therefore, the output data width is reduced by half correspondingly. Deducing the example to the commodity DRAM, the fine-grained activation can severely destroy the n-bit prefetching and thus reduce the data bandwidth of one bank. To mitigate the reduction on bandwidth, it is straightforward to increase the narrow data width of a MAT for compensation, similar to what the selective bitline activation [3] does. However, this approach is not practical in that the data width is constrained by the limited routing resources as well as the available number of HFFs and secondary latches used as drivers for the long intra-chip data path. In other words, it is difficult to increase the data width without incurring large area overhead, which has also been pointed out by T. Vogelsang [30]. Since the data frequency does not change, the only way to deliver an atomic 64Byte data chunk[7] is to increase the burst length accordingly. However, this will increase the

[7]Given a 64b I/O width and a burst length of eight, one memory request can deliver 64×8=512b data, which

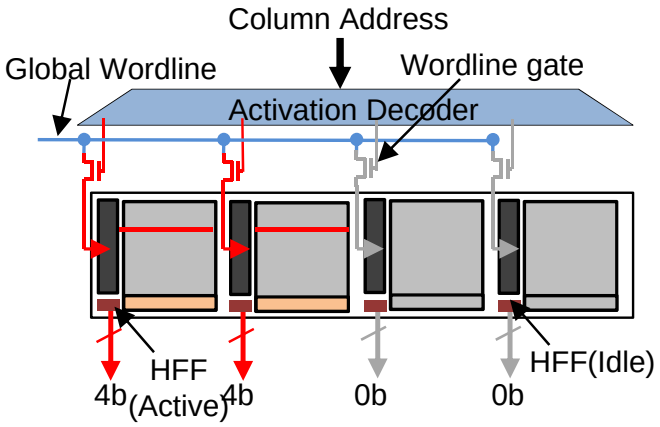


Figure 3.4: Fine-grained activation in prior work [2, 3]. An activation decoder is introduced to control the number of active MATs. However, halving the active MATs also halves the data width and thus reduces bandwidth.

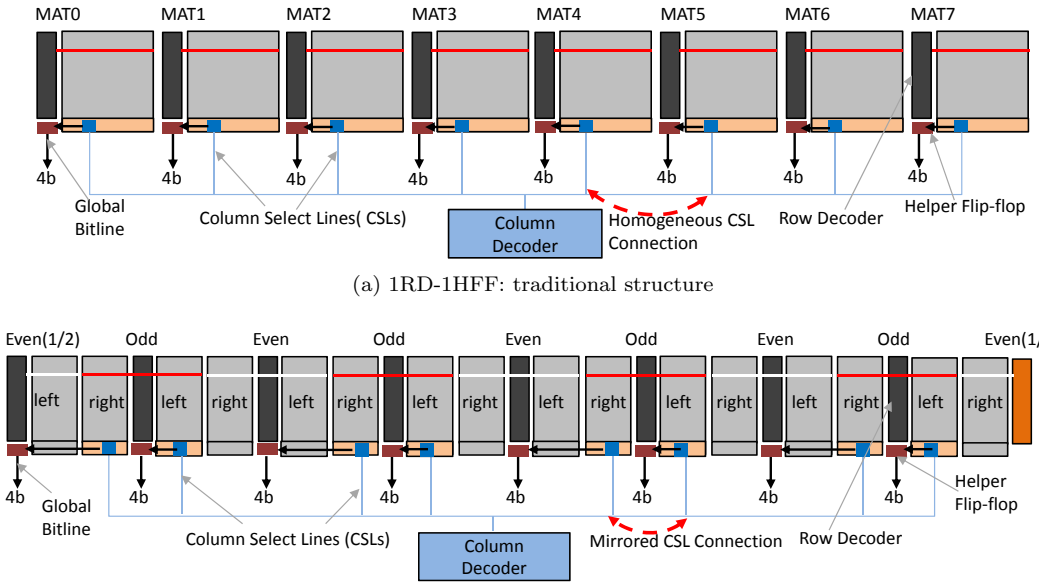
data transfer time on a DRAM bus because the bursts are serialized, and therefore will degrade the memory performance.

To evaluate the impact of data bandwidth loss on the system performance, we run the `STREAM` benchmark [36] as a representative of memory-intensive applications. Figure 3.2b shows the respective results for the cases when only 1/2, 1/4, or 1/8 of a bank can be activated at a time. Compared to the full bank activation, the original method suffers a lot from the narrow data width a MAT can provide. In particular, the 1/2 bank activation without careful design can experience a 30% performance loss. Even worse, an unacceptably high 84% performance loss is observed when 1/8 bank is activated (four MATs involved as the example in Figure 3.3). Therefore, we conclude that **naively activating fewer MATs to reduce activation power can destroy the n-bit prefetching and thus cause significant memory bandwidth drop**. In the extreme scenario of 1/8 bank activation, 8-bit prefetching is completely destroyed and 7/8 of the bandwidth is wasted. The wasted bandwidth explains the significant performance degradation as it now needs 64 bursts to deliver the amount that could be transferred within 8 bursts originally (i.e., a 64Byte last-level cache line).

Due to the aforementioned issues, fine-grained activation must be carefully rethought to make sure it does not degrade memory bandwidth nor incur large area overhead. In this work, Half-DRAM is proposed to make better compromise among performance, power and area. A novel fine-grained memory architecture is designed so as to activate half the MATs in one sub-array while still providing full 128b data with negligible area overhead.

---

is equal to 64Byte.



(a) 1RD-1HFF: traditional structure

(b) 1RD-2HFF: Half-DRAM structure (proposed)

Figure 3.5: The reorganization of sub-array in Half-DRAM. (a) Traditional sub-array. The one-to-one relationship exists between row decoder and HFF since the row decoder drives the entire wordline of one MAT; (b) A MAT is split into "left" and "right" block and they are driven by different row address decoders. The sub-array is further divided into Odd and Even groups.

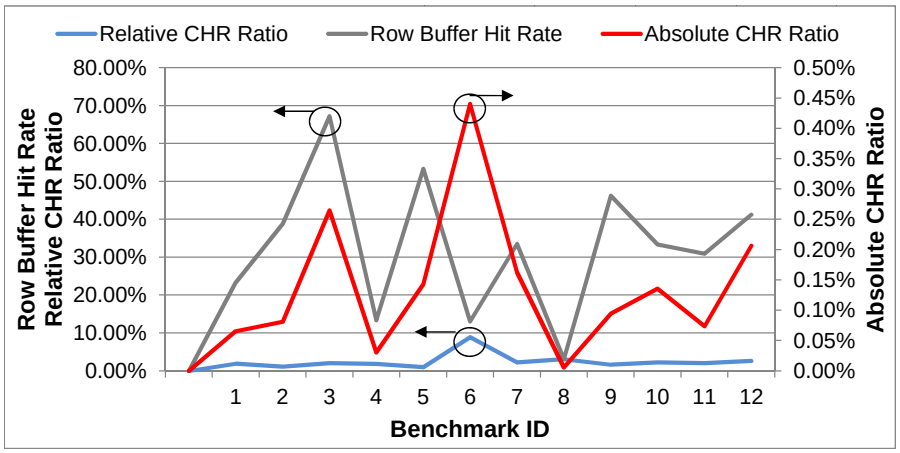
3.2 The Design Of Half-DRAM

From our observation, the major problem in prior fine-grained activation techniques is the incapability of reusing the row buffer and the associated functional components on the data path (e.g., HFFs and global bitlines). The incapability stems from the one-to-one relationship between row decoder/row buffer and the HFF.

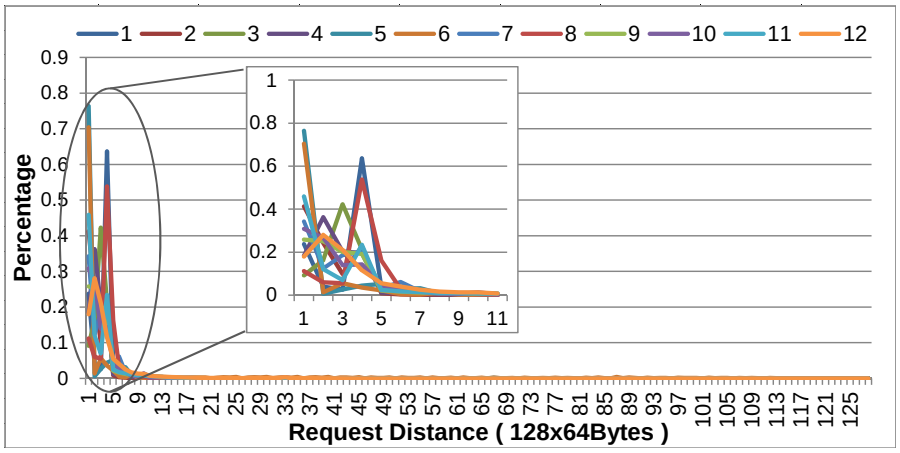
Figure 3.5 shows the reorganization of sub-array in Half-DRAM with comparison to the baseline DRAM design. As shown in Figure 3.5a, the local row decoder drives the associated wordline, which further activates the entire row data into the row buffer. All bits in the row buffers compete for a common HFFs shown on the left. We name such one-to-one relationship as "1RD-1HFF", which means one HFF group is dedicated to one row decoder and vice versa. This 1RD-1HFF hardware organization in traditional DRAM is kept the same in the prior work of fine-grained activation [2, 3], which is the substantial reason why the bandwidth cannot be unleashed.

3.2.1 1RD-2HFF vs. 1RD-1HFF

We propose a novel structure to enable the fine-grained activation without bandwidth loss. Figure 3.5b shows the basic idea of Half-DRAM. First of all, Half-DRAM breaks the original 1RD-1HFF restraint by splitting one MAT into two identical slabs: **left** and **right**. Distinct from 1RD-1HFF, a row decoder is now driving the left and right slabs that originally belong to two



(a) CHR Ratio: Four-core



(b) Request Distance Distribution

Figure 3.6: Impact of half-size row buffer. All benchmarks have quite high data locality so that only <0.5% accesses cross half row during a burst of row buffer hits, which indicates the great opportunity for Half-DRAM.

neighbor MATs. As a consequence, once the row decoder selects a wordline, both MATs are activated but each with half a row. In other words, *even if every other row decoder is disabled* (white wordlines in 3.5b), *all HFFs can still be active whilst only half of a row has valid data* (red wordlines). Therefore, the sub-array is **logically** divided into two groups, which are respectively labeled as *Odd* and *Even* as Figure 3.5b shows. Similar to previous work, a transistor is deployed to control the gating of the undesired MATs, which is shown in Figure 3.4. Note that there are two slabs standing alone at the head and tail of a sub-array, which actually belong to the Even group. An additional row decoder is introduced to drive the tail (left) slab for simplicity. The extra row decoder can even be removed to eliminate area overhead as we will elaborate in Section 3.3. In this way, Half-DRAM builds a new "1RD-2HFF" relationship between row decoder and HFF.

3.2.2 Half Active DRAM

Figure 3.7 shows the logic views of various Half-DRAM designs. By leveraging the column select line (CSL) gate shown in Figure 3.7a, fine-grained activation can be easily enabled in 1RD-2HFF as the Even group is active while the Odd group is idle, or vice versa. The MSB of column address is used to determine which half of a bank should be activated. The posted-CAS technique allows ACT and CAS commands to be issued back-to-back, which can be leveraged by Half-DRAM to know column address in advance. Once such fine-grained activation is applied to commodity DRAM, only half of a DRAM chip is active for each memory request (so-called "Half-DRAM"). Distinct from prior work [2, 3], all components on the global data paths, including the HFFs and the global bitlines, can be fully utilized to sustain the 8-bit prefetching and provide the required 128b data of each DRAM chip (8 bursts of 16 bits).

In addition, the column select logic should be redefined to make sure only the valid data can be selected out of the row buffer. Figure 3.5 also illustrates the different column select logic for this goal. For the traditional 1RD-1HFF, every MAT has the same semantic to interpret the input of CSLs. This is achieved by the homogeneous CSL connection to the output of column decoder. Alternatively, in Half-DRAM the semantic is completely mirrored in any two neighboring MATs. If we assume one MAT still complies with the original column select semantic, then the neighbor MAT needs to have the reversed semantic for the correct selection. Fortunately, this can be simply achieved by mirroring the CSL connection without any overhead (see Section 3.3).

Even though Half-DRAM intelligently mitigates the bandwidth reduction problem, it has only half of the row buffer size, which may induce performance overhead due to degraded row buffer hit rate. Fortunately Half-DRAM can be easily extended to the traditional full bank activation without any extra design effort to provide full row buffer size. The second activation does not have to bear the tRRD because the row address has been pre-decoded by the previous activation. Consequently, it neither incurs timing penalty as shown in Figure 3.7b.

Depending on the row buffer management policy, Half-DRAM can render different behaviors. In this work, two close-page policies are evaluated. The first close-page policy, *Restrict-ClosePage*, closes the row immediately after the access is completed. No row buffer hit can be exploited in this policy. The other close-page policy, *Relaxed-ClosePage*, closes the row only when there is no more requests in the queue that can benefit from row buffer hit. Obviously, for the Restrict-ClosePage, Half-DRAM can gain maximum power saving without performance overhead. For Relaxed-ClosePage, two half-row activations are issued if and only if two requests in the queue reference to both Odd and Even group. In this way, Half-DRAM enables a fine-grained, on-demand-activation memory system.

3.2.3 Half-Size Row Buffer: Challenge Or Opportunity

Even though Half-DRAM can effectively reduce the activation power, it may induce more activations because of the smaller row buffer size. To justify the feasibility of Half-DRAM, we evaluate the performance impact of half-size row buffer. We first define the *ratio of crossing half*

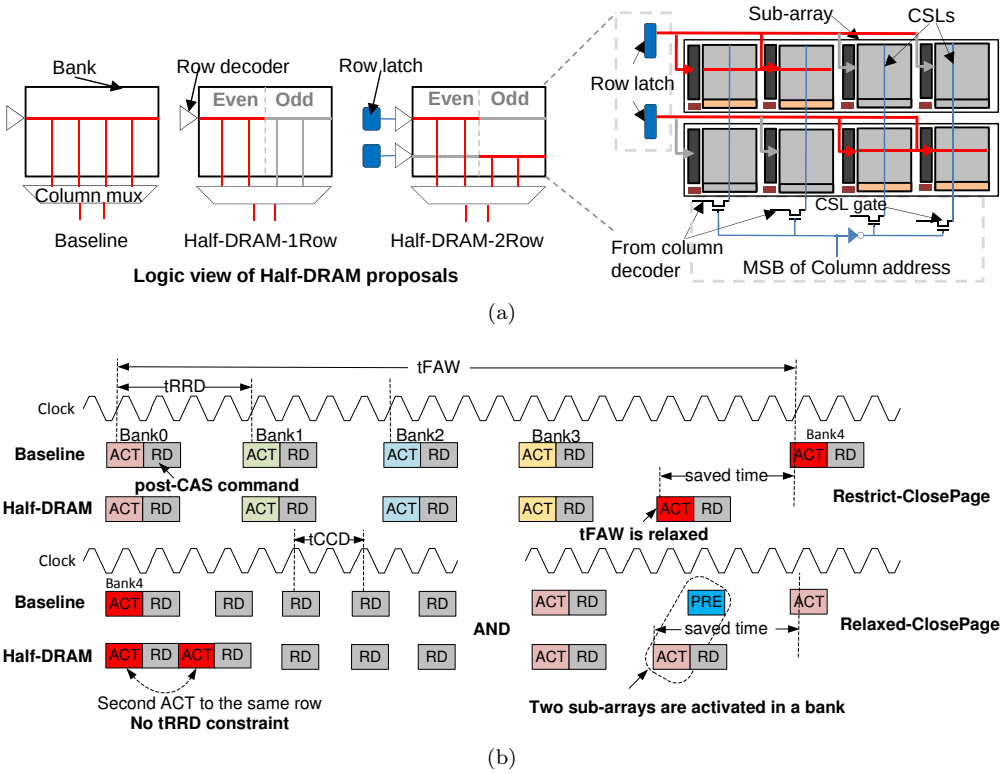


Figure 3.7: The proposed Half-DRAM models with the timing illustration. (a) The logic view of Half-DRAM-1Row and Half-DRAM-2Row (left) and the row/column control design for Half-DRAM-2Row; (b) Timing diagram to illustrate the relaxation of tFAW as well as the integration of sub-array level parallelism [4] for performance improvement.

row (CHR ratio) as the percentage of two requests referencing to the same row but different half rows when Relaxed-ClosePage policy is applied. The lower the ratio is, the more requests are served within a half row. Specifically, zero CHR ratio means all requests go to the same half row at each memory access and there is no performance penalty at all. Note that we only count the requests that have row buffer hits, other requests that close the row immediately (due to row conflicts) have been excluded.

$$CHR\_Ratio_{absolute} = \frac{NUM_{CHR}}{NUM_{total\_req}} \tag{3.2}$$

$$CHR\_Ratio_{relative} = \frac{NUM_{CHR}}{NUM_{row\_buffer\_hit}} \tag{3.3}$$

Two CHR ratios are defined in Equation 3.2 and 3.3. The *absolute CHR ratio* represents the percentage of half row crossing over all requests, while *relative* CHR ratio shows the actual number of requests that have data accesses across half rows upon a row buffer hit. Therefore, relative CHR ratio filters out the noise from requests with low row buffer hit rates. Figure 3.6a

shows the results of the 4-core simulation. Obviously, all benchmarks have quite low absolute CHR ratios (<0.5%, red curve in Figure 3.6a). Even the relative CHR ratio (blue curve) is very low, which is no more than 8%. The low absolute and relative CHR ratio indicates the great opportunity for Half-DRAM to save power without performance loss. Also, note that the CHR ratio and row buffer hit rate are not necessarily correlated. For example, test3 has high row buffer hit rate (67.2% in gray curve) but it only has 0.26% row crossing rate. On the other hand, test6 has 13% row buffer hit rate while it shows the highest CHR ratio as 0.44%.

To understand the cause of low CHR ratio, we also quantify the request distance, which is defined as the distance between two requests that access the minimum and maximum column address during a series of row buffer hits. The distance distribution is shown in Figure 3.6b with 64Byte sampling unit. Given our baseline has 64Kb row in a rank (4chip×16Kb), there are totally 128 sampling units on the X-axis. As shown, the requests have pretty high data locality as >99% requests occur within 8-unit distance. As a result, the CHR ratio is minimized by the high data locality in a row and thus reveals promising opportunity for Half-DRAM to reduce activation power.

3.2.4 Chance Of Improving Memory Parallelism

To prevent too frequent activations from generating a high current that exceeds the predefined current threshold, two timing parameters are defined in commodity DRAM. While the row-to-row activation delay (tRRD) specifies the minimum interval of two successive activations, the four-activation-window constraint requires that there must be no more than four activations in the rolling window tFAW as shown in Figure 3.7b. Half-DRAM is promising in that it can relax these power constraints (manifested as timing constraints), resulting from its design to activate fewer bitlines each time and thus have less activation power. Specifically, the four-activation window constraint can be relaxed since now eight half-row activations are permitted in any tFAW window. This is meaningful for the memory with Restrict-ClosePage applied since the performance of such memory is mainly limited by tFAW [24].

In addition, *Half-Bank level parallelism* is introduced in Half-DRAM to further improve the memory parallelism. Different from the basic design, two half bank latches are utilized to decouple Odd and Even groups from each other. As a result, sub-array level parallelism [4], which uses sub-arrays as independent memory operation corresponder, can be integrated into Half-DRAM seamlessly. As shown in Figure 3.7b, two half-rows in different sub-arrays can be activated without data path contentions as long as they belong to different half-rows, which effectively doubles the memory parallelism.

In this work, two Half-DRAM schemes are proposed as shown in Figure 3.7a with the baseline illustrated at the leftmost. The intermediate Half-DRAM that only permits one half-row activation is presented in the middle (*Half-DRAM-1Row*), meaning that no further activation can be issued even if the next activation goes to the inactive half-row. Alternatively, the Half-DRAM that allows any two half-rows to be active is given at the rightmost side of the figure

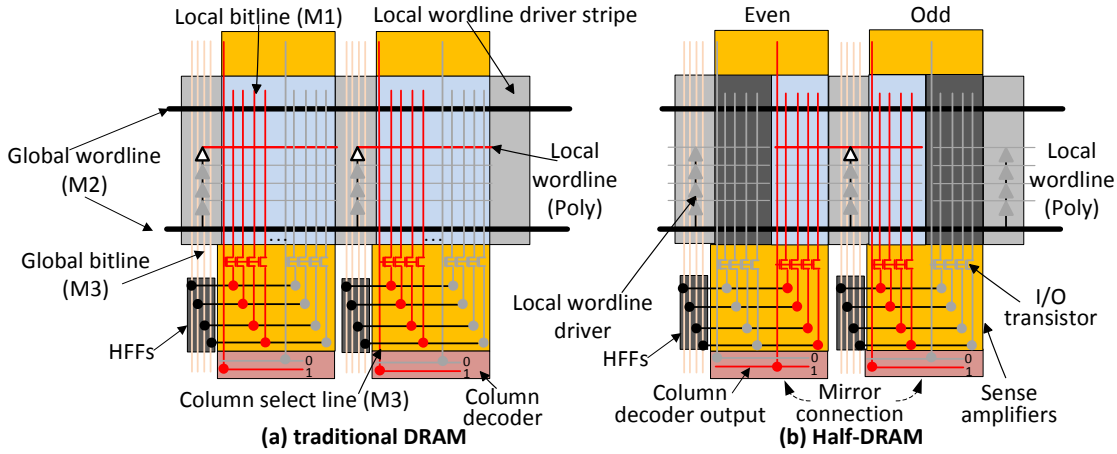


Figure 3.8: The circuit design of Half-DRAM-1Row. All metals layers are given for the illustration. Note that how data is selected out and relayed on HFFs [5]. Obviously, Half-DRAM-1Row can completely reuse all components and wires without any overhead. The connection of column decoder output in an Even MAT is mirrored in an Odd MAT.

