ADDRESSING PROLONGED RESTORE CHALLENGES IN FURTHER SCALING DRAMS

by

Xianwei Zhang

B.E. in Software Engineering, Northwestern Polytechnical University, 2011

Submitted to the Graduate Faculty of
the Kenneth P. Dietrich School of
Arts and Sciences in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Xianwei Zhang

It was defended on

July 14, 2017

and approved by

Dr. Youtao Zhang, Department of Computer Science, University of Pittsburgh

Dr. Bruce R. Childers, Department of Computer Science, University of Pittsburgh

Dr. Wonsun Ahn, Department of Computer Science, University of Pittsburgh

Dr. Jun Yang, Electrical and Computer Engineering Department, University of Pittsburgh

Dr. Guangyong Li, Electrical and Computer Engineering Department, University of Pittsburgh

Dissertation Director: Dr. Youtao Zhang, Department of Computer Science, University of Pittsburgh

ADDRESSING PROLONGED RESTORE CHALLENGES IN FURTHER SCALING DRAMS

Xianwei Zhang, PhD

University of Pittsburgh, 2017

As the *de facto* memory technology, DRAM has enjoyed continuous scaling over the past decades to keep performance growth and capacity enhancement. However, DRAM further scaling into deep sub-micron regime faces significant challenges. Among the induced issues, prolonged restore time is expected to be one of the major concerns, but it has been paid little attention. Aiming at restore issue, this thesis performs pioneering studies to characterize the problems, and presents techniques from different perspectives to overcome them.

First, our experimental studies quantify the significant restore process variations, causing serious degradations on yield and/or performance. To solve the problem, we propose schemes to expose the variations to the architectural levels. Fast restore chunks can thus be constructed utilizing DRAM organization, and they can be exposed to the memory controller to effectively compensate the performance loss. Further, we maximize the improvement by applying restore-time-aware rank construction and hotness-aware page allocation schemes to fully utilize the fast regions.

Second, in addition to simply expose the variations to higher levels, we investigate DRAM cell structures and behaviors finding that refresh and restore are two strongly correlated operations. Whereas are being fully restored after each read or write access, DRAM cells are always being fully charged by periodical refresh operations, providing an opportunity to early terminate restore. With the insight, we first propose to truncate a restore using the time distance to next refresh. Further, to provide more truncation opportunities, we integrate the multirate-refresh concepts to shorten the distance by increasing the refresh

rate of recently accessed regions.

Lastly, we explore higher to the application level with the inspiration that a large set of applications can well tolerate output accuracy loss and runtime errors, enabling us to exploit approximate computing to mitigate prolonged restore. By utilizing the variance in restore timing exhibited at different row segments, we reduce the restore time such that only partial segments are fully reliable. We then map the critical data onto the reliable segments to keep the application-level errors low. Atop of the approximation-aware technique, we further generalize it to support precise computing as well.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF EQUATIONS

PREFACE

I would like to thank all the fantastic people I have met and worked with in the past wonderful years. First and foremost, I would like to thank my advisor, Professor Youtao Zhang, for great guidance and for providing all research resources. During the Ph.D program, he has put tremendous amount of effort to guide me through each step to explore and research. I am also grateful for the excellent supervisions from Professor Bruce R. Childers and Professor Jun Yang. They taught me lots of things about doing research, and constantly provided insightful discussions and feedbacks both on my research projects and conference presentations.

I want to thank all the members of my thesis committee: Professor Youtao Zhang, Professor Bruce R. Childers, Professor Jun Yang, Professor Wonsun Ahn, and Professor Guangyong Li, for their time, efforts and valuable inputs into both the thesis proposal and final dissertation. Thanks for the constructive suggestions ranging from high-level research directions to detailed implementations. Without the great help, this dissertation would not be possible.

All members in the labs of Professor Youtao Zhang and Professor Jun Yang deserve my special thanks for the valuable and enjoyable discussions, and for the great help both in and outside research. Further, it is also my privilege to work on GPU in Nvidia Architecture Research Group. I learnt a lot from my mentors and colleagues both on GPU knowledge and research skills. Moreover, I also want to express my sincere thanks to all the friends I have met in Pittsburgh, Austin and conference venues for the great time and pleasure.

And last, but not least, my deepest appreciation goes to my family for their endless support and always understand. To my beloved parents, who always encourage me to pursue dreams with infinite love. To my wife, Feng, a great life companion, who has always been by my side to share happiness and to overcome challenges.

This work has been generously sponsored by National Science Foundation (NSF) under grants CCF-1422331, CNS-1012070, CCF-1535755 and CCF-1617071, and by University of Pittsburgh under Teaching Assistant Scholarship and Andrew Mellon Fellowship, etc.

1.0 INTRODUCTION

As a fundamental component of computing systems, main memory bridges the fast processors to the slow massive storage devices. Servicing as the primary repository of runtime data and instructions, main memory plays significant role on the whole system performance and cost. Since being invented in 1960s, dynamic random access memory (DRAM) has been the *de facto* standard in the past few decades thanks to its structural simplicity, and is now widely deployed in almost all systems, including servers, desktops, mobile devices and embedded systems. After generations' evolutions, DRAM has reached a right combination on intertwining aspects like high capacity, short latency and low cost, which cannot be effectively achieved in the potential alternative memories like PCM [Lee et al., 2009a], STT-RAM [Kultursay et al., 2013] and 3D-XPoint [Patterson, 2015].

Wheres DRAM's capacity and bandwidth have been significantly increased in each generation, the growth rates are much slower than that of processor performance, and the unbridgeable gap is still widening [Wilkes, 2001; Jevdjic et al., 2014]. Memory footprints of running applications continue enlarging, and emerging data intensive platforms and applications, such as in-memory database [sap] and Deep Learning applications [Diamos et al., 2016], place greater need for high capacity; more and increasingly massive multithreading cores, e.g., GPGPUs [Lindholm et al., 2008] and heterogeneous cores [Vijayarahavan et al., 2017], keep demanding high capacity and bandwidth; meanwhile, DRAM latency and power consumption has been one of the most significant bottlenecks in modern computing, and will be a profound challenge in future Exascale computing [DoE, 2015].

1.1 PROBLEM STATEMENT

DRAM's great success is tremendously contributed by its continuous technology scaling to grow DRAM chip capacity and bandwidth, to reduce power consumption and to lower the per-bit cost. While scaling-down has been the conventionally effective method, it is starting hit a brick wall where DRAM cells cannot be made smaller without jeopardizing their robustness.

As scaling down, DRAM cells, composed of one transistor and one capacitor (as introduced in Chapter 2), have been shrinking to smaller dimensions, which results in size reduction of access transistor, storage capacitor and peripheral circuits. Firstly, smaller capacitor translates into a lower capacitance, reducing the stored charge; secondly, scaled DRAMs apply lower supply voltage [Mukundan et al., 2013; Zhang et al., 2016a], which further decreases per-cell charge and also worsens the gate induced drain leakage [Nair et al., 2013b]; meanwhile, at smaller dimensions, adjacent cells are likely to electrically disturb each other [Kim et al., 2014]; moreover, smaller cell geometry increases the resistance on both access transistor and bitline [Mukundan et al., 2013; Wang, 2015], obstructing the cell charging process. In addition to charge decrease, the input offset voltage on the sense amplifiers is also expected to exacerbate [Zhang et al., 2016a; Mukundan et al., 2013], which makes it more difficult to sense data content stored in the cells.

Moreover, scaling to smaller technology nodes grow the fabrication complexity and makes it more challenging to precisely control the dimensions of DRAM cells, which introduces severer process variations (PVs). Accordingly, DRAM cells are expected to show more statistical behaviors in a much wider range, as shown by the example restore distribution in Figure 1.1; and, such uncertainty might cause increasingly more cells to violate the design specifications, which is beyond the tolerance of existing mechanisms such as row/column redundancy and ECC [Nair et al., 2013b; Jacob et al., 2007]. As a result, a large number of chips would fail the manufacture testings.

The above phenomena of reduced dimensions and increased variability together adversely affect the data retention time, charge restoration and voltage sensing, which impact DRAM performance, reliability and cost from several aspects: (i) lower stored charge and higher

leakage current introduces more leaky cells, resulting in more frequent refresh operations that harms performance, energy and reliability; (ii) slower charging process means longer restoring operation and potentially failing the standard specifications, which could hurt both performance and yield; (iii) sensing difficulty lengthens the normal read/write accesses, and might cause access failures; (iv) close proximity of cells leads to electrical coupling effects, which introduces erroneous data.

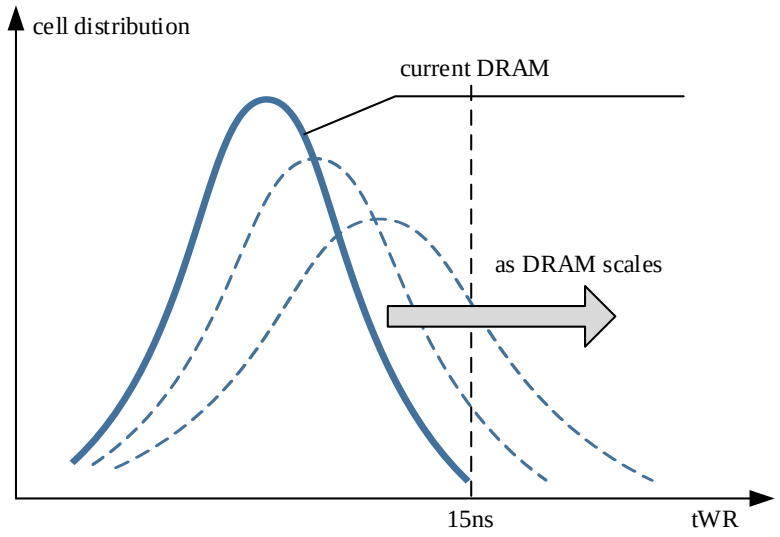


Figure 1.1: The cell restore time prolongs as DRAM scales.

Although technology scaling becomes increasingly challenging, DRAM is still irreplaceable and hence it must keep advancing to satisfy the overwhelming requirements from systems and applications. To tackle the scaling issues, whereas significant researches have been put into DRAM retention studies [Hamamoto et al., 1998; Wong et al., 2008; Kim and Lee, 2009], refresh optimizations [Ghosh and Lee, 2010; Stuecheli et al., 2010; Liu et al., 2012a, 2013; Mukundan et al., 2013; Nair et al., 2013b,a; Agrawal et al., 2014; Khan et al., 2014; Qureshi et al., 2015; Bhati et al., 2015b], and sensing time reductions [Son et al., 2013; D. Lee et al., 2013; Shin et al., 2014; Lee et al., 2015], restoring has not been paid attention to until recently and very few explorations [Kang et al., 2014; Choi et al., 2015] have been performed.

1.2 RESEARCH OVERVIEW

In this thesis, our research objective is to explore the restoring trends in further scaling DRAM, and propose schemes to mitigate the induced issues like performance loss and yield degradation. Specifically, this thesis plans to answer the following questions: (i) how serious will be the restore scaling in future DRAMs? (ii) how to solve the enlarged timing constraints utilizing the memory organization characteristics? (iii) can we utilize the DRAM cell behaviors to find solutions by integrating the restore with other memory operations? (iv) what kind of assistance can we get from above-architecture layers like operating system and the applications? Generally, this thesis tries to explore DRAM scaling from restoring perspective with collaborative efforts from different layers as shown in Figure 1.2, covering hardware architecture ❶, operating system ❷ and applications ❸, etc.

1.2.1 Achieve Fast Restore via Reorganization

Conventionally, a single set of timing constraints is applied to the whole memory system, which becomes undesirable in smaller feature size, because of the enlarged worst-case timing values. To preserve high chip yield, we have to relax the timings to allow the worst cells reliably finishing operations, which indicates a longer bank occupancy and thus degraded performance.

Fortunately, the variation characteristics also provide a large portion of healthy cells, which can be potentially utilized to compensate the bad ones. Without modifying the DRAM cell structure, we can choose to expose the timing variations to architectural level. The memory controller can thus be enhanced to be variation-aware to mitigate the performance loss. The exploration can be performed either on coarse chip-level or fine chunk-level, depending the overhead budgets and improvement goals.

Nevertheless, given the facts that each DRAM logical row is physically composed of multiple rows from different chips, and each chip row consists of thousands of cells, naively exposing timings is likely to offer very limited help. Accordingly, we choose to form logical rows by clustering similar fine-granularity chunks from different chips [Zhang et al., 2015a].

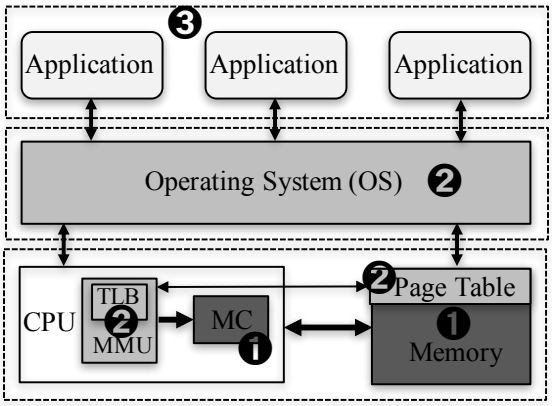


Figure 1.2: Overall view of thesis work. While placing a focus on memory structures, we explore restoring effects from other layers, including page translation from operating system and inherent behaviors of applications.

As a result, more fast memory regions can be constructed. In addition, we extend the idea to assembly phase in manufacturing stage, where compatible chips are grouped together to form ranks, and hence bad chip contamination could be well controlled [Zhang et al., 2017b].

From processor perspective, the aforementioned variation exploration provides non-uniform memory access (NUMA) characteristics, which can be utilized by operating system to speed up program execution [Zhang et al., 2017b]. Following the idea, we profile workloads to identify frequently accessed pages, and then allocate them to fast memory regions to maximize performance gains.

1.2.2 Shorten Restore using Refresh

Normal read/write accesses restore charge into cell capacitors to store data values. Due to DRAM's intrinsic leakage, the charge leaks over time, causing the stored data to be lost. To prevent this, periodical refreshes are required to recharge the cells. Given that cell voltage monotonically reduces between two refreshes and each refresh always fully charge the cells, complete charge is thus unnecessary if the access is close in time to the next refresh.

The observation motivates us to propose refresh-distance-aware partial restore [Zhang et al., 2016a]. Restore operation is performed with respect to the distance to the coming

refresh, and the closer to next refresh, the less charge is needed and thus the earlier the restore operation can be terminated. For ease of management, the idea is implemented by partitioning one refresh window (typically 64ms) into multiple sub-windows, each of which has its own restoring goal, i.e., different timings.

Whereas DRAM necessities frequent fresh to avoid data loss, the vast majority of cells can hold the data for much longer time, which inspires designs of multiple rate refresh [Kim and Lee, 2009; Liu et al., 2012a; Bhati et al., 2015b]. Compared to refresh operations, restore contributes more critically to memory access latency and overall system performance, and hence we could achieve more truncation opportunities by upgrading refresh rate. Correspondingly, we apply higher refresh frequency, e.g., 64ms, to those less leaky rows with longer refresh window, e.g., 256ms, to further shorten restore timings.

However, blind upgrading introduces more refresh operations, which not only prolongs memory unavailable period but also consumes more energy. As a compromise, we selectively upgrade the refresh rate of selected bins, those were recently touched, and thus incur modest refreshing overhead to the system.

1.2.3 Mitigate Restore in Approximate Computing Scenario

While much performance loss in further scaling DRAMs can be recouped with the help of the proposed memory designs and the operating system, this might be over qualified for nowadays popular applications in domains like computer vision, machine learning and big data analytics. Those applications commonly have intrinsic resilience to inaccuracy, which provides a good opportunity to tolerate the slow-to-restore cells in memory [Zhang et al., 2016b, 2017a].

With the insights, this thesis applies approximate computing to tolerate slow restore timings by mapping critical data bits to fast row segments at a low application error rate. To exploit the restore variation across different row segments of a DRAM row, we reduce the restore timings such that only partial segments are fully reliable, i.e., other segments might contain faulty bits. With the reduced timings, we then map the critical bits of annotated variables [Sampson et al., 2011, 2013; Miguel et al., 2014, 2015] to reliable memory regions

to keep low application-level errors.

Given the fact that not all applications can tolerate accuracy loss, we generalize the technique to support precise-computing by stitching together fast, reliable row segments to form a faster row. And further, we integrate to have a hybrid scheme speeding up both approximate and precise computing.

1.3 CONTRIBUTIONS

This thesis makes the following contributions:

- We perform pioneering studies on DRAM restoring in deep sub-micron scaling to confirm and characterize the problems. We build models to simulate restoring behaviors, and then generate DRAM devices to faithfully repeat the manufacturing process and perform architectural-level studies.
- Targeting at restoring issues, we propose schemes from different perspectives. On device and architectural levels, we apply chunk remapping and chip clustering techniques to exploit the exposed variance, and further to achieve fast memory accesses; on system level, we maximizing performance improvement by allocating hot pages of the running workloads to fast regions.
- Going further, we investigate applications in domains like machine learning and computer vision, and find that they can tolerate final output quality loss, and thus can be utilized to mitigate restore issues. Hence, we apply approximate computing onto further scaling DRAMs to strike a good balance among performance, energy and accuracy.
- Our principles in the thesis are universal and thus can be used to explore wider topics in memory and even other components; the principles include examining the problems at varying levels from different perspectives, designing for common and specific cases instead of the worst case, and co-operating memory and system, hardware and software, etc.

1.4 OUTLINE

The rest of this thesis is organized as follow: Chapter 2 introduces the DRAM structures, operations and scaling issues, and also presents related work on timing reduction, restore and approximate computing, etc. In Chapter 3, we build models to study restoring effects, and then propose a series of techniques to shorten restoring timing values from memory organization perspective. In Chapter 4, we utilize the correlation between restoring and refresh to seek the opportunities to early terminate restore operations. In Chapter 5, we explore restore in approximate computing scenario, and devise schemes to strike the balance between performance and accuracy. Summary and future work are discussed in Chapter 6.

2.0 BACKGROUND AND RELATED WORK

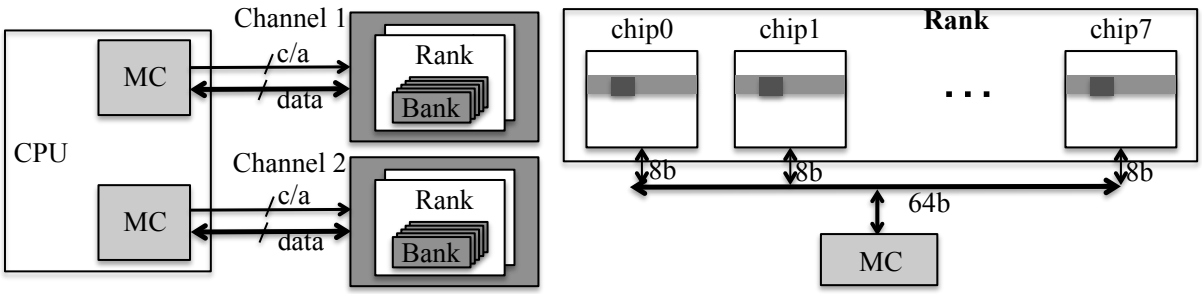
2.1 DRAM STRUCTURE AND ORGANIZATION

DRAM-based main memory system is logically organized as a hierarchy of channels, ranks and banks, as illustrated by Figure 2.1(a). `Bank` is the smallest structure to be accessed in parallel with each other, which is termed as bank-level parallelism [Mutlu and Moscibroda, 2008; Lee et al., 2009b]. And, `rank` is formed by clustering multiple, typically eight [1] , banks which operate in lockstep, i.e., all banks in a rank respond to a single command received from memory controller (MC). Lastly, one `channel` is composed of an on-chip memory controller and several ranks that share the same narrow command/address and wide data bus.

Physically, DRAM is provided as DIMM (dual in-line memory module), typically with 2 ranks on each side; each rank is composed of multiple chips, inside which eight banks are deployed as cell arrays. The logical bank, as shown in Figure 2.1(a), is physically made up of the same numbered bank from all chips. For instance, `bank 0` of a rank contains `bank 0` [2] residing in all chips in the rank. Likewise, a DRAM row is dispersed across chips, as shown in Figure 2.1(b). In normal accesses to a rank, each chip provides 8 bits at a time simultaneously, which together satisfy the total data bus width of 64-bit, termed as a `beat`. In addition, to amortize memory access overhead on processor side and also to bridge the giant gap between DRAM core frequency (about 200 MHz) and bus frequency (over 1000 MHz), n-bit prefetch and burst access is supported [Yoon et al., 2011; Zhang et al., 2014b].

[1] For illustration purpose, we assume the memory chips are x8, i.e., 8 data I/O pins. The overall structure keeps the same for x4 and x16, except the number of chips in a rank.

[2] Without specific comment in the rest of the thesis, `bank` refers to a logical bank, which is across chips in a rank; for banks residing in a chip, we would specifically call as chip bank to differentiate. The same rule goes with `row`.



(a) Logical hierarchy **(b)** Rank organization

Figure 2.1: DRAM high-level structure.

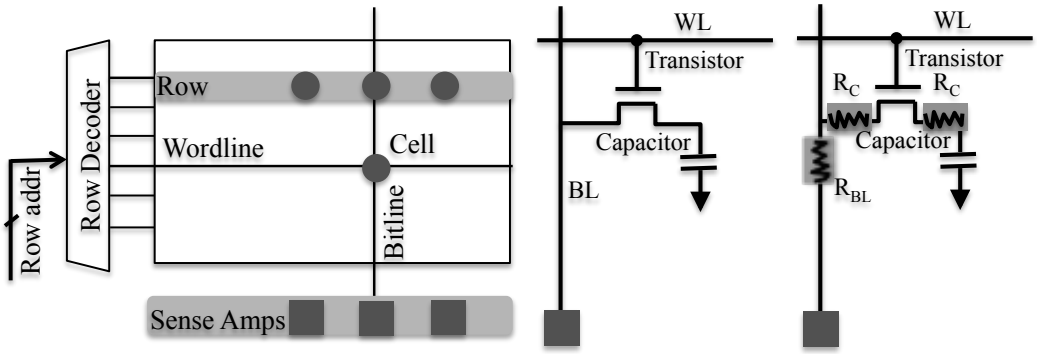
n is 8 for commodity DDR3, which translates into a granularity of 64B (64b×8), the popular cache block size.

In more detailed level, DRAM cells are packed into 2D arrays, as Figure 2.2 shows, where each cell can be uniquely located by a vertical bitline and a horizontal wordline. Each cell consists a capacitor to store electrical charge, and one access transistor to control the connection to wordline. Upon receiving a row address, DRAM fetches the target row content, through charge sharing and sensing [Lee et al., 2015; Shin et al., 2014], into the row buffer, which contains thousands of sense amplifiers to detect the voltage change on bitline.

2.2 DRAM OPERATIONS AND TIMING CONSTRAINTS

DRAM supports three types of accesses — read, write, and refresh. An on-chip memory controller is responsible to receive requests from processors and decompose them into a series of commands such as `ACT`, `RD`, `WR` and `REF`, etc. The commands are then sent to DRAM modules sequentially following the predefined timing constraints in DDRx standard [JEDEC, 2009b]. We briefly summarize the involved commands and timing constraints as follow:

READ: as illustrated in Figure 2.3(a), read access starts with an ACTIVATE (ACT)



(a) Array **(b)** Cell **(c)** Circuit

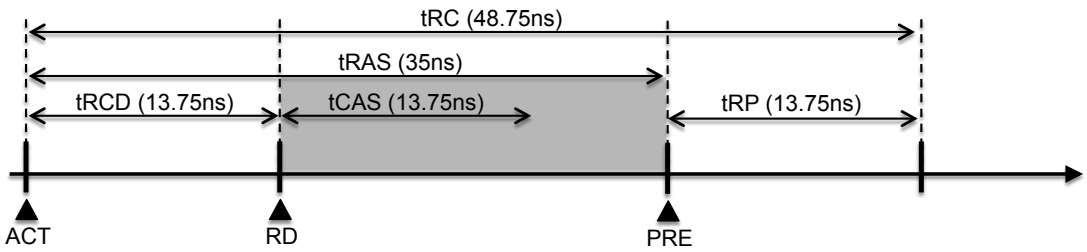
Figure 2.2: DRAM detailed organization. (a) is the high-level structure of DRAM array, (b) shows cell structure, and (c) illustrates the equivalent circuit where R_c is contact resistance and R_{BL} is the bitline resistance.

command to bring the required row into the sense amplifiers; then, a READ (RD) command is issued to fetch data from the row buffer. The interval between ACT and RD is constrained by `tRCD`. DRAM read is destructive, and hence the charge in the storage capacitors needs to be restored. The restore operation is performed concurrently with RD, and a row cannot be closed until restoring completes, which is determined by `tRAS-tRCD`. Once the row is closed, a PRECHARGE (PRE) can be issued to prepare for a new row access. PRE is constrained by timing `tRP`. The time for the whole read process is thus `tRC=tRAS+tRP`.

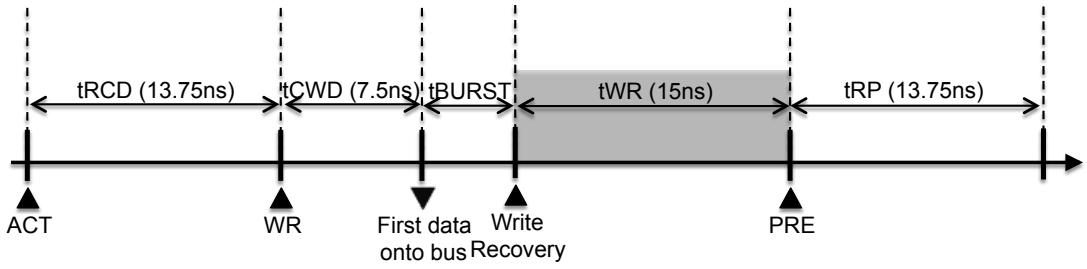
WRITE: write works similarly to read, with ACT as the first command to be performed. After `tRCD` has been elapsed, a WRITE (WR) is issued to overwrite the content in the row buffer, and then update (restore) the value back into the DRAM cells. Before issuing PRE, the new data overwritten in the sense amps must be safely restored into the target bank, taking `tWR` time. To summarize, both RD and WR commands involve the restoring operation [3] , and hence a change in restore time shall affect both DRAM read and write accesses.

Refresh: DRAM needs to be refreshed periodically to prevent data loss. According to JEDEC [JEDEC, 2009b], 8K all-bank auto-refresh (`REF`) commands are sent to all DRAM

[3]Whereas restoring after write is represented by `tWR`, that after read is included in `tRAS`. For ease of presentation, we discuss with a focus on `tWR` and always adjust `tRAS` accordingly.



(a) Read operation



(b) Write operation

Figure 2.3: Commands and timing constraints involved in DRAM accesses. (Timing values are from [JEDEC, 2009b])

devices in a rank within one retention time interval (`Tret`), also called as one refresh window (`tREFW`) [Bhati et al., 2015a; Mukundan et al., 2013], typically 64ms for DDRx. The gap between two `REF` commands is termed as refresh interval (`tREFI`), whose typical value is 7.8μs, i.e. 64ms/8K. If a DRAM device has more than 8K rows, rows are grouped into 8K **refresh bins**. One `REF` command is thus used to refresh multiple rows in a bin. An internal counter in each DRAM device tracks the designated rows to be refreshed upon receiving `REF`. The refresh operation takes `tRFC` to complete, which proportionally depends on the number of rows in the bin.

2.3 DRAM TECHNOLOGY SCALING

With continuously increasing demands on DRAM density and capacity, the cell dimensions keep scaling downward. Past decades saw DRAM's rapid development of 4x density every 3 years [Patterson and Hennessy, 2008]. Along scaling path from over 100nm to nowadays 2x nm, DRAM also experiences the drop of supply voltage [Zhang et al., 2016a], more severe signal noise [Ryan and Calhoun, 2008; Mukundan et al., 2013] and shorter retention time [Nair et al., 2013b; Wang, 2015]. However, for reliable operations in DRAM, cell capacitor must be sufficiently large to hold charge, access transistor is required to be large enough to exert effective control [Lee et al., 2009a], resistance should not be too large to obstruct cell charging process, and sub-threshold leakage should be small to safely hold data for a long time.

The intertwining requirements make the scaling jeopardy. For instance, smaller technology nodes provides smaller contacts of transistor and capacitor, and also narrower bitlines, both of which result in increased resistance (shown in Figure 2.2(c)), which lengthens the restoring time, and further the overall access latency. The growing number of slow and leaky cells has a large impact on system performance. There are three general strategies to address this challenge:

- The first choice is to keep conventional hard timing constraints for DRAM, which makes it challenging to handle slow and leaky cells. Cells that fall outside of guardbands could

be filtered (not used). With scaling, however, this approach can incur worse chip yield and higher manufacturing cost. Because the DRAM industry operates in an environment of exceedingly tight profit margins, reducing chip yield for commodity devices is unlikely to be preferred.

- A second choice is to expose weak cells, falling outside guardbands, and integrate strong yet complex error correction schemes, e.g., *ArchShield* [Nair et al., 2013b]. Due to the large number of cells that violate conventional timing constraints such as `tRCD`, `tWR`, significant space and performance overheads are expected.
- A third choice is to relax timing constraints [Kang et al., 2014; Zhang et al., 2015a]. This approach is compelling because it can easily maintain high chip yield at extreme technology sizes. However, relaxing timing, without careful management, can cause large performance penalties.

Because the third choice is compatible with the need for both high chip density and yield, we adopt it in this thesis. We relax restore timing and strive to mitigate associated performance degradation. Our design principle is also applicable to the second strategy if exposed errors can be well managed. We leave this possibility to future work.

2.4 RELATED WORK

Around the performance and power design goals, DRAMs have been optimized from varying perspectives, e.g., timing reductions, refresh scheduling. We briefly discuss the prior arts and compare against our proposed designs as follow:

2.4.1 Timing Reduction

Reducing timing constraint values can effectively improve memory performance. `TL-DRAM` [D. Lee et al., 2013] creates row segments with low ACT and PRE latencies. `CHARM` [Son et al., 2013] reduces sensing time by attaching fewer cells to each bitline. `MCR` [Choi et al., 2015] is a recent work that reduces both sensing time and restore time. While above schemes

are designed mainly for existing commodity DRAM, this thesis targets at the restoring issues in deep sub-micron DRAMs to keep high yield. Moreover, the schemes proposed in this thesis do not need to modify the DRAM internal structures.

To achieve high yield and reliability, timing constraint values are set with excessive margins, which reflect the worst-case. Most DRAM chips can perform normally with smaller timing values. Chandrasekar *et al.* [Chandrasekar et al., 2014] proposed to identify the excess in process-margins for DRAM devices at runtime. `AL-DRAM` [Lee et al., 2015] analyzes the timing reduction opportunities and exploits the large margin of DRAM timing parameters to improve performance. `NUAT` [Shin et al., 2014] exploits the electric charge variation caused by leakage to design a non-uniform access time memory controller. Differently, this thesis focuses on the prolonged restore issue in the future DRAMs, and thus it is orthogonal to `AL-DRAM` and `NUAT`, as discussed in Section 4.2.3.

2.4.2 DRAM Restore

While write recovery time (tWR) keeps at 15ns across all generations from DDR to DDR4 [JEDEC, 2000, 2009a,b, 2012], it has to be lengthened in deep sub-micron technology nodes, which was first recently discussed by Kang et al. [2014]. As the first academic work on restore issues in further scaling DRAM, our paper [Zhang et al., 2015a, 2017b] studied the variation behaviors and proposed to utilize chunk remapping to lower restoration durations. Afterwards, patents on restore were granted: Son et al. [2014] raised the idea to adjust timings with respect to temperature, and Wang [2015] claimed that tWR can be increased from 15ns to 60ns, and then raised the idea of exploring backward compatibility. The patents' findings on restore trends are well aligned with our studies.

Whereas the restore scaling issue has been identified in industrials, little academic research have been performed. Restoration has been an silent issue util recently; people started to utilize the reserved timing margins [Chandrasekar et al., 2014; Lee et al., 2015], with restoring being included. Besides, later work [Choi et al., 2015] took use of charge variation to relax some timing constraints. However, none of these work targets at future DRAM technologies. Compared to the very few existing work, this thesis performs comprehensive

studies on restore issue, and proposes effective schemes from multiple perspectives.

2.4.3 DRAM Refresh

As DRAM device capacity increases, refresh is expected to introduce larger performance and energy overheads [Liu et al., 2012a; Nair et al., 2013a; Bhati et al., 2015a]. To address the issue, a bunch of schemes have been proposed from different directions, with representative ones like `Smart Refresh` [Ghosh and Lee, 2010], `Elastic Refresh` [Stuecheli et al., 2010], and `Refresh Pausing` [Nair et al., 2013a], etc.

While weak cells require frequent refreshes, the majority of the cells on a DRAM chip can hold the data for a much longer time, which makes it viable to adopt multi-rate refreshes [Kim and Lee, 2009; Wong et al., 2008; Agrawal et al., 2014; Liu et al., 2012a; Wang et al., 2014a; Bhati et al., 2015b]. Particularly, Liu et al. [2012a] proposed `RAIDR` to group DRAM rows into several partitions and enable different refresh rates for different partitions. Wang et al. [2014a] and Bhati et al. [2015b] optimized `RAIDR` to make it compatible with modern DRAM standards. `ArchShield` was designed to tolerate high error rate in future DRAMs. It can be utilized to cover leaky cells and reduce refresh rate.

The proposed restore truncation schemes in Chapter 4 targets at the correlation of restore and refresh, and they can be integrated with the existing refresh innovations to find a better tradeoff between overall performance improvement and refresh penalty. We adopt the practical `REFLEX` [Bhati et al., 2015b] to implement multi-rate refresh. And, the experimental results show that our proposed refresh-related schemes are capable to beat all refresh reduction designs, as reported in Section 4.6.

2.4.4 Approximate Computing

Approximate computing is an emerging paradigm that exploits the inherent error resilience of many modern applications where a small number of hardware and software errors have little impact on the quality of program output [Liu et al., 2011; Sampson et al., 2011]. For these applications, user data are often categorized as either *critical* or *non-critical*. While non-critical data has error tolerance, critical data are protected to ensure correctness.

For main memory, most approximate computing approaches focus on DRAM refresh energy. Flikker [Liu et al., 2011] refreshes non-critical memory regions less frequently to save energy. Arnab et al. [2015] further divide memory pages into quality bins with different refresh and error rates, and enable quality-aware data allocation to the bins. Sparkk refreshes different chips in a DIMM with different rates to reduce refresh power [Lucas et al., 2014]. Compared to refresh operations, the degradation of restore time with scaling slows down both read and write operations, leading to more significant performance degradation. In addition, approximate computing caches have been proposed to improve effective cache capacity and achieve energy savings [Miguel et al., 2014, 2015, 2016], and optimize storage density and lifetime [Sampson et al., 2013; Guo et al., 2016].

To support approximate computing analysis, EnerJ uses type qualifiers to map approximate data to low power storage [Sampson et al., 2011]. Schmoll et al. [2013] presented a flexible software-based error handling. Khudia et al. [2015] and Mahajan et al. [2016] proposed to dynamically monitor errors and adjust computation accuracy to meet the quality demand on the final results. These schemes are orthogonal to our work so that they can be combined with our design to further improve system performance.

3.0 ACHIEVE FAST RESTORE VIA REORGANIZATION

3.1 MODELING RESTORE EFFECTS

Modeling and simulation are required to perform the studies on further scaling DRAM. On device level, the model needs to capture the critical components including transistor, capacitor, sense amplifier, and other peripheral circuits; and the model should also cover the primary parameters and dimensions, such as transistor length/width, capacitance and voltage, etc. Following the principles, we built SPICE modeling on basis of a Rambus tool [Vogelsang, 2010], and simulated data write operation.

Further, to involve process variation effects, the models should be inherently statistical following certain distributions. Using the aforementioned cell model, we generate 100K samples and curve fit using log-normal distribution. Similar to recent process variation (PV) studies [Liu et al., 2012a; Agrawal et al., 2014], we include bulk distribution to depict the normal variation that dominates the majority of cells, and tail distribution to depict random manufacturing defects [1]. Table 3.1 summarizes the parameters for bulk and tail distributions after curve fitting with our cell samples.

Table 3.1: Process Variation Modeling Parameters

tech node	μ_{bulk}	σ_{bulk}	μ_{tail}	σ_{tail}	ϕ	random weight
20nm	2.031	0.21	3.081	0.063	0.3	0.5
14nm	2.048	0.247	3.283	0.0735	0.3	0.5

[1] Note that not all cells following the tail distribution are treated as defects. The worst ones are covered by conventional redundant repairs [Agrawal et al., 2014; Jacob et al., 2007].

To obtain the chip maps, we use the VARIUS tool [Sarangi et al., 2008] to involve both within-die (WID) and die-to-die (D2D) process variations. Similar to prior PV studies [Karnik et al., 2004; Agrawal et al., 2014], we assume the same share of systematic and random components, and choose $\phi = 0.3$ meaning that the correlation range equals to 30% of the chip's side length, as shown in Table 3.1. With the constructed models and collected parameters, we can move forward to generate chips, and then form ranks and DIMMs using the pool of chips. Next, architectural explorations can be conducted on the collected memory system.

Conventionally, each timing constraint for DRAM has a single fixed value, e.g., tWR keeps at 15ns in existing DRAM standard [JEDEC, 2000, 2009a,b, 2012]. Given that more cells in deep sub-micron are likely to violate tWR, it is beneficial to relax it to allow most such cells finish restoring operations after a destructive read or write operation, which helps to preserve high chip yield. However, a larger tWR indicates longer bank occupancy and lower bank throughput.

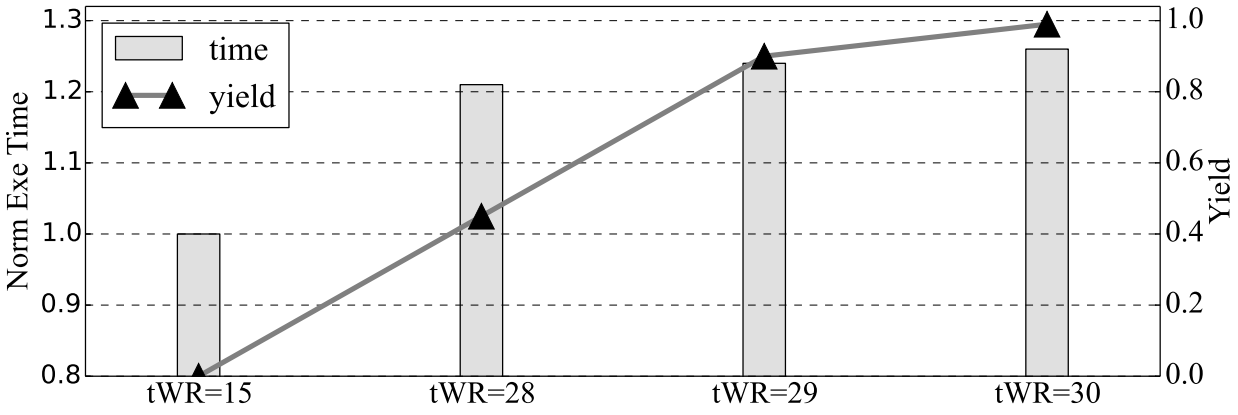


Figure 3.1: Comparing performance and yield with different tWR values (20nm).

Figure 3.1 compares the performance and yield with different relaxed tWR values at 20nm technology node. If tWR is set to 15ns, the scaling effect would lead to no chip satisfying the existing specification, i.e., yield rate is 0%. At 20nm technology node, the majority of chips have large tWR values in a tight range (28-29). To achieve 99% yield rate, tWR has to be relaxed to 30ns, which prolongs the execution by over 25%; A smaller degradation, e.g., 21%, can be observed when tWR is relaxed to 28ns. However, the yield is

seriously lowered to 45%. From the figure, we can see that it is challenging to achieve high chip yield while minimizing its impact on system performance.

3.2 PROPOSED DESIGNS

In this section, we elaborate the proposed designs. First, we discuss the post-fabrication schemes in terms of both coarse chip-level and fine chunk-level restoring managements; then, we extend the schemes to assembly phase to deliberately form ranks by clustering compatible chips; and finally, the schemes are integrated with OS-level page allocation to maximum performance gains.

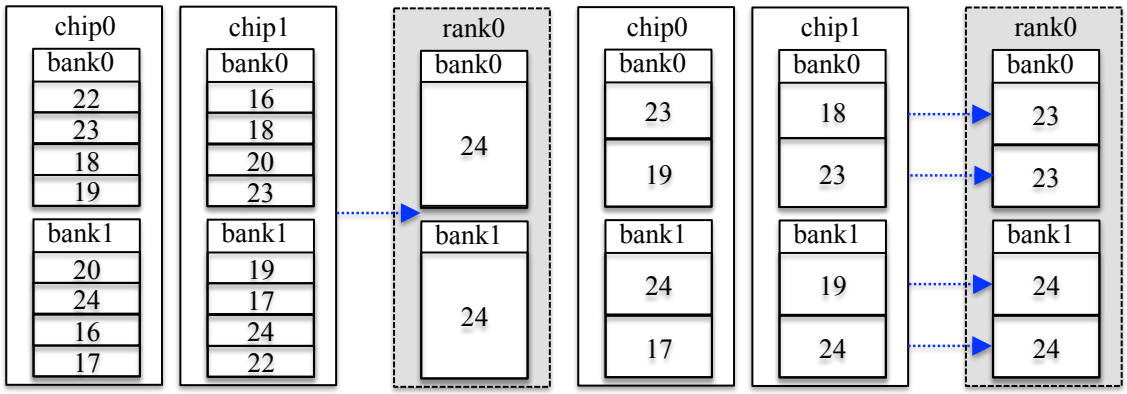
3.2.1 Chip-specific Restoring Control

We start with a simple enhancement to the current DRAM standard of adopting one tWR [2] — by exposing chip variations, we may set different tWRs for different chips. For this purpose, a post-fabrication test process is performed by the manufacturer to determine the tWR of each chip while a DIMM is then constructed using chips with the same or similar tWRs. Each DIMM derives its tWR from the chip-row [3] that has the worst tWR of the entire DIMM (as shown in Figure 3.2(a)), or the worst one after adopting a small number of spares to rescue those slowest chip-rows [Jacob et al., 2007].

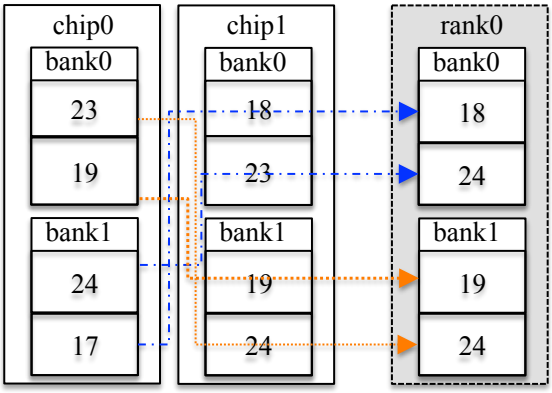
The chip-specific tWR design helps to improve chip yield rate as otherwise a chip with tWR=24ns would be discarded if tWR is set as 23ns or less in the standard. While technically all fabricated chips can now be treated as good ones, those with very large tWR (e.g., twice as large as the expected tWR) should still be marked as failed chips as DIMMs constructed from them tend to have very low performance.

[2] For discussion purpose, we focus on tWR relaxation while tRAS is relaxed accordingly in each design

[3] A chip-row refers to the portion of cells of a row that reside on one chip. A DIMM-row refers to all the cells of a row.



(a) Chip-specific **(b)** Chunk-specific



(c) Chunk-specific w/ remap

Figure 3.2: Comparison of different schemes: (a) The chip-specific tWR; (b) The chunk-specific tWR; (c) The chunk-specific tWR with chunk remapping. For illustration purpose, each rank consists of two chips while each chip contains two four-row banks. One **DIMM-row** (i.e., the row exposed to the OS) consists of two **chip-row** segments — the number in each chip-row indicates its corresponding tWR, i.e., the tWR of the weakest cell.