(*Half-DRAM-2Row*). In fact, Half-DRAM-1Row can be easily extended to *Half-DRAM-1Row-Demand*, in which the other half of the same row can be activated immediately as long as no precharge is issued. Note that we retain the same assumptions that no more than eight half-row activations can be issued in the tFAW rolling window and two continuous half-row activations must comply with the tRRD constraint whenever they go to different rows.

3.3 Design Overhead Analysis

As DRAM is competing at a thin profit margin, and the cost (known as $/bit) is very sensitive to area increase, any change in DRAM structure should be assessed with area overhead analysis. In this section, we will conduct a detailed analysis on DRAM area overhead to justify the practicality of our Half-DRAM design. Typically, the commodity DRAM is implemented with three metal layers [1, 30]. This design can be well leveraged by our Half-DRAM design. Figure 3.8 presents the circuit design of Half-DRAM. Instead of driving a local wordline that traverses all 512 bitlines in a single MAT (Figure 3.8a), the wordline can be horizontally shifted and cover a half row from both neighboring MATs, respectively. Notice that the two half-row wordlines share a common row logic stripe (Figure 3.8b). In other words, the local wordline driver becomes bi-directional, and drives the wordline (or row) at both directions. Obviously, this design does not incur extra area or routing overhead, given that it only involves a horizontal shift of the metal wordline and only requires a few additional metal vias to connect to the driver at circuit level.

Figure 3.8 also shows the proposed design change of column select lines (CSLs) routing. In the original DRAM layout as Figure 3.8a shows, the CSLs are organized in an order that is identical between MATs. One CSL connects to four I/O transistors to select data out. Therefore, each MAT can have only one $\times 4$ column selected and buffered in the HFFs assigned to the MAT. In

Half-DRAM, in order to select the desired data within one even or odd DRAM row, the connection of CSLs to the output of column decoder is mirrored between even and odd MATs. For example, two CSLs are shown in the figure with each from the left and right slab, respectively. Originally, if the output of column decoder is '10', then the left columns in both MATs are selected due to the homogeneous connection (in red color in Figure 3.8a). Instead, as the CSL connection is mirrored, the same decoding output '10' selects the right slab of the first MAT (Figure 3.8b). In this way, two columns (or 2×4=8 bits) in each activated even (or odd) row are selected and transferred to their corresponding HFFs. Note that these "mirrored" CSL design will apply to each pair of MAT across the entire sub-array. Because the mirrored wire connection can be done outside the cell array, there is no circuit or routing overhead incurred.

The only overhead in Half-DRAM-1Row is the extra row decoder at the tail of each sub-array. According to Table 3.1, the total footprint of the row decoder is 6.84 mm^2 (local wordline driver + row predecoder), which is 18.4% of the DRAM die area (37.129 mm^2). Since the ratio of extra row decoder is 1/32, the area overhead is roughly 0.58% (18.4%×1/32). In addition, as the original row decoders already consume much smaller power than the bitline and SA, the power overhead caused by the additional row decoders is negligible. We denote this design option as *Power-OPT* (power optimization) since it maximizes the power saving with modest area overhead. Furthermore, even this marginal area overhead can be saved by removing the extra tail row decoders. Instead, the half-row at the tail can be driven by the last Odd row decoder and this decoder is always activated once the row is selected, similar to what the baseline does. We denote this design option as *Area-OPT* (area optimization). Of course, Area-OPT is at the cost of lowering the energy savings from roughly 1/2 to 29/64 as the last three half MATs are always activated. The elimination of area overhead, however, encourages us to apply Area-OPT in our work. In summary, Half-DRAM-1Row does not incur any area overhead.

On the other hand, Half-DRAM-2Row requires extra row and column logic shown in Figure 3.7a. Similar to sub-array level parallelism [4], two row address latches are deployed and each is sized around 40b. Note that **Half-DRAM-2Row naturally avoids the global bitline contention due to the exclusive column selection**. Therefore it removes the designated-bit latches used in multiple activated sub-arrays [4]. This is a unique advantage of Half-DRAM-2Row. In the column decoder, the most-significant bit of column address is used to gate half of CSLs and its reverse signal gates the other half. As one bank has 1,024 CSLs, 8,192 gates are needed for a whole chip. Similarly, 8,192 gates are also needed for wordline gating. Since the gate is implemented by a single transistor, its area overhead is completely negligible (remember one chip has >1G bits and each bit has a transistor). In addition, we use Design Compiler [29] to synthesize the row latches with TSMC 45nm-1.05V process. The results show that these latches only occupy 786μm^2 and consume 427µW in a chip. Compared to the activation power that can be as much as 24mW (=16mA×1.5V), the power overhead of the latches is trivial. The wire routing overhead is also negligible since the area of the second 40b address bus is only about 0.081μm^2 at 45nm node. In summary, the total area and power overhead caused by Half-DRAM is conservatively estimated as less than 0.003%, which is negligible.

3.4 Evaluation Results

In this work, we use gem5 [34] as our simulation platform. We integrate the NVMain [58] into gem5 as the DRAM model. Table 5.2 shows the simulation setup. The DRAM timing and power parameters are excerpted from Micron's data sheet [7]. Based on the power analysis in Section 3.1.3, the IDD0 of an 8Kb row (42mA) is used as the half-row activation current. FR-FCFS memory scheduling policy [59] is deployed in the memory controller with separate read/write queue. The selected SPEC2006 CPU benchmarks with reference input size [35] and STREAM with all functions [36] are evaluated as multi-programmed tests. Eight benchmarks that have high MPKIs (miss per kilo instructions) are selected and each benchmark is either duplicated or mixed for the four-core simulation. The four-core benchmarks are listed at the bottom of Table 5.2, where each of them is given a test number. We run all benchmarks for 500 million instructions for cache warmup and then the following 100 million instructions for statistics. The weighted IPC (instructions per cycle) defined in Equation 4.3 is used as the performance criteria for the four-core simulation. The aforementioned two Half-DRAM models: Half-DRAM-1Row and Half-DRAM-2Row, are evaluated and compared to the baseline. Obviously, they represent the lower and upper bound of performance improvements, respectively.

$$WeightedSpeedup = \sum_{i=1}^{n} \frac{IPC^{i}_{multi-core}}{IPC^{i}_{standalone}} \tag{3.4}$$

3.4.1 Performance Analysis

The performance results of the four-core simulation are shown in Figure 3.9a. To show the advantage of Half-DRAM, the result of the prior work with 1/2 bank activation is also evaluated and denoted as *FGA-1/2Bank*. The burst length in FGA-1/2Bank is set to 16 (8×2) to compensate the bandwidth loss discussed in Section 3.1.4. All tests have been normalized to the baseline where full bank activation and the Relaxed-ClosePage policy are applied. First, all tests suffer severe performance degradation, from 20% (test8) to 36% (test3). Since test3 has high row buffer hit rate and intensive memory accesses, the reduced bandwidth leads to much more contention on the data bus, which can explain the performance drop.

In contrast, thanks to the extremely low CHR ratio, no tests suffer from performance drop in Half-DRAM-1Row. Moreover, test6 that has the highest CHR ratio even shows 3% performance improvement. The reason of the improvement is that it has a low row buffer hit rate so that Half-DRAM-1Row can take advantage of the relaxation of Four-activation-window constraint to overwhelm the slight increase of activation number. On average, Half-DRAM-1Row can improve the performance by 1.3%. Even though the performance improvement is trivial, Half-DRAM-1Row does not induce performance degradation. In addition, Half-DRAM-2Row shows a promising performance improvement over baseline by leveraging the sub-array level parallelism. In particular, test1 can achieve as much as 19% performance improvement. The performance gain comes from the relatively low data locality, which can utilize the half-bank

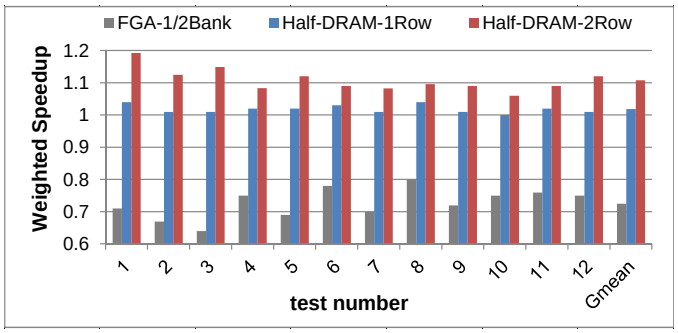
Table 3.3: Simulation Platform Configuration

Cores	4, ALPHA, out-of-order
CPU Clock Freq.	3 GHz
LDQ/STQ/ROB Size	32 / 32 / 128 entries
Issue/Commit Width	8 / 8
L1-D/L1-I Cache	32kB / 32kB 4-way 2-cycle latency
D-TLB/I-TLB Size	64 / 48 entries
L2 Cache	Shared, Snooping, 4MB, LRU 8-way, 10-cycle latency
Memory	
General	DDR3-1600, 4GB, 64bit I/O, 4 ranks, 2Gb chip, 8 banks(×16), FR-FCFS, RD/WR queues, 64/32 entries,
Timing (in cycle)	tRCD-tCAS-tRP-tWR: 11-11-11-12, tRAS-tCCD-tRRD-tFAW: 28-4-6-32
Power (fast-exit PD)	49/**42mA**(IDD0), 15mA(IDD2P), 23mA(IDD2N), 37mA(IDD3N), 135mA(IDD4R), 146mA(IDD4W), 182mA(IDD5A), 1.5V(V_{DD})
$^{test\#}$**Benchmarks (SPEC2006+STRAM)**	
[1]STREAM×4, [2]bwaves×4, [3]gobmk×4, [4]leslie3d×4, [5]libquantum×4,, [6]lbm×4, [7]mcf×4, [8]milc×4, [9]STREAM-gobmk-lbm-libquantum, [10]bwaves-leslie3d-mcf-milc, [11]lbm-libquantum-bwaves-leslie3d, [12]STREAM-gobmk-mcf-milc	

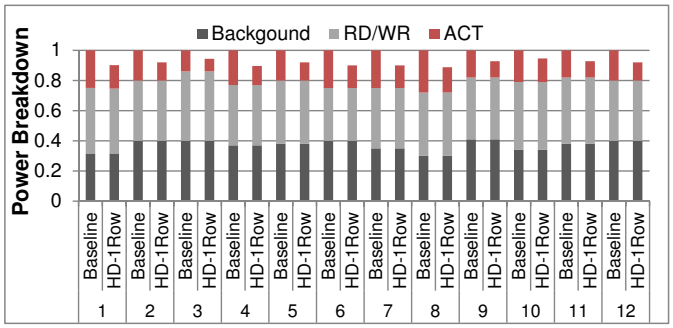
parallelism well. Again, the relaxation of tFAW also boosts the performance gain. The average performance improvement in Half-DRAM-2Row is 10.7%.

3.4.2 Power Analysis

To verify the power reduction of Half-DRAM, DRAMSim2 collects the active, standby, and powerdown cycle numbers as well as the read and write request numbers that are required by Micron's power calculator [52]. Note that the IDD0 of half-bank activation is set as 42mA so that every activation can save 43.7% power as discussed in Section 3.1.3. The generated four-core power results are shown in Figure 3.9b with the power breakdown in detail. Since Half-DRAM-1Row has almost the same runtime as the baseline, its power is given so that we can concentrate on the power reduction from activation. As shown in the figure, the power efficiency varies among the benchmarks. For instance, test1 and test6 have significant power reduction, which is up to 9.1% and 10%, respectively. In contrast, the power reduction from test3 and test9 is relatively small (5.8% and 7%). The effect of fine-grain activation in Half-DRAM is mainly determined by the intensity of activations at runtime. From Figure 3.9b, it is obvious that the activation power in test1 is big enough to reflect the power efficiency of Half-DRAM. On the other hand, the power gain is limited because the high buffer hit rate can effectively amortize the activation power, like what happens in test3. In general, Half-DRAM-1Row can achieve 8.4% improvement on power efficiency over the baseline.



(a) Speedup



(b) Power

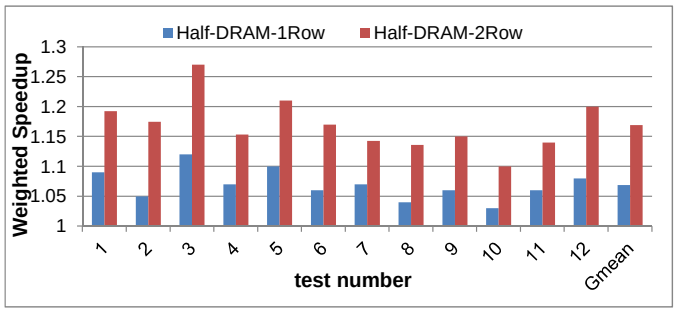
Figure 3.9: Four core simulation results. All tests are run with the Relax-ClosePage scheme and all results are normalized to baseline. The FGA-1/2Bank corresponds to 1/2 bank activation in prior work [2, 3]. The burst length of FGA-1/2Bank is 16.

3.4.3 The Effect Of The Relaxation of Power Constraint

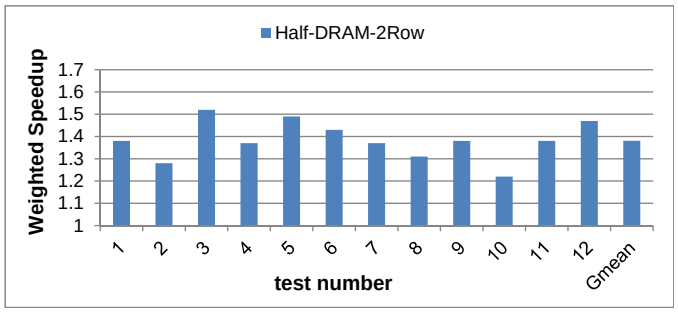
In this section, we first present the performance impact of Half-DRAM by relaxing the Four-activation-window constraint. Then, we show a case study in which Half-DRAM is integrated into Wide-IO to relax the Two-activation-window constraint.

3.4.3.1 Impact Of Four-Activation-Window Constraint

Compared to Relaxed-ClosePage policy, the performance of Restrict-ClosePage policy is limited by Four-activation-window constraint because any memory request requires an activation/precharge while the constraint limits the activation frequency [24]. According to the value of tRRD and tFAW shown in Table 5.2, there can be 5.3- ($=\frac{tFAW}{tRRD}$) activations in a tFAW if no Four-activation window constraint is applied. In other words, DRAM loses 25% ($=\frac{tFAW-4\times tRRD}{tFAW}$) activation bandwidth due to the tFAW constraint. As Half-DRAM alleviates the power constraints, the activation rate could be improved accordingly. To verify the advantage of Half-DRAM, we rerun the simulation and the results are shown in Figure 3.10a. The average speedup in Half-DRAM-1Row is 6.9%. Compared to the results of Relaxed-Close Policy is applied (Figure 3.9a), Half-DRAM-1Row has even higher performance improvement, which unsurprisingly comes from the relaxation of four-activation-window constraint. In particular, test3 has 11.8% improvement



(a) Restrict-ClosePage + Half-DRAM



(b) Wide-IO + Half-DRAM-2Row

Figure 3.10: Simulation results of power constraint study. Restrict-ClosePage management policy is applied. (a) Weighted speedup by relaxing tFAW constraint; (b) Better performance improvement due to higher tFAW penalty in Wide-IO.

as the requests going to the same rows that originally have to wait for the row re-activation can now be served earlier. Even better, Half-DRAM-2Row has further performance improvement over all tests. The additional improvement is from enhanced memory parallelism as expected. At this time, two requests that go to the Odd and Even group separately can be served in parallel. In general, 16.9% performance improvement is observed in Half-DRAM-2Row.

3.4.3.2 A Case Study Of Half-DRAM

So far, it is clear that the performance advantage of Half-DRAM is promising in a memory that suffers from the restricted power constraint. Wide-IO [11] is such a memory in which the Four-activation-window is further restricted to Two-activation-window (tTAW) due to the challenging power delivery issue in 3D-stacked memory [60, 61]. In other words, in any tTAW there can be only two activations allowed. Considering the larger window in Wide-IO (tTAW=50ns and tRRD=10ns), the power constraint more severely suppresses the performance[8]. We apply Half-DRAM-2Row to Wide-IO to assess the possible performance gain. Note that all requests still go to one channel because the Two-activation-window is applied within each channel. The results are given in Figure 3.10b. Not surprisingly, the average performance improvement is up to

[8] Wide-IO augments the channel-level parallelism to alleviate the constraint as it has four independent channels. Our concern, however, is that the power constraint has effect within a channel so that it eventually suppresses the performance when DRAM utilization is high.

38.1% while test3 and test5 gain as much as 52.4% and 49.2% speedup, respectively. Therefore, Half-DRAM is very effective in Wide-IO to exploit the potential performance benefit.

3.5 Related Work

Several fine-grained DRAM structures have previously been proposed in conventional 2D DRAM. For example, Fujitsu implements the Fast-cycle RAM (FCRAM) that has a sub-bank structure and achieves faster access speed and lower power consumption than the baseline 2D DRAM [15]. Unfortunately, FCRAM has low cell density that leads to limited memory capacity. Similarly, Reduced-latency DRAM (RLDRAM) [14] was introduced with a smaller bank size for low access latency. RLDRAM, however, also induces large area overhead so that the capacity was only about 40% of commodity DRAM. As mentioned earlier in the paper, Cooper-Balis *et al.* [2] and Udipi *et al.* [3] proposed to leverage fine-grained access in the commodity DRAM to reduce power. However, both work fail to comply with the n-bit prefetching, and the implementation overhead to sustain full data bandwidth is significant [30].

Sub-rank level parallelism has been studied as another level of fine-grained structure. Zheng *et al.* introduced a bridge chip MRB to split the original rank into mini-ranks [62]. The mini-rank design not only significantly reduces power consumption, but also improves the memory parallelism so as to compensate the potential performance degradation from narrowing down the data bus. In spite of the power optimization, the extra MRB increases the DRAM cost and it still suffers from the bandwidth loss. Leveraging the mini-rank structure, D. Yoon *et al.* implemented a memory system that has adaptive access granularity at rank level [63]. The adaptive granularity memory system, however, requires the co-design of a corresponding fine-grained cache architecture, which significantly affects its flexibility. To summarize, without reasonable optimizations in DRAM core, it is hard to achieve a good trade-off between performance (bandwidth) and power. All above approaches always improve one aspect by sacrificing the other. Distinguished from the previous work, Half-DRAM takes a holistic consideration of both performance and power and thus can achieve better compromise in between.

3.6 Summary

The power consumption for DRAM memory can be a significant percentage of the total power consumption in modern computing systems, especially for server and data center systems. Therefore, reducing DRAM power is critical for the power-efficient computing. In this chapter, *Half-DRAM* is introduced to only activate only half a bank for activation/precharge power reduction. Compared to previous work [2, 3] that may induce severe performance or area overhead due to the reduced data bandwidth, Half-DRAM can enable the fine-grained activation with full data bandwidth by leveraging the "1RD-2HFF" structure. In addition, by exploiting the sub-array level parallelism [4], half-bank level parallelism is developed to take advantage of the relaxed power constraint to further improve the memory performance. Depending on the row buffer man-

agement policy, the experimental results show that Half-DRAM can achieve 10.7% and 16.9% performance improvement with 8.4% power saving, with negligible hardware overhead. By integration with Half-DRAM, Wide-IO can obtain 38.1% performance gain that is promising for its success. As a result, Half-DRAM can be a promising enhancement to the conventional DRAM for future power-efficient computing.

Chapter 4

Lazy Precharge: The Reduction Of Precharge Overhead

As the rich transistor resource enables chip multi-processor (CMP) architecture, the memory requests from multiple cores render high random behavior so that low row buffer locality can be exploited [3, 64]. The poor spatial locality makes it more challenging to design a memory controller with satisfied performance. Given the low row buffer locality, the *Close-Page* policy that closes the entire row buffer immediately after the data access is preferable in the CMP. The memory access latency in a memory system with Close-Page policy is mainly determined by the *Row Cycle* (tRC). Unfortunately, the row cycle is almost constant through the JEDEC-DDR evolutions (DDR-DDR4). Considering the memory clock frequency doubles in each generation, the unchanged row cycle indicates the memory becomes relatively slower (in cycle). For example, the row cycle of DDR3-1600 is 41 cycles while it becomes 54 cycles in DDR4-2400 [8, 6].