3.2.2 Chunk-specific Restoring Control

Even though tWR exhibits a wide range of variations when scaling in deep sub-micron regime, only a small number of cells need long recovery time. Setting a DIMM's tWR based on the chip-row that has the worst tWR is still too pessimistic. We therefore propose to partition each memory bank into a number of smaller chunks and set the chunk level tWR based on the worst chip-row within the chunk. The chunk level tWR is then exposed to the memory controller to aid scheduling.

In Figure 3.2(b), one chunk consists of two rows. Since the first chunk has 23ns and 18ns tWRs for its two chip-rows, its chunk tWR is set to 23ns. By take advantage of these fast chunks, a chunk-tWR-aware memory controller can speed up memory accesses that fall into the fast chunks [4].

3.2.3 Chunk-specific with Remapping

The previous design can only form a DIMM-chunk from the same-index chip-chunks, which can be optimized to further reduce tWR values. This is because the chip-chunks that are of the same index may exhibit significant tWR difference. It would be beneficial to form a chunk using chip-chunks that are of the same or similar tWRs.

For the example in Figure 3.2(c), if we form the first DIMM-chunk using the 4th chip-chunk from chip 0 and the 1st chip-chunk from chip 1, the tWR of this chunk can be as low as 18ns. Constructing a number of such fast chunks helps to speed up the average row access time of the given DIMM.

The chunk remapping is done in two steps: (1) after detecting the tWR for each chip-chunk, we compute the averaged tWR for each chip-bank, and sort chip-banks independently on each chip. A **DIMM-bank** consists of chip-banks that are of the same index on the sorted list; (2) For chip-chunks within each chip-bank, we sort them again such that each **DIMM-chunk** consists of chip-chunks that are of the same index on the sorted list.

While only one access is allowed to access one bank at any time, the multiple banks

[4]For discussion purpose, a *chip-chunk* is referred to as one chunk within one chip; a *DIMM-chunk* is referred to as the set of same-index chip-chunks from different chips of the DIMM. For example, the 2nd DIMM-chunk consists of the 2nd chip-chunk from each chip.

in a DIMM can be accessed simultaneously. To maintain the same bank level parallelism, we treat the chip-chunks from one bank as a group in chunk remapping. In Figure 3.2(c), DIMM-chunk 0 and 1 belong the DIMM-bank 0. Since DIMM-chunk 0 is constructed using chip-chunk 3 on chip 0, DIMM-chunk 1 needs to use chunks from the same group, i.e., chip-chunk 2 on chip 0. In this way, simultaneously accessing two different DIMM-banks will never compete for the same chip-bank on any chip.

3.2.4 Restore Time aware Rank Construction

A DIMM rank is composed of multiple chips, which work in lockstep fashion. The access speed of one logical row is determined by its worst chip-row. While chunk-remapping does not have to form a DIMM-row using the chip rows that of the same physical index, it may still be ineffective when one of the chips that form a rank contains many slow rows. A bad chip would lead to a slow rank no matter how the chunks are remapped.

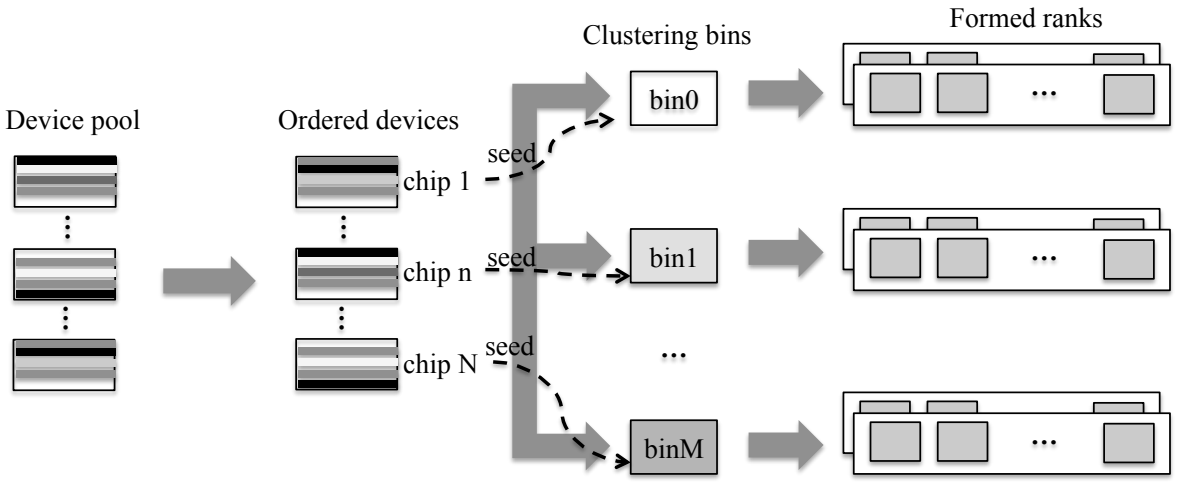


Figure 3.3: Rank construction consists of three steps — (1) chip sorting and seed chip selection; (2) distributing chips to bins; (3) constructing DRAM ranks using chips from each bin.

We further propose to construct DRAM ranks using compatible chips, rather than random chip selection in the baseline design. Given N DRAM chips, our goal is to construct a better rank set (and each rank contains R chips). The rows in each chip are divided into K chunks and we use M bins to assist rank construction.

We first compute the average chip level tWR, which uses the chunk level tWR values of each chip. The latter can be collected during post-fabrication testing. We sort the chips based on their average tWR values, and choose M seed chips, i.e., the chips on the sorted chip list whose indice can be divided by $\lfloor N/M \rfloor$. The seed chips are distributed to M bins.

We then place the rest of chips into M bins based on their similarity to the seed chip of each bin. The chunk level tWR values of each chip are treated as a K-item vector. The similarity of two chips is calculated using the Hamming distance of the two K-item vectors. The candidate chip is placed in the bin whose seed chip has the smallest Hamming distance, i.e., the highest similarity, to the candidate chip.

Once a bin reaches its size limit, i.e., $n \times R$ where $n = \lfloor N/M/R \rfloor$, and $n \times R \leq N/M$, it can no longer accept new chips. In the algorithm, an extra bin Bin_{M+1} is used to hold the leftover chips. When filling chips to each bin, we construct a rank if a bin has R chips (the seed chip is used to form a rank in the last batch).

Since the algorithm, as depicted in Alg 1, needs to scan each chip and compute its similarity with all seed chips, the time complexity is $O(N \times M \times K)$. Here M and K are constant. M is usually small ($M << N$) while K can be relatively large, e.g., K=1024. Therefore, the time complexity is linear to the number of candidate chips. This is a light weight rank construction scheme, compared that in [Wang et al., 2015]. Our experiments show that the two algorithms achieve similar rank level tWR results.

Table 3.2 compares the average tWR values when using different rank construction algorithms. We constructed ten ranks and the rows in each bank are divided into 4K chunks. We list the rank tWR values and the average tWR values at 20nm technology node, and the average tWR values at 14nm technology node. From the table, we find that, without chunk remapping, the slow rows can significantly affect the rank level tWR values. For example, for `Baseline` at 20nm node, the average tWR of all ranks is 28.25ns while the average tWR of rank-1 is 30.1ns. Performing chunk remapping after bin-based rank construction is the most effective scheme. In the experiments, we set the number of bins to 5. Varying the bin counts from 2 to 10 shows similar results. In addition, we compare the results with the scheme that perform heavy weight rank construction [Wang et al., 2015], and observe similar average tWR values of all ranks.

Algorithm 1: Bin-based Rank Construction

Input: Chunk level tWR values of all candidate chips

Arguments : N-total number of candidate chips; K-chunks in each chip; M-total clustering bins; R-chips in a rank;

Output: N/R formed ranks

1. Preprocessing: calculate the average tWR of each chip, and order the chips;
2. Seed pick-up: assign one seed chip per bin by selecting one every N/M chips;
3. Scan the remaining non-seed chips, and put each into a bin;

$chipID = 0$; bin size = $\lfloor N/M/R \rfloor \times R$;
create one extra bin, Bin_{M+1};
repeat
  fetch the $Chip_{chipID}$ from the pool;
  min_dist = IMAX; min_id = -1;
  for $id = 0$; $id < M$; $id{+}{+}$; **do**
    if *(Bin_{id} is full)* **then**
      continue;
    end
    $dist$ = Hamming_distance(seed chip of Bin_{id}, $Chip_{chipID}$);
    if *($dist < min\_dist$)* **then**
      $min\_dist = dist$; $min\_id = id$;
    end
  end
  if *($min\_id\ != -1$)* **then**
    put the chip into Bin_{min_id};
  else
    put the chip into Bin_{M+1};
  end
  $chipID$ ++;
until $chipID >= N - M$;

4. Sequentially scan the classified chips of each bin, and form R-chip ranks;

Table 3.2: Comparison of average tWRs for 20nm and 14nm nodes (10 ranks were constructed and each chip has 4096 chunks)

rankID	Baseline[1](ns)	Chunk[2](ns)	ChunkBin[3](ns)	ChunkSortBin[4](ns)
0	25.9	22.4	25.9	27.3
1	30.1	27.1	30.7	27.3
2	25.9	22.2	25.9	22.4
3	30.7	27.2	25.9	22.2
4	28.8	27	31.8	22.2
5	28.8	27	25.9	22.2
6	28.8	27.1	25.9	22.2
7	28.8	27.1	25.9	22.3
8	25.9	22.3	25.9	22.2
9	28.8	27.0	25.9	22.2
avg(20nm)	28.25	25.64	26.97	23.25
avg(14nm)	35.04	31.71	33.33	28.73

[1] `Baseline` is the baseline that constructs ranks using random grouping.

[2] `Chunk` is the scheme that conducts chunk remapping on basis of `Baseline`.

[3] `ChunkBin` is the scheme that adopts our proposed restore time aware rank construction. Chunks within each chip are not remapped.

[4] `ChunkSortBin` is the scheme that remaps chunks after the rank construction.

3.2.5 Restore Time aware Page Allocation

The translation of virtual to physical address is supported in hardware by Memory Management Unit (MMU), and the virtual-physical mapping is determined by operating system (OS). Traditional page allocation is restore time oblivious as all physical pages have the same access latency. However, when a set of fast DRAM chunks are constructed and exposed to the memory controller, it is beneficial to exploit the access latency difference to speed up program execution.

Clearly, the memory system can be more effective if fast chunks are assigned to service performance-critical pages. In this thesis, the page criticality is estimated using its access frequency [Son et al., 2013; Lee et al., 2001]. Studies have shown that it is usually a small subset of pages, referred to as hot pages, that are frequently accessed in modern applications [Bhattacharjee and Martonosi, 2009; Ramos et al., 2011; Ayoub et al., 2013]. We adopt the offline profiling approach as in [Son et al., 2013] to identify hot pages.

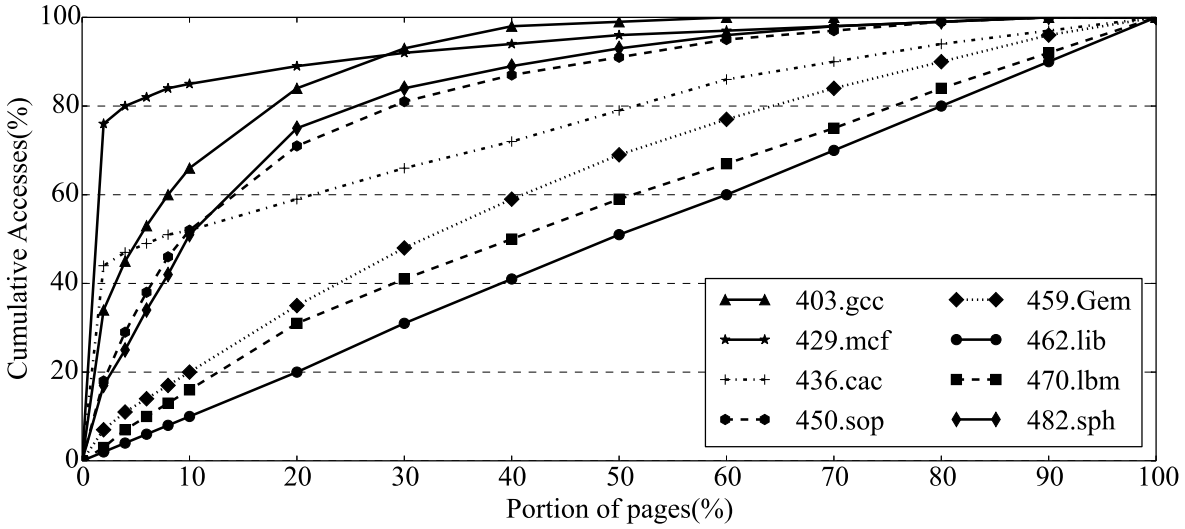


Figure 3.4: The page access distributions in SPEC CPU2006.

Figure 3.4 studies the page access distribution of a set of SPEC CPU2006 applications [SPEC, 2006]. The figure shows that different applications have very different access behaviors: for some workloads, e.g., 459.*Gem* and 470.*lbm*, accesses are evenly distributed such that the number of accumulative requests grows linearly with the number of touched pages;

for some other applications, e.g., 429.*mcf* and 403.*gcc*, most memory accesses come from a small subset of hot pages. The hot pages are the ones to be allocated in fast DRAM chunks.

The benefit of restore time aware page allocation also depends on the number of hot pages, i.e., if the hot page set can all be allocated in the fast chunks. Figure 3.5 compares the number of touched pages of different benchmarks. From the figure, the majority touch less than 1/8 of the total memory space, while some benchmarks (i.e., 459.*lbm* and 429.*mcf*) use up all available space.

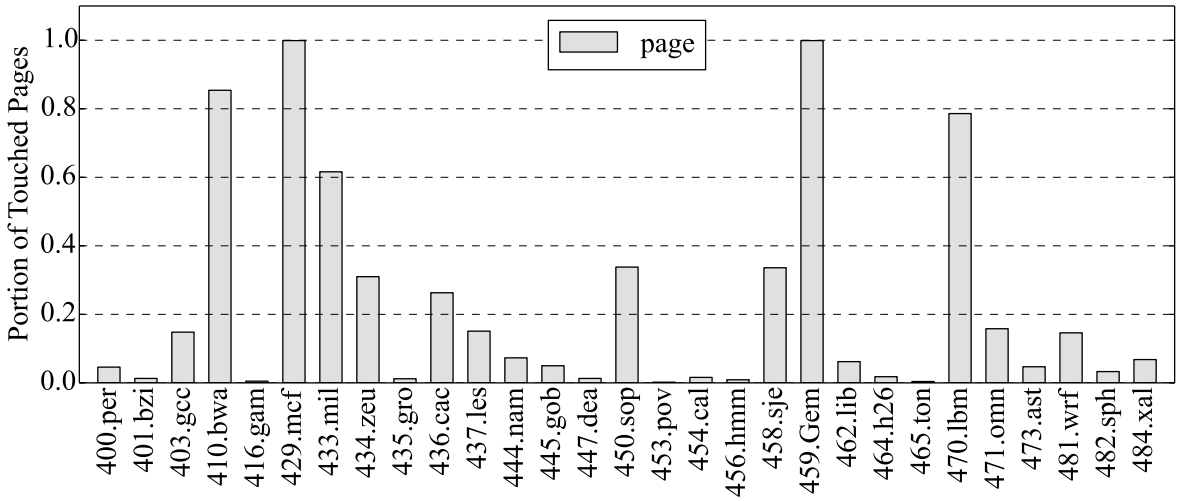


Figure 3.5: The portion of touched pages for each benchmark. The memory capacity is 2GB, which can be divided into 52K 4KB-page.

In this article, our goal is to illustrate that a restore time aware page allocator can take advantage of the latency difference of the DRAM chunks. For this purpose, we adopt a simple strategy that profiles program execution offline and statically allocates hot pages to fast chunks. In the case if profiles are not accurate, we may need to design and enable more flexible strategies, e.g., such as the detailed behavioral synthesis [J. Cong and Zou, 2011] and data migration and compression [Ozturk and Kandemir, 2008]. We leave this as our future work.

3.3 ARCHITECTURAL ENHANCEMENTS

In order to exploit restore time variations at either chip or chunk levels, a post-fabrication testing needs to be performed to detect restore time at fine-granularity. Given that cell restore time is thermal dependent — study showed that it becomes worse at low temperature [Kang et al., 2014], the manufacturer needs to examine different types of data patterns [Venkatesan et al., 2006; Liu et al., 2012a, 2013]. to record the worse timing constraints under chip's allowed working conditions. The values are organized as a table (with each entry in the table recording affected timing constraints tWR/tRAS of its corresponding DIMM chunk) and saved in non-volatile storage in the DIMM [Seshadri et al., 2013].

The memory controller loads this table at boot time and schedules memory accesses accordingly to maximize bandwidth and avoid conflicts. As an example, two READ operations cannot be scheduled back to back to a DIMM bank if the later one accesses a fast chunk and shall compete with the preceding READ for using the data bus.

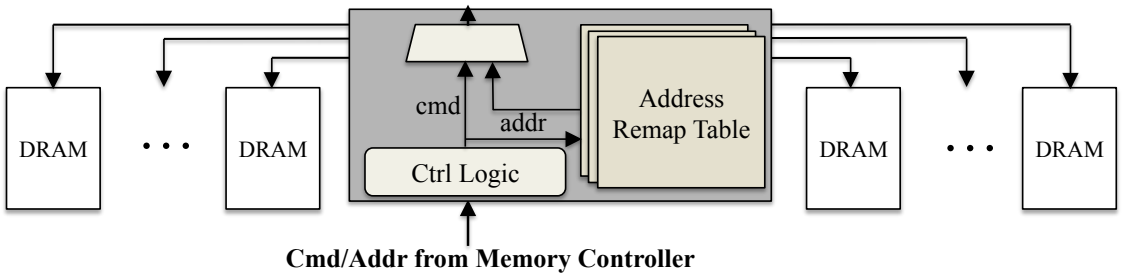


Figure 3.6: The on-DIMM architectural enhancement.

To enable chunk re-organization, we need one extra chunk remapping table as shown in Figure 3.6. Similar as HP's MC-DIMM [Ahn et al., 2009] and Mini-rank [Zheng et al., 2008], our design integrates an enhanced registering clock driver (RCD) [JEDEC, 2009c] to remap and redrive the physical address. Differing from rank-subsetting designs where only partial chips are involved for each memory access, our proposed design follows the chips' original lockstep working fashion, but only requires each chip to receive a dedicated row address. Compared to conventional DIMM RCD, additional register, together with some pins, and

DIMM PCB traces [Micron, 2008] are needed to implement address remapping. For the chunk remapping table, each entry maps the corresponding DIMM-chunk to the chip-chunk at each chip. Given the following table, when the bridge chip receives a request asking for data in the 10th DIMM-chunk, it translates the requests to asking for segment data from the 1220th chunk from chip 0, the 124th chunk from chip 1, etc.

Table 3.3: Remap table

DIMM_chunk	chip0_chunk	chip1_chunk	...	chip7_chunk
...	...	...	...	...
10	1220	124	...	256
...	...			

3.4 EXPERIMENTAL METHODOLOGY

3.4.1 Configuration

To evaluate the effectiveness of our designs, we compared them to traditional repair solutions [Jacob et al., 2007] using an in-house chip-multiprocessor system simulator. We modeled a quad-core system with the parameters shown in Table 3.4. For both 20nm and 14nm technology nodes, we used VARIUS to generate 90 chips, and then form ranks in different fashion as discussed in Table 3.2. The memory system to be simulated is composed of two ranks. We constructed five rank pairs and tested the proposed designs with all pairs.

The DRAM timing constraints follow Hynix DDR3 SDRAM data sheet [Hynix, 2010] and are summarized in Table 3.5. For the schemes exploiting chunk level timing constraints, we added two CPU cycles to access the timing table. For the schemes performing chunk-remapping, we added one extra DRAM cycle to access the mapping table.

Table 3.4: System Configuration

Processor	four 3.2Ghz cores; four-issue; 128 ROB size
Cache	L1(private): 64KB, 4-way, 3 cycles L2(shared): 2MB, 6-way, 32 cycles 64B cacheline
Memory Controller	Bus frequency: 1066 MHz 128-entry queue; close page
DRAM	1channel, 2ranks/channel, 8banks/rank, 16K rows/bank, 8KB/row, 1066 MHz, tCK=0.935ns, width: x8

Table 3.5: Memory Timings

Timing Parameters	Time(ns)	DRAM Cycles(CLK)
CL	13.09	14
tRCD	13.09	14
tRC	46.09	50
tRAS	33.0	36
tRP	13.09	14

3.4.2 Workloads

We used SPEC CPU2006 [SPEC, 2006] and simulated 1 billion instructions after skipping the warm-up phase of each benchmark. The Read and Write MPKI (memory accesses per kilo instructions) for each workload is profiled to indicate the memory intensiveness. Based on MPKI, the applications are classified into three categories (Spec-High/Spec-Med/Spec-Low), as shown in Table 3.6. The workloads are running in rate mode, where all cores execute the same task.

We performed timing simulation until all cores finish the execution, and averaged the execution time of all the four cores. We constructed five rank pairs, i.e., DIMMs. One simulation run used one DIMM while the reported results are the average of the runs using different DIMMs.

3.5 RESULTS

We evaluated the following schemes:

— `Baseline`. The baseline sets tWR to 15ns, the same as existing DRAM specification. It is the ideal baseline due to scaling. The results of other schemes are normalized to the baseline for comparison.

— `Relax-`x. Given that scaling in deep sub-micron regime leads to worse timing, this scheme relaxes time constraints to achieve x% yield. We relaxed tWR and set tRAS/tRC accordingly. We tested x=85 and x=100, respectively.