Intuitively, the larger memory latency suppresses the memory level parallelism since a bank must spend more cycles to deliver the data. To compensate the reduced memory parallelism, sub-array level parallelism (SALP) [4] has been proposed to hide the latency with fine-grained memory structure and tiered-latency DRAM (TL-DRAM) [65] have been studied to shorten the row cycle in the *Near segment* of a bank. Both prior arts, however, requires additional logic in the memory chip, which leads to increasing cost per bit ($/bit). Distinguished from those work, we propose a simple but effective scheme, named *Lazy Precharge* (LaPRE), to smartly leverage the legacy of existing sub-array infrastructure to reduce the precharge overhead as well as the row cycle. In summary, our contributions in this work are:

- We propose LaPRE as an effective solution to reduce the precharge overhead and thus shorten the row cycle for performance improvement. LaPRE does not incur any design overhead in the memory chip. The zero overhead makes LaPRE promising to be adopted by the industry.

- We propose three memory scheduling schemes in the memory controller (MC) to leverage LaPRE to obtain performance improvement. The impact of address mapping schemes and power constraint are also evaluated as sensitivity study.

4.1 Background

As mentioned in Section 1.4, DRAM consists of four basic commands: `ACT`, `CAS`, `PRE`, and `REF`. In this work, we focus on command `ACT` and `PRE`. As DRAM depends on electrical (dis-)charging, we explain each command with its unique electrical behavior as follows.

As shown in Figure 4.1 ①, at the beginning all sub-arrays in a bank are precharged as the signal *EQ* in the precharge logic (equilibration logic) is asserted. Therefore, the voltage of bitline (BL) is driven to $V_{dd}/2$ during idle. Once an `ACT` command is coming, *EQ* becomes inactive to prevent the precharge logic from driving the bitline (so not shown in Figure 4.1②③). The sub-array then goes through *wordline opening*, *charge sharing*, *data sensing*, and *data restoring* in sequence. First of all, the row address decoder asserts one wordline based on the row address as shown in Figure 4.1 ② (red line). Correspondingly, all access transistors in the row are open. Afterwards, the sub-array enters charge sharing. If the voltage of storage cell is V_{dd} (0V), the cell (bitline) shares charges with the bitline (cell). After the charge sharing, the bitline has a positive (negative) voltage difference δV from the reference bitline that is still under $V_{dd}/2$. Next, in the sensing logic signal *ACT* goes to V_{dd} and *NLAT* goes to 0V for the data sensing. The sensing logic in the sense amplifier (SA) can recognize the voltage difference and amplifies the sensing result to '1' ('0')(Figure 4.1②③). After the data sensing, the data is locked in the SA for the following data burst by `CAS-R/W`. Note that the original data in the cell has been destroyed due to the charge sharing. As a result, SA needs to drive the bitline to restore the data into the cell[1].

It is obvious that the bitline voltage is no longer $V_{dd}/2$ after an activation. As a consequence, no further `ACT`s are permitted in the active sub-array since SA may not work correctly. To open another row in the same sub-array, a `PRE` must be issued at first shown in Figure 4.1 ④. A `PRE` mainly does two things: wordline closing and bitline/SA reset. Once there is a `PRE` command, the row address decoder desserts the wordline so that the access transistors in the row are completely closed. Then, signal *EQ* becomes active again so that the precharge logic drives the bitline back to $V_{dd}/2$ for the next `ACT`. In the commodity DRAM, all sub-arrays in a bank share signal *EQ* so that one `PRE` precharges the whole bank even though in fact only one sub-array is precharged. Note that a precharge can only be issued after the completion of data restoring since the wordline closing can early terminate the restore and thus results in data loss. As no data transfer occurs during a precharge, it inevitably induces performance overhead in DRAM.

[1]The reader can refer to [5] for more details of DRAM operations.

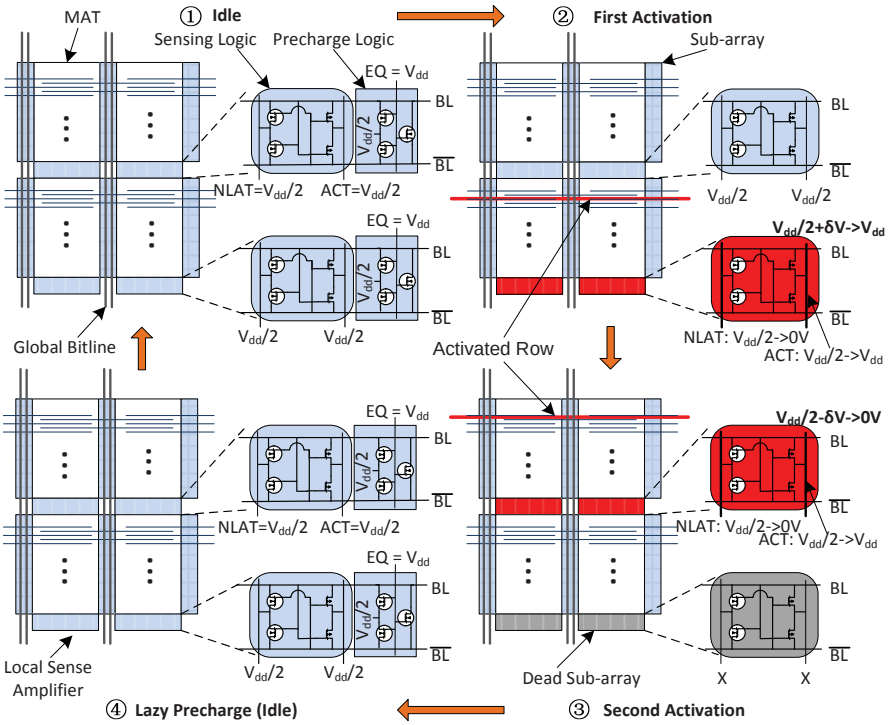


Figure 4.1: The circuit operation of activation and precharge

4.2 Design Of Lazy Precharge

In this section, we first define the overhead of precharge with Close-Page policy as the motivation. The design of LaPRE is then presented and the corresponding memory scheduling policies are extended to Open-Page policy.

4.2.1 Overhead Of Precharge

Figure 4.2 (top) shows the timeline of three requests to the same bank in the baseline. Even though they refer to different sub-arrays, all requests have to be serviced in sequence as we assume no SALP is applied. In particular, every request has a dedicated precharge due to the Close-Page policy. According to DRAM timing constraint, we define *overhead of precharge* (OH_{PRE}) as the ratio of precharge time in a row cycle as shown in Equation 4.1 and 4.2, where the overhead for read and write is calculated separately. Based on the values of timing parameters in Table 4.1, the overhead of precharge can be up to 30% and 25.6% for read and write, respectively. The large overhead indicates an opportunity to improve the memory performance by reducing the precharge overhead.

$$OH_{PRE}^{read} = \frac{tRP}{tRAS + tRP} = \frac{tRP}{tRC} \tag{4.1}$$

$$OH_{PRE}^{write} = \frac{tRP}{tRCD + tCWD + tBURST + tWR + tRP} \tag{4.2}$$

Table 4.1: Timing Parameters from Micron Data sheet [8]

Timing Params.	Value	Timing Params.	Value
tRAS	36ns	tRP	15ns
tRCD	15ns	tCWD	7.5ns
tBURST	6ns	tWR	15ns

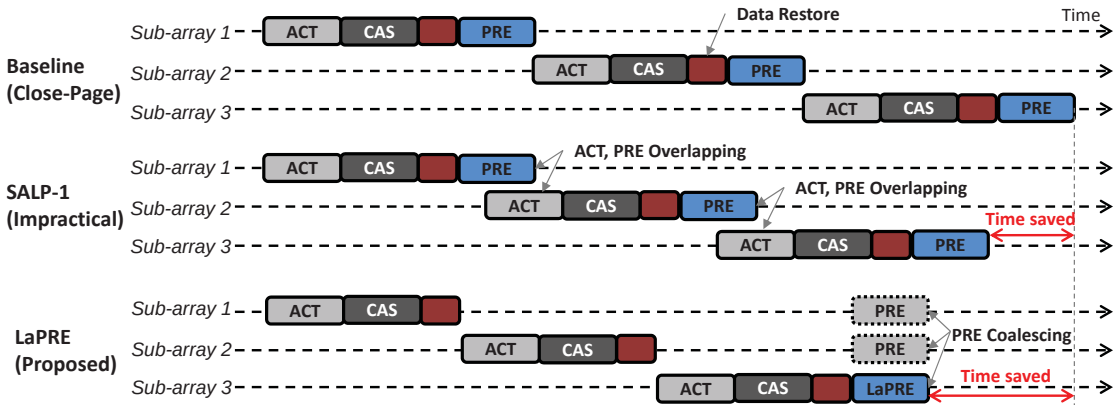


Figure 4.2: Timing diagram of the three different memory access schemes

In fact, SALP-1 has been proposed in [4] to reduce the precharge overhead. As shown in the middle of Figure 4.2, SALP-1 relaxes the precharge overhead by overlapping the precharge of one sub-array with the activation of another sub-array. SALP-1, however, incurs the conflict between activation and precharge in a bank for two reasons. The first reason is the conflict of wordline opening and closing in the sub-array that an activation is undergoing. As mentioned in [4] (Section 5.2), "... a PRECHARGE is designed to lower all wordlines within a bank to zero voltage..." while an activation should assert one of the wordlines. The second reason is that all sub-arrays are precharged simultaneously resulting from the sharing of signal *EQ*. As a consequence, SALP-1 is in fact impractical to be implemented without extra logics for selective sub-array-level precharge, like SALP-2 or MASA in that paper.

Distinct from SALP-1, LaPRE eliminates both conflicts by removing the overlap of activation and precharge. Instead, LaPRE delays the precharge and allows multiple activations before a coalesced precharge, which is named *Lazy Precharge* (LaPRE). Figure 4.2 (bottom) illustrates the basic idea behind LaPRE. As shown, the `ACT` to sub-array 2 (3) is able to start immediately after the data has been restored in the prior active sub-array 1 (2). At the end, a lazy precharge is issued to precharge all three sub-arrays. In this way, only one precharge is needed compared to the baseline that requires three precharges.

4.2.2 DRAM Chip Design

The commodity DRAM chip has provided the infrastructure for LaPRE and thereby LaPRE does not incur any hardware overhead in DRAM chip. As shown in Figure 4.1②, once an `ACT` is received, the wordline of the first sub-array is open for data burst. After the data has been

restored into the cell, the wordline can be closed without data loss. At this time, the second `ACT` can be issued to open another sub-array without a precharge, which is shown in Figure 4.1③. Since only one output of the row address decoder is asserted, the second `ACT` automatically triggers the closing of the old wordline and the opening of a new wordline. Note that no further request can be issued to the first sub-array before a lazy precharge. As a consequence, the row buffer of the first sub-array becomes invisible to MC. We name such a sub-array as a *dead sub-array*.

The aggressive access scheme in LaPRE re-defines the timing constraints on the activation sequence. From the perspective of the first active sub-array, it requires the following `ACT` to be issued after its wordline completely closes. Therefore, the second `ACT` should fulfill the same timing constraint as a `PRE` should obey, which is shown in Table 4.2. In particular, the second `ACT` can only be issued tRTP cycles after a read command `CAS-R`; or, tBURST+tWR cycles after a write command `CAS-W`. Notice that the tRRD and tFAW constraints are still applied to meet the power constraint.

Table 4.2: New Timing Constraints on Activation

Prev. Cmd	Curr. Cmd	Timing Constraint	Comments
CAS-R	ACT	tRTP	same bank
CAS-W	ACT	tBURST + tWR	same bank
ACT	ACT	Five-ACT-Window	same rank
I_{ACT}	32.1mA	I_{PRE}	6.2mA

4.2.3 DRAM Controller Design

LaPRE should be exposed to the MC so that the MC can make use of the enabled SALP. In particular, *when to precharge a bank* is a critical question to answer when design an MC. In commodity DRAM, Close-Page and Open-Page are two classic row buffer management policies to address the question[2]. In Close-Page, the precharge is initiated immediately after the desired data burst. No benefit of row buffer locality can be exploited. Alternatively, Open-Page policy requires a precharge *if and only if* 1) no request in the queue that addresses the same row (row buffer hit) or four requests have been serviced with row buffer hits; AND 2) there is a request pointing to a different row (row buffer miss). Note that we adopt four row-buffer-hit requests as the threshold to prioritize a row-buffer-miss request based on the results of prior study [64], in which less than five row-buffer-hit requests can be serviced before a precharge. This also takes into account the concern of request starvation and fairness of request scheduling.

Different from the traditional MC, LaPRE-MC has more flexibility to close a bank because multiple sub-arrays can be activated before it issues a lazy precharge. In this work, we propose three memory scheduling schemes to leverage LaPRE for the performance improvement. The three schemes are described as follows.

[2]Throughout the paper, Close-Page and Open-Page are dedicated to the row buffer management policies used in the commodity DRAM memory controllers. The proposed policies are noted with prefix "LaPRE".

•**LaPRE-Idle-First**: Similar to the traditional Close-Page policy, LaPRE-MC seeks a memory request in the queue that points to an idle sub-array based on first-come-first-serve (FCFS) policy. A lazy precharge is issued if and only if no such request can be found in the queue.

•**LaPRE-RBH-First**: Similar to Open-Page policy, LaPRE-MC prioritizes the requests that can explore row buffer hit (RBH). A sub-array switching occurs if no such request can be found or the number of requests that have been serviced reaches the threshold to avoid remaining requests' starvation. Finally, a lazy precharge is issued when no request that has row buffer hit or accesses an idle sub-array can be found in the queue.

•**LaPRE-DS-First**: As an extension of LaPRE-RBH-First, LaPRE-MC issues a lazy precharge *once* it detects the head request in the queue points to a dead sub-array (DS). In LaPRE-RBH-First scheme, such head request may be delayed for a long time because LaPRE-MC can continue to drain other requests in the queue. LaPRE-DS-First eliminates the potential delay with more precharges as the payment.

Figure 4.3 illustrates how the three scheduling schemes work. At the beginning, there are nine requests in the queue (older request has smaller number). The requests having the same color point to the same sub-array and row while the requests with different colors accesses different sub-arrays. For instance, Req-1/3/4/5/6 (light gray blocks) access the same sub-array and same row so that row buffer hit can be exploited. Req-2/7 (dark gray block) or Req-8/9 (black block) access different sub-arrays so that LaPRE can be applied.

Since LaPRE-Idle-First only searches for the request to an idle sub-array, a lazy precharge is needed after the completion of Req-1/2/8. For the same reason, the second lazy precharge is inserted after the completion of Req-3/7/9. Then, each of the remaining requests (Req-4/5/6) needs a precharge because they go to the same sub-array. Compared to the traditional Close-Page policy that needs eight precharges (dashed blue line), LaPRE-Idle-First reduces the number of precharge to four.

When it comes to LaPRE-RBH-First, Req-1/3/4/5 are prioritized due to row buffer hit. Afterwards, Req-2/7 are selected since four requests have been serviced for the previous sub-array. Req-8/9 are then scheduled consecutively. Prior to Req-6, a lazy precharge must be issued to reset the dead sub-array. Finally, Req-6 can be serviced after the precharge. Compared to Open-Page policy, LaPRE-RBH-First reduces the number of precharge from three to one. Also notice that Req-6 is delayed until Req-7/8/9 finish. To avoid the long delay, LaPRE-DS-First issues a precharge once it detects Req-6 comes to the head of queue. After the precharge, the remaining requests are scheduled accordingly. Obviously, LaPRE-DS-First may degrade the performance since it can early terminate a series of requests that are originally featured row buffer hit (e.g., Req-2/7). In addition, LaPRE-DS-First sometimes induces more precharges than LaPRE-RBH-First. For example, another precharge is introduced if the position of Req-6 and Req-8 are swapped in the queue.

In addition to the row buffer management policy, LaPRE-MC has to account for the power consumed by the simultaneous precharges of multiple sub-arrays. We use the power analysis tool from Rambus [30] to obtain the current dissipation of an activation (I_{ACT}) and a precharge

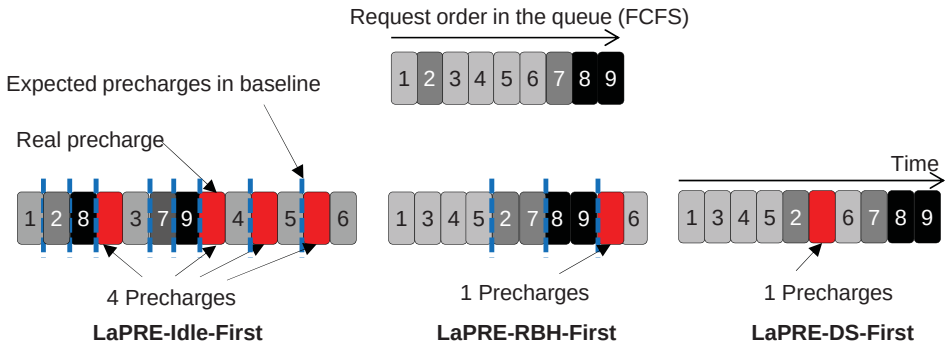


Figure 4.3: The proposed memory scheduling schemes

(I_{PRE}), respectively. The result shown in Table 4.2 reveals that the precharge current of one sub-array is around 1/5 of the activation current. Therefore, a new Five-ACT-Window constraint is introduced to limit the maximum current of a lazy precharge, where at most five activations can be issued between two precharges.

4.3 Evaluation Results

In this work, gem5 [34] is adopted as the simulation platform. We integrate NVMain [58] into gem5, which is a cycle-accurate memory simulator for both DRAM and non-volatile memories. Table 5.2 shows the setup of gem5 and NVMain, respectively. All DRAM timing parameters are excerpted from Micron's data sheet [8]. Three memory controllers with the proposed memory scheduling schemes are implemented in NVMain for the evaluation (see Section 4.2.3). Since both Close-Page and Open-Page policies are used in the baseline, these three models are further divided into two classes, where LaPRE-Idle-First is compared to the baseline that employs Close-Page policy while LaPRE-RBH-First and LaPRE-DS-First are compared to the baseline with Open-Page policy.

The selected SPEC2006 CPU benchmark with reference input size [35] and STREAM with all functions [36] are evaluated as multi-programmed testbench. We classify the benchmarks into three categories and select four benchmarks as the representatives for each category, which are symbolized as H, M, and L for high (>10), medium ([1,10]), and low (<1) miss per kilo instructions (MPKI) of last level cache (LLC). We simply duplicate four copies of each benchmark for the four-core simulation. We run all benchmarks for 500 million instructions for the cache warmup and then the following 100 million instructions for the performance statistics. The weighted instructions-per-cycle (IPC) defined in Equation 4.3 is used as the performance criteria throughout the simulation.

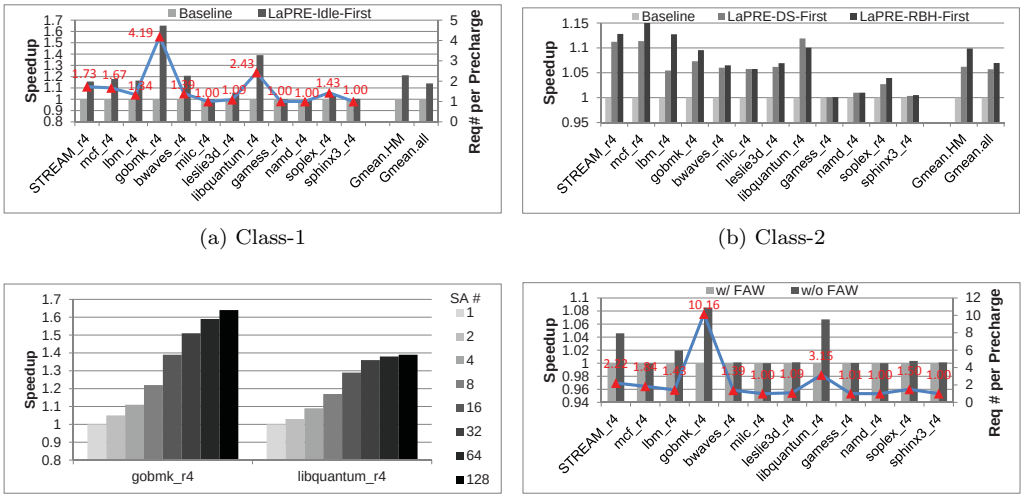
$$WeightedSpeedup = \sum_{i=1}^{4} \frac{IPC^{i}_{multi-core}}{IPC^{i}_{standalone}} \tag{4.3}$$

Table 4.3: Simulation Configuration

System Configuration	
Cores	4, ALPHA, out-of-order
CPU Clock Freq.	3 GHz
LDQ/STQ/ROB Size	32 / 32 / 128 entries
Issue/Commit Width	8 / 8
L1-D/L1-I Cache	32kB / 32kB 4-way 2-cycle latency
D-TLB/I-TLB Size	64 / 48 entries
L2 Cache	Shared, Snooping, 4MB, LRU 8-way, 10-cycle latency
Memory	JEDEC-DDR3-1333, 8GB, 64bit channel, 2 ranks, 4Gb chip(×8), 8 banks(64K×8K), tRCD-tCAS-tRP-tWR 10-10-10-10, FR-FCFS, 32-entry queue
Benchmark Classification (MPKI)	
H	STREAM(34.96), mcf(16.26) lbm(31.92) gobmk(38.35)
M	bwaves(6.07), milc(5.13), leslie3d(4.30), libquantum(6.95)
L	gamess(0.02), namd(0.12), sphnix3(0.09), soplex(0.3)

4.3.1 Performance Analysis

The performance comparison between LaPRE-Idle-First and the traditional Close-Page policy is shown in Figure 4.4a. All *H* benchmarks and two *L* benchmarks have more than 15% performance improvement. On average 14% performance improvement is observed and the improvement is up to 21% among *H* and *M* benchmarks. In contrast, little performance gain is observed in *L* benchmarks. In particular, `gobmk` and `libquantum` achieve 65.1% and 39% performance gain, respectively. To figure out the source of performance gain, we collect the average number of requests per precharge. Baseline requires one request per precharge due to the Close-Page policy. Alternatively, LaPRE-Idle-First can effectively reduce the overall amount of precharge so that one precharge can service more than one request. For example, `STREAM` has 1.73 requests per precharge while `namd` still has 1 request per precharge. Specifically, 4.19 and 2.43 requests per precharge are observed in `gobmk` and `libquantum`, respectively. The larger request number reflects that more requests share the precharge and thus the precharge overhead is significantly reduced. Even though `soplex` produces 1.43 requests per precharge, the improvement is modest due to low memory intensity.



(a) Class-1

(b) Class-2

(c) Impact of Number of Sub-arrays

(d) Impact of Power Constraint

Figure 4.4: Performance results. (a) Class-1: LaPRE-Idle-First vs. Close-Page; (b) Class-2: LaPRE-DS-First and LaPRE-RBH-First vs. Open-Page; (c) Impact of Number of Sub-arrays; (d) Impact of Power Constraint.

On the other hand, the performance results of LaPRE-DS-First and LaPRE-RBH-First are shown in Figure 4.4b. Compared to Figure 4.4a, LaPRE-DS-First and LaPRE-RBH-First provide less performance improvement, where the overall improvement is only 6.9% (5.7%) for LaPRE-RBH-First (LaPRE-DS-First) and the H and M benchmarks can obtain 9.8% (6.2%) improvement on average. The reason of less performance gain is two-fold. Firstly, the baseline already leverages Open-Page policy that can service multiple requests for one precharge. Therefore, Open-Page policy effectively offsets the benefit of LaPRE. In addition, the address mapping scheme also affects the effectiveness of LaPRE (see Section 4.3.2). Therefore, only Close-Page and LaPRE-Idle-First are employed in the following sensitivity study.

4.3.2 Sensitivity Study

In this section, we first explore the impact of number of sub-arrays. We then evaluate the impact of power constraint on the Lazy Precharge. Finally, we show the impact of address mapping at the end.

4.3.2.1 Impact Of Number Of Sub-arrays

We first sweep the number of sub-arrays in a bank from 1 to 128. Figure 4.4c shows the results. It is clear that better performance can be obtained if more sub-arrays are employed. It is straightforward that more sub-arrays can reduce the possibility of sub-array conflict that occurs when two requests try to access the same sub-array. LaPRE is always beneficial to `gobmk` as more sub-arrays are deployed. In contrast, the performance improvement is little for `libquantum` beyond 32 sub-arrays. As LaPRE does not induce any extra logics, it can simply leverage the

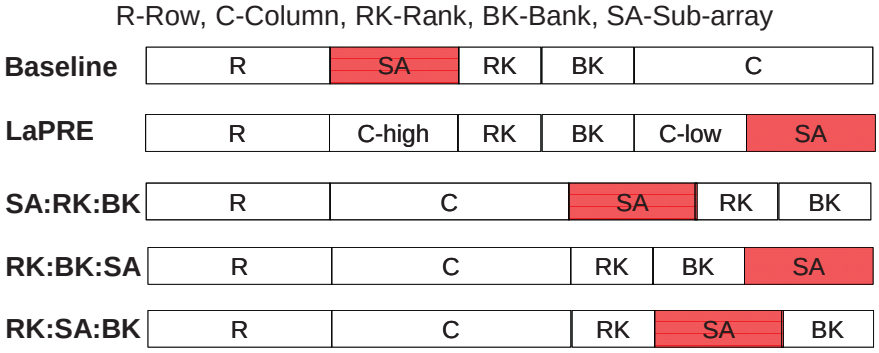


Figure 4.5: The address mapping schemes in this work

maximum number of sub-arrays to help the performance. Note that this is different from the prior work [4] in which more sub-arrays introduce larger area overhead.

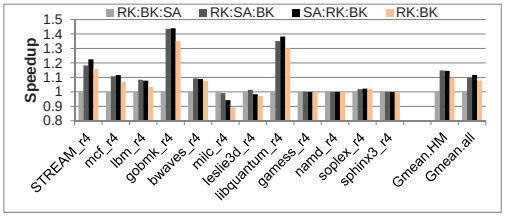
4.3.2.2 Impact Of Power Constraint

To examine the impact of power constraint, LaPRE without Five-ACT-Window constraint is simulated and compared to the counterpart where the Five-ACT-Window constraint is applied. As shown in Figure 4.4d, only `STREAM, gobmk` and `libquantum` obtain 4.6%, 8.6% and 6.7% performance improvement, respectively. The additional improvement stems from higher request/precharge rate, which implies these benchmarks fit LaPRE well. The performance gain for other benchmarks, however, are negligible due to little change of request/precharge. As a consequence, the power constraint does not severely degrade the performance.

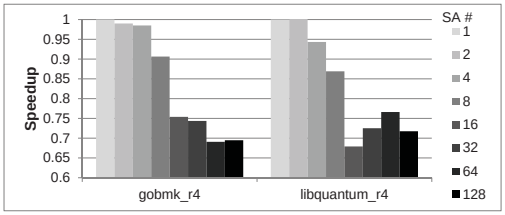
4.3.2.3 Impact Of Address Mapping Scheme

Figure 4.5 presents the address mapping schemes used in this paper. *Baseline* scheme is applied to Close-Page, Open-Page, LaPRE-RBH-First, and LaPRE-DS-First to maximize the row buffer hit while *LaPRE* scheme is employed in LaPRE-Idle-First to maximize the sub-array level parallelism. To eliminate the interference of rank-/bank-level parallelism, the position of bank and rank segment in baseline scheme is aligned with the bank and rank segment in LaPRE scheme. The baseline address mapping scheme can help explain the less performance gain in Figure 4.4b. As shown, the least significant bits (LSBs) are assigned to the column bits while the sub-array bits are placed in the relatively higher position. As a result, less sub-array level parallelism can be exploited for the performance improvement.

Among rank-/bank-/sub-array-level parallelism, two experiments are designed to figure out which one is the most important factor for the performance. The rank/bank/sub-array bit segments are alternately placed in the LSB of physical address. The corresponding address mapping schemes are shown in the last three lines of Figure 4.5, which are labeled SA:RK:BK, RK:BK:SA, and RK:SA:BK. Figure 4.6a shows the evaluation results. In sum, SA:RK:BK and RK:SA:BK outperforms RK:BK:SA so that bank-level parallelism is most important for the



(a) Impact of Address Mapping



(b) Evidence of the importance of bank-level parallelism

Figure 4.6: Performance results. (a) Class-1: LaPRE-Idle-First vs. Close-Page; (b) Class-2: LaPRE-DS-First and LaPRE-RBH-First vs. Open-Page; (c) Impact of Number of Sub-arrays; (d) Impact of Power Constraint; (e) Impact of Address Mapping; (f) Evidence of the importance of bank-level parallelism.

memory performance because banks can completely be accessed in parallel. Furthermore, the sub-array-level parallelism in LaPRE can at least work as well as rank-level-parallelism when we compare RK:SA:BK with SA:RK:BK and sometimes it even gets better results (`milc` and `leslie3d`). Moreover, under SA:RK:BK scheme, we compare SA:RK:BK to the baseline without LaPRE (labeled as RK:BK). The results show that LaPRE achieves 4.4% and 3.6% performance gain, which indicates LaPRE is still useful even if the sub-array-parallelism is minimized.

To verify the conclusion, we further conduct another experiment by sweeping the number of sub-arrays with RK:BK:SA scheme. As shown in Figure 4.6b, the result is distinct from Figure 4.4c: more sub-arrays lead to significant performance degradation. Once more sub-arrays are deployed, the width of sub-array segment in the physical address becomes larger. As a consequence, the bank-level parallelism is weakened due to the left shift of its segments. Even though the performance of `libquantum` is improved as the number of sub-arrays increases from 16 to 64, the overall performance still drops. Therefore, it demonstrates that bank-level parallelism is more important for the performance. More banks, however, requires more peripheral logics and thus reduces area efficiency of the cell and increases the corresponding cost ($/bit). As the clock frequency keeps scaling up, less ranks can be populated in a single channel [24]. Therefore, considering the zero cost, our LaPRE is still meaningful for the performance improvement.

4.4 Related Work

Several work has been done to leverage the fine-grained memory architecture. In the industry, FCRAM [15] and RLDRAM [14] make use of smaller banks for low-latency memory access. The drawback of these memory modules is the significantly reduced area efficiency, which in turn increases the total cost per bit. On the other hand, fine-grained activation is proposed in [2] so that only a portion of sub-array is activated to reduce activation power. The performance overhead that results from the reduced data bandwidth, however, is not evaluated in this work. Similarly, selective bitline activation (SBA) and single sub-array access (SSA) are proposed in [3]. Nonetheless, the performance results could be misleading because the severe bandwidth reduction

and the unaffordable layout overhead are neglected while the problem has been pointed out in [30].

The sub-array level parallelism (SALP) has been explored in [4]. In addition to the SALP-1 that is impractical because of the conflict between activation and precharge in a sub-array, SALP-2 and MASA are also proposed in [4]. SALP-2 prevents the conflict by enabling *selective precharge*. Extra logic is introduced so that the sub-arrays no longer share the same *EQ* signal for the precharge. Instead, each sub-array can have a dedicated *EQ* so that one sub-array can be precharged safely while another sub-array is open. Furthermore, the selective precharge is also applied to MASA with the enhanced isolation logic of the global data path. Compared to LaPRE, both methods requires additional logics and modification to the DRAM chip. As a result, they are supposed to be more costly than LaPRE.

4.5 Summary

As memory performance becomes the system bottleneck in a CMP architecture, high memory parallelism is mandatory to improve the overall memory performance. To reduce the precharge overhead, Lazy Precharge (LaPRE) is proposed as a novel memory architecture, which can effectively reduce the precharge overhead and thus improve memory parallelism. Compared to the previous work [4] that can cause the conflict of activation and precharge, LaPRE eliminates the conflict by allowing multiple sub-arrays to be activated without the intervention of precharge. A lazy precharge is then shared by all active or dead sub-arrays. In addition, three request scheduling policies are proposed and evaluated to leverage the enhanced memory parallelism. The experimental results show that LaPRE can achieve as much as 65% performance improvement. On average it can obtain 14% and 6.9% performance gain under different scheduling policies. Taking into account the negligible hardware overhead, LaPRE can be favorably adopted by the industry.

Chapter 5

3D-SWIFT: Enabling Sub-array Level Parallelism In 3D Wide-IO DRAM

As an early adoption of 3D integration, 3D-stacked DRAM is a promising technology for overcoming the barriers in DRAM scaling, thereby offering an opportunity to break the "memory wall" with improved DRAM cell density (capacity) and wire routing resources (connectivity), as well as reduced wire length (latency/power). In recent years, several 3D DRAM prototypes have been demonstrated [11, 13, 41, 42, 66, 67], and several 3D DRAM architectures have been explored by directly leveraging 3D die-stacking technology [68, 69, 70, 71, 72].

In particular, Wide-IO memory [11] has been standardized by JEDEC as a high bandwidth and low-power 3D DRAM for embedded SoC system. A Wide-IO has four channels that are independent of each other. Each channel is 128-bit wide with single data rate. The Wide-IO follows the low-power design methodology of LPDDR so that it removes the delay lock loop (DLL) and on-die termination (ODT) logic and applies lower supply voltage to achieve low static power.

Unfortunately, Wide-IO does not take full advantage of the die-stacking since it has to comply with conventional DRAM structure where the bank size is large and the number of banks is small. In addition, the increasing number of channels in turn mandates more memory controllers (MCs) and interconnects between them, bringing in issues that may hinder its popularity. For example, hybrid memory cube (HMC) isolates a complicated MC on the logic die to relax customers' design effort. However, it is difficult for system designers to perform further optimization on memory controllers due to its transparency to the system [73]. In this work, we propose *3D-SWIFT* to enable a fine-grained memory architecture so as to improve performance in Wide-IO. The contributions of our work can be summarized as follows.

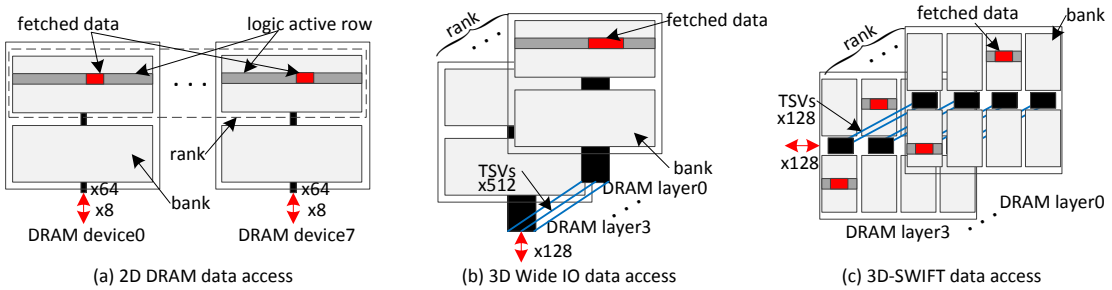


Figure 5.1: Data access mechanisms in different memories. (a) DDR3: the data is distributed over all devices in a rank. Only a small fraction of row is fetched in each memory access; (b) Wide-IO: the data is placed in a large row and no individual device exists. Still only a portion of data is fetched; (c) 3D-SWIFT: the width of fetched data matches cacheline width. The rest of row is idle.

- **Fine-grained 3D DRAM structure.** We find that the power constraint can suppress the performance of Wide-IO. To eliminate the power constraint, a bank in 3D-SWIFT is further divided into multiple sub-banks and each sub-bank has the ability to independently serve a memory request and provide an entire cacheline. Once there is a memory access, only the target sub-bank is activated. In this way, the fine-grained access reduces the activation/precharge current and thus enables higher memory parallelism. To our best knowledge, this is the first work that takes into account power constraint in designing high-performance Wide-IO.

- **Sub-bank "autonomy".** Leveraging the close-page row-buffer management policy, sub-bank autonomy is developed to combine the commands `RAS`, `CAS` and `PRE` as `REQ` to activate a sub-bank, carry on the data burst, and close the sub-bank *automatically*. We devise a packet-based interface protocol accordingly to simplify the memory transaction. Moreover, by making use of the rich routing resource on the logic die, a wide data bus is employed to deliver a full cacheline in one cycle.

- **Simplified memory controller design.** 3D-SWIFT takes into account the design complexity of MCs. By leveraging the sub-bank autonomy and packet-based protocol, a MC of 3D-SWIFT can be integrated into the processor die so that it is visible to the processor for the system-level optimization.

5.1 Background And Motivation

In this section, we review the limitations of 2D DRAM as the background. We then discuss the disadvantage of Wide-IO as the motivation of this work.

5.1.1 The Limitations Of 2D DRAM

In conventional DDRx family, the pin count constraint is a major factor that limits the memory bandwidth due to the packaging limitation. The long off-chip memory bus implemented with the transmission line on the motherboard introduces the trade-off between a higher operating frequency and the channel capacity because of the limited load. On the other hand, on-die termination (ODT) and delay lock loop (DLL) consume lots of power whilst they are mandatory to the signal integrity and synchronization when operating in a high bus frequency of DDR. In contrast, neither ODT nor DLL is needed in LPDDR, which is featured by low static power due to a lower operating frequency and supply voltage. To improve memory bandwidth, rank- and bank-interleaving are used to increase memory parallelism. Rank-level parallelism, however, is overlooked in DDR3 due to the rank-to-rank-switching penalty tRTRS [1]. Furthermore, the rank number is limited in DDR3 due to the load constraint in one channel, which also weakens rank-level parallelism.

In addition, since all devices in a rank work in lockstep, DDRx manifests an extremely large logic row while only a small fraction of data is delivered for each memory access (see Figure 5.1b). Even though the data overfetching is beneficial for the applications that have high spatial locality, the device association limits bank-level parallelism because fewer banks can be scheduled by the MC. For instance, an 8-device-8-bank rank has only eight logic banks while actually 64 "independent" banks are deployed. Furthermore, the overfetching consumes extra dynamic power because all devices should be activated/precharged at the same time. The power consumption exacerbates as the processor enters the multi-core era, where the interference among cores can drastically lower the row buffer hit rate [3], which results in the power inefficiency of such overfetching due to increased activations/precharges.

5.1.2 The Disadvantage Of Wide-IO

Once we relocate the DRAM from off-chip to on-chip, multiple DRAM dies can be stacked together to increase the cell density. The Wide-IO (Figure 5.1b) is applicable for memory bandwidth improvement for two reasons. First of all, the pin-out constraint is eliminated and the on-chip I/O bus replaces the long off-chip transmission lines. Moreover, the compact DRAM layout reduces memory access latency as well as power consumption. Despite the benefits a Wide-IO provides, some new problems emerges and need to be solved carefully:

• **More restricted power and current density constraints.** A new problem in Wide-IO is the requirement of low-power design, which is critical for the success of 3D DRAM due to the increasing power density [32]. Since no DLL or ODT is employed, the background power is dramatically reduced and the burst and activation/precharge power now starts to dominate in the Wide-IO DRAM. As illustrated in Figure 5.1b, the entire row is activated but only a portion of data is fetched, which still indicates large power redundancy. In addition, considering

[1] Due to the space limitation, we do not explain the DDR timing parameters in detail. Readers can refer to [24] to have a comprehensive knowledge on the command timing constraints.

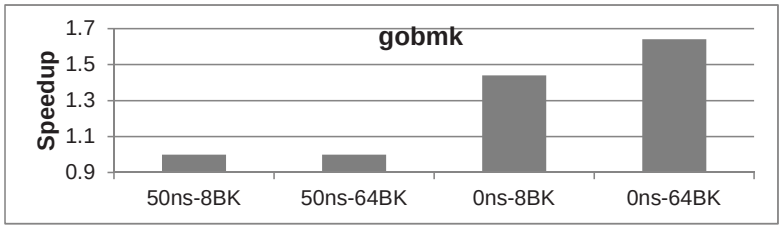


Figure 5.2: The impact of power constraint on 3D DRAM. The simulation is done with 8/64 banks(BK) and tTAW=50ns/0ns (see Section 5.4 for the details of simulation settings).

the power supply challenges in 3D ICs [61, 60], the power constraint becomes more restricted in 3D context. Therefore, Wide-IO has conservatively switched the restriction on memory accesses from four-bank activation window (tFAW) to two-bank activation window (tTAW) and extended the window width from 30ns to 50ns (the lower the better for performance) [11].

As shown in Figure 5.2, with a large tTAW constraint (tTAW=50ns), no further improvement is observed even if the bank-level parallelism is augmented as the bank number increases from 8 to 64. In other words, the power constraint can significantly suppress the bank-level-parallelism in a rank (channel), which has also been verified by Jacob *et al.* [24] when close-page row buffer management policy is applied. Once the power constraint is eliminated (tTAW=0ns), the performance gain can be up to 44% when increasing the bank number from 8 to 64. Moreover, by getting rid of tTAW constraint, the bank-level parallelism can further provide 14% improvement. As a result, neglecting the power constraint and simply increasing the bank number or bank size in 3D DRAM is not a wise choice in terms of either performance or power. Instead, 3D DRAM design should carefully cope with the power constraint to make sure it does not incur performance degradation.

• **Increased design complexity of memory controller.** The second problem is that the increasing number of channels one Wide-IO provides can result in higher design complexity of memory controllers. For example, eight independent channels have been implemented by Tezzaron [41]. While more channels can effectively improve the bandwidth, the need of multiple MCs induces noticeable area overhead when MCs are deployed in the processor die. These MCs are required to have either peer-coordination [74] or smart application assignment [75] to achieve better memory performance, which aggravates the design complexity. Alternatively, HMC [13] puts the memory controller on the logic layer, which leaves little room for a system designer to conduct further system optimization on the MC for the better communication with 3D DRAM [73].

5.1.3 Rationale Of 3D-SWIFT

To address the performance and power issues in Wide-IO DRAM, 3D-SWIFT is proposed as an essential solution. The basic idea is straightforward: since the total current of an activation/precharge is roughly proportional to the row size, the fine-grained memory architecture that shrinks the row size can effectively reduce the power consumption by activation [62, 2, 3], which can further help to alleviate or even eliminate the current constraint. In addition, the

fine-grained DRAM operation can increase the memory parallelism [15, 62, 63, 4] as well. Note that even though [2, 3] made use of the fine-grained architecture to achieve power efficiency, they did not explore the potential performance gain resulting from the relax of tTAW.