— `Spare-`x. One commonly adopted post-fabrication repair approach is to integrate sparing rows/columns to mitigate performance and yield loss. It is implemented by using a laser programmable link to disconnect the abnormal rows/columns and connect the spare ones [Jacob et al., 2007]. In our experiments, we set the spare density as high as 16 spare rows per 512-row block, which resides in the aggressive spectrum [Kirihata et al., 1996; Koren and Krishna, 2010]. Given that spares may be reserved for high-priority repairs, such as defects and retention failures, we testsed x=0, 2, 8, 16 spares out of each 512-row block,

Table 3.6: Benchmark Characteristics

Class	Workload	Read MPKI	Write MPKI
Spec-High	429.mcf	58.07	4.56
	470.lbm	31.31	23.38
	450.sop	26.77	2.69
	433.mil	24.85	8.38
	471.omn	19.26	0.04
	459.Gem	19.04	2.66
	462.lib	17.48	0.52
	484.xal	16.89	0.45
	403.gcc	14.69	0.52
	482.sph	14.14	1.11
	410.bwa	10.04	1.67
Spec -Med	437.les	8.70	2.48
	481.wrf	5.18	1.40
	436.cac	5.13	1.52
	434.zeu	4.69	1.22
	401.bzi	3.92	1.72
	473.ast	3.53	0.96
	447.dea	3.15	0.20
	456.hmm	3.11	2.88
Spec -Low	400.per	1.73	0.22
	464.h26	1.47	0.66
	445.gob	1.27	0.92
	435.gro	1.06	0.28
	458.sje	0.90	0.45
	454.cal	0.66	0.35
	444.nam	0.65	0.62
	465.ton	0.30	0.03
	416.gam	0.18	0.04
	453.pov	0.01	0

respectively.

— `ECC`. ECC is implemented by placing one extra ECC chip to correct errors in data chips. Though ECC is conventionally used to correct soft errors, it can be potentially used to tolerate weak cells. Exploiting ECC chips to rescue slow rows sacrifices soft error resilience and hurts reliability [Su et al., 2005].

— `Chunk-`x. This scheme implements the chunk-specific restore time control, with each bank being divided into x chunks. Each DIMM chunk has its own timing constraints, which are exposed to the variation-aware memory controller.

— `ChunkSort-`x. This scheme implements the chunk-specific restore time control with chunk remapping, with each bank being divided into x chunks. Similar as `Chunk-`x, the timing constraints of each chunk are exposed to the memory controller.

— `ChunkBin-`x. This schemes is similar as `Chunk-`x. The difference is that it constructs ranks using the proposed bin-based matching scheme.

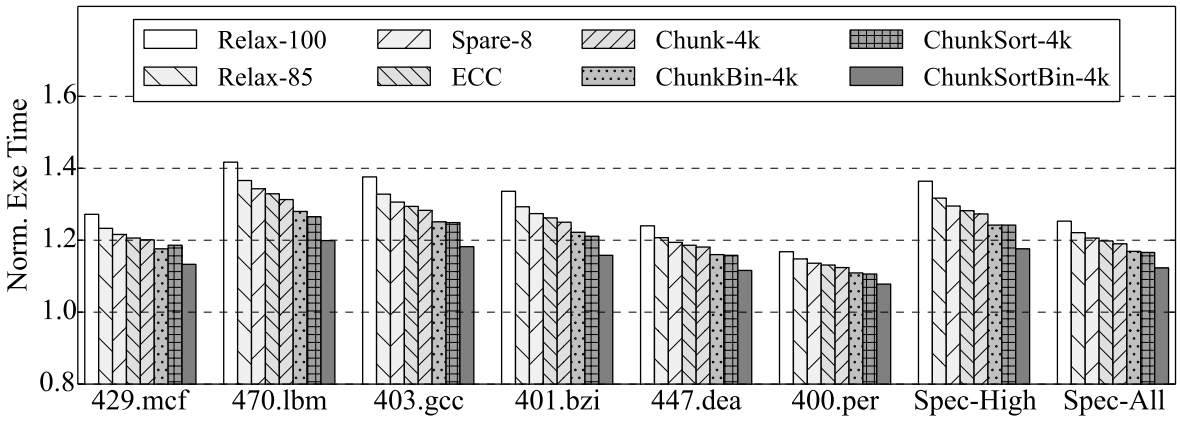
— `ChunkSortBin-`x. This schemes is similar as `ChunkSort-`x. The difference is that it constructs ranks using the proposed bin-based matching scheme.

We compared these schemes on memory access latency and system performance, and studied their sensitivity to different system configurations.

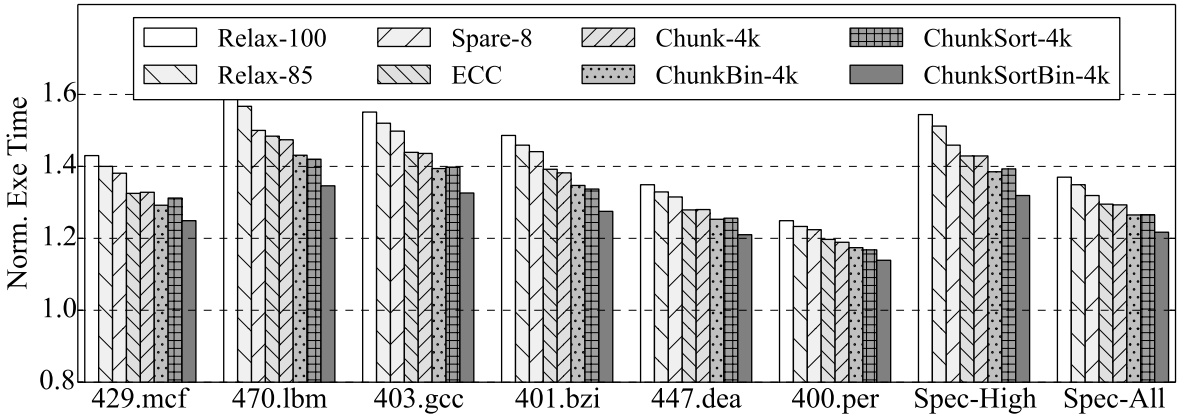
3.5.1 Impacts on Program Execution Time

Figure 3.7 compares the execution time with random page allocation policy under different schemes for 20nm (Figure 3.7(a)) and 14nm (Figure 3.7(b)) technology nodes. The execution time is normalized to the ideal baseline, i.e., tWR is 15ns. The figure reports the results from representative benchmarks of all classified categories (e.g., `Spec-High` is the set of highly memory-intensive subset), and the full set (`Spec-All`).

3.5.1.1 20nm Technology Node From Figure 3.7(a), we observed that (1) DRAM scaling has a large impact on restore time. To maintain a high yield rate, the timing constraints have to be vastly relaxed from 16 cycles to over 30 cycles, which significantly hurts performance. On average, `Relax-100` and `Relax-85` prolong the execution time by 25.3%



(a) 20nm normalized average execution time



(b) 14nm normalized average execution time

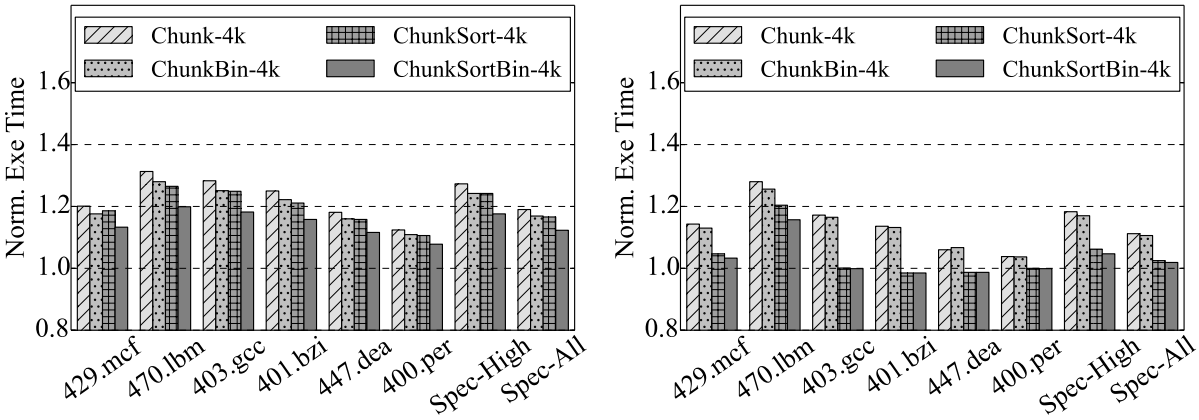
Figure 3.7: The execution time comparison of different schemes for 20nm and 14nm technology nodes under random page allocation policy. Representative applications and the geometric means for highly memory-intensive (`Spec-High`) and all applications (`Spec-All`) are presented here.

and 22.1%, respectively. Highly memory-intensive applications tend to have large degradation (i.e., over 30%). (2) Adding spare rows helps to mitigate performance losses: `Spare-8` is 20.6% worse than the ideal. (3) `ECC` works only slightly better than `Spare-8`. This is because SEC-DED ECC can only correct one bit in each 64-bit block. Since there might be multiple cells violating timing constraints within such a 64-bit block, ECC lacks the ability to effectively adapt restore time variations. (4) `Chunk-4k` is 1% better than `ECC` as it exposes chunk-level restore time variations. There are a small number of chunks that have smaller tWRs than the single tWR in `ECC`. Due to random page allocation policy, the exposed fast chunks cannot be fully exploited, and thus the performance improvement is pretty limited. (5) `ChunkSort-4k` works better than `Chunk-4k` because it helps to construct more fast chunks. On average, `ChunkSort-4k` helps to reduce the performance loss from 25.3% in `Relax-100` to 16.6%, and 3% better than `Chunk-4k` for `Spec-High`.

In addition, restore time aware rank construction helps to reduce tWR — `ChunkBin-4k` is 2% better than `Chunk-4k` while `ChunkSortBin-4k` is 4.3% better than `ChunkSort-4k`. Interestingly, `ChunkBin-4k` and `ChunkSort-4k` achieve comparable performance improvements over the baseline. While both schemes require post-chip-fabrication testing to extract chunk level tWR values, the former needs rank clustering, which imposes extra step and cost during fabrication; the latter needs to embed mapping table and thus introduces extra runtime overhead. `ChunkSortBin-4k` achieves the best performance while it incurs both extra fabrication cost and runtime overhead.

3.5.1.2 14nm Technology Node Figure 3.7(b) shows the normalized execution time for 14nm Technology node. Comparing with Figure 3.7(a), the performance difference from the ideal increases as the technology node scales down. For example, `Relax-100` exhibits a 25% loss at 20nm node while a 37% loss at 14nm node. In general, highly memory intensive applications, e.g., 403.*gcc*, show large losses. Due to the fact that more memory cells violating timing constraints, it becomes increasingly difficult to mitigate performance loss at small technology nodes.

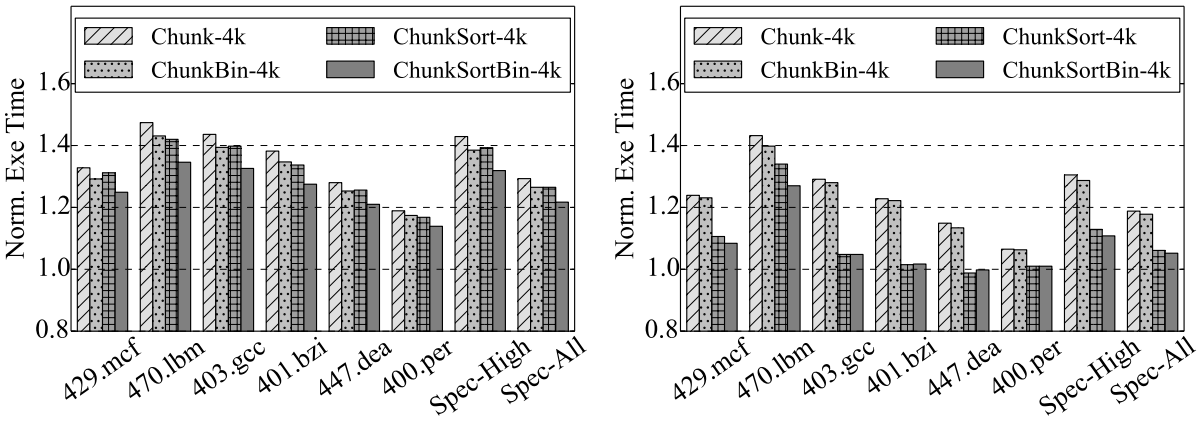
With more slow cells at 14nm, both ECC and Sparing become less effective and their impacts on performance become smaller. `ChunkSort-4k` shows a tendency of better im-



(a) With random page allocation

(b) With restore time aware page allocation

Figure 3.8: The execution time comparison of different schemes at 20nm technology node.



(a) With random page allocation

(b) With restore time aware page allocation

Figure 3.9: The execution time comparison of different schemes at 14nm technology node.

provement over traditional solutions and `Chunk-4k`. For Spec-High, `ChunkSort-4k` executes 7% faster comparing to `Spare-8` and `ECC`. `ChunkSortBin-4k` achieves over 20% performance improvements over `Spare-8` and `ECC`, and 27.5% over `Relax-100`.

3.5.2 Restore Time Aware Page Allocation

Figure 3.8 and 3.9 compare the results using random and restore-time-aware page allocation schemes at 20nm and 14nm technology nodes, respectively. From the figure, by making better use of fast chunks, restore time aware page allocation speeds up the execution of all chunk based schemes, e.g., for `ChunkSortBin-4k`, restore time aware allocation achieves 10% and 15% improvement over random allocation for 20nm and 14nm nodes respectively. Restore time aware allocation is very effective for most benchmark programs — on average, `ChunkSortBin-4k` is only 2% worse than the ideal `Baseline`.

In the figure, 470.*lbm* achieves small improvement because it evenly accesses a large number of memory pages and lacks very hot pages. Given that a small number of chunks have shorter than 15ns tWR values, it is not surprising to find that some benchmark programs, e.g., 403.*gcc* and 400.*per*, have their hot pages fit in these fast chunks and thus outperform `Baseline`.

Also in the figure, we observed that the effectiveness of restore time aware rank construction is diminishing after adopting restore time aware allocation. For example, on average, `ChunkSort-4k` and `ChunkSortBin-4k` have less than 1% difference when using restore time aware allocation at 20nm node. Nevertheless, those benchmarks with large footprint and relatively uniform access pattern, e.g., 470.*lbm*, can still achieve distinct benefits.

3.5.3 Impacts on Memory Access Latency

Figure 3.10 compares the average memory access latencies under different schemes. Among these schemes, `ChunkSortBin-4K` achieves the lowest latency, which is 7% lower than `ChunkBin-4K`, about 12% lower than `Spare-8` and `ECC`, and 17% lower than `Relax-100` for 20nm technology node with random page allocation. There is a clear latency increase for 14nm technology node, e.g., the average memory access latency of `ChunkSortBin-4K` increases from 280ns to

434ns. This is mainly due to further relaxed timing constraints. In addition, restore time aware allocation offers great help to lower the latency for chunk-based schemes.

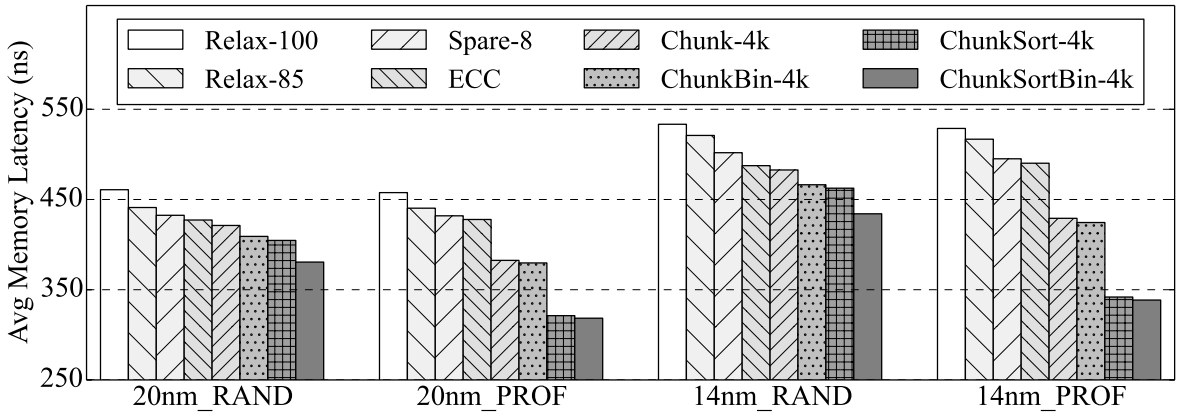


Figure 3.10: Comparing memory latencies under different schemes (values are averaged over all SPEC benchmarks).

3.5.4 Sensitivity Studies

The effectiveness of conventional Sparing technique strongly depends the sparing levels being used [Jacob et al., 2007]; the proposed chunk-based schemes depends on the chunk granularity. We conducted the sensitivity studies on these two key parameters.

3.5.4.1 Varying Variation Correlation As discussed in Section 3.1, restore variation is the combination of systematic and random components: systematic part is characterized by the spatial correlation, which is depicted by ϕ; random variation is reflected by the weight of sigmas, i.e., $w = \sigma_{rand}/\sigma_{sys}$. To study the correlation effects, we sweep over different combinations of σ-w, as reported in Figure 3.11. As ϕ decreases and w increases, cells' restoring becomes more randomly distributed; to the opposite, combination $1.0-0$ gives the extreme case that all cells in the chip are correlated and the distribution is solely systematic.

Figure 3.11 shows that our proposed scheme is always efficient because of the exposed cell variation, which agrees with the existing observation that manufacturing defects always provide some weak cells [Agrawal et al., 2014]. In addition, the achieved results are even

better under some extreme cases, e.g., 0.0-10 and 1.0-0. The reason is that both provide more fast cells and thus larger remap opportunities to expose fast regions to improve performance.

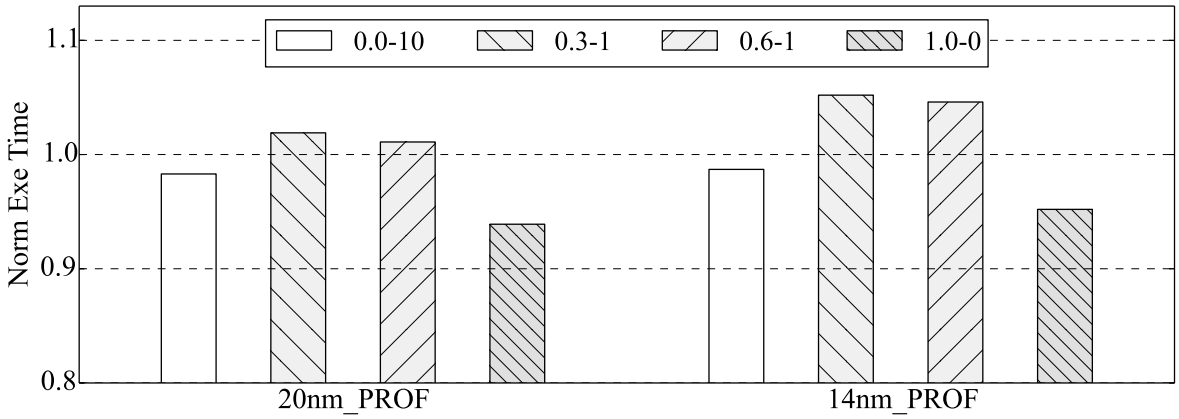


Figure 3.11: Sensitivity study of scheme *ChunkSortBin-4k* for both 20nm and 14nm tech nodes, under different cases (ϕ-w), in terms of spatial correlation parameter (ϕ) and ratio of systematic and random distributions (w).

3.5.4.2 Varying Sparing Levels Figure 3.12(a) compares the performance with different spare rows [5] in each 512-row block. The X axis shows the results for `Spec-High` and `Spec-All` benchmark sets at 20nm and 14nm nodes. While better performance can be achieved with more spares, the improvement gradually diminishes. For example, the difference between `Spare-8` and `Spare-16` is only 0.4% at 14nm node. The reason is that after rescuing a very small number of super slow rows, we expect to encounter a relatively large number of slow rows (due to scaling effect), which is beyond the ability that a sparing approach has.

3.5.4.3 Varying the Number of Chunks Figure 3.12(b) compares the performance with different numbers of chunks for chunk-specific restore time scheduling. The X axis `Anm_B` indicates the results for the schemes using `B` chunks at `A`-nm technology node. The results

[5]Note that `Spare-0` is different from `Relax-100` — while `Spare-0` places no spares, and uses the chip-specific tWR, `Relax-100` sets tWR to ensure all chips can work. As a result, `Spare-0` usually has a lower tWR.

are normalized to the ideal `Baseline`. From the figure, all schemes show better performance when using more chunks. For example, ranging from 1K to 16K chunks, `ChunkSortBin-16K` reduces the performance losses from 15% to 2% at 14nm node. `ChunkSortBin-16K` achieves better than `Baseline` performance at 20nm node because exposing tWR at fine-granularity leads to more faster chunks, which reduces the average memory access latencies.

We observed that the performance gap between `Chunk` and `ChunkBin` reduces significantly when there are more than 4K chunks. This is because restore time aware allocation makes better use of fast chunks, exposing tWR at 8K granularity is often sufficient to identify the set of fast chunks to service the hot page set of the tested benchmark programs. The gap between `ChunkSort` and `ChunkSortBin` reduces similarly.

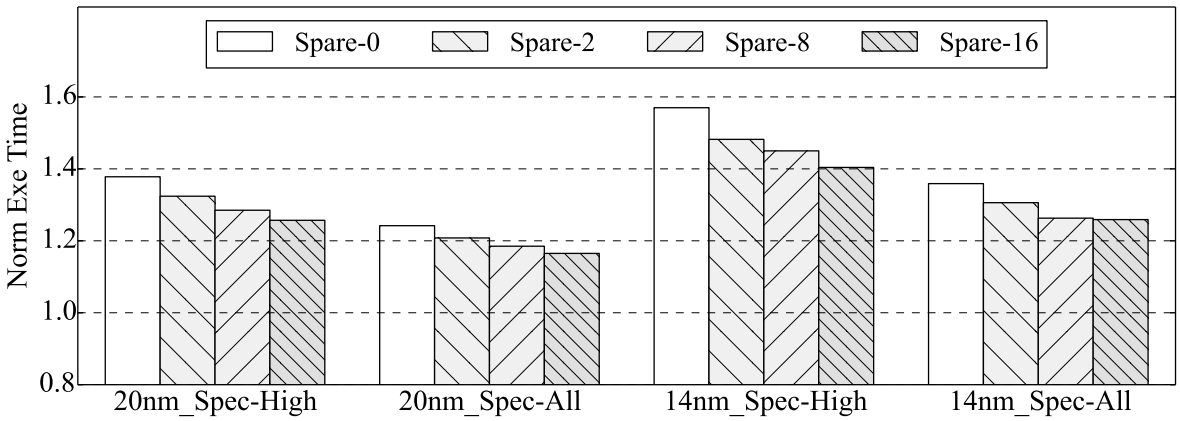
The performance gap enlarges between `Chunk` and `ChunkSort` with more chunks. This is because remapping finer granularity chunks helps to form a number of faster chunks, which brings down the average memory access latency when they are fully exploited through restore time aware allocation. From the figure, we also found that `ChunkSort-4K` is better than `Chunk-8K` while `ChunkSortBin-4K` is better than `ChunkSort-8K`, showing that chunk remapping is very effective in reducing average memory access latency.