The fundamental of 3D-SWIFT is shown in Figure 5.3. To enable the fine-grained 3D DRAM, a bank is partitioned into multiple smaller slices, which we call *sub-bank*. The length of the wordline is reduced based on the simple partition[2]. As shown in the figure, each sub-bank is further folded into multiple dies to reduce the length of the bitline. In this way, the access latency is shortened by the wire length reduction. Distinguished from 2D DRAM design, a shared bus replaces the original column select logic to support the improved memory parallelism. On the other hand, every sub-bank has dedicated MATs to store the ECC code (blue block, only one is shown) so that the protected data and the corresponding ECC code can be fetched within a sub-bank simultaneously. As a result, the distributed ECC code does not incur performance degradation that could occur in other 3D DRAMs where the central ECC bank is accessed sequentially [63]. Note that the data placement is different from 2D DRAM where the data is scattered across multiple devices or the state-of-the-art Wide-IO DRAM where the data resides in a large row. As a consequence, power from activation and precharge can be dramatically reduced, relaxing tTAW constraints. Furthermore, as the sub-banks are independent of each other, more memory requests could be served simultaneously to improve memory parallelism.

The improved memory concurrency, however, may incur large hardware overhead. In particular, the design complexity of MCs significantly increases to effectively exploit the memory parallelism. As a consequence, more storage resources are needed to track the status of each sub-bank and more complex control logics are necessary to guarantee there is no command or data conflict on the bus. Therefore, the design overhead significantly limits the ability of fine-granularity access. As number of sub-bank keeps increasing along with the growing memory capacity, the poor scalability of MCs will likely overwhelm the advantages brought by 3D-SWIFT, which can affect the popularity of this novel 3D DRAM. Therefore, 3D-SWIFT takes into account the design challenge and strives to reduce the complexity with smaller area overhead. We will show the design details of 3D-SWIFT in the following sections to mitigate these overheads.

5.2 3D-SWIFT–A Novel Wide-IO DRAM

As shown in Figure 5.4, 3D-SWIFT re-maps a 2D DRAM that has nine devices (eight for data and one for ECC) and eight banks per device into the 3D-stacked DRAM. With respect to the process optimization, 3D-SWIFT separates control and interface logics from DRAM cells and put them in the logic layer as other state-of-the-art 3D DRAM does. Note that 3D-SWIFT is extensible when more ranks are employed. Those ranks can be either stacked vertically or placed horizontally as neighbors. Like HMC, multiple 3D-SWIFTs can be populated on the interposer to increase the memory capacity.

[2]Alternatively, SALP[4] did the partition on the bitline (gray dashed line)

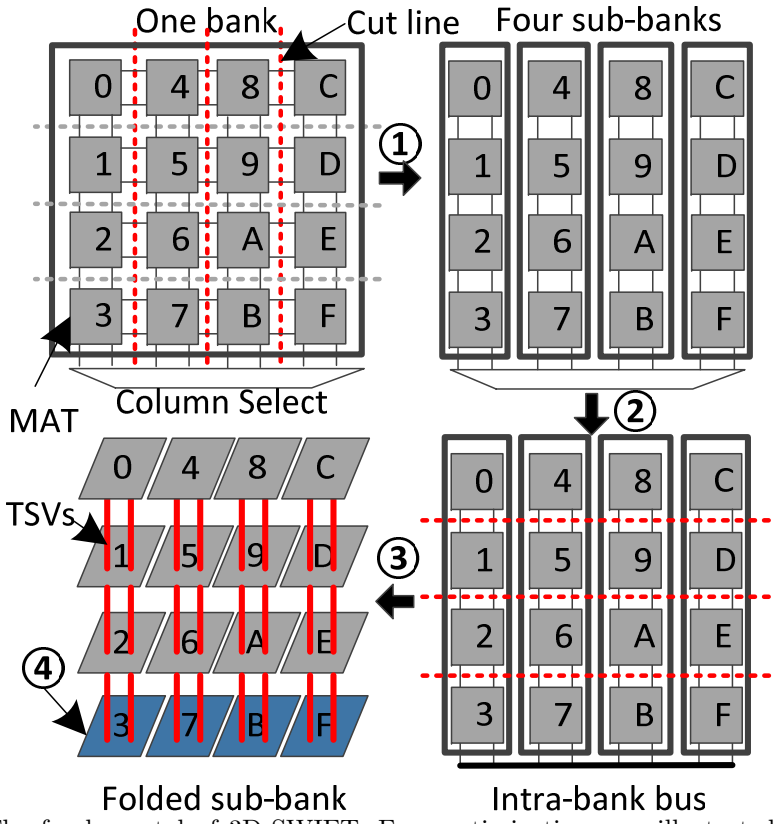


Figure 5.3: The fundamental of 3D-SWIFT. Four optimizations are illustrated: 1) A bank is partitioned into four sub-banks; 2) The column multiplexing is replaced by intra-bank bus; 3) A sub-bank is further folded into four layers; 4) A local MAT is dedicated to store ECC code.

5.2.1 Fine-Grained Memory Architecture

Without loss of generality, we use an example DRAM design with specific configurations to demonstrate the 3D-SWIFT design. To enable the fine-grained memory access, a 128M-bit (16K×8K) bank is further split into 16 identical sub-banks so that each sub-bank is sized by 16384×512 to provide a full cache line. As a consequence, a device with eight banks has 128 sub-banks and totally 2,048 sub-banks are available in 4 channels within the four layers. As multiple DRAM layers are provided, each sub-bank can be further folded [72]. For example, when there are four layers, only four sub-banks belonging to the same bank are on one layer. Since the sub-bank can serve a memory request independently, 3D-SWIFT can significantly improve the memory parallelism. In particular, *TAW constraint can be eliminated* since ideally 3D-SWIFT allows as many as 2×16=32 sub-banks to be activated in pipeline due to the reduction of active row size[3], which always holds with 1/cycle request rate and tRC row cycle (in this work, tRC=36ns=15cycles). Furthermore, the 16 identical sub-banks guarantee that 3D-SWIFT can provide sustained bandwidth even in the the worst case, where all requests access the same bank but no sub-bank conflict occurs.

[3]In fact, the number of concurrent active sub-banks in 3D-SWIFT is less than 16 because of the 2-cycle data burst and the read/write turn-around overhead.

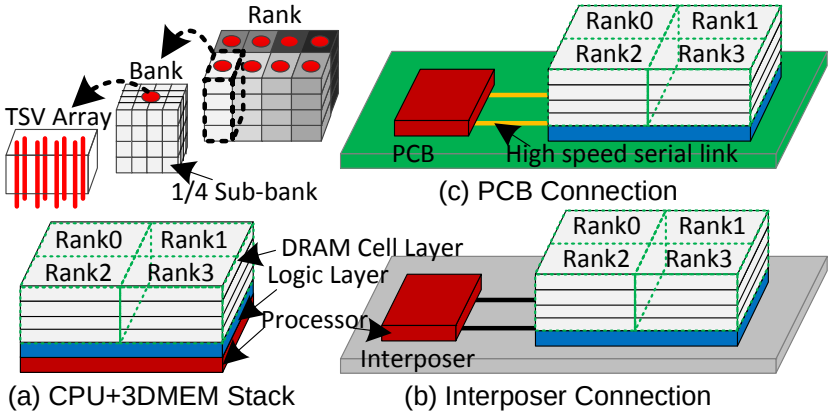


Figure 5.4: 3D-SWIFT memory subsystem and possible applications. (a) 3D-SWIFT directly stacks on the top of CPU; (b) 3D-SWIFT is connected to CPU by interposer; (c) 3D-SWIFT is placed on PCB with high speed links. Multiple 3D-SWIFTs can be populated to provide large memory capacity (not shown).

5.2.2 Adoption Of Close-Page Policy

Open-page row-buffer management policy needs a bank status table to record bank status and the current active row address for command scheduling and timing check. In conventional 2D DRAM, the table size is affordable due to the limited rank and bank number. The table, however, can be extremely large as the bank number grows. For example, considering 2,048 sub-banks and each sub-bank needs 3-byte to track the status and open row address, the bank status table should be 6KB, which leads to noticeable area overhead. Moreover, the advantage of open-page policy, which opens a large memory page as data overfetching to exploit data locality, is diminished in 3D-SWIFT as the memory page is equal to a cache line. As a result, 3D-SWIFT adopts close-page row-buffer management policy, that is, the row is closed immediately after the access.

Consequently, one sub-bank should go through a state transition loop shown in Figure 5.5. In spite of the longer access latency caused by tRC, close-page policy can provide deterministic access latencies, which is simplified as shown in (5.1)–(5.3). $L_{request}^{closepage}$ is the total latency for a memory request that is equal to a row cycle. It defines the minimum time interval between two successive requests to the same sub-bank. $L_{data}^{closepage}$ is the latency between the beginning of a request and the data appears on the bus. $L_{precharge}^{closepage}$ is the minimum interval that an automatic pre-charge can be applied to an active sub-bank. Since 3D-SWIFT implements the early sub-bank select technique [2], it can reduce CL to one cycle, including TSV gating and I/O bus propagation latency. More details are discussed in Section 5.2.5.

$$L_{request}^{closepage} = tRAS + tRP = tRC \tag{5.1}$$

$$L_{data}^{closepage} = tRCD + CL = tRCD + 1 \tag{5.2}$$

$$L_{precharge}^{closepage} = tRAS \tag{5.3}$$

5.2.3 Sub-bank Autonomy

In traditional DDRx protocol, MC and DRAM work in a **master-slave** manner. As a master, MC must send various commands, including `RAS` (row activation), `CAS` (column read/write) and `PRE` (precharge), to order the target DRAM bank to complete a data transaction. These commands can only be issued under various timing constraints (tRAS, tRCD, tRP, etc.). As MC is a queuing system, commands can be rescheduled to maximize row buffer hit rate and/or bank-level parallelism. The growing design complexity of the scheduler, however, incurs large hardware overhead and makes MC error-prone.

To offload MC's complexity, 3D-SWIFT employs the sub-bank autonomy, in which each sub-bank can automatically go through the state transition loop without intervention from MC. To enable the sub-bank autonomy, one sub-bank should be aware of the timing stamp to complete the state transition. Thanks to the deterministic access latency listed above, a transition generator can be easily deployed to signal the sub-bank when to move. As a result, a new **client-server** relation is established between MC and 3D-SWIFT: Whenever there is a new memory request, MC (client) only needs to send the request to 3D-SWIFT and then wait for the sub-bank's response. Once a sub-bank (server) detects a request, the sub-bank carries on the transition and completes the data transfer automatically. As a result, 3D-SWIFT eliminates the complicated and area-consuming scheduler in MC.

5.2.4 Packet-Based Interface Protocol

With respect to the sub-bank autonomy, a simple packet-based interface protocol is developed to simplify MC's interface design. As shown in Figure 5.5, MC sends a `REQ` with a full memory address and read/write signal to 3D-SWIFT. After $L_{data}^{closepage}$ cycles, MC should receive the data on the data bus if it is a read, or put the write data on the data bus in the case of a write. Multiple memory requests can be issued in pipeline as long as these requests do not cause sub-bank conflict, which happens once a request reference to a busy sub-bank. As a result, even though overfetching is disabled, a series of back-to-back requests can be issued to mimic the open-page overfetching. In addition, the request pipeline can also be used to support a DMA transfer that usually has a larger data size than a cacheline, in which case MC generates multiple memory requests in burst to 3D-SWIFT.

5.2.5 Hierarchical Bus

3D-SWIFT employs a hierarchical I/O bus to deliver address and data. The hierarchical bus is composed of intra-bank bus, TSV bus, inter-bank bus, and inter-rank bus. Figure 5.6 illustrates the hierarchical bus in 3D-SWIFT. The intra-bank bus connects the sense amplifier of each sub-bank to TSV arrays. To support the full cacheline delivery, both intra-bank bus and TSV array

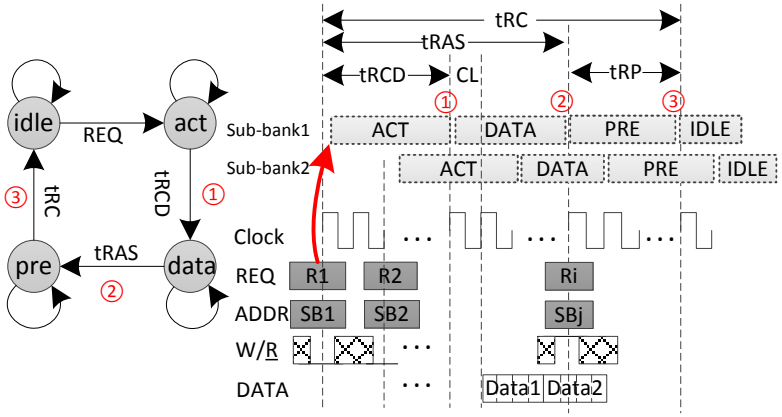


Figure 5.5: Packet-based interface protocol. Memory controller only issues REQ to initiate a memory access. The sub-bank can complete the data transfer and precharge to close the row automatically.

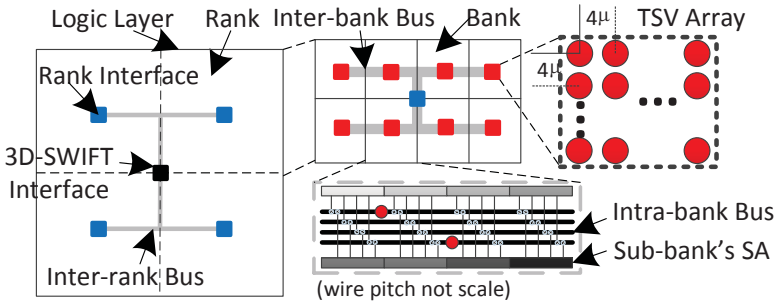


Figure 5.6: Hierarchical I/O bus. The intra-bank bus, TSV array and inter-bank bus are 512-bit and inter-rank bus is 128-bit. Each rank interface has a data buffer for the 4-beat burst (not shown).

are 512-bit wide. Afterwards, the H-tree inter-bank bus leverages the rich routing resources on the logic die to make the connectivity of all banks in a rank. One rank interface can be used as an I/O port so that multiple channels are enabled independently. If necessary, the inter-rank bus can be employed to provide a universal memory interface for the processor. In this figure, the inter-rank bus is a H-tree since ranks are placed horizontally. Once multiple ranks are stacked, the inter-rank bus can be directly implemented by TSV. Note that the width of inter-rank bus is only 128-bit for the 4-beat data burst.

5.2.6 Address Decoding

According to the hierarchy of 3D-SWIFT, dedicated decoders for rank, device, bank and sub-bank are necessary to conduct the full address decoding. Similar to [4], additional row address register is required to decouple the local row activation within a sub-bank from the rest of system (Figure 5.7). In addition, a sub-bank is selected to enable the row address register by asserting "row enable" signal. Then, the sub-bank can open the row for the data access. Since the sub-bank is selected before the row activation, we call it early sub-bank select. Note that the rank, device and bank decoders are placed in the logic layer so that they can be shared by all sub-banks.

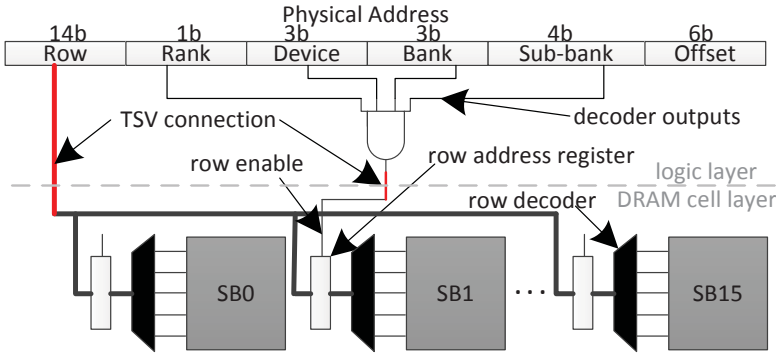


Figure 5.7: Address decoding and early sub-bank select. The full address is decoded by rank, device, bank, and sub-bank decoders and a "row enable" signal is asserted to latch the row address. MC is responsible for the freedom of sub-bank conflict so that no second "row enable" is asserted to an active sub-bank.

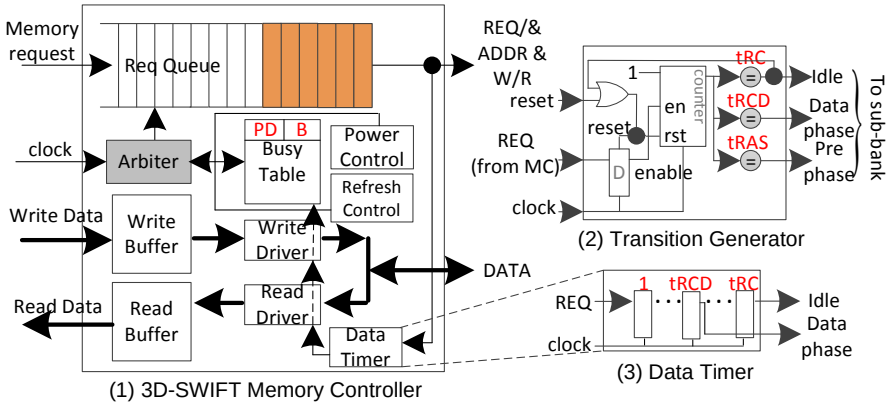


Figure 5.8: Memory controller design in 3D-SWIFT. (1) Memory controller; (2) Transition generator; (3) Data timer. The length of the counter and shifter are determined by the cycle number of tRC.

To enable the sub-bank autonomy, the automatic state transition shown in Figure 5.5 must be designed so that the sub-bank knows how to move forward to complete data transaction. A dedicated transition generator is deployed as a timer to signal the sub-rank to complete transition (see Figure 5.8). According to the cycle number of tRC, (15cycles), a 4-bit counter is sufficient to count the cycle number in the transition generator. Whenever one of the control signals "Data phase", "Pre phase" or "Idle" is asserted, the sub-bank automatically moves to the next state till the time it returns to idle.

5.2.7 The Design Of Memory Controller

As mentioned, the proposed packet-based interface protocol can significantly reduce the design complexity and area overhead of MC. Unlike a conventional MC that needs to maintain a bank status table for guarding the timing constraints, a 3D-SWIFT MC (Figure 5.8) only needs to affirm the freedom of sub-bank conflict with simple logic. In addition, command queues are

removed to improve area efficiency. Considering the negligible footprint of data timer, 3D-SWIFT is able to reduce MC's area and make it simpler for verification.

Different from HMC, of which the MC is transparent to the user, the smaller MC in 3D-SWIFT can be integrated into the processor die with few pin-outs. As a consequence, processor designers can apply system-level optimization on the "visible" MC. We believe this is meaningful for the wide adoption of 3D-SWIFT.

Busy Table. As MC is responsible to avoid sub-bank conflict, a busy table is deployed to track sub-banks' status. As shown in Figure 5.8, one table entry has two bits: 1) the "busy" bit tells whether the sub-bank is idle ('0') or busy ('1'); and 2) the "PowerDown" bit indicates if the sub-bank is in PowerDown mode. Since the size of busy table is relatively small (4,096bits), a 512B multi-port (4R4W) register file is simply employed. The area and power of the busy table is estimated based on [76].

Data Timer As mentioned in Section 5.2.3, MC and 3D-SWIFT have the agreement that they know exactly when to put the data on the data bus. Different from a sub-bank that gets the information from the transition generator, MC completes the data transfer with the assistance of a data timer. The data timer is in fact a bit shifter that has tRC bits. Figure 5.8 illustrates the function of the data timer. A valid request sets the left-most bits to indicate a data transfer is scheduled. Once the "Data phase" is asserted, MC knows there should be a data transfer in the following cycle. The "Idle" notifies MC to clear the busy bit in busy table. Two data timers are used to distinguish the data read and write. They also help MC meet the bus turn-around constraint.

5.3 Design Overhead Analysis

Since DRAM is cost-sensitive and area is the major contributor to the memory cost, we strive to minimize the area overhead of our sub-bank design in 3D-SWIFT. In this section, we elaborate the floorplan of the 3D-SWIFT with sub-bank design in addition to the description in Section 5.2.

5.3.1 Sub-bank DRAM Floorplan

In general, our goal is to increase the data width per mat to output all bits in the sense amplifier (i.e., row buffer) without introducing extra metal layer or area overhead on wire interconnect. Figure 5.9 shows the 3D-SWIFT floorplan inside a bank which consists of 16 sub-banks. As shown, each sub-bank has 8 mats on one layer, with a total size of 8Mb over four layers. Note that we keep a consistent mat size with the same 512 wordlines and 512 bitlines as in the conventional DRAM design. To minimize the overhead on intra-bank wiring incurred by the sub-bank partitioning, we leverage similar design as in the prior fine-grained 3D DRAM design [72].

In the conventional 2D DRAM, since each mat outputs 4 data bits, there are 512/4=128 column select lines per mat. Note that the numbers of column select lines (M3) and global data lines (M3) in a mat are complementary, meaning that the product of the two numbers is in fact

Table 5.1: Hardware Implementation Summary

Name	Type	Area	Power
Rank decoder & Device decoder	5-32dec	58.92μm^2	9.40μW
Bank decoder	3-8dec	13.41μm^2	2.05μW
Subbank decoder	4-16dec	27.87μm^2	4.68μW
Transition generator	counter	55.75μm^2	18.71μW
Data timer	shifter	93.25μm^2	35.85μW
Row address register	register	101.61μm^2	35.92μW
Busy table [76]	regfile	0.02mm^2	25mW

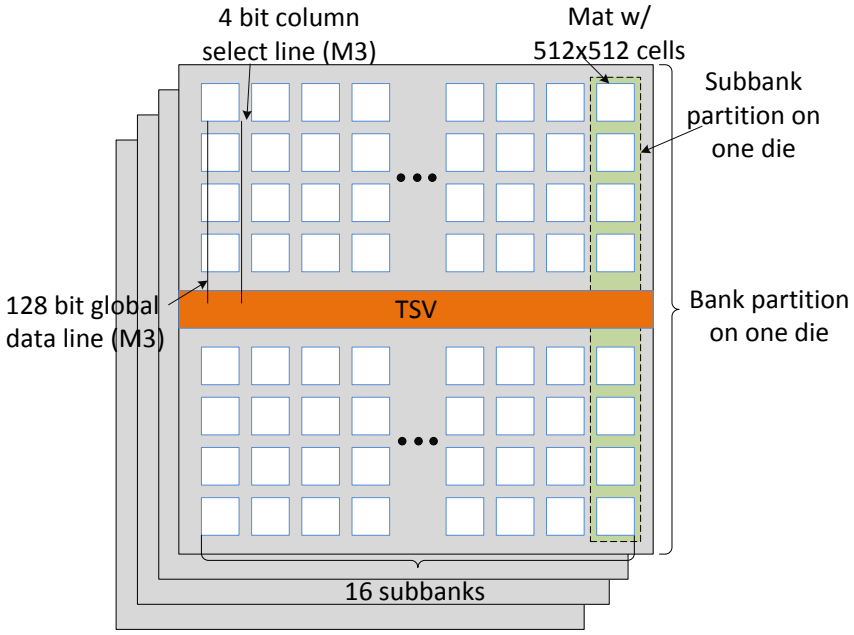


Figure 5.9: The Floorplan of 3D-SWIFT. A bank consists of 32 mats and each mat is 256Kb (512×512).

the total number of bitlines of a mat. Alternatively, in 3D-SWIFT we can switch the usage of the wires and have 128 data lines and 4 column select lines. This is easily accomplished by flipping the direction of the three-state driver that connects the column select line and data line (the driver input is the sense amplifier output). In this way, the output of a mat is increased by 128/4=32 times without incurring any area overhead or wire routing overhead. In order to provide the full 512 bits from each mat's sense amplifier to feed a entire cache line, we use an internal burst length of 4.

Simply switching the column select line and global data line is at the risk of increasing the data line access latency, because the original column select lines are usually densely routed above the mats and are relatively narrow and slow. Specifically, the new global data line has a pitch of only 8F (while F is feature size=45nm). Fortunately, because the we can partition the bank and sub-bank into different die layers (connected with TSVs), the length of the global dataline (equal to the height of the bank) is effectivelly shortened by four folds. In addition, we can layout the

TSVs in the middle ground between the mats and share the TSVs between the half sub-banks on two sides as shown in Figure 5.9, which further reduces the data line length by half. In this way, the wire latency and power consumption is effectively reduced to 1/8 of the original.

5.3.2 TSV Overhead

Based on the above-mentioned design, we basically eliminate the data wire routing overhead. The area overhead now comes from two parts: 1) the address/command buses and the row address latches routed to the individual sub-banks because they need to be accessed independently. 2) The TSV layout overhead. Here we assume that all sub-banks in a bank share a common 128-bit wide TSV data bus. We modified CACTI-3DD [1] to model the sub-banked 3D DRAM with the aforementioned configurations, and compared the results against a Wide-IO design with the same memory capacity and mat size. The results shows that our design has only a negligible 1.6% area overhead with each DRAM die size as 29.3mm^2.

Specifically for the TSV overhead part, according to ITRS' projection [77], the size of TSV will quickly shrink to 2-4µm. Based on the published 3D DRAM prototype [66, 67], the TSV used in this work is set as 2µm wide, 6µm high, and has 4µm pitch. The TSV count is calculated as follows. In one bank, there are 128 TSVs for data delivery. In addition, one redundant TSV is inserted to address the TSV reliability issue [78]. Given 14-bit row address and 5-bit control signals, one bank needs 147 TSVs and 4,704 TSVs are employed in each layer, which consumes about 0.075mm^2 as one TSV takes 16µm^2.

In addition, the wire model in CACTI-3DD is used to calculate the bus delay, where 1mm wire introduces 0.087ns delay and TSV has 0.03ns delay under 45nm technology. Since the RC delay is proportional to the wire length, the longest path, which crosses four layers and goes from the corner to the center, is around 5.35mm and thus results in about 0.58ns delay. Compared to 2.5ns clock period, the signal propagation delay does not incur extra bus latency (in cycle).

Control Logic Overhead. Except the busy table, all control logics in the 3D-SWIFT and MC have been implemented and verified by ModelSim. All designs are further synthesized by Design Compiler with TSMC 45nm technology for the area and power analysis. Table 5.1 shows the the synthesis results.

As the MC of 3D-SWIFT can be integrated into the processor die, it does not consume any area in 3D-SWIFT. The main area overhead caused by the control logic is from the dedicated row address register. Given 2,048 sub-banks in 3D-SWIFT, the row address register consumes 0.21mm^2 area in total (four layers). Similarly, the area overhead of transition generator is 0.12mm^2, if each sub-bank also has a dedicated generator. In fact, the number of transition generator can be reduced to the value of tRC (in cycle) by sharing the generators among all sub-banks, as at least one transition generator is free after a row cycle. However, the shared transition generators induce the internal multiplexing logic and distribution network. Therefore, we insist on the dedicated transition generator. Accounting for the address decoder, the area overhead caused by control logic circuit is 0.35mm^2, which is only 1.2% of one layer.

Table 5.2: Simulation Platform Configuration

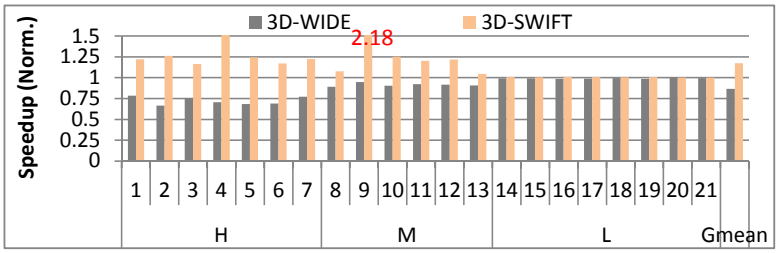
System	
Cores	4, ALPHA, out-of-order
CPU Clock Freq.	3 GHz
LDQ/STQ/ROB Size	32 / 32 / 128 entries
Issue/Commit Width	8 / 8
L1-D/L1-I Cache	32kB / 32kB 2-way 2-cycle latency
D-TLB/I-TLB Size	64 / 48 entries
L2 Cache	Shared, Snooping, 4MB, LRU 8-way, 15-cycle latency
Memory	
2D	JEDEC-DDR3, 2GB, 2 ranks 8 banks(×8), 64-bit bus, 800MHz (1.6GHz DDR) tRAS-tRCD-tRP-tTAW: 28-10-10-24
3D-WIDE	128-bit/channel, 4-channel, 400MHz tRAS-tRCD-tRP-tTAW: 20-8-8-20
3D-SWIFT	128-bit/channel, 4-channel, 400MHz tRAS-tRCD-tRP-tTAW: 20-8-8-**0**

5.4 Experiment

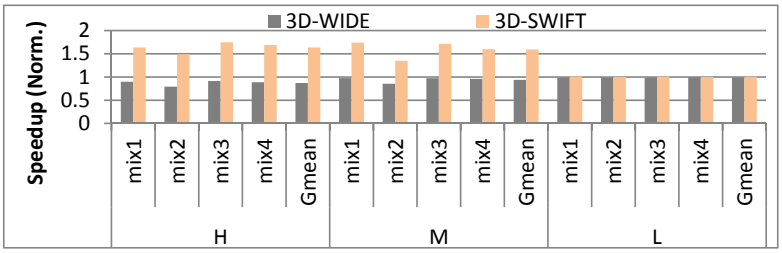
In this work, we adopt gem5 [34] as our simulation platform. We modified the DRAMSim2 [26] and successfully integrated it to gem5 as the DRAM model. SPEC2006 [35] and STREAM [36] benchmarks are employed as multi-programmed benchmarks. Normalized Instructions-Per-Cycle (IPC) is used as the speedup criteria. Table 5.2 shows the gem5 setup during the simulation. In this work, two memory systems are developed as our reference models. **2D** is the base model that simulates a commodity JEDEC DDR3-1600 SDRAM. **3D-WIDE** is a Wide-IO DRAM model that leverages the wider memory bus to deliver the burst data. For the fairness of comparison, even though 3D-WIDE has four independent channels, all channels are used as ranks and connected to a single MC. In addition, the data bus in 3D-WIDE run at 400MHz with double data rate, which is envisioned as the next generation Wide-IO. FR-FCFS scheduling scheme [59] is used in 2D and 3D-WIDE, to maximize the row buffer hit rate, whilst FCFS is deployed in 3D-SWIFT due to the close-page policy. The key timing parameters are shown at the bottom of the table. Note that the tTAW of 3D-SWIFT is 0 to indicate it eliminates the power constraint.

5.4.1 Single-Core Simulation

Firstly, we characterize the benchmark in the single-core simulation. We run each SPEC2006 CPU benchmark with reference input size for 100 million instructions to warm up the cache and another 100 million instructions for the statistics. According to the miss per kilo instructions (MPKI) of last level cache (LLC), we classify the benchmarks into three categories. The symbols H, M, and L stand for the applications that have high (>10), medium ([1, 10]), and low memory intensity (<1), respectively. Only eight benchmarks are selected as the representatives of L class because the rest have almost the same results. Table 5.3 lists the classification and the corresponding MPKIs. All selected benchmarks are numbered as shown in the table.



(a) Single-core



(b) Four-core

Figure 5.10: Performance results of single-core and 4-core simulation. All results are normalized to 2D DRAM design.

Table 5.3: Benchmark Classification

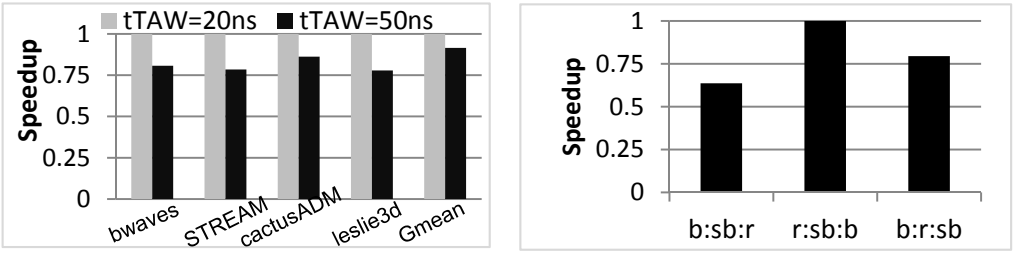
	#**Benchmarks(MPKI)**
H	[1]bzip2(45.89), [2]bwaves(36.62), [3]zeusmp(19.37), [4]gobmk(37.69), [5]sjeng(33.51), [6]lbm(26.44), [7]STREAM(34.43)
M	[8]milc(5.13), [9]cactusADM(7.85), [10]leslie3d(8.33), [11]libquantum(6,94), [12]wrf(7.33), [13]astar(1.01)
L	[14]soplex(0.16),[15]gamess(0.12), [16]gromacs(0.14),[17]sphinx3(0.08), [18]gcc(0.12),[19]hmmer(0.08),[20]GemsFDTD(0.001),[21]namd(0.04)
Mix*H*	mix1: bzip2, bwaves, zeusmp, lbm mix2: lbm sjeng, gobmk, stream mix3: bzip2, sjeng, zeusmp, stream mix4: zeusmp , bwaves, gobmk, lbm
Mix*M*	mix1: milc, cactusADM, leslie3d, wrf mix2: astar, libquantum, wrf, milc mix3: cactusADM, leslie3d, astar, libquantum mix4: wrf, libquantum, leslie3d, astar
Mix*L*	mix1: gcc, gamess, gromacs, namd mix2: GemsFDTD, soplex, hmmer, sphinx3 mix3: gcc, soplex, gromacs, sphinx3 mix4: GemsFDTD, gamess, hmmer, namd

The performance result of signle-core simulation is shown in Figure 5.10a. 3D-WIDE experience 13.5% performance drop on average. The performance degradation of 3D-WIDE mainly stems from its larger access latency, which adversely affects the memory parallelism. The results from H and M benchmarks indicate that 3D-WIDE has poor performance with memory-intensive applications. In contrast, 3D-SWIFT only has small performance loss in some H and M benchmarks. On average 3D-SWIFT achieve 17.8% and 38.4% performance improvement over 2D and 3D-WIDE, respectively.

In particular, `cactusADM` has 2.18X improvement in 3D-SWIFT. Note that `cactusADM` has

Table 5.4: DRAM Configurations

	Structure			Timing Parameters	
	Capacity	(Sub-)Bank Size	I/O Width	tCK	tRAS-tRCD-tRP-tTAW
2D (base)	2GB	16384×8192	64b	1.25ns	28-10-10-24
3D-WIDE	2GB	16384×8192	128b	5ns	10-4-4-10
3D-WIDEx2	2GB	16384×8192	128b	2.5ns	20-8-8-20
3D-SWIFT	2GB	16384×512	128b	2.5ns	10-4-4-**0**



(a) Impact of tTAW

(b) Impact of address mapping

Figure 5.11: Sensitivity Study. (a) Impact of tTAW; (b) Performance with different address mapping schemes. "r"–row, "b"–bank, "sb"–sub-bank.

absolutely random accesses as the row buffer hit rate is 0. Therefore, 3D-SWIFT can fully take advantage of sub-bank activation to maximize the memory concurrency. This observation indicates that 3D-SWIFT is more promising in a multi-core system in which the intensive memory requests are more likely to be random due to the interference among applications.

5.4.2 Four-Core Simulation

We randomly select benchmarks from each category for four-core simulation. The mixed benchmarks are listed at the bottom of Table 5.3 and Figure 5.10b presents the results. Similar to the single-core simulation, 3D-SWIFT performs better in H and M benchmarks. In general, 3D-SWIFT achieves 64.7% (58.2%) and 87.6% (67.3%) improvement over 2D, and 3D-WIDE, respectively for H-Mix (M-Mix) benchmarks. The application interference that destroys the data locality and makes the memory request more random is the main reason for the performance improvement, as observed in [3].

5.4.3 Impact Of Power Constraint (tTAW)

As mentioned, the tTAW becomes larger (50ns) in [11]. We intentionally apply a 20ns tTAW to 3D-WIDE to see the impactof tTAW. As shown in Figure 5.11a, tTAW has significant impact on the memory-intensive applications. Particularly, STREAM has 21.5% performance drop compared with larger tTAW. On average, 8.5% performance degradation is observed, which results from the more restricted power constraint on Wide-IO. As a result, 3D DRAM design should carefully cope with the power constraint and 3D-SWIFT is one of potential solutions.

5.4.4 Address Mapping

As shown in Figure 5.11b, we evaluate the impact of three address mapping schemes: "bank(b):subbank(sb):row(r)", "b:r:sb", and "r:sb:b" and compare it to the baseline "r:b:sb" which maximizes the sub-bank-level parallelism. First of all, there is no difference to prioritize the sub-bank-level (r:b:sb) or bank-level parallelism (r:sb:b) since both levels can be accessed in parallel. In contrast, both b:r:sb and b:sb:r introduces significant performance drop. The reason is these two schemes suppress the sub-bank-level parallelism. In particular, b:sb:r is the worst case since it maps a large continuous memory space to a single sub-bank, where lots of sub-bank conflicts are fired so that following memory requests must stall and access the sub-bank sequentially. As a result, the overall performance drops by 33.5% and some applications even experiences 75% degradation. Therefore, 3D-SWIFT adopts r:b:sb as the solution.

5.5 Related Work

In this section, we give a brief review of the prior arts that are related to our proposal. In general, the prior arts can be classified into two categories: optimizations in 2D and 3D DRAM architectures, respectively.

5.5.1 Related Work in Conventional 2D DRAM Architecture

Several fine-grained DRAM structures have been proposed in conventional 2D DRAM. For example, Fujitsu implemented the Fast-cycle RAM (FCRAM) that had similar sub-bank structure and achieved faster access speed and lower power consumption than the baseline 2D DRAM [15]. However, FCRAM has limited memory capacity and parallelism due to the physical constraint. Cooper-Balis *et al.* leveraged post-CAS to enable the fine-grained access in the commodity DRAM [2]. The earlier decoded column address helped to activate a small fraction of row. However, this work fails to make use of the fine-grained structure for performance improvement. Instead, one access still locks the whole bank so that it limits the memory parallelism and increases access latency. A. Udipi *et al.* also proposed single sub-array access (SSA) [3]. It is impractical to be implemented in 2D DRAM due to large area overhead [30]. In contrast, 3D-SWIFT take advantage of TSVs to enable the fine-grained access.

Zheng *et al.* introduced a bridge chip MRB to split the original rank to mini-ranks [62]. Similar to 3D-SWIFT, the mini-rank not only significantly reduced power consumption but also improved the memory concurrency to compensate the potential performance degradation by narrowing down the data bus. In spite of the power optimization, the extra MRB increases the DRAM cost with possible performance drop. Leveraging the mini-rank structure, Yoon *et al.* implemented a memory system that had adaptive access granularity [63]. Recently, Kim *et al.* proposed a memory architecture that explored the sub-array-level parallelism (SALP) [4]. As mentioned, SALP could overwhelm the MC design complexity as the number of sub-array keeps increasing. The tFAW and tRRD constraint should significantly limit the performance

gain as well. In addition, SALP did not reduce the page size, which may exacerbate the power consumption. Again, all approaches mentioned above play with the DRAM physical constraints so that they can only provide low memory bandwidth.

5.5.2 Related Work in 3D DRAM Architecture

Numbers of studies have been published to exploit the advantages of 3D DRAM. In particular, 3D Wide-IO DRAM has been standardized by JEDEC [11]. However, based on the experimental result, the current Wide-IO memory can only provide low bandwidth if multiple channels are not employed. As a result, it is more likely to be used in the mobile devices rather than the workstation that has high memory bandwidth requirement. On the other hand, Loh proposed several novel 3D DRAM architectures: Loh explored the impact of increasing the number of ranks, channels and MSHRs (miss status handling register) in 3D DRAM [68]. He also leveraged the white space in logic layer to implement a large row buffer "cache" to increase the row buffer hit rate [71].

Weis *et al.* proposed a 3D DRAM that could provide fine-grained memory access. The 3D DRAM interface can be configured for flexible bandwidth and burst length so that it can provide on-demand memory bandwidth [70]. The I/O power can be reduced when a narrower data is delivered because it effectively deactivates some of I/O drivers. The large row (1KB), however, needs to be activated even if the flexible bandwidth is enabled. Obviously, this fine-grained 3D DRAM should still abide with tTAW power constraint. It did not exploit the the opportunity to improve memory parallelism. Woo *et al.* raised a novel 3D DRAM structure to enable the row buffer cache in 3D DRAM [72]. The folded-bank structure is implemented by the trade-off between layout area overhead and energy consumption. Different from the folded-bank scheme that loads the whole page into row buffer cache, the TSV bus in 3D-SWIFT is shared by all banks.

5.6 Summary

In this work, we propose a fine-grained 3D-stacked Wide-IO DRAM architecture–3D-SWIFT, to exploit high memory parallelism for improving performance and power efficiency. Sub-bank autonomy and packet-based interface protocol are devised to simplify the MC design. The experiment results show that 3D-SWIFT can achieve 64.7% and 87.6% performance improvement than conventional 2D DRAM and 3D Wide IO, respectively. The results indicates the promising future of 3D-SWIFT.

As part of future work, the impact of various data placement strategies and address mapping policies on 3D-SWIFT wil be explored to find the optimal solution. The effectiveness of applying 3D-SWIFT to non-volatile memory (NVM) (e.g., STT-RAM and PCRAM) will be also evaluated as another follow-up work.

Chapter 6

The Prototype Chip Tapeout Of A 3D SoC With 3D-Stacked DRAM

As mentioned, 3D-stacked DRAM is promising to mitigate the memory wall in the traditional 2D DRAM architecture. To demonstrate the feasibility of 3D-stacked DRAM, a System-on-Chip (SoC) design for H.264 application using 3D DRAM memory stacking is presented in this chapter. In addition, to fully leverage the benefit of on-chip 3D DRAM stacking, a dedicated memory controller is designed with simplified double data rate (DDR) protocol. Several optimizations, such as parallel access policy and TSV clustering, have been implemented to make better use of on-chip DRAM and 3D stacking.

6.1 Chip Overview

As shown in Figure 6.1, the 3D chip is composed of two logic tiers and three DRAM tiers. Tier Logic-1 is on the top and Logic-2 lies in the middle. Both logic tiers are sized by $2.5\times5.0\text{mm}^2$ while the DRAM tiers are $12.3\times21.8\text{mm}^2$. Logic-2 is much thinner than other tiers since most of silicon substrate of this tier is burnished to expose TSVs. Note that all of the I/O pads are on the back surface of DRAM tier, which mandates TSVs to account for data and power delivery between DRAM and logic tiers as well.

6.1.1 The Architecture Of 3D SoC

To support real-time H.264 applications, a 3D SoC is proposed as shown in Figure 6.2. AMBA AHB is employed as the system bus [79]. An H.264 encoder, a USB On-The-Go controller, and a RISC processor UniCore-2 are also integrated into this SoC [80] [81]. A dedicated 3D DDR controller is developed to communicate with the 3D DRAM. Additionally, a JTAG interface is used to load the initial instructions into memory for the system bootup.