3.5.5 Testing Overhead

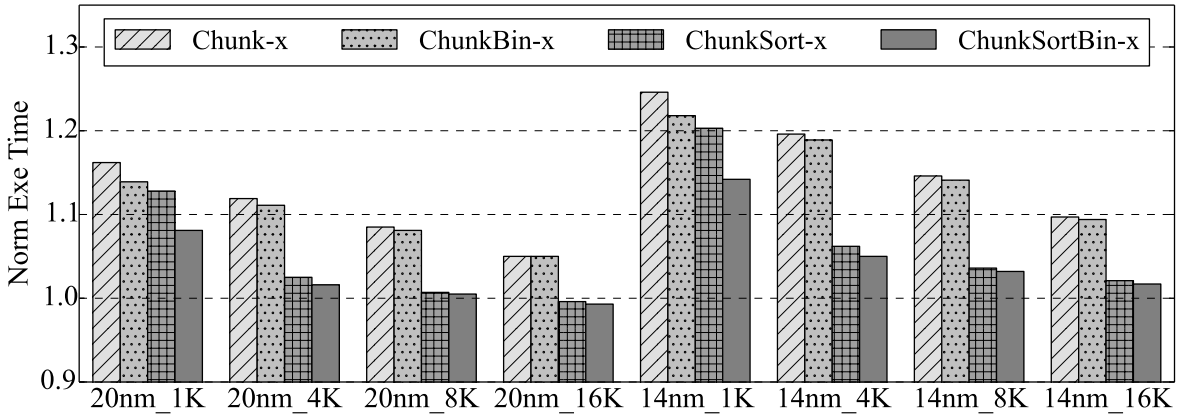
The proposed method requires memory manufacturers to test the restoring time of each chunk [6], and then to cluster chips into different ranks/DIMMs. Hence, the manufacturing time will be definitely lengthened. To avoid dramatical increase, chunk-level timing measurement can be performed in a binary search fashion, i.e., starting from middle value (e.g., 22ns), and then halving to left (e.g., 16-21ns) or right part (e.g., 23-30ns). And further, the number of chunks is fixed to be smaller than that of rows, and thus the testing time of each chip is $O(1)$.

For rank construction, thousands of chips should be clustered into different bins, as discussed in Section 3.2.4, for later rank formation. To make the algorithm more practical, we propose to run the algorithm for a group of, e.g. 10K, already manufactured chips,

[6] As [Wang et al., 2015], manufacturing testing results are unnecessary to be saved inside DRAM.



(a) Sensitivity on sparing levels



(b) Sensitivity on number of chunks

Figure 3.12: Sensitivity study using different spares and chunks under hot page allocation policy.

instead of running until after one-day production. Apparently, the algorithm cost would not be too high given the moderate number of chips, and the chip production would have tolerable impact.

3.5.6 Storage, Area and Latency Overheads

To enable variation aware memory scheduling, we added two tables: one is for extracting the timing constraints of each chunk while the other is for chunk address remapping. Table 3.7 summarizes the storage overhead with different chunks.

Table 3.7: Tested Chunk Configuration

#CKs	MC area (KB)	DIMM area (KB)	total area (KB)
1k	24	224	248
4k	96	896	992
8k	192	1792	1984
16k	384	3584	3968

From Table 3.7, `ChunkSort-4k` requires 896KB DIMM storage to save the chunk mapping. The table is evenly divided among the banks in the DIMMs, i.e., 56KB per bank. We used CACTI 5.3 [HP, 2008] to model the table as a direct-mapped cache with 8B line size. For 32nm [7] technology, it has 0.348ns access latency, $0.229mm^2$ area overhead, 0.0181W standby leakage power, and consumes 0.012nJ dynamic energy per access. Similarly, for the 96KB cache at the memory controller side, we have 0.414ns access latency, $0.349mm^2$ area overehead, 0.015nJ energy per access, and 0.03W standby leakage power. The area, power and energy overheads are very moderate for the DIMM and the memory controller.

Remap table occupies about $1.83mm^2$ area in total, which is trivial compared to conventional RCD area of $108mm^2$ [JEDEC, 2009c] and DIMM area of over $4000mm^2$ [Micron, 2009b]. The overheads of added register and DIMM PCB traces are moderate [Ahn et al., 2008], and the overheads can be even smaller by combining adjacent chips into groups [Ahn

[7] Different from sub-20nm DRAM chips, caches are constructed with 32nm process technology, which is supported by CACTI tool.

et al., 2008]. Based on the latency overheads calculated by CACTI, we charged 1 memory cycle [8] and 2 CPU cycles to access the chunk mapping table and the timing table, respectively. We observed less than 1% performance overhead on average.

3.6 CONCLUSION

In this chapter, we modeled DRAM process variation, studied the scaling effects on restore time, and showed that future DRAM chips need relaxed timing constraints to maintain high yield and to keep the manufacturing cost low. Existing approaches are worst-case determined, and thus they are ineffective to address the performance losses. We proposed schemes to expose restore time variations at chunk level and devised architectural enhancements to enable find-grained variation-aware scheduling. We then proposed restore time aware rank construction and page aware page allocation schemes to make better use of fast chunks. The experimental results show that our schemes achieve as high as 25% average performance improvement over traditional solutions.

[8] One memory cycle is enough because only one eighth of the table is being looked up for each access, whereas the total size of the remap table is hundreds of KBs.

4.0 SHORTEN RESTORE USING REFRESH

4.1 MOTIVATION OF PARTIAL RESTORE

Scaling DRAM to 20nm and below faces significant manufacturing difficulties: cells become slow and leaky [Son et al., 2014; Wang, 2015] and exhibit a larger range of behavior due to process variation (i.e., there is a lengthening of the tail portion of the distribution of cell timing and leakage) [Kang et al., 2014; Zhang et al., 2015a; Mukundan et al., 2013].

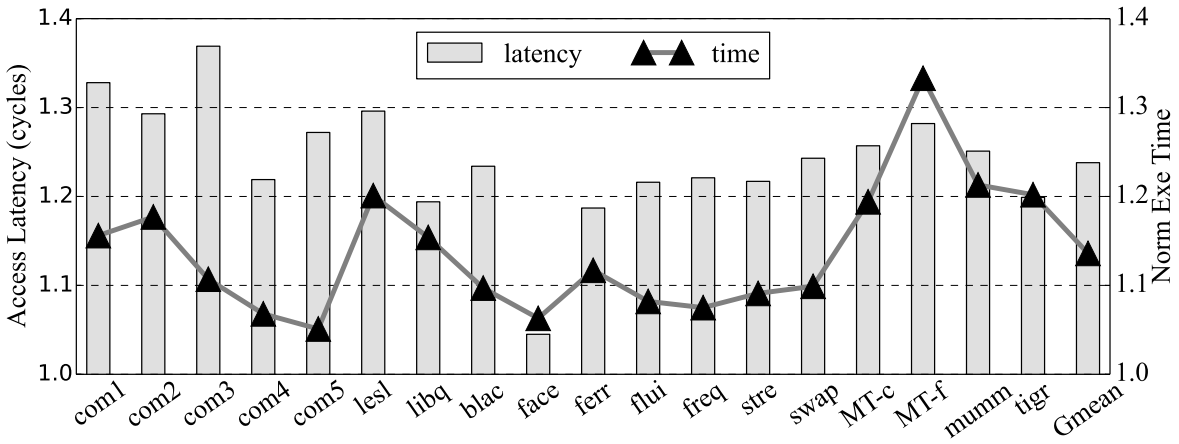


Figure 4.1: Access latency and execution time increase due to relaxed restore timing. `Baseline` adopts standard timing constraints in specifications.

As bit cell size is reduced, the supply voltage V_{dd} also reduces, causing cells to be leakier and store less charge [Mukundan et al., 2013]. For instance, DDR3 commonly uses 1.5V V_{dd}, while DDR4 at 20nm uses 1.2V [Samsung; Mukundan et al., 2013]. Performance oriented DRAM enhancements, such as high-aspect ratio cell capacitors [Kang et al., 2014; Mukundan et al., 2013], often worsen the situation. DRAM scaling also increases noise along bitline and

sensing amplifier [Mukundan et al., 2013; Ryan and Calhoun, 2008; Kurinec and Iniewski, 2013], which leads to longer sensing time. Scaling also degrades DRAM restore operation due to smaller transistor size, lower drivability and larger resistance [Kang et al., 2014; Zhang et al., 2015a; Mukundan et al., 2013; Wang, 2015].

The growing number of slow and leaky cells has a large impact on system performance. Among the three general strategies, as discussed in Section 2.3, to address this challenge, we choose to relax timing constraints to strive for high chip density and yield.

Partial restore. Due to intrinsic leakage, the voltage level of a DRAM cell reduces monotonically after a full restore. The solid curve in Figure 4.2 illustrates the voltage decay of an untouched cell (i.e., not accessed) within one refresh window. Stored data is safe as long as the voltage remains above V_{min} ($0.73V_{dd}$ here, discussed in Section 4.4) before the next refresh. If a read or write access occurs, the post-access restore operation fully charges the cell by default, as shown in the figure. However, the full charge is often unnecessary if the access is *close in time* to the next refresh, which will fully restore the cell anyway.

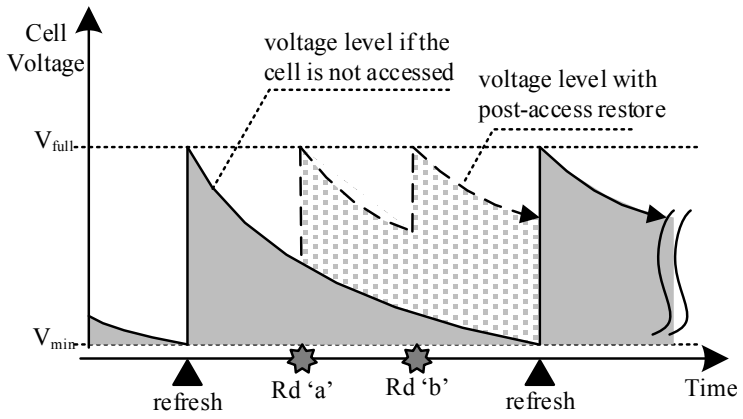


Figure 4.2: DRAM cell voltage is fully restored by either refresh commands or memory accesses. (V_{full} indicates fully charged; V_{min} is the minimal voltage to avoid data loss).

Based on this observation, we propose that post-access restore *partially charges* a cell's voltage to the level that the cell would have if the cell had been untouched in one refresh window. The restore operation is truncated when this target voltage level is reached.

The cell charging curve starts with a deep slope and flattens when approaching V_{full} [Lee et al., 2015; Demone, 2011], as shown in SPICE modeling and simulation results in

Section 4. Hence, reducing target voltage can drastically shorten restore time. For example, SPICE modeling indicates that restoring a cell's charge to $0.89V_{dd}$ rather than $0.975V_{dd}$ (fully charged) reduces `tWR` from 25 to 15 DRAM cycles, i.e., a 40% reduction.

We next describe two schemes, `RT-next` and `RT-select`, to enable partial restore. These schemes are applied by the memory controller.

4.2 PROPOSED DESIGNS

4.2.1 RT-next: Refresh-aware Truncation

`RT-next` truncates a long restore operation according to the time distance to its next refresh. The sooner the next refresh is, the less charge the cells in the row need, and the earlier the restore operation can be terminated.

Table 4.1: Adjusted restore timing values in `RT-next` (using the model in Section 4.4)

sub-window	Distance to next refresh			Target restore voltage (V_{dd})	`tRAS`	`tWR`	`tRCD`
	64ms-row	128ms-row	256 ms-row		(DRAM cycles)		
1st	[64ms, 48ms)	[128ms, 96ms)	[256ms, 192ms)	0.975	42	25	15
2nd	[48ms, 32ms)	[96ms, 64ms)	[192ms, 128ms)	0.92	27	18	15
3rd	[32ms, 16ms)	[64ms, 32ms)	[128ms, 64ms)	0.86	21	14	15
4th	[16ms, 0ms)	[32ms, 0ms)	[64ms, 0ms)	0.80	18	11	15
No Truncation				0.975	42	25	15

`RT-next` works as follows. We partition one refresh window into multiple sub-windows. While accesses falling in different sub-windows use different sets of timing parameter values, those falling in the same sub-window use the same set of values. In the following, we use four sub-windows to discuss our proposed schemes — Table 4.1 lists the adjusted timing values for the device that we model in this thesis. The smaller the timing values are, the larger truncation opportunity the truncation has. While distinguishing more sub-windows provides

finer-grained control and the potential to exploit more truncation benefits, it complicates the control and provide little further benefits as shown in our experiments.

When servicing a read or write access, `RT-next` uses the following formula to calculate the time distance to the next refresh command and determine the sub-window that the access falls in. It then truncates its restore operation using the adjusted timing parameters, e.g., the right most columns in Table 4.1.

$$Distance = ((8192 + Bin_c - Bin_a)\%8192 + 1) \times \frac{64ms}{8192} \tag{4.1}$$

where $\texttt{Bin}_c$ is the last bin that was refreshed; $\texttt{Bin}_a$ is the refresh bin to which the row being accessed belongs. In multi-rate scenario, the calculation is adjusted to include the further 64ms refresh rounds, which will be discussed later.

The above calculation needs the mapping from row address to bin address. While the bin counter is maintained in the memory controller and incremented sequentially, the actual row addresses (responding to each bin-refresh command) are generated internally inside DDRx devices [JEDEC, 2009b, 2012]. This mapping may be non-linear because of vendor's full flexibility to implement the refresh. Recent studies [Bhati et al., 2015b; Kim et al., 2014] assume this mapping can be made known to the memory controller. We make the same assumption in this thesis.

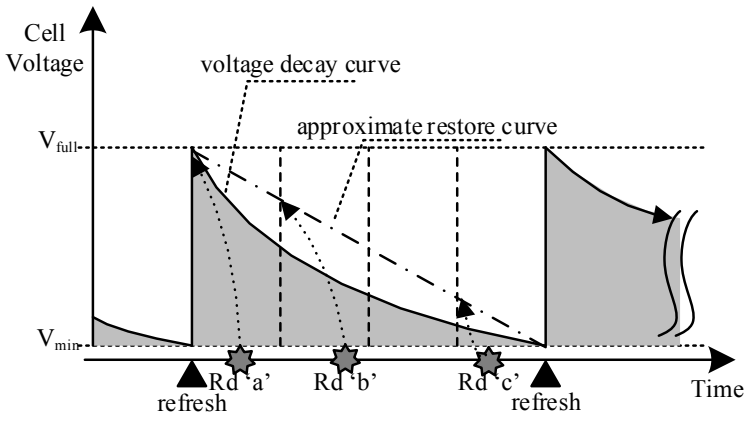


Figure 4.3: `RT-next` safely truncates restore operation.

The memory controller also needs to consider the page policy (open or close). A restore

is truncated by a PRE command from the memory controller. For a closed-page policy, every access can potentially benefit from restore truncation. For an open-page policy, truncating restore of a preceding access may not beneficial if its following access is a row buffer hit. We evaluate both policies in the experiments.

To adapt to cell variations within a DRAM row, RT-next takes a conservative approach, as illustrated in Figure 4.3. In the example, reads 'a', 'b', and 'c' are serviced in the first, the second, and the fourth sub-windows, respectively.

- RT-next assumes the worst case scenario, i.e., the currently accessed row has weak cells that barely meet timing constraints and these weak cells are leaky enough that their voltage levels are reduced to V_{min} before the next refresh. The weak cells are difficult to restore because fully charging them requires long latency. The adjusted restore timings in Table 4.1 ensure that slow and leaky cells can accumulate charge more than what they have under natural decay, i.e., they are not accessed in one refresh window.
- RT-next restores to the target level at the beginning of each sub-window. In particular, while it is possible to partially restore an accessed row in the first sub-window, e.g., read 'a' in Figure 4.3, RT-next conservatively fully restores the row, i.e., with no truncation.
- Due to a slightly faster rate of leakage at higher voltage (as shown in Section 4), a DRAM cell has an exponential decay curve that is close but always below the linear line between V_{full} and V_{min} in Figure 4.3. This curve varies from row to row, which implies that different restore timing values are needed. To simplify the control in memory controller, RT-next conservatively sets up the voltage restore targets, at the beginning of each sub-window, as the voltage levels on the linear line, rather than on the curve. This allows RT-next to use the same timing parameters for all rows.

RT-next in multi-rate refresh. Applying RT-next in a multi-rate refresh environment works similar to the case that has only one rate. To calculate the distance between a memory access and the next refresh to its bin, RT-next uses the same formula except adding the extra refresh rounds for low rate, i.e., 128/256ms, bins. Here we assume the underlying multi-rate refresh scheme has profiled and tagged each bin with its best refresh rate, e.g., 64ms, 128ms, or 256ms.

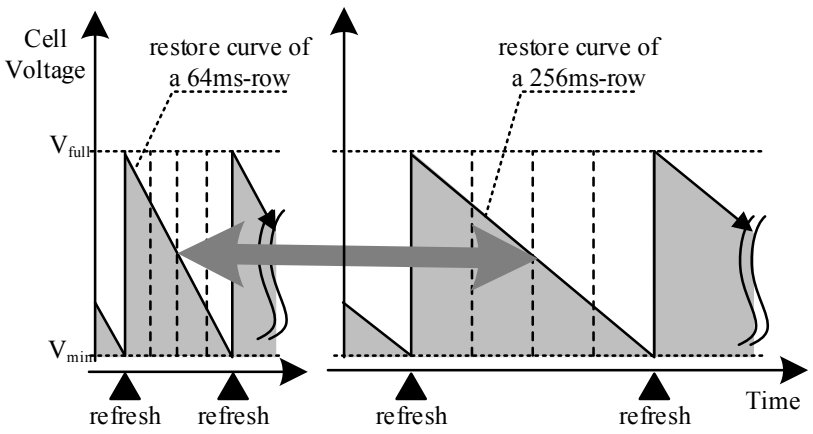


Figure 4.4: Restoring voltage according to linear line simplifies timing control in multi-rate refresh — a 64ms-row and a 256ms-row share the same timing values in each correspond sub-window.

As shown in Figure 4.4, it simplifies the timing control in memory controller by restoring a cell's post-access voltage according to the linear line between V_{full} and V_{min} (rather than the exponential decay curve). Given a 64ms-row and a 256ms-row, accesses falling in the same corresponding sub-window can use the same timing values in Table 4.1.

4.2.2 RT-select: Proactive Refresh Rate Upgrade

We next present `RT-select`, a scheme that upgrades refresh rate for more truncation opportunities. Refresh and restore are two correlated operations that determine the charge in a cell. Less frequently refreshed bins can be exploited to further shorten the post-access restore time.

Figure 4.5 illustrates the benefit of refreshing a 256ms-row (in multi-rate refresh) at 128ms rate. Given that this row is a 256ms-row, its voltage level decreases to V_{min} after 256ms. As shown in Figure 4.5(a), the refresh commands sent at +64ms, +128ms, and +192ms marks are dummy ones. The access "Rd" appears in the 2nd sub-window; it has a distance within [192ms, 128ms) to the next refresh command. According to `RT-next`, the restore can be truncated after reaching $0.92V_{dd}$ (using the 256ms-row column in Table 4.1).

Now, suppose the dummy refresh at +128ms is converted to a real refresh, i.e., the row is

"upgraded" to a 128ms-row. With this earlier refresh, the access "Rd" is at most 64ms away from the next refresh. Using the 128ms-row column in the timing adjustment table, `RT-next` can truncate the restore after it reaches $0.86V_{dd}$, achieving significant timing reduction for the restore operation (Figure 4.5(b)).

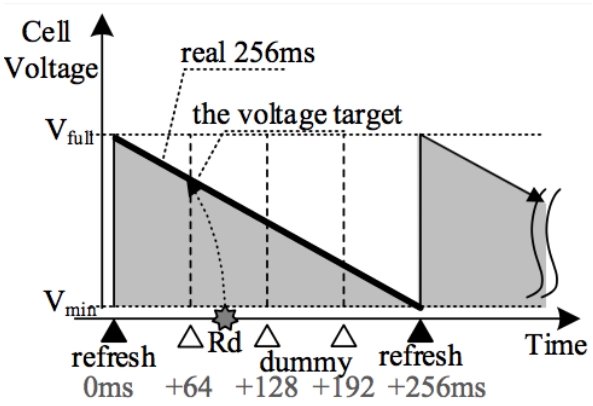
Refreshing a 256ms-row at 128ms rate exposes more truncation benefits, as shown in Figure 4.5(c). For access "Rd", it is sufficient to restore the voltage to $0.80V_{dd}$ rather than $0.86V_{dd}$ in above discussion. This is because a 256ms-row, even if being refreshed at 128ms rate, leaks slower than a real 128ms-row. We can adjust the timing values by following the solid thick line in 4.5(c), rather than the dashed thick line from a real 128ms-row, as shown in 4.5(b). In particular, as summarized in Table 4.2, a row access, even if it is 128ms away from the next refresh to the row, just needs to restore the row to $0.86V_{dd}$, rather than V_{full} ($=0.975V_{dd}$) for a real 128ms-row.

RT-select scheme. While upgrading refresh rate reduces restore time, it generates more real refresh commands, which not only prolongs memory unavailable period but also consumes more refresh energy. Previous work shows that refresh may consume over 20% of the total memory energy for a 32Gb DRAM device [Bhati et al., 2015a; Liu et al., 2012a]. Blindly upgrading the refresh rate of all rows is thus not desirable.