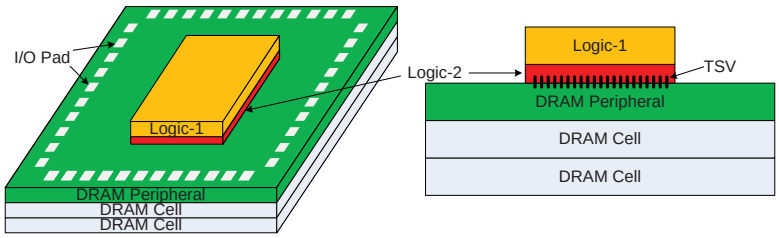


Figure 6.1: 3D SoC with On-chip Memory. The 3D SoC consists of five dies: two logic dies and three DRAM dies. The two logic dies are stacked together via micro-bumps in a fashion of Face-to-Face (F2F). In addition, Logic-2 is thinned to expose TSV for the communication with the 3D DRAM. The 3D DRAM is composed of one peripheral die and two cell dies. It can provide eight independent channels while we use four of them in this work.

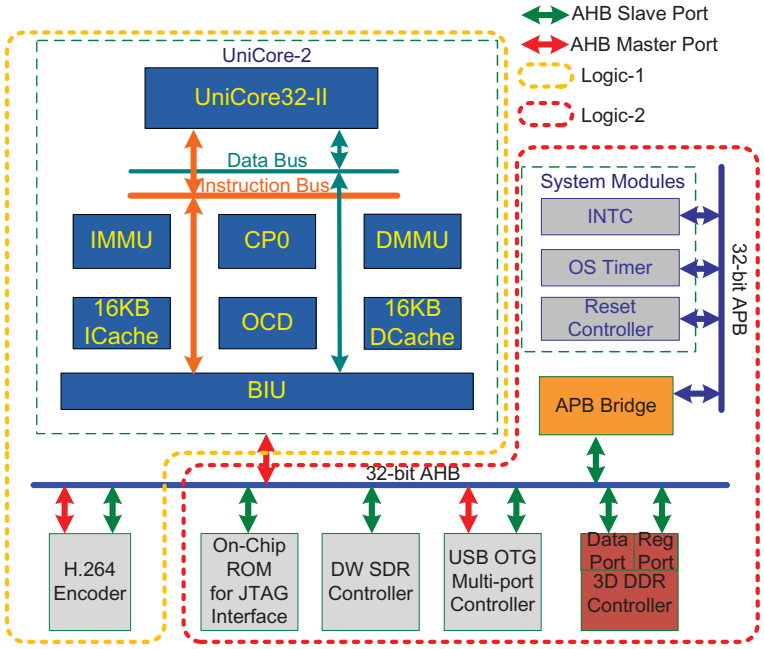


Figure 6.2: Block View of 3D SoC. We reuse the SoC from Peking University, where it employs a UniCore RISC processor. The SoC is partitioned into two parts that are mapped to Logic-1 and Logic-2, respectively. Note that the 3D DRAM controller is placed on Logic-2 so that it is close to the 3D DRAM.

6.1.2 The Use Of 3D-Stacked DRAM

The 3D DRAM used in this work is the state-of-the-art product from Tezzaron [41]. To achieve both high performance and high cell density, the DRAM chip is separated into one peripheral tier and multiple cell tiers, where each tier can be optimized separately by different technologies. The total capacity of this 3D DRAM is 2Gb with 1Gb on each cell tier. Eight data channels with separate write/read ports (128b) are used to attain high memory bandwidth. By using

the simplified DDR protocol, every data channel allows 256-bit data transfer in a single memory cycle. A prominent feature of these channels is that each channel is fully independent of the others, which means all channels can be accessed in parallel at different frequencies. In addition, a MailBox channel is used to initialize the DRAM when the system is powered on. Table 6.1 lists the basic parameters of this 3D DRAM.

Table 6.1: Parameters of 3D DRAM

# of Tiers	3
Total Capacity	2 Gb (256 MB)
Clock Frequency	1 GHz (Max.)
Refresh Mode	Automatic
Refresh Rate	64 ms
# of Data Channel	8
Data Width Per Channel	128 b
# of Pins Per Channel	294
Burst Length	4 or 8

6.2 3D Optimization

In this section, we show two optimizations implemented in our 3D SoC.

6.2.1 Parallel Access Policy

From our observation, AHB bus can only have at most two outstanding transactions at the same time. Due to this limitation, even though the DRAM has eight independent channels, a two-channel *parallel access policy* is sufficient to improve the performance. We categorize those back-to-back AHB transactions as: Read-After-Read (RAR), Read-After-Write (RAW), Write-After-Write (WAW), and Write-After-Read (WAR). Based on our understanding, except for WAR, we have the opportunity to optimize the first three patterns by allowing the second outstanding transaction to be detected and processed without any stall, if they have different bank requests.

In other words, under certain conditions, the DRAM controller does not necessarily wait for the completion of the first memory access but can initiate the second access command to the 3D DRAM immediately. For example, Figure 6.3 shows the case of RAR, where the second read follows the first read. As shown, the DDR controller originally can only process each transaction in sequence even if the second read requests the data from a different bank. As a result, the second read needs to stall for a few cycles before it can be served. In contrast, with the parallel access policy, the second transaction can be processed immediately if it references a different data bank as shown in Figure 6.3.

Another DRAM finite state machine (FSM) is replicated to allow the second transaction to be processed. The replica is illustrated as the dashed green block in Figure 6.4a. After AHB

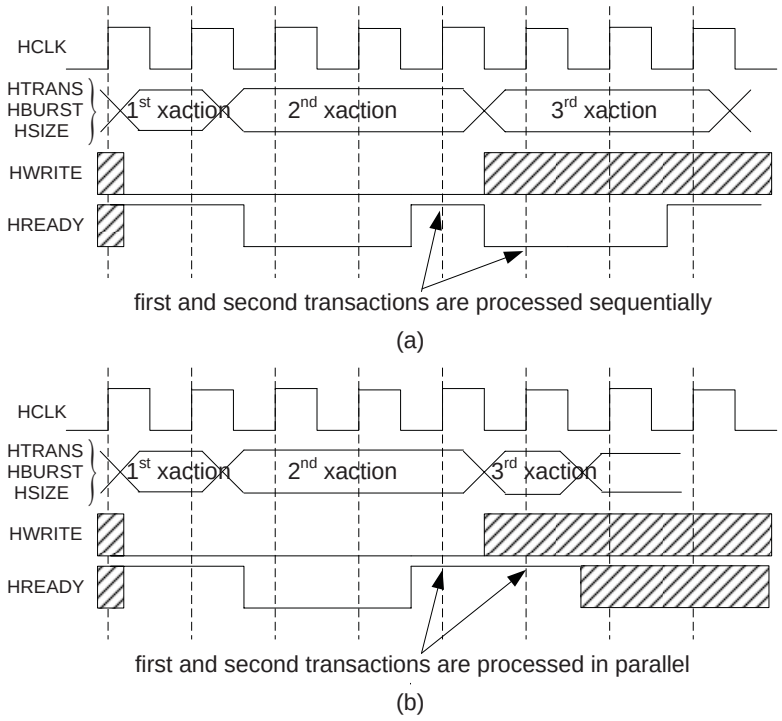
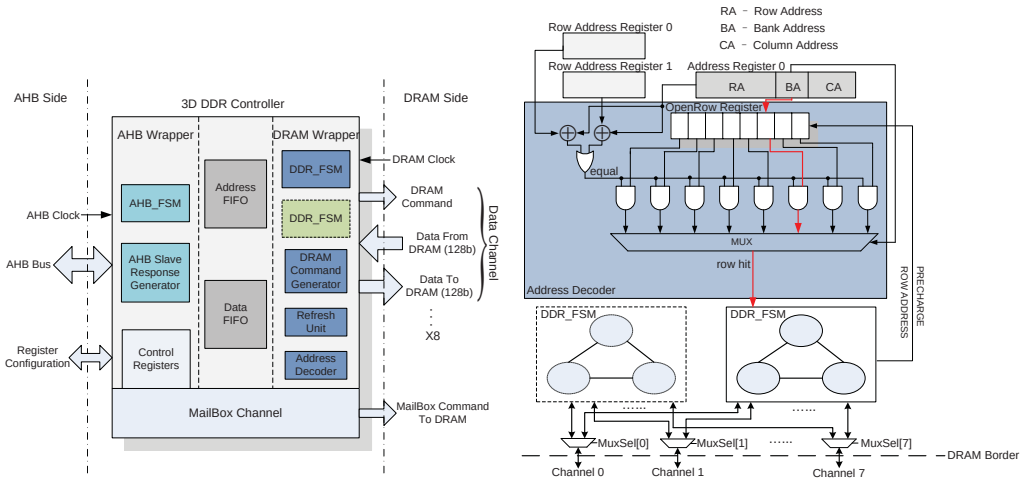


Figure 6.3: Parallel Access Policy. a) the transaction in the conventional memory controller. The first and second transactions are processed sequentially, which cannot be hidden due to the lack of parallelism; b) the transaction in the proposed 3D memory controller. The first and second transaction can be issued in parallel due to the increasing channels in 3D DRAM.

wrapper detects the second transaction, it informs DDR wrapper that the second access request is coming. DDR wrapper then wakes up the replica to respond to the second memory access. Sometimes, the first access may suffer a row miss while the second access enjoys a row hit. In this case, the second access will be retired prior to the first one. Considering the in-order property of AHB protocol, the DDR controller should hold the second result until the first access finishes. Figure 6.4b shows the additional hardware to support the parallel access.

6.2.2 TSV Clustering

TSVs are inserted between Logic-2 and DRAM as signal carriers. As presented in Figure 6.7, all of TSVs must be capped with backside metal bondpoints on one end. The backside metal bondpoint then connects to the bondpoint on DRAM with back-to-back bonding. Bunches of dummy TSVs are also inserted into Logic-2 to meet the requirement of TSV density. In addition, to achieve high reliability, multiple TSVs are aggregated to form a TSV cluster for signal delivery. Two types of TSV clusters are employed to deliver data as well as power. *DRAM Cluster* is utilized for dedicated DRAM signals, while *I/O Cluster* is used to connect to the I/O pads on the surface of DRAM. Each DRAM cluster contains 10 TSVs and each I/O cluster contains 26



(a) Duplicated FSM (b) Hardware Support

Figure 6.4: Block Diagram of 3D DRAM Controller. a) two FSMs are deployed to improve the overall bandwidth. Asynchronous FIFOs are used to deal with the synchronization between AHB clock domain and DRAM clock domain; b) The design detail of the memory controller.

TSVs (Figure 6.7). Table 6.2 lists the physical characteristics and the total number of effective TSVs (excluding dummy TSV) used in this work.

Table 6.2: TSV Parameters

Diameter	1.2 μm
Pitch	4.0 μm
Depth	6.0 μm
Resistance	$<$0.6 Ω
Capacitance	2–3 fF
Num. per DRAM Cluster	10
Num. per I/O Cluster	26

6.3 The Proposed Design Flow

The comprehensive design flow used in this work is presented in Figure 6.5. Analogous to the conventional 2D flow, we simply categorize it into frond-end and back-end. Only differences from conventional 2D flow are listed in this section to highlight the changes in 3D chip design.

6.3.1 Divide and Conquer Methodology

Divide-and-Conquer methodology is used by applying traditional 2D physical design methods to each tier. The floorplanning, placement, wire routing as well as clock tree generation are all done

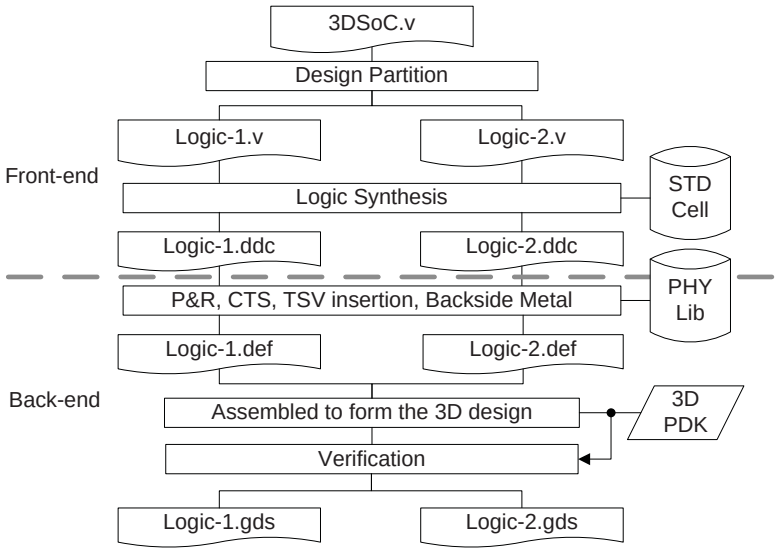


Figure 6.5: 3D SoC Design Flow. The flow is divided into front-end and back-end. The front-end flow mainly contains the design partition and logic synthesis. The back-end flow runs various design rule checking and TSV insertion. The merge of two netlists is also done in the back-end.

within a single tier. As an example, Figure 6.6 illustrates the CTS flow. Even though a lot of work have been done to exploit the design space of 3D-aware clocking synthesis, the TSV Both Logic-1 and Logic-2 conducts the conventional CTS by SoC Encounter. In the figure, the source clock from I/O pad is firstly delivered onto Logic-2 and further propagates to Logic-1 through the logic bondpoint covered in next paragraph.

6.3.2 Front-End Design

In the front-end, the logic partition is applied to the SoC architecture. Currently, the partitioning is primarily based on the power and area budget of each tier. According to power and area evaluation results, UniCore-II and H.264 encoder, which consume more power and area, are placed on Logic-1, and the rest of components, including the DRAM controller, are grouped into Logic-2 (shown in Figure 6.2). In this way, the hotter tier gets closer to the heat sink, which help mitigate the thermal issue in 3D ICs. After partitioning, the logic synthesis is applied to each tier to generate the gate-level netlists by Synopsys™ Design Compiler® [29].

6.3.3 Back-End Design

Two logic tiers are stacked face-to-face as shown in Figure 6.7. The top metal layer is pre-defined and no modification is allowed to guarantee the alignment. Additionally, we take the shared-bus design into account to make it much easier to combine two tiers. The shared bus reveals an advantage in the bonding due to the characteristics of standardization and centraliza-

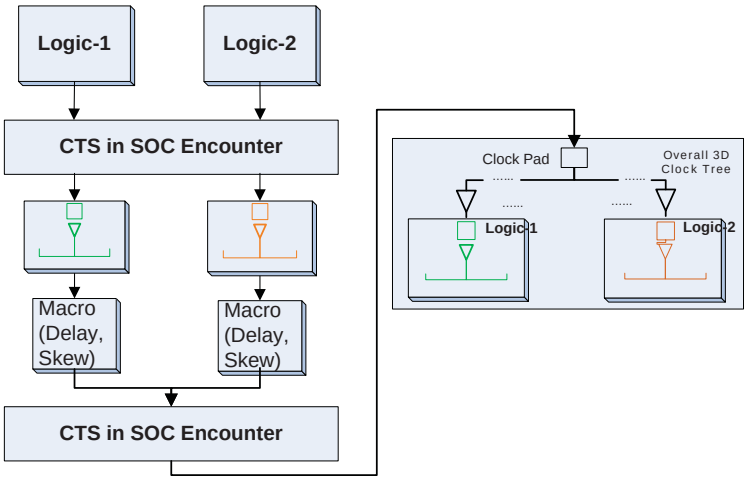


Figure 6.6: An example of Divide and Conquer methodology. Firstly, CTS is conducted in Logic-1 and Logic-2 separately. Then the overall clock tree is built once the two netlists are integrated.

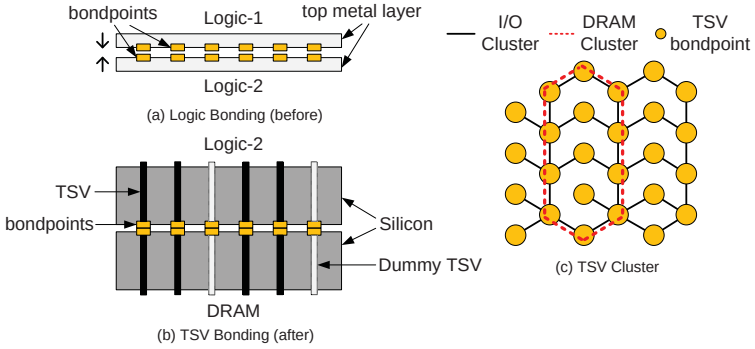


Figure 6.7: 3D Bonding. a) logic bonding. Logic-1 and Logic-2 are bonded face-to-face. The bond-point is predefined in the top metal layer of each die; b) Similar to logic bonding, the TSV is bonded in the back-side metal layer; c) multiple TSVs compose a TSV cluster to enhance the reliability and connectivity.

tion. Otherwise, the distributed inter-tier communication incurs more efforts on logic bonding to achieve the accurate alignment and thus introduces more complexity and potential failures. In this work, AHB is placed in the center of Logic-2 by allocating two sets of AHB master interfaces for bonding (Figure 6.9). Meanwhile, both AHB interfaces of Unicore-II and H.264 encoder are correspondingly placed in the center of Logic-1 so that they can adhere to Logic-2 directly.

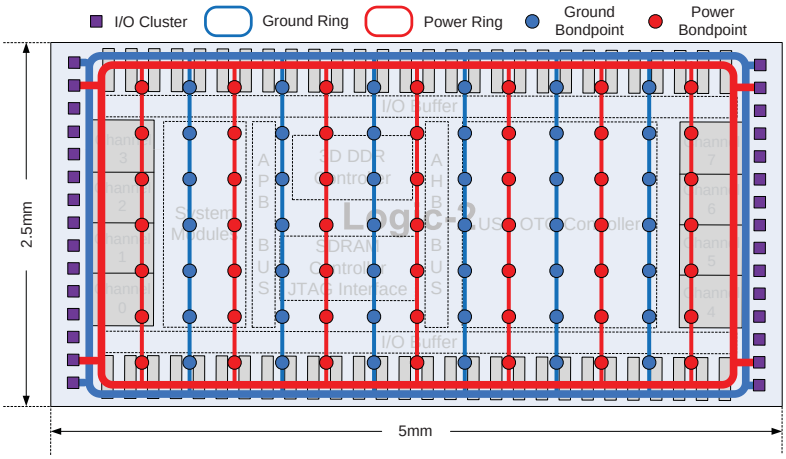


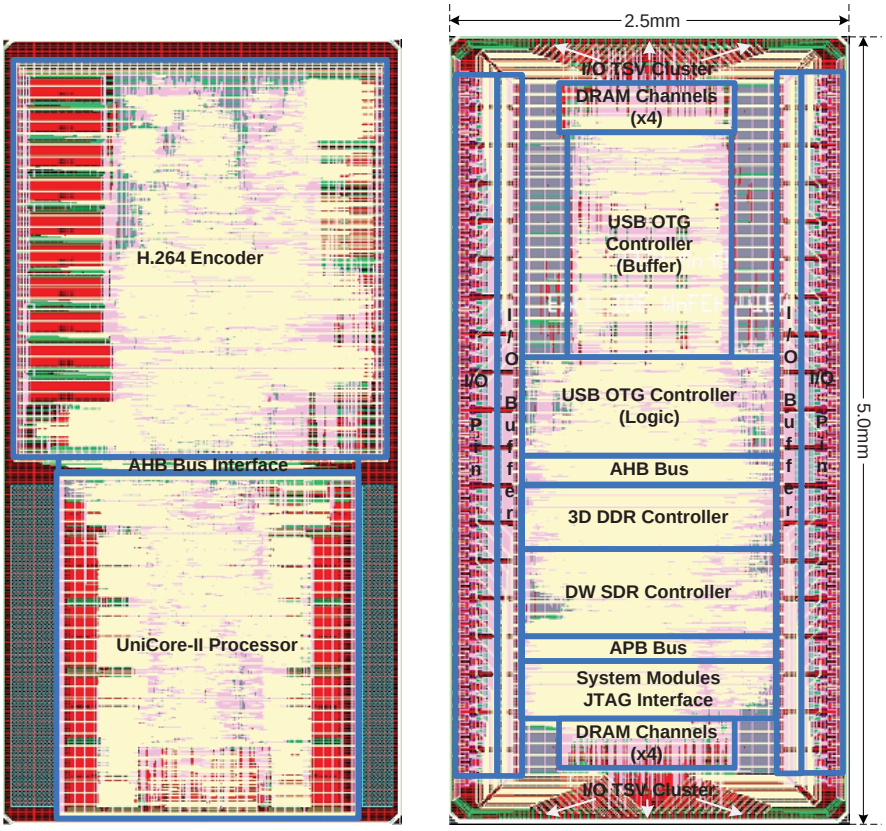
Figure 6.8: Power Delivery Network on Logic-2. The power and ground ring are placed at the edge of the chip while they are placed alternatively in the middle to supply the power to the devices.