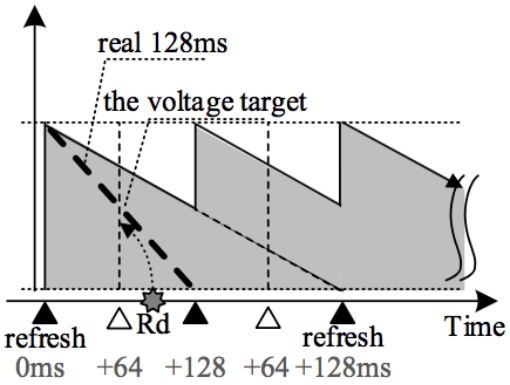
`RT-select` upgrades the refresh rate of *selected bins*, those were touched, for *one refresh window*. It works as follows. A 3-bit rate flag is attached to each refresh bin to support multi-rate refresh. When there is a need to upgrade, e.g., from 256ms to 128ms rate, `RT-select` updates the rate flag as shown in Section 3.5, which converts the dummy refresh at +128ms in Figure 4.5. A real refresh command rolls the rate back to its original rate, i.e., `RT-select` only upgrades the touched bin for one refresh window, which incurs modest refreshing overhead to the system.

4.2.3 Discussion

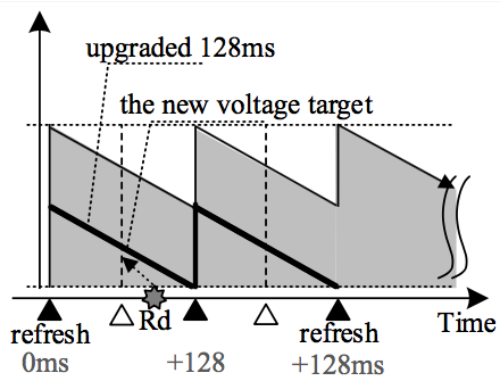
Restore truncation (`RT`) is an orthogonal design to the state of the art DRAM enhancements. We have discussed how to support multi-rate refresh [Bhati et al., 2015b; Liu et al., 2012a]. As another example, `NUAT` [Shin et al., 2014] reduces `tRCD/tRAS` timings by exploiting the



(a) the voltage target of a 256ms-row



(b) the voltage target of a 128ms-row



(c) the voltage target decreases when a higher refresh rate (128ms) is employed

Figure 4.5: The voltage target can be reduced if a strong row is refreshed at higher rate.

Table 4.2: Adjusted restore timing values in `RT-select`

Upgrade	Distance to Next refresh	Target restore voltage (V_{dd})	`tRAS`	`tWR`	`tRCD`
			(DRAM cycles)		
256ms→128ms	[128ms, 64ms)	0.86	21	14	15
	[64ms, 0ms)	0.80	18	11	15
256ms→64ms	[64ms, 0ms)	0.80	18	11	15
128ms→64ms	[64ms, 32ms)	0.86	21	14	15
	[32ms, 0ms)	0.80	18	11	15

potential between V_{min} and the voltage of a cell under natural decay. `RT` restores a cell's post-access voltage to no less than this level and thus is fully compatible with `NUAT`.

`RT` also works with strong error correction (e.g., `ArchShield` [Nair et al., 2013b]) and the recently proposed restore time mitigation scheme [Zhang et al., 2015a]. As shown in experiments, these schemes improve performance close to the one with no timing degradation. `RT` can be integrated with these designs. `RT` does not take advantage of thermal impact on restore timing. The timing parameters used in the thesis ensure reliable operation in the chip's recommended temperature range. Additional truncation opportunities may be exploited if thermal behavior is considered.

Recent studies revealed the complication in profiling the retention time of DRAM modules, which comes from two phenomena: data pattern dependence (DPD) and variable retention time (VRT) [Liu et al., 2013; Mukundan et al., 2013]. DPD can be alleviated by repeated testing and ECC protection. VRT can be alleviated by enhancing profiling through ECC and guardbanding techniques [Khan et al., 2014; Qureshi et al., 2015]. Our study works in conjunction with existing profiling techniques on these issues.

4.3 ARCHITECTURAL ENHANCEMENTS

To enable `RT-next` and `RT-select`, we enhance the memory controller, as shown in Figure 4.6. `RT` adds a truncation controller, to adjust the timing for read, write, and refresh accesses. This control is similar to past schemes that change timings [D. Lee et al., 2013; Son et al., 2013; Shin et al., 2014]. The memory controller has a register that records the next bin to be refreshed, referred to as $\texttt{Bin}_c$, which rolls over every 64ms. It can also infer the mapping from row address to refresh bin, the same as that in [Bhati et al., 2015b; Kim et al., 2014].

Table 4.3: Refresh rate adjustment table

Profiled refresh rate	Rate flag	Flag after rate upgrade
64ms	000	n/a
128ms	01A	128→64ms: 010
256ms	1BC	256→128ms: 1BC⊕0B0 256→64ms: 100

To support multi-rate refresh, the memory controller keeps a small table that uses 3 bits to record the refresh rate of each refresh bin, similar to that in [Liu et al., 2012a; Bhati et al., 2015b]. As shown in Table 4.3, a 64ms-/128ms-/256ms- bin is set as '000'/'01A'/'1BC', respectively. Here 'A' and 'BC' are initialized to ones and decrement every 64ms. While the refresh bin counter increments every in 7.8μs(=64ms/8K), a real `REF` command is sent to refresh the corresponding bin only if its bin flag is '000', '010', or '100'. 'A' and 'BC' are changed back to '1' and '11' afterwards, respectively.

When upgrading the refresh rate of a refresh bin, we update the rate flag according to the last column in Table 4.3. For example, when upgrading a 128ms-bin to 64ms rate, we set the rate flag as '010', which triggers the refresh in the next 64ms duration and roll back to '011' afterwards. This effectively upgrades for one round. Upgrading 256ms-row to 128ms rate sets the flag as '1BC⊕0B0', which always sets the middle bit to zero to ensure that

the refresh distance is never beyond 128ms, and thus the sub-window can only be `3rd` and `4th` referring to Table 4.1. In general, the distance calculation in `RT-select` is adjusted by adding value in Equation (4.1) with the further refresh rounds indicated by the two least significant bits (LSB) of rate flag.

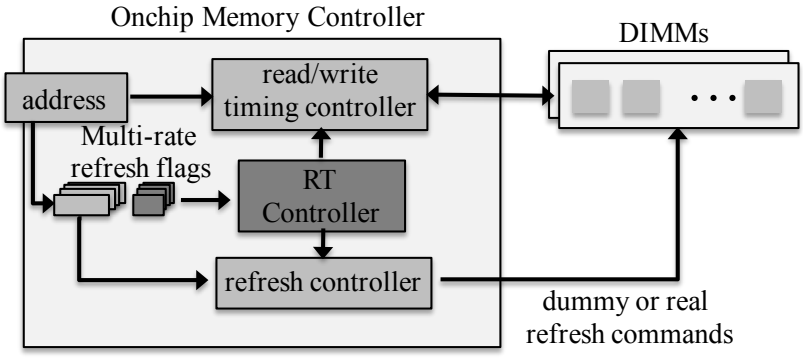


Figure 4.6: The RT architecture (the shaded boxes are added components).

To enable multi-rate refresh, the rate table is accessed before each refresh to determine if a real or dummy command should be sent. To enable `RT-select`, the rate table is also accessed before each memory access to decide the refresh distance, and then to complete the upgrade after the access. The extra energy and latency overheads are minimal, as shown in Section 4.6.4.

4.4 MODELING DETAILS

In this section, we present the details of modeling DRAM that are the underpinning of our `RT` schemes, including sensing delay, restore time, and retention time.

4.4.1 Voltage Drop

The stored charge in a DRAM cell capacitor leaks over time through its access transistor. The leakage current I_{leak} is mainly sub-threshold leakage [Chen and Peh, 2003; Jeon et al., 2010],

and it is exponentially relates to V_{cell}, which indicates that the cell voltage drops following an exponential curve. We further built a SPICE model and reported the cell voltage drop within a normal refresh window in Figure 4.7, which confirms the exponential decay.

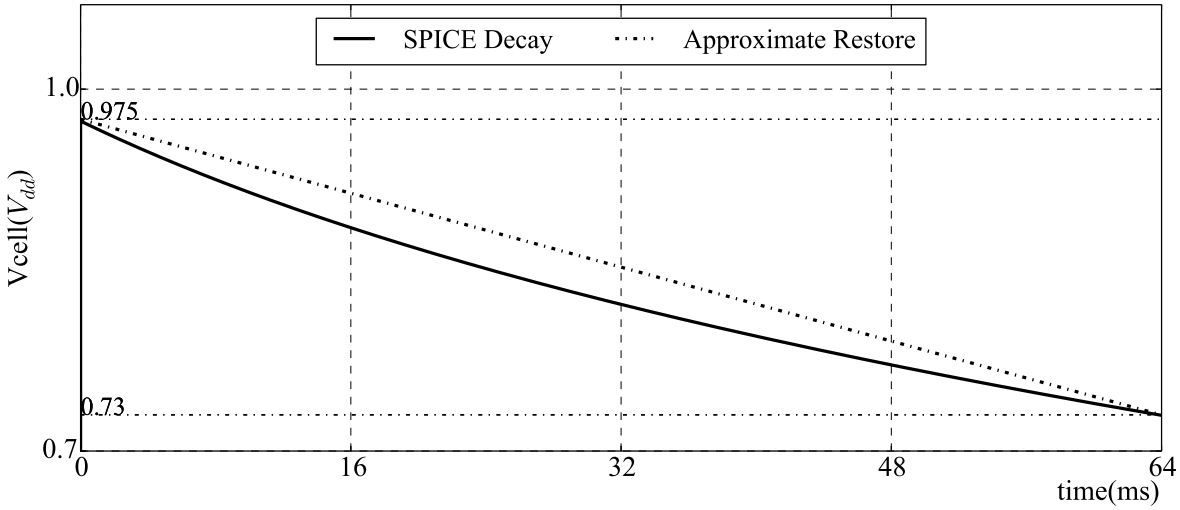


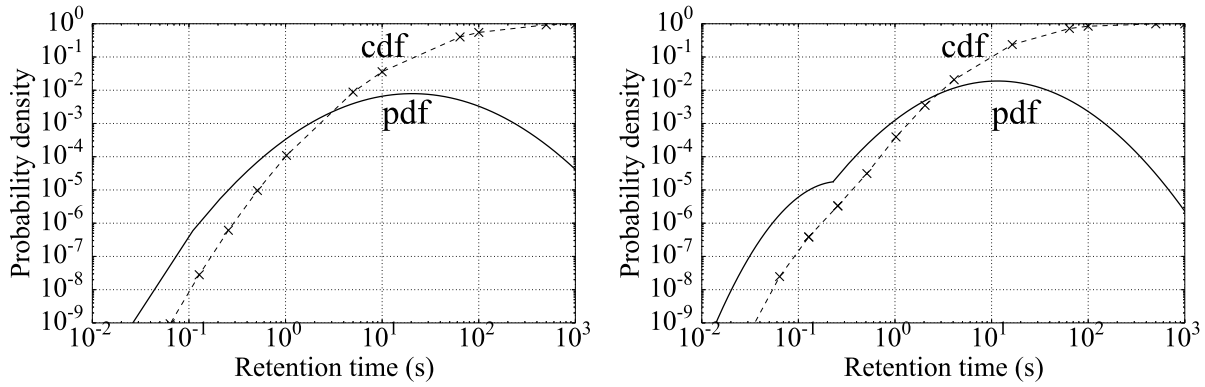
Figure 4.7: SPICE modeling of cell voltage drop. `SPICE Decay` is the exponential curve from SPICE simulation; `Approximate Restore` is a linear line from V_{full} to V_{min}, which is exploited to set up restore voltage targets in each refresh sub-windows.

4.4.2 Retention Time and Refresh

The amount of time that a DRAM cell can safely retain data is defined as *retention time*, T_{ret}, which is determined by the magnitude of the leakage current and the total charge that is allowed to lose [Hu, 2009; Wong et al., 2008; Uyemura, 2002]. Following previous work [Agrawal et al., 2014; Wong et al., 2008], we define T_{ret} as the time until the capacitor charge/voltage leaks to the minimum sustainable value (i.e., $(0.975-0.73)V_{dd}$, which is more conservative than the 60% maximum lose used in [Agrawal et al., 2014; Wong et al., 2008]). T_{ret} can be denoted as

$$T_{ret} = \frac{(V_{cell} - V_f) \times C_{cell}}{I_{leak}} = \frac{(V_{cell} - 0.73V_{dd}) \times C_{cell}}{I_{leak}} \tag{4.2}$$

where, I_{leak} is the leakage current, and V_f is the minimum readable stored voltage, which is $0.73V_{dd}$.



(a) Current technology **(b)** Future technology

Figure 4.8: T_{ret} trend as DRAM scales down. [Kim and Lee, 2009; Liu et al., 2012a]

We modeled DRAM retention time distribution based on [Kim and Lee, 2009; Liu et al., 2012a; Nair et al., 2013b], and reported the the results of current and future technologies as shown in Figure 4.8. Similar to prior works [Liu et al., 2012a; Bhati et al., 2015b; Wang et al., 2014a], leaky cells are randomly distributed throughout DRAM. We derive the weak row distribution by converting the weak cell failure probability into failure row probability.

Our modeling results show that (i) while cells are becoming more leaky, the number of cells and rows that have less than 64ms retention time is still very small, which can be corrected by enhanced error rescue schemes, like *ArchShield* [Nair et al., 2013b]. Hence, refreshing all DRAM chip rows per 64ms is sufficient to prevent data loss. (ii) The retention timing for cells in current commodity DRAM chips is conservative, which inspired designs to tighten timing for performance improvement [Chandrasekar et al., 2014; Lee et al., 2015]. The opportunity is diminishing in future chips as more cells become leaky.

4.4.3 Sensing and Restoring Time

DRAM scaling has negative impact on sensing and restore time. DRAM cell is read out by sensing the voltage difference on bit line after charge sharing. The difference is given by the expression[Keeth et al., 2007; Jiang et al., 1994; Kurinec and Iniewski, 2013]:

$$\Delta V_{BL} = \frac{(V_{cell} - V_{BL}) \times C_{cell}}{C_{BL} + C_{cell}} \tag{4.3}$$

where ΔV_{BL} is the small voltage increase on bitline, and C_{BL} (V_{BL}) and C_{cell} (V_{cell}) are the capacitance (voltage) of bitline and cell, respectively. However, offset noise [Mukundan et al., 2013; Liu et al., 2013] weakens ΔV_{BL} [Hong et al., 2002], which might lead to read failure. To correctly read the cell content, the effective signal is required to be larger than zero:

$$\Delta V_{effective} = \Delta V_{BL} - \Delta V_{noise} > 0 \tag{4.4}$$

The noise voltage of existing DRAM [Shin et al., 2014; Lee et al., 2015; Vogelsang, 2010] is conservatively set to 25mV as shown in [Hong et al., 2002]. For further scaling, a smaller V_{dd} of 1.0V is used referring to [Sinha et al., 2012; Iwai, 2009]. In this thesis, we make a conservative assumption that doubles the existing offset value of 25mV to 50mV, which indicates a $0.73V_{dd}$ minimum cell voltage following Equations (4.3) and (4.4).

DRAM restore time is degraded due to deceasing transistor drivability. While `tWR` has been kept at 15ns across generations, it is challenging to continue this value for sub-20nm technologies [Kang et al., 2014]. We followed [Zhang et al., 2015a] to obtain the distribution of `tRAS` and `tWR`, where extremely slow cells/rows are rescued by existing redundancy techniques.

We built core circuits for a DRAM array, including cell, sense amplifier, write driver and column mux, etc, and simulated them in SPICE. The circuits have generic topologies [Jacob et al., 2007] and their transistor parameters were taken (and projected) from a DRAM modeling tool [Vogelsang, 2010]. The obtained circuit curve is as shown in Figure 4.9. From the figure, we can see that both `tRCD` and `tRAS` are increased in future DRAM.

In this thesis, we focus on the relationship between restore and retention. Consequently, unrelated timing values, such as `tRCD`, are unchanged [1].

[1] Whereas this thesis places a particular focus on restoring time study, the proposed schemes are orthogonal to `tRCD` reduction designs.

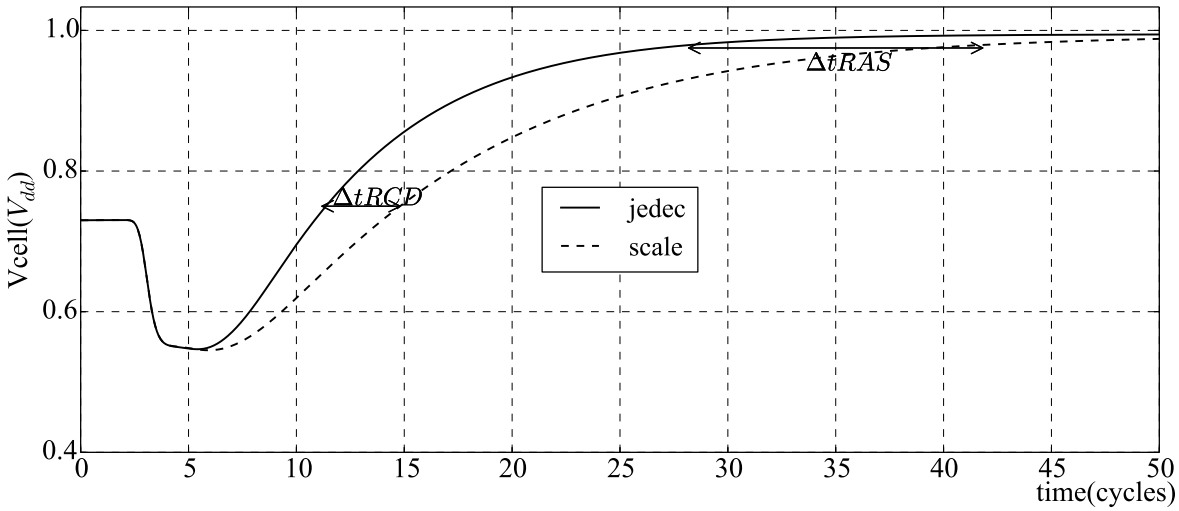


Figure 4.9: SPICE modeling on `tRCD` and `tRAS`. `jedec` follows the JEDEC specification and `scale` meets the projected values. `tRCD` ends at $0.75V_{dd}$ [Shin et al., 2014; Lee et al., 2015], and `tRAS` completes at $0.975V_{dd}$.

4.5 EXPERIMENTAL METHODOLOGY

4.5.1 System Configuration

To evaluate the effectiveness of our proposed designs, we performed the simulation using the memory system simulator USIMM [Chatterjee et al., 2012], which simulates DRAM system with detailed timing constraints. USIMM was modified to conduct a detailed study of refresh and restore operations.

We simulated a quad-core system with settings listed in Table 5.3, similar to those in [Nair et al., 2013a; Shin et al., 2014]. The DRAM timing constraints follow Micron DDR3 SDRAM data sheet [Micron, 2009a]. By default, DRAM devices are refreshed with 8K `REF` within 64ms, and `tRFC` is 208 DRAM cycles, which translates into a `tREFI` of 7.8 μs (i.e., 6240 DRAM cycles). As [Nair et al., 2013a], the baseline adopts closed page policy, which is preferred in multicore systems [Liu et al., 2012b].

Table 4.4: System Configuration

Processor	four 3.2Ghz cores; 128 ROB size
	Fetch width: 4, Retire width: 2, Pipeline depth: 10
Memory Controller	Bus frequency: 800 MHz
	Write queue capacity: 64
	Write queue high watermark: 40
	Write queue low watermark: 20
	Address mapping: rw:cl:rk:bk:ch:offset
	Page management policy: closed-page with FRFCFS
DRAM	2channels, 1rank/channel, 8banks/rank,
	64K rows/bank, 8KB/row, 64B cache line
	tCK=1.25ns, width: x8
	tCAS(CL): 13.75ns, tRCD: 13.75ns, tRC: 48.75ns
	tCWD: 6.25ns (5 cycles), tBURST: 5.0ns (4 cycles)
	tRAS: 35ns, tRP: 13.75ns, tFAW: 24 cycles,
	tRRD: 5 cycles, tRFC: 208nCK, tREFI: 7.8μs

4.5.2 Workloads

Table 4.5 lists the workloads for evaluation. They are from the Memory Scheduling Championship [MSC, 2012], and cover a wide variety of benchmarks, including five commercial applications *comm1* to *comm5*, nine benchmarks from PARSEC suite and two benchmarks each from the SPEC suite and the Biobench suite. Among them, *MT-fluid* and *MT-canneal* are two multithreaded workloads. As [Nair et al., 2013a], the benchmarks are executed in rate mode, and the time to finish the last benchmark is computed as the execution time.

Table 4.5: Workloads

COMMERCIAL	comm1, comm2, comm3, comm4, comm5
PARSEC	Black, face, ferret, fluid, freq, stream, swapt, MT-canneal, MT-fluid
SPEC	leslie, libq
BIOBENCH	mummer, tigr

4.6 RESULTS AND ANALYSIS

4.6.1 Schemes to Study

To evaluate the effectiveness of `RT` schemes, we studied the following schemes:

— `Baseline`. This scheme adopts the projected relaxed timing (`tRCD`=15, `tRAS`=42, and `tWR`=25) in future DRAM chips. The timing is applied to all rows and chips, and fits the worst-case.

— `ConvTm`. This scheme assumes the conventional timings (`tRCD`=11, `tRAS`=28, and `tWR`=12) for future DRAM chips. This is an ideal scheme as it is unclear how to efficiently handle the large number of weak cells that cannot meet these timings.

— `NoRefresh`. This scheme assumes no refresh activity in `Baseline`, which eliminates its impact on performance as well as energy consumption. It marks the performance upper bound of multi-rate refresh and other enhancement designs, including `RAIDR`[Liu et al., 2012a], `RefreshPausing` [Nair et al., 2013a] and `ArchShield` [Nair et al., 2013b].

— `RT-next-f64/-var`. This scheme is built on top of `Baseline`, and truncates a long DRAM restore operation based on its distance to the next refresh event. Whereas `RT-next-f64` assumes that all rows adopt 64ms refresh rate, `RT-next-var` explores the retention time variation to truncate refresh operations.

— `RT-all-up128/-up64`. This scheme is to trade refresh for restore truncation benefits by converting dummy refresh commands to real ones. It upgrades the refresh bins that have lower than 128/64ms rate to use 128/64ms rate.

— `RT-sel-up128/-up64`. This scheme is similar to `RT-all-`. The difference is that it does upgrade only for the touched bins.

Next, we compared the schemes on system performance, memory access latency and energy, and then studied their sensitivities to different configurations. To study different aspects of RT, we analyze different set of schemes in each part.

4.6.2 Impact on Performance

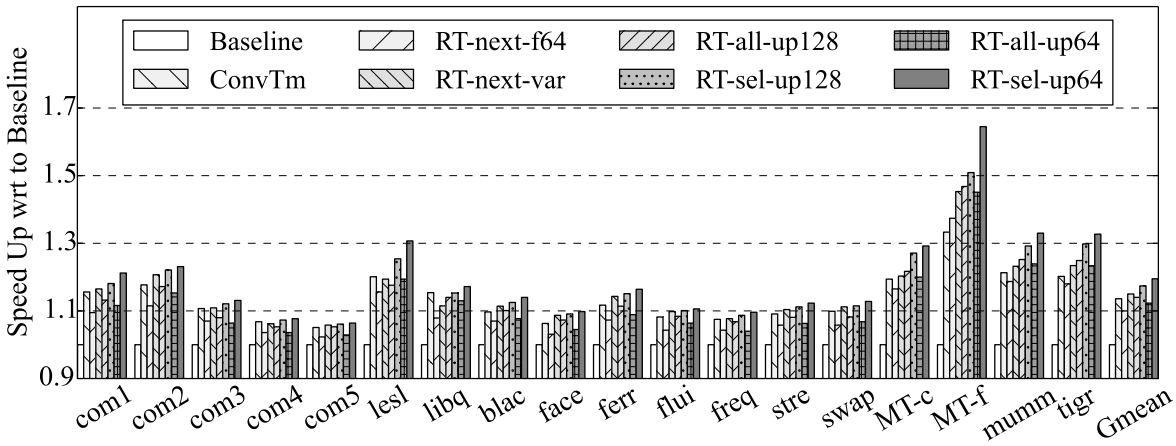


Figure 4.10: Performance comparison of different schemes.

Figure 4.10 compares the execution time of different schemes. The results are normalized to `Baseline`. In the figure, `Gmean` is the geometric mean of the results of all workloads.

On average, `RT-next-f64` achieves 10% improvement over `Baseline` by truncating restore time. `RT-next-var` identifies more truncation opportunities in multi-rate refresh DRAM modules and achieves better, i.e., 15%, improvement. While `RT-all-up128` truncates more restore time through refresh rate upgrade, it introduces extra refresh overhead and thus is slightly worse than `RT-next-var`. `RT-sel-up128` achieves 2.4% improvement over `RT-next-var` by balancing refresh operations and restore benefits. The performance gap between upgrading all rows and selective upgrading is even larger when we aggressively upgrade refresh rate to 64ms. `RT-sel-up64` achieves the best performance — it is 19.5% speedup over `Baseline`, or 4.5% better than `RT-next-var`. The performance trend across the schemes demonstrates that our restoring schemes achieves a good balance between refresh and restore.

Generally, memory access intensive workloads such as *com1*, *libq* and *mumm* benefit most from the reduced restore timing. Particularly, *MT-f* obtains the largest performance improvement because of the parallel access patterns and relatively tight gaps between accesses, which greatly enlarges the effect of shortened `RAS` and `WR`.

4.6.3 Impact on Access Latency

Figure 4.11 compares the memory access latencies using different schemes. The average access latency of `Baseline` is 283 DRAM cycles. Restore time reduction effectively reduces the latency for all workloads. `RT-all-up64` is worse than `RT-all-up128` due to more real refresh operations slowing down normal memory accesses. `RT-sel-up64` reduces the average latency to 210 DRAM cycles, indicating a 25.8% reduction over `Baseline`.

4.6.4 Energy Consumption

Figure 4.12 compares the energy consumption of different schemes. We reported the energy consumption breakdown — background (`bg`), activate/precharge (`act/pre`), read/write (`rd/wr`) and refresh (`ref`). We summarized the results according to benchmark suites, where

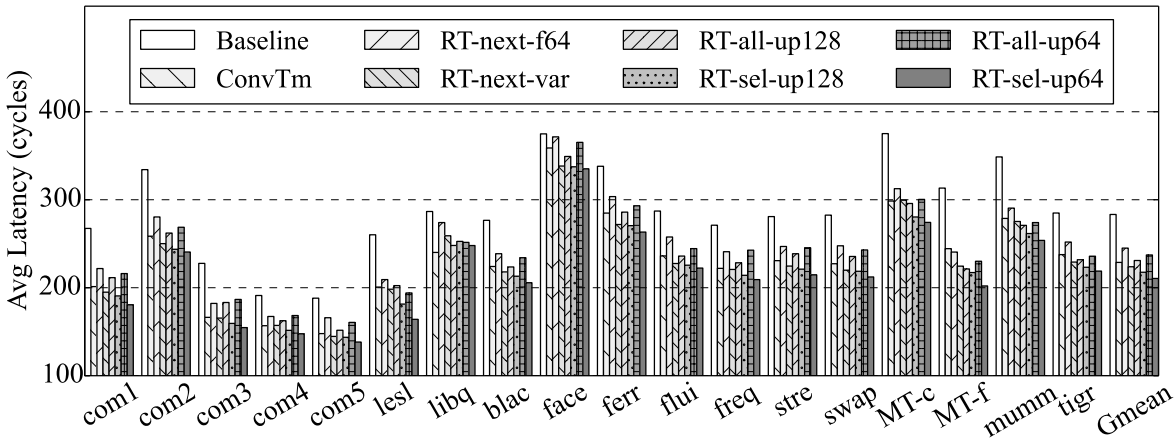


Figure 4.11: Access latency comparison of different schemes.

results are averaged over workloads within each suite. We used the Micron power equations [Micron, 2007], and the parameters from vendor data sheets [Micron, 2009a] with scaling.

To enable truncation in multi-rate refresh DRAM modules, we need to query the refresh rate for each access. The refresh rates for 8K bins are organized as 3KB direct mapped cache with 8B line size. We used CACTI5.3 [HP, 2008] to model the cache with 32nm technology — it requires 0.22ns access time, occupies 0.02mm^2 area, consumes 1.47mW standby leakage power, and spends 3.33pJ energy per access. The extra energy is trivial (less than 0.5%) and is reported together with `bg`.

From the figure, we observed that the device refresh energy for 4Gb chips is small. Due to increased refresh operations, `RT-all-up128/-up64` consume more refresh energy than `RT-sel-up128/-up64`, respectively. `RT-sel-up64` saves 17% energy compared to `Baseline`, and consumes slightly lower energy than `NoRefresh` due to decreased execution time. And, as expected, `RT-sel-` refresh schemes is more energy efficient than `RT-all-` refresh peers.

4.6.5 Comparison against the State-of-the-art

Figure 4.13 compares `RT` with three related schemes in the literature.

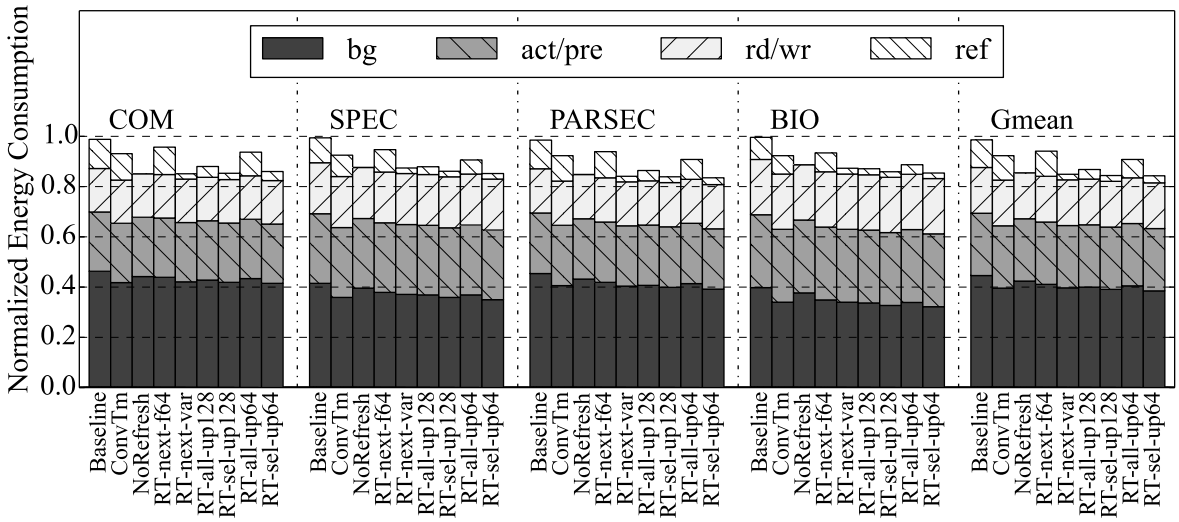


Figure 4.12: Comparison of memory system energy.

- `Archshield+` implements a scheme that treats all the cells with long restore latency as failures and adopts Archshield [Nair et al., 2013b] to rescue them.
- `MCR` is the recently proposed scheme that trade DRAM capacity for better timing parameters [Choi et al., 2015]. $2x$ `MCR` and $4x$ `MCR` are the two options that reduce DRAM capacity to 50% and 25% of the original, respectively.
- `ChunkRemap` implements the scheme that differentiates chunk level restore difference and constructs fast logic chunks through chunk remapping [Zhang et al., 2015a].

The figure shows that `Archshield+` and `ChunkRemap` are approaching `ConvTm` while `RT-sel-up64` is 5.2% better than `ConvTm`, exploiting more benefits from reduced restore time.

`MCR` shares similarity with `RT-select`, i.e., we share the observation that a line that is refreshed more frequently can be restored to a storage level lower than V_{full}. `MCR` exploits this with significant DRAM capacity reduction while `RT-select` takes a light weight design that upgrades used bins only for one refresh window, and leaves all the other bins being refreshed at original rates.

From the figure, $4x$ `MCR` outperforms `RT-sel-up64` by a modest percentage. This is be-

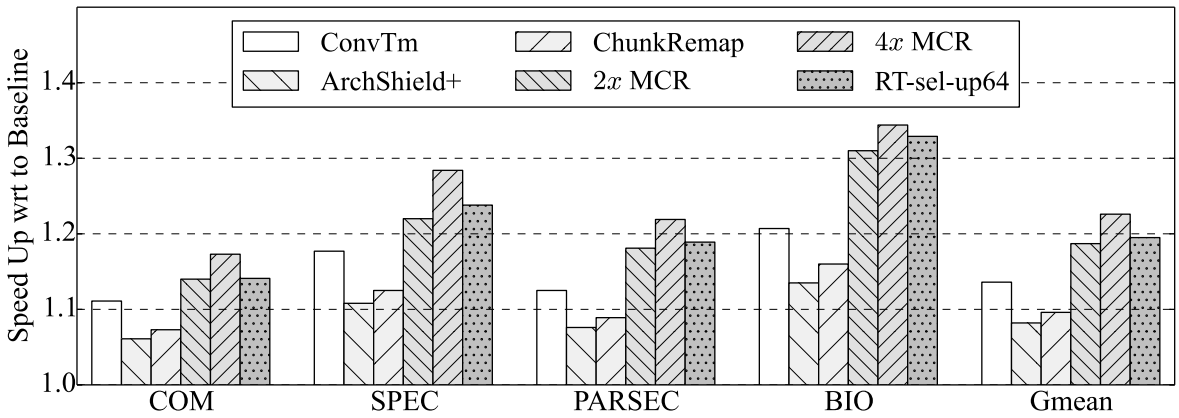


Figure 4.13: Comparison against the state-of-the-art.

cause `MCR` improves the baseline by reducing not only restore time but also sensing time while `RT-sel-up64` focuses only restore time. `RT-sel-up64` works better than $2x$ `MCR` because it upgrades the refresh rate of a bin for one refresh window at a time, which significantly reduces refresh overhead (as shown with the difference from `RT-all-up64`).

Table 4.6: Comparing EDP between RT and MCR (lower is better).

Cases	`ConvTm`	`RT-sel-up64`	$2x$ `MCR`	$4x$ `MCR-4`
`Same Chip`	1.0×	0.715×	0.753×	0.713×
`Same Capacity`	1.0×	0.715×	0.918×	1.068×

Given that `MCR` improves performance at a significant capacity reduction. We next comparing the energy-delay-product (EDP) — "Same Chip" is optimistic assumption as $4x$ `MCR` has only 25% available capacity, which is likely to have more page faults in practice. "Same capacity" enlarges the raw chip in `MCR` by two/four times, which introduces more background power. `RT-sel-up64` shows good potential as its EDP closely matches that of `MCR` under "Same chip" setting, and is much better under "Same capacity" setting.

4.6.6 Comparison against the Ideal

Table 4.7: Bound schemes to study.

Schemes	tRAS	tWR	tRCD	Refresh rate
Baseline	42	25	15	64ms
NoRefresh	42	25	15	-
ConvTm	28	12	11	64ms
BestInterval	18	11	15	64ms
BestIntNoRef	18	11	15	-

To further evaluate the effectiveness of RT, Figure 4.14 compares the proposed RT schemes against several *ideal* schemes. These schemes are *ideal* because they are infeasible in practice — Table 4.7 summarizes their timing values. NoRefresh eliminates all refresh operations in Baseline to set the performance upper bound of refresh enhancement schemes; ConvTm adopts conventional timings. BestInterval uses the the best timings (tRAS=21, tWR=11 and tRCD=15) reported in Table 4.1; BestIntNoRef further eliminates refresh operations in BestInterval.

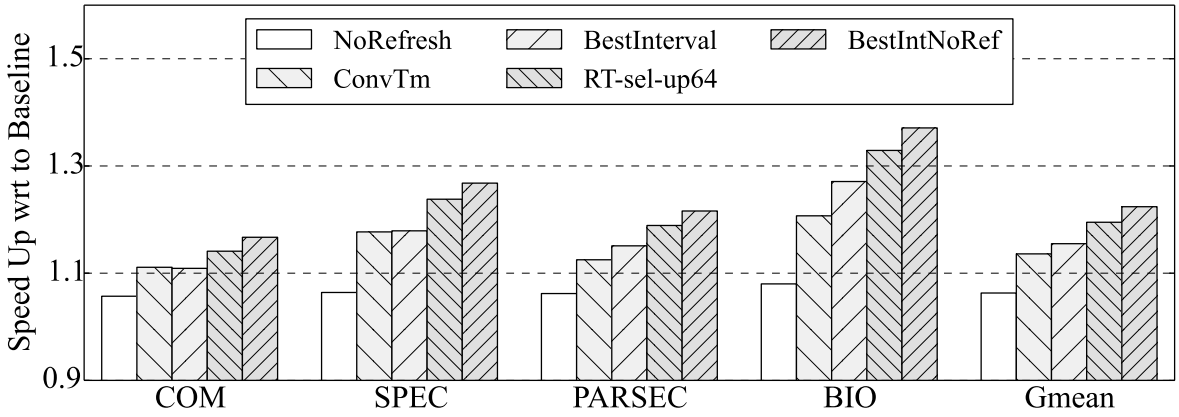


Figure 4.14: Comparison of RT-sel-up64 to candidate ideal schemes.

From the figure, NoRefresh and ConvTm are 6.3% and 13.6% better than Baseline, respectively, showing the large performance impacts from refresh activities and degraded

timing. `RT-sel-up64` is 12.4% better than `NoRefresh`, indicating that improving restore timing is more valuable than reducing refresh operations. `RT-sel-up64` beats both `ConvTm` and `BestInterval`. This is promising because, even though matching the conventional timing becomes challenging in future DRAM, more benefits can be gained by exposing and exploiting restore time differences. For most benchmarks, the gap to `BestIntNoRef` is less than 3%.

4.7 SENSITIVITY STUDIES

Next, we evaluated the performance sensitivity by varying configurations including chip density, refresh granularity, refresh sub-window division and page management policy.

4.7.1 Chip Density

Given that one refresh command is sent to refresh all rows in one refresh bin, the larger the chip capacity, the more rows the command needs to refresh, and further the larger `tRFC` is. [Micron, 2009a; Mukundan et al., 2013; Nair et al., 2013a] show that `tRFC` may grow from 260ns for 4Gb chips to 640ns for 32Gb chips.

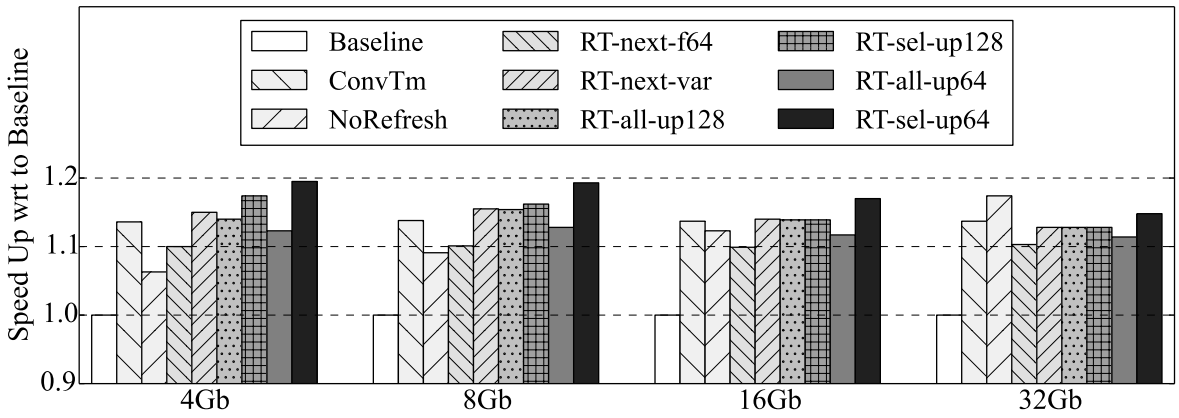


Figure 4.15: Impact of chip size on performance.

Figure 4.15 compares the results under different chip densities. The results show that

refresh penalties go up as chip size increases. RT-sel-up64, while outperforming NoRefresh and ConvTm at 4Gb chip, shows decreasing benefits in larger chip sizes, and is beaten by NoRefresh and is only slightly better than the ideal ConvTm at 32Gb chip size. In addition, we observed that RT-all-up128 and RT-sel-up128 are the same as RT-next-var at 16Gb and 32Gb. The reason is that as bin size increases, from 64 to 256 and then to 512 rows, it is hard to find a bin that can be refreshed at 256ms or lower rate. This leaves no difference for RT-all-up128 and RT-sel-up128. While the improvement of RT-sel-up64 diminishes as chip size increases, it still achieves the best result among schemes except the ideal NoRefresh. And hence, RT-sel-up64 still serves as an effective scheme to mitigate restoring issues in the long future.

4.7.2 Refresh Granularity

DDR4 standard starts to support fine-grained refresh (FGR) modes [JEDEC, 2012; Mukundan et al., 2013; Bhati et al., 2015b], that is, by lowering refresh interval (tREFI) by a factor of 2× or 4× to reduce tRFC, it can send 2x (i.e., 16K) or 4x (i.e., 32K) refresh commands, instead of the 8K commands in the normal setting.

Figure 4.16 compares the performance of schemes using different FGR modes (X-axis). 1x/2x/4x schemes maintain 8K/16K/32K refresh bins, respectively. In Row scheme, the number of bins equals the number of rows in one bank.

Whereas RT-all- schemes observe degraded performance, all other schemes achieve better results. This is because less number of rows per REF in FGR modes help to expose more non-leaky bins, which can be further utilized to shorten the restoring timings; to the contrary, blindly refresh rate upgrading in RT-all- introduces more refresh operations, and thus lead to performance degradation. And, as the figure shows, the performance gap between RT-sel- and RT-all- schemes becomes larger at finer mode. Particularly, RT-sel-up64 achieves 32.9% performance improvement over Baseline at row granularity.

4.7.3 Sub-window Division

RT adopts sub-window based timing adjustment. It becomes more complicated for the mem-

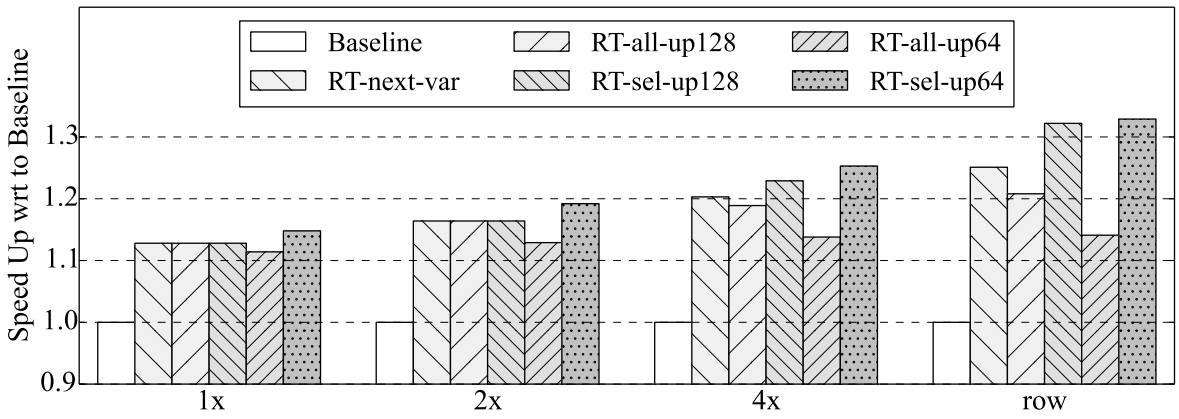


Figure 4.16: Sensitivity of refresh granularity on 32Gb chip (using multi-rate refresh).

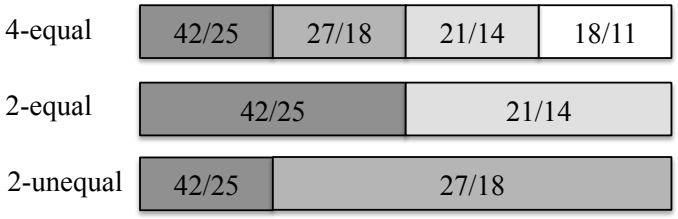


Figure 4.17: Using different sub-windows. Timings values are denoted as `tRAS/tWR` in each grid.

ory controller to schedule memory requests if the number of sub-window is large. We next study the impact of the number of sub-windows on performance.