Table 6.3: System Design Summary

Area	6.2 mm^2 (Logic-1) 7.3 mm^2 (Logic-2)
Power	101 mW (Logic-1) 70 mW (Logic-2)
Clock Freq.	60 MHz
Temperature	25℃–85℃
Supply Voltage	1.5 V
# of Data Pad	61
# of P/G Pad	128
# of DRAM Channels	4
Data Bandwidth	4.25 GB/s (w/o Parallel Access) 8.5 GB/s (with Parallel Access)

6.3.4 Power Delivery

Like the data signals, all power/ground signals are delivered from power I/O to tier Logic-2 first through I/O clusters. Figure 6.8 illustrates the power delivery network on Logic-2. As shown in the figure, regular power/ground rings are generated along the edges of logic tiers. To deliver power to Logic-1 layer, multiple power/ground rows are implemented on Logic-1 and Logic-2. One power/ground row is composed of an array of bonding points in both top metal layers. The interleaved power/ground rows cover the whole chip to meet the density requirement of the top vias.



(a) Logic-1 (b) Logic-2

Figure 6.9: Layout of Two Logic Tiers. I/O TSVs are placed on the top and bottom edge of Logic-2. In addition, the eight DRAM channels are divided into two groups and placed on the top and bottom as well.

6.4 Summary

The prototype chip has been fabricated with GlobalFoundries' 130nm low-power process together with Tezzaron's TSV bonding technology. Table 6.3 summarizes the design result. The supply voltage of this chip is 1.5V. Figure 6.9 shows the layout of logic tiers. Stacking on the 3D DRAM chip, the DRAM controller can communicate with DRAM via eight data channels. In Figure 6.9, eight DRAM channels are divided into two groups and placed on the top and at the bottom of tier Logic-2, respectively. Half of them, however, are used in practice due to the area limitation and routing complexity. The lack of PLL results in a single clock to drive the chip. Considering the quality of the input clock, the frequency of the whole SoC is set at 60MHz, which is sufficient to run desired multimedia applications.

Chapter 7

Conclusion and Future Work

DRAM has been the *de facto* main memory technology in the past five decades. Even though several non-volatile memories (NVM) have emerged in recent years, DRAM has the unique advantages of balanced read and write operation, high write speed, low write energy, small footprint of sense amplifier. As a result, DRAM is still considered as the mainstream of main memory technology in the future. Due to the diverging exponential increase in the performance of processor and DRAM, however, DRAM becomes the bottleneck of the entire computing system, which is known as the "Memory Wall". DRAM is also hitting the scalability issue since its evolution is approaching the physical limitation. On the other hand, as more cores are integrated in a single chip, the system requires DRAM to provide low access latency, high data bandwidth and low power consumption. As a result, it is critical to optimize the contemporary DRAM at architecture level so that DRAM can survive in the future.

In this work, several optimizations to DRAM are proposed to make better trade-off among performance, power, and cost. In chapter 2, we first review the increasing refresh overhead as the cell density of a DRAM chip is improved. In theory, the refresh can lead to 11.2% performance loss in DDR3 DRAM when DRAM temperature is between 85°C and 95°C. To mitigate the refresh overhead, a concurrent refresh aware DRAM architecture (CREAM) is presented, which consists of sub-rank level refresh (SRLR) and sub-array level refresh (SALR). SRLR is used to eliminate the power constraint on the conventional all-bank refresh. Therefore, SRLR opens up the opportunity of a fine-grained refresh scheme. On the other hand, SALR is designed to isolate a refresh from the normal memory accesses. As long as there is no sub-array conflict, memory access and refresh can be executed in parallel. In this way, the number of memory accesses that are blocked by the refresh is significantly reduced and thus the refresh overhead can be hidden effectively in CREAM. The experimental results show that CREAM can achieve as much as 12.9% and 7.1% performance gain over the conventional memory and the memory in which the elastic refresh scheduling policy is applied.

Secondly, we develop Half-DRAM architecture in chapter 3, which redesigns the DRAM array to support the fine-grained activation. Even though the fine-grained activation has been proposed

by prior work [2, 3], we confirm that the existing fine-grained activation seriously ruins the n-bit prefetching. As a result, it induces either unacceptable performance overhead (30% performance drop when half-bank activation is applied) or unaffordable area overhead. Distinct from the prior fine-grained activation proposals, Half-DRAM enables half-bank activation by splitting one MAT into two slabs. One row address decoder now drives two slabs from two neighboring MATs. As a consequence, Half-DRAM improves the "1RD-1HFF" relationship to "2RD-1HFF", which means all helper flip-flops (HFFs) can be active while only half of a row is activated. Therefore, Half-DRAM enable half-row activation without bandwidth loss. Since only half of a row is activated, Half-DRAM can relax the power constraint tRRD and tFAW naturally. In addition, with the integration of sub-array level parallelism (SALP [4]), Half-DRAM can improve DRAM performance as well. The experimental results show that Half-DRAM can achieve 10.7% and 16.9% performance improvement and 8.4% power saving, with negligible hardware overhead. When it is applied to Wide-IO, Half-DRAM can obtain 38.1% performance gain.

In chapter 4, we present Lazy Precharge to reduce the precharge overhead. Precharge is an overhead in a row cycle because no data transfer involves. In theory, precharge can lead to 30% and 25.6% performance overhead for read and write accesses, respectively. Lazy Precharge is able to reduce the number of precharge by allowing multiple sub-arrays to share a single precharge command. Compared to the prior work [4], Lazy Precharge does not incur the conflict between activation and precharge, which makes it more practical to be implemented. As a result, it can significantly reduce the overall precharge overhead. Furthermore, three request scheduling policies are explored and evaluated. The experimental results validate the advantage of Lazy Precharge: On average 14% and 6.9% performance gain are observed under different scheduling policies.

3D-SWIFT is proposed in chapter 5 to exploit high memory parallelism for performance improvement. Due to the challenge of power delivery in 3D-stacked chips, Wide-IO [11] has more restricted power constraint that adversely affects the performance. To relieve the power constraint, 3D-SWIFT further splits a bank into multiple sub-banks. Each sub-bank can provide an entire cacheline via TSV. In addition, sub-bank autonomy is developed with the packet-based interface protocol to simplify the memory transaction. Correspondingly, we show a simplified memory controller design. The experimental results confirm that 3D-SWIFT can achieve 64.7% and 87.6% performance improvement than the commodity DRAM and Wide-IO, respectively.

Finally, a 3D SoC is illustrated in chapter 6 to demonstrate the feasibility of 3D ICs. The in-depth detail of front-end and back-end flow is shown separately, including the design partition, clock tree synthesis, power delivery and pad allocation. Moreover, a custom 3D DRAM controller is designed to communicate with the 3D-stacked DRAM through 4 channels. Parallel access policy is employed to make use of the improved bandwidth. The chip has been fabricated with GlobalFoundries' 130nm low-power process together with Tezzaron's TSV bonding technology. The chip is supposed to run at 60MHz with 8.5GB/s bandwidth provided by Tezzaron's 3D DRAM.

The war against the "Memory Wall" will never stop. As DRAM technology is approaching the physical limitation, 3D-stacked DRAM is promising in the future. Wide-IO [11], High Bandwidth Memory (HBM) [12], Hybrid Memory Cube (HMC) [13], and Octopus [41] have been sampled and supposed to be commercialized soon. How to fully make use of 3D-stacked DRAM, however, is still unclear. In particular, we need the answer from the community to decide the optimal connection scheme between processor and 3D-stacked DRAM. On the other hand, the read-write turnaround penalty becomes even larger as the operating frequency increases. As a result, it is critical to hide the turnaround overhead in the future DRAM chip.

Bibliography

[1] Chen, K., S. Li, N. Muralimanohar, J. H. Ahn, J. Brockman, and N. Jouppi (2012) "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in *DATE'12*, pp. 33–38.

[2] Cooper-Balis, E. and B. Jacob (2010) "Fine-Grained Activation for Power Reduction in DRAM," *MICRO*, **30**(3), pp. 34–47.

[3] Udipi, A. N., N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi (2010) "Rethinking DRAM Design and Organization for Energy-constrained Multi-cores," in *ISCA'10*, pp. 175–186.

[4] Kim, Y., V. Seshadri, D. Lee, J. Liu, and O. Mutlu (2012) "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA'12*, pp. 368–379.

[5] Keeth, B., R. J. Baker, B. Johnson, and F. Lin (2007) *DRAM Circuit Design: Fundamental and High-Speed Topics*, Wiley-IEEE Press.

[6] JEDEC Solid State Technology Association (2012), "JEDEC Standard: DDR4 SDRAM," http://www.jedec.org/sites/default/files/docs/JESD79-4.pdf.

[7] Micron, "MT41J128M16HA-125 Data Sheet," http://www.micron.com/products/dram/.

[8] Micron, "MT41J512M8RA-15E Data Sheet," .

[9] Wulf, W. A. and S. A. McKee (1995) "Hitting the Memory Wall: Implications of the Obvious," *SIGARCH Comput. Archit. News*, **23**(1), pp. 20–24.

[10] JEDEC Solid State Technology Association (2013), "LPDDR4 Moves Mobile," http://www.jedec.org/sites/default/files/D_Skinner_Mobile_Forum_May_2013_0.pdf.

[11] JEDEC Solid State Technology Association (2011), "JEDEC Standard: Wide I/O Single Data Rate Specification," http://www.jedec.org/standards-documents/results/jesd229.

[12] JEDEC Solid State Technology Association (2013), "JEDEC Standard: High Bandwidth Memory (HBM) DRAM," http://www.jedec.org/standards-documents/results/jesd235.

[13] Hybrid Memory Cube Consortium (2013), "Hybrid Memory Cube Specification 1.0," http://hybridmemorycube.org.

[14] Micron, "MT44K32M18RB-093 RLDRAM3," http://www.micron.com/products/dram.

[15] Fujitsu (2010) "Memory Consumer FCRAM 512M Bit MB81EDS516445," .

[16] JEDEC Solid State Technology Association (2003), "JEDEC Standard: Double Data Rate (DDR) SDRAM Specification," http://www.jedec.org/standards-documents/docs/jesd-79-c.

[17] JEDEC Solid State Technology Association (2012), "JEDEC Standard: LPDDR3," http://www.jedec.org/standards-documents/results/jesd209-3.

[18] JEDEC Solid State Technology Association (2010), "JEDEC Standard: LPDDR2," http://www.jedec.org/standards-documents/docs/jesd209-2e.

[19] seung Kang, Y. (2014), "Samsung to mass produce advanced computer memory," http://english.yonhapnews.co.kr/business/2014/03/11/0501000000AEN20140311004600320.html.

[20] Stuecheli, J., D. Kaseridis, H. C.Hunter, and L. K. John (2010) "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *MICRO'10*, pp. 375–384.

[21] JEDEC Solid State Technology Association (2009), "JEDEC Standard: DDR3 SDRAM Specification," http://www.jedec.org/standards-documents/docs/jesd-79-3d.

[22] Micron, "MT41J128M8JP-15E Data Sheet," http://www.micron.com/products/dram.

[23] Micron, "MT41J1G8THE-15E Data Sheet," .

[24] Jacob, B., S. W. NG, and D. T. Wang (2007) *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann.

[25] JEDEC Solid State Technology Association (2011), "DDR4 Mini Workshop," http://www.jedec.org/sites/default/files/JS_Choi_DDR4_miniWorkshop.pdf.

[26] Rosenfeld, P., E. Cooper-Balis, and B. Jacob (2011) "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, **10**(1), pp. 16–19.

[27] Ipek, E., O. Mutlu, J. F. Martínez, and R. Caruana (2008) "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA'08*, pp. 39–50.

[28] Micron, "TN-41-15: Low-Power Versus Standard DDR SDRAM," http://download.micron.com/pdf/technotes/DDR/tn4615.pdf.

[29] Synopsys, "Design Compiler," http://www.synopsys.com.

[30] Vogelsang, T. (2010) "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO'10*, pp. 363–374.

[31] Chatterjee, N., N. Muralimanohar, Bal, A. Davis, and N. P. Jouppi (2012) "Staged Reads : Mitigating the Impact of DRAM Writes on DRAM Reads," in *HPCA'12*, pp. 1–12.

[32] Ghosh, M. and H.-H. S. Lee (2007) "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *MICRO'07*, pp. 134–145.

[33] Liu, J., B. Jaiyen, R. Veras, and O. Mutlu (2012) "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA'12*, pp. 1–12.

[34] Binkert, N., B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, et al. (2011) "The gem5 Simulator," *Computer Architecture News*, **39**(2), pp. 1–7.

[35] Standard Performance Evaluation Corporation, "SPEC2006 CPU," http://www.spec.org/cpu2006.

[36] McCalpin, J. D., "STREAM Benchmark," http://www.cs.virginia.edu/stream.

[37] Liu, S., B. Leung, A. Neckar, S. O. Memik, G. Memik, and N. Hardavellas (2011) "Hardware/Software Techniques for DRAM Thermal Management," in *HPCA'11*, pp. 515–525.

[38] Zhu, Q., X. Li, and Y. Wu (2008) "Thermal Management of High Power Memory Module for Server Platforms," in *ITHERM'08*, pp. 572–576.

[39] Liu, S., K. Pattabiraman, T. Moscibroda, and B. G. Zorn (2011) "Flikker: Saving DRAM Refresh-power through Critical Data Partitioning," in *ASPLOS'11*, pp. 213–224.

[40] Brandt, T., T. Morris, and K. Darroudi (2007) "Analysis of the PASR Standard and its usability in Handheld Operating Systems such as Linux," *Intel Technical Report*.

[41] Tezzaron, `http://www.tezzaron.com`.

[42] Kang, U., H. Chung, S. Heo, S.-H. Ahn, H. Lee, et al. (2009) "8Gb 3D DDR3 DRAM Using Through-Silicon-Via Technology," in *ISSCC'09*, pp. 130–131.

[43] Kirihata, T., P. Parries, D. Hanson, H. Kim, J. Golz, et al. (2005) "An 800MHz Embedded DRAM With a Concurrent Refresh Mode," *IEEE Solid State Circuits*, **40**(6), pp. 1377–1385.

[44] Akesson, B., K. Goossens, and M. Ringhofer (2007) "Predator: A Predictable SDRAM Memory Controller," in *CODES+ISSS'07*, pp. 251–256.

[45] Bhat, B. and F. Mueller (2011) "Making DRAM Refresh Predictable," *Real-Time System*, **47**(5), pp. 430–453.

[46] Hoelzle, U. and L. Barroso (2009) *The Datacenter as a Computer*, Morgan & Claypool Publishers.

[47] Lim, K., J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch (2009) "Disaggregated Memory for Expansion and Sharing in Blade Servers," in *ISCA'09*, pp. 267–278.

[48] Malladi, K., F. Nothaft, K. Periyathambi, B. Lee, C. Kozyrakis, and M. Horowitz (2012) "Towards Energy-Proportional Datacenter Memory with Mobile DRAM," in *ISCA'12*, pp. 37–48.

[49] Yoon, D. H., J. Chang, N. Muralimanohar, and P. Ranganathan (2012) "BOOM: Enabling Mobile Memory based Low-Power Server DIMMs," in *ISCA'12*, pp. 25–36.

[50] Sudan, K., K. Rajamani, W. Huang, and J. B. Carter (2012) "Tiered Memory: An Iso-Power Memory Architecture to Address the Memory Wall," *IEEE Transactions on Computers*, **61**(12), pp. 1682–1696.

[51] Micron, "DDR3 Power Calculator," http://www.micron.com/products/dram.

[52] Micron, "TN-41-01: Calculating Memory System Power for DDR3," http://www.micron.com/products/dram.

[53] Hur, I. and C. Lin (2008) "A Comprehensive Approach to DRAM Power Management," in *HPCA'08*, pp. 305–316.

[54] David, H., C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu (2011) "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC'11*, pp. 31–40.

[55] Malladi, K., I. Shaeffer, L. Gopalakrishnan, D. Lo, B. Lee, and M. Horowitz (2012) "Rethinking DRAM Power Modes for Energy Proportionality," in *MICRO'12*, pp. 131–142.

[56] Lee, Y., S. Kim, S. Hong, and J. Lee (2013) "Skinflint DRAM System: Minimizing DRAM Chip Writes for Low Power," in *HPCA'13*, pp. 25–34.

[57] Kaseridis, D., J. Stuecheli, and L. K. John (2011) "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *MICRO'11*, pp. 24–35.

[58] Poremba, M. and Y. Xie (2012) "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *ISVLSI'12*, pp. 392–397.
URL `http://www.nvmain.org`

[59] Rixner, S., W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens (2000) "Memory Access Scheduling," in *ISCA'00*, pp. 128–138.

[60] Healy, M. B. and S. K. Lim (2011) "Power-Supply-Network Design in 3D Integrated Systems," in *ISQED'11*, pp. 223–228.

[61] Jain, P., D. Jiao, X. Wang, and C. H. Kim (2011) "Measurement, Analysis and Improvement of Supply Noise in 3D ICs," in *VLSI Circuits Digest*, pp. 46–47.

[62] Zheng, H., J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu (2008) "Mini-rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency," in *MICRO'08*, pp. 210–221.

[63] Yoon, D. H., M. K. Jeong, and M. Erez (2011) "Adaptive Granularity Memory Systems: A Tradeoff between Storage Efficiency and Throughput," in *ISCA'11*, pp. 295–306.

[64] Kaseridis, D., J. Stuecheli, and L. K. John (2011) "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *MICRO'11*, pp. 24–35.

[65] Lee, D., Y. Kim, V. Seshadri, J. Liu, and O. Mutlu (2013) "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA'13*, pp. 615–626.

[66] Zhang, T., Y. Xie, et al. (2010) "A Customized Design of DRAM Controller for on-chip 3D DRAM Stacking," in *CICC'10*, pp. 1–4.

[67] Kim, D. H., K. Athikulwongse, M. Healy, M. Hossain, M. Jung, et al. (2012) "3D-MAPS: 3D Massively Parallel Processor with Stacked Memory," in *ISSCC'12*, pp. 188–190.

[68] Loh, G. H. (2008) "3D-Stacked Memory Architectures for Multi-core Processors," in *ISCA'08*, pp. 453–464.

[69] Woo, D. H., N. H. Seong, D. L. Lewis, and H.-H. S. Lee (2010) "An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth," in *HPCA'10*.

[70] Weis, C., I. Loi, L. Benini, and N. Wehn (2012) "An Energy Efficient DRAM Subsystem for 3D Integrated SoCs," in *DATE'12*, pp. 1138–1141.

[71] Loh, G. H. (2011) "A Register-file Approach for Row Buffer Caches in Die-stacked DRAMs," in *MICRO'11*, pp. 351–361.

[72] Woo, D. H., N. H. Seong, and H.-H. S. Lee (2011) "Pragmatic Integration of an SRAM Row Cache in Heterogeneous 3-D DRAM Architecture Using TSV," *TVLSI*, pp. 1–13.

[73] HPCWire, "NVIDIA Bill Dally Talks 3D Chips and More at GTC," http://www.hpcwire.com/hpcwire/2012-05-16/nvidia%E2%80%99s_bill_dally_talks_3d_chips_and_more_at_gtc.html.

[74] Kim, Y., D. Han, O. Mutlu, and M. Harchol-Balter (2010) "ATLAS: A Scalable and High-performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA'10*, pp. 43–54.

[75] Muralidhara, S. P., L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda (2011) "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning," in *MICRO'11*, pp. 374–385.

[76] Shah, T. (2010) "FabMem: A Multiported RAM and CAM Compiler for Superscalar Design Space Exploration," *Thesis*.

[77] International Technology Roadmap for Semiconductors (2009), http://www.itrs.net/.

[78] Hsieh, A.-C., T. Hwang, M.-T. Chang, M.-H. Tsai, C.-M. Tseng, and H.-C. Li (2010) "TSV Redundancy: Architecture and Design Issues in 3D IC," in *DATE'10*, pp. 166–171.

[79] ARM (1999), "AMBA Specification," .

[80] Chen, J.-W., C.-Y. Kao, and Y.-L. Lin (2006) "Introduction to H.264 Advanced Video Coding," in *ASPDAC'06*, pp. 736–741.

[81] Cheng, X., X. Wang, J. Lu, J. Yi, et al. (2010) "Research Progress of UniCore CPUs and PKUnity SoCs," *Journal of Computer Science and Technology*, **25**(2), pp. 200–213.

[82] Jacob, B. (2009) *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*, Morgan & Claypool Publishers.

Vita

Tao Zhang

Tao Zhang is a six-year Ph.D candidate in the Department of Computer Science and Engineering, Pennsylvania State University. Before he joined Penn State in 2008, he had received B.S. and M.S. degree from Peking University. He is now working with Dr. Yuan Xie in the Microsystems Design Laboratory (MDL). His research interests mainly fall into Computer Architecture, 3D IC and On-chip Networks. In particular, he focusses on the solution of memory subsystem, including commodity DRAM and emerging 3D-stacked memory. He has published 16 conference papers. He has also served as a peer reviewer for several journals and conferences in the field of Computer Architecture, VLSI design, and EDA, including ACM Transactions on Embedded Computing Systems (TECS), IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), IEEE Transactions on Circuits and Systems II (TCASII), IEEE Transactions on Computers (TC), IEEE Transactions on Very Large Scale Integration Systems (TVLSI), IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC), IEEE International Symposium on High Performance Computer Architecture (HPCA). In 2013, he joined Nvidia Corporation and serves as a senior architect in the Tegra Memory Subsystem team.