Due to the exponential (close to linear) voltage drop curve and the long-tail charge restore curve, the timing difference of the sub-window in the second half of refresh window becomes small. And Figure 4.14 has shown that our 4-sub-windows division is very close to the best case `BestInterval`. As such, it is often not necessary to differentiate more windows. Figure 4.17 shows the settings with two or four equal/non-equal sub-windows. The adjusted `tRAS/tWR` timing parameters are also listed in each sub-window.

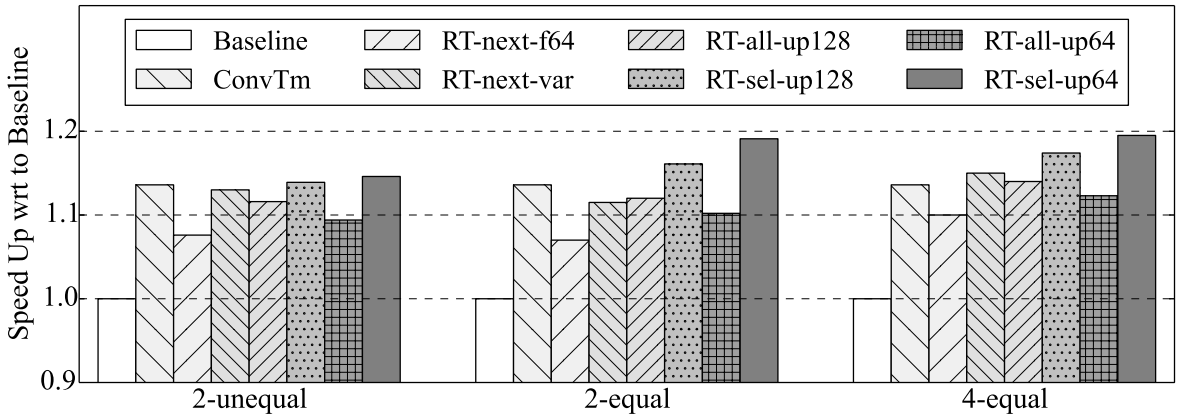


Figure 4.18: Comparing different numbers of sub-windows.

Figure 4.18 compares the performance improvements with different sub-window settings. There is a large gap between 2- and 4-sub-window designs for most schemes. And, in general, adopting four sub-window, i.e., `4-equal`, achieves better performance. `RT-sel-up64` is less sensitive to a small number of sub-window because `RT-select` charge the voltage of an upgraded row to a level much lower than V_{full}, which exploits most performance benefits.

4.7.4 Page Management Policy

By default, `RT` schemes adopt closed-page policy. We next evaluated its integration with open page-policy. We followed the recent adjustment on open-page policy [Kaseridis et al., 2011; Bhati et al., 2015b].

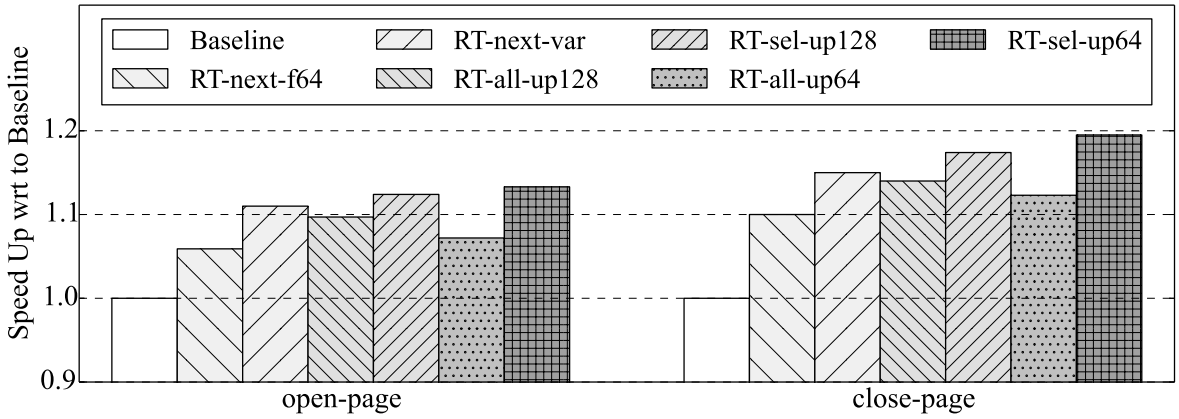


Figure 4.19: Performance comparison of 4Gb chip under open- and closed-page policies.

Figure 4.19 compares the results using different policies at 4Gb chip size. In open-page policy, the access hits in row buffer are not constrained by `tRAS`/`tWR`. Comparing to closed-page, not every access in open-page can benefit from restore truncation. Therefore, lower performance (13.3% open-page v.s. 19.5% closed-page for `RT-sel-up64`) improvement is expected, as shown in the figure. Similar results were observed on 8/16/32Gb chips.

4.8 CONCLUSION

In this thesis, we studied the restoring issues in further scaling DRAM, identified the fact that both post-access restore and periodical refresh will charge the cells. Further, with the insight that restore is more critical than refresh on performance impact, we proposed to early truncate restore operations with respect the distance in time to the coming refresh. The nearer to the refresh, the lower charge goals and thus shorter restore timing constraints. Two restore truncation (RT) schemes were proposed to divide the refresh windows (each with a unique set of timings), and also to combine multi-rate refresh concept in smart way to balance the refresh overheads and restore benefits. Our experimental results showed that, on average, RT improves performance by 19.5% and reduces energy consumption by 17%.

5.0 MITIGATE RESTORE IN APPROXIMATE COMPUTING SCENARIO

Whereas the performance loss can be effectively mitigated using the proposed techniques discussed in the previous chapters, prolonged restore time (PRT) remains a major performance bottlenecks in DRAM systems. Recently, approximation for DRAM has been proposed to improve energy consumption by trading off computation accuracy [Liu et al., 2011; Lucas et al., 2014]. By exploiting the intrinsic error resilience of many modern applications, a DRAM sub-system can save approximate data, while still achieving satisfactory computational results. Existing works on DRAM, e.g., *Flikker* [Liu et al., 2011], focus on refresh energy reduction, which unfortunately has limited impact on improving memory access latency [Arnab et al., 2015; Lucas et al., 2014]. Consequently, these schemes cannot mitigate the large performance degradation due to PRT. In addition, the benefits of these schemes can only be realized through error-resilient applications, which greatly limits their applicability for general-purpose computation.

This chapter proposes a fine-grained precision-aware restore scheduling technique, DrMP, that aggressively reduces restore time to achieve high performance. DrMP is a suite of progressively capable techniques to support approximate, precise and hybrid approximate-precise computing.

5.1 MOTIVATION OF RESTORE-BASED APPROXIMATE COMPUTING

To motivate DrMP, we first examine the severity of PRT. We followed the models in [Zhang et al., 2015a, 2016a] as described above. Although PRT appears as a major scaling challenge, the number of weak cells (i.e., cells that take a longer time to restore) actually remains small

in our observations. However, the portion of slow cells is significantly beyond the desired DRAM reliability in modern computer systems. For example, recent in-filed study [Sridharan et al., 2013] shows that the reliability is as low as 25 FIT (failure in time per billion device hours) per Mbit, making it impractical to integrate ECC with practical space overhead to rescue the weak cells.

Rather than trying to rescue the weak cells, an alternative solution to boost system performance is to aggressively reduce tWR for approximate computing. This approach exploits the error-resilience of many modern applications. Intuitively, if the errors induced by restoring bit cells faster than their required restore timing cause a tolerable number of application errors (similar to the errors induced by reducing refresh frequency [Liu et al., 2011]), then we may adopt existing approximate designs to mitigate PRT.

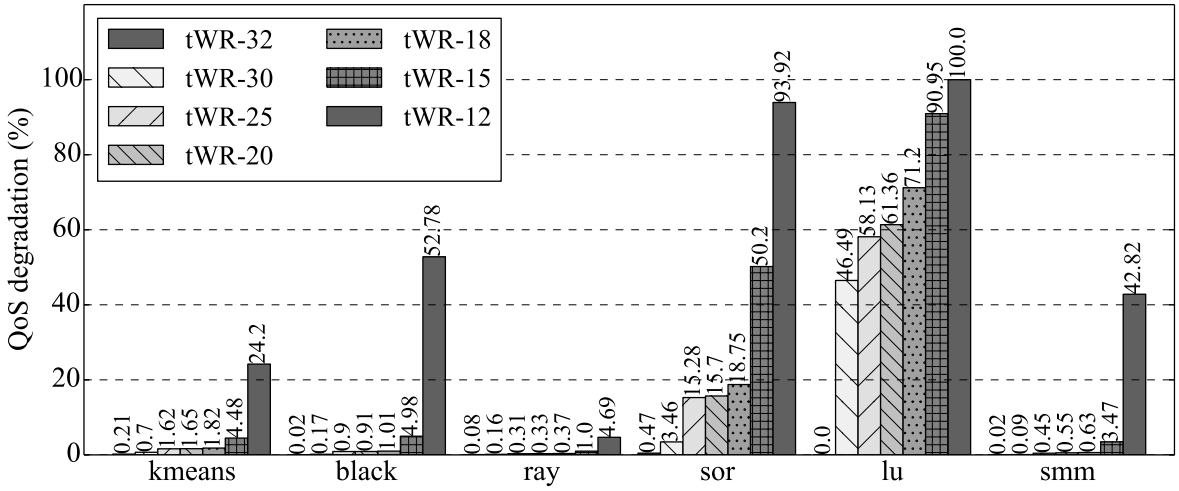


Figure 5.1: The QoS degradation with larger tWR values.

We did an experiment to gradually reduce tWR to evaluate the QoS degradation [1] for a suite of benchmark programs that were used for approximate computing from the literature. The results are summarized in Figure 5.1. The experimental setting is described in Section 5.3. From the figure, it is clear that application-level errors increase with decreasing tWR. This effect happens because, given the same DRAM row, restoring data with smaller

[1]QoS degradation is used to evaluate the output quality of approximate computing, which is calculated by comparing the approximated outputs to precise ones. The metrics are application dependent [Sampson et al., 2011; Miguel et al., 2014]

tWR leads to more cells failing to reach the target voltage level for reliable operation. The figure also illustrates that different programs exhibit vastly different error resilience. Some benchmarks, such as `smm`, only begin to suffer from high application-level errors when tWR is reduced to 15ns or smaller. Other applications, such as `lu`, begin to suffer from significant errors at a larger tWR value of 20ns. The high error rate (over 60%) observed for `lu` at 20ns is generally considered unacceptable for approximate computing [Sampson et al., 2011; Miguel et al., 2014].

The figure shows that programs are sensitive to the selected tWR value, and consequently, it is impractical to uniformly reduce tWR. Instead, we propose to reduce tWR at finer row granularity. DrMP exploits process variation (PV) within and across DRAM rows such that it can perform precision-aware restore scheduling.

5.2 PROPOSED DESIGNS

5.2.1 DrMP Design Details

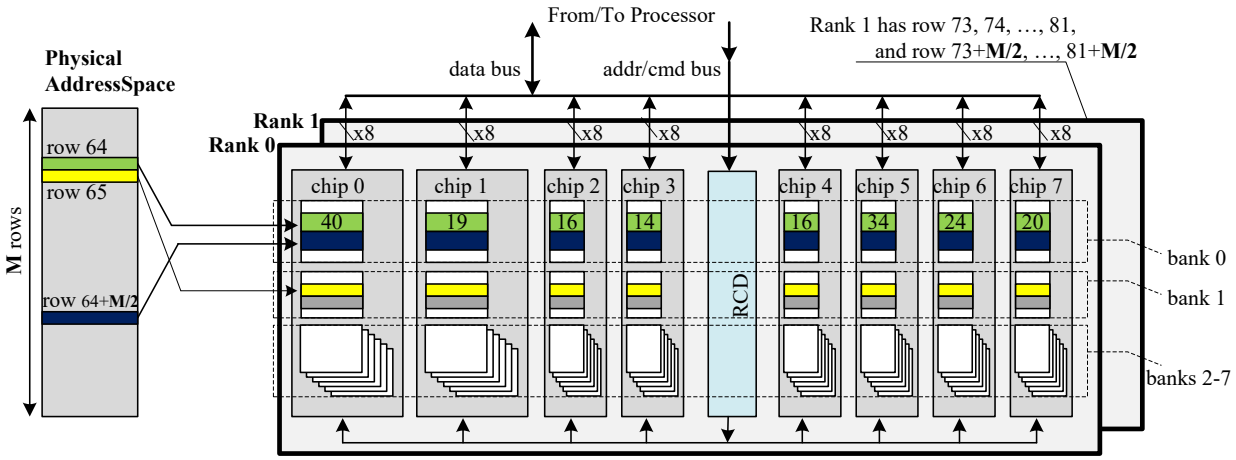


Figure 5.2: Baseline DIMM configuration. One row consists of eight row segments, one segment per chip. The number in the row segment indicates the segment's tWR value (in memory cycles).

5.2.1.1 Baseline Memory Organization

Figure 5.2 depicts the baseline memory organization in the thesis. A DIMM has two ranks while each rank has eight banks that are spread across eight DRAM chips (the baseline has no ECC chip). We assume linear row layout across all banks and ranks as follows. One row has 8KB data.

- Consecutive *physical* rows (i.e., the rows in physical address space) are mapped to different banks. For example, rows `0,1,...,15` are mapped to memory banks `0,...,7` in rank `0` and then to banks `0,...,7` in rank `1`.
- The physical address space is divided in half. Rows from the two halves are interleaved. That is, after mapping rows `0,...,15`, rows `0+K,...,15+K` are mapped to banks `0,...,7` in rank `0` and banks `0,...,7` in rank `1`, respectively. We then continue to map rows `16,...,31` and so on. There are `2K` memory rows in the memory space.

In this configuration, each DRAM row has 8 segments spread across 8 chips. Each segment is termed a ***row segment***.

The memory controller sends addresses and commands to the Register Clock Driver (RCD) on a DIMM, which enables the synchronous operation of all eight DRAM chips. The data are sent from each chip to the data bus independently. The baseline follows JEDEC's DDR3 specification [JEDEC, 2009b]; we give timing details in the experiments (see Section 5.3).

5.2.1.2 DRAM Restore Time Profiling

DrMP requires timing information about the memory. Although both reads and writes are affected by PRT, for simplicity, we describe only the design issues for tWR. In the experiments, the affect on reads (tRAS) and writes (tWR) are both fully considered and evaluated.

To determine the required information, post-fabrication profiling is done. In essence, this profiling tests memory under different settings of tWR and tRAS to find the best restore timing for DRAM row segments. The memory controller and the OS are enhanced to do the profiling similarly to prior work [Lee et al., 2015; van de Goor and Tlili, 1998]. The enhancements are: (1) the OS and the memory controller can alter timing values (tWR and tRAS) to check whether specific timing values work correctly; (2) a March test [van de Goor

and Tlili, 1998] checks the data integrity of each row by writing, reading and verifying test bit patterns in different access orders; and, (3) the test bit patterns are checked multiple times to ensure reliability with given timing values.

Since DRAM restore has worse timing at lower temperature [Kang et al., 2014; Son et al., 2014], profiling after a cold boot is safe. The timing profile is then used in the online operation of DrMP. For higher assurance, a temperature sensor could monitor memory temperature and disable DrMP if the temperature falls below the one during profiling, likewise to LPDDR [Micron, 2001]. Profiling may need to address VRT through enhanced ECC and large guardband, as described in prior work [Khan et al., 2014; Qureshi et al., 2015].

Since DRAM cells use a similar charging process for both reads and writes, tWR and tRAS are correlated, i.e., a row segment having faster tWR also has a faster tRAS. Thus, we only need to test a subset of typical combinations. For example, tRAS and tWR have ranges *[19,42]* and *[12,25]*, respectively, in cycles. When setting tRAS=19, we try tWR=12 or 13. In total, we conduct binary search for 30 tRAS and tWR combinations, resulting in around 5 tries to find the best timing. Past work has demonstrated that a March test can be performed at high speed, e.g., 0.4ms per row [van de Goor and Tlili, 1998; Rahman et al., 2014]. Therefore, the complete profiling can be done within 20 minutes.

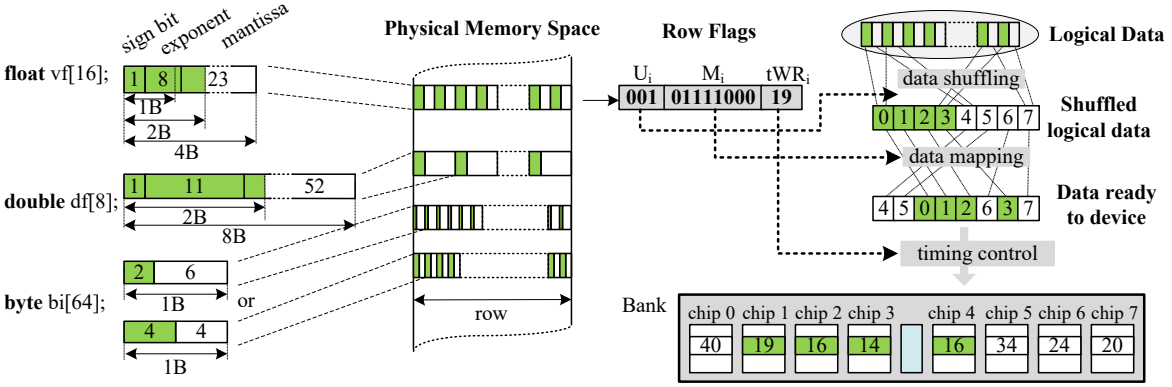
The profile keeps two timing values (tWR and tRAS, 6 bits each) for each row segment, i.e., 12B per row, or 6MB of profile data for a 4GB DIMM. The OS saves the profile (including processed data, as we show next) in system storage, i.e., hard drive or SSD, and loads it at boot-up.

5.2.2 DrMP-A: Approximate DRAM Restore

This section presents DrMP-A for high-performance approximate computing. In analyzing the ineffectiveness of the simple approximation scheme in Figure 5.1, we observe that restore errors may occur at random places in a row. Although these rows store non-critical data, depending on their data types, the importance of erroneous data bits varies. As an example, the sign bit and exponent bits of a floating point representation tend to be more important than the last several bits of the mantissa [Lucas et al., 2014; Guo et al., 2016]. For an integer

that records RGB colors, the first two or four bits of each byte are often more important.

Figure 5.3(a) shows the importance of bits of a memory row located at different places for different data types. If tWR is reduced below a reliable value, then errors from the "too fast restore" will occur at fixed positions in an application data value. If these positions are important bits, then catastrophic errors can be induced. Based on this observation, we propose DrMP-A to enable per row approximate computing that achieves extremely low error rate. Figure 5.3(b) illustrates how DrMP-A works.



(a) High order bits (green colored boxes) are more important (b) Map important bits to faster segments (no restore errors)

Figure 5.3: The details of DrMP-A.

Assume an approximate data declaration "**float** vf[...]", where the first 16 bits of each array element are tagged as important. The OS support and the process to tag important bits is discussed in later sections. DrMP-A attaches three flags $(U_i,\ M_i,\ tWR_i)$ to each row `i` in the memory space ($0 \leq i < 2K$). The flags are used by the memory controller:

(i) U_i is a 3-bit usage flag to indicate how the important bits of a row are categorized. DrMP-A uses the flags in Table 5.1; more flags can be added if needed.

(ii) M_i is an 8-bit bit vector to record the four fastest row segments. For example, we have M_i="0111100" in Figure 5.3(b), which indicates that the row segments from chips 1/2/3/4 are faster.

(iii) tWR_i is a 6-bit tWR value that records the tWR of the second or the fourth (depending on U_i) fastest row segment of a row. In the example, tWR_i=19 indicates that reducing

the row tWR to 19 memory cycles ensures there is no restore error in row segments from chips 1/2/3/4 while there might be errors in other row segments.

Table 5.1: Definition of Usage Flag

"001"	the first 2B of each 4B data are important
"010"	the first 4b of each 1B data are important
"101"	the first 2B of each 8B data are important
"110"	the first 2b of each 1B data are important
"000"	all bits are important; this is the baseline
"111"	all bits are important; used in DrMP-P and DrMP-U

The memory controller schedules requests using the flags as depicted in Figure 5.3(b). First, the controller uses U_i to shuffle data bits into groups of important bits. Second, the controller uses M_i to map the important bits to four reliable faster row segments, i.e., the segments from chips 1/2/3/4 in the example. At this point, the data are ready to be sent to the memory module. Lastly, the memory controller uses tWR_i to determine when to schedule the next memory command to maximize memory bandwidth usage.

DrMP-A is designed to achieve a good trade-off between memory performance and computation accuracy. Using the fourth fastest row segment's tWR to determine the tWR of the whole row ensures quick access to the row, improving performance over a fully reliable baseline using the worst-case row segment tWR. The majority of cells can be reliably accessed with the fast tWR. In addition, by mapping important bits to row segments with tWR values less than or equal to the row tWR, DrMP-A reduces the impact of restore errors, which minimizes the error rate at the application level.

5.2.3 DrMP-P: Pairing Rows for Fast Precise Computing

While DrMP-A speeds up approximate computing by shortening the restore time, its effectiveness is often limited by the amount of non-critical data. We next reuse this hardware enhancement to speedup precise general-purpose computing.

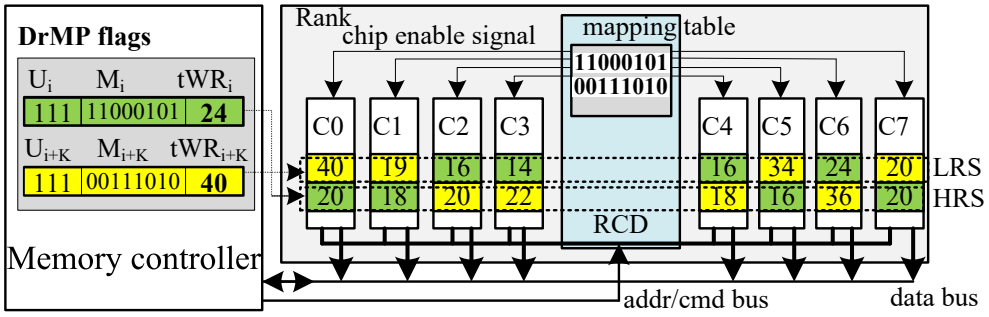


Figure 5.4: DrMP-P constructs one fast row for each row pair. Each chip stores two row segments: LRS and HRS.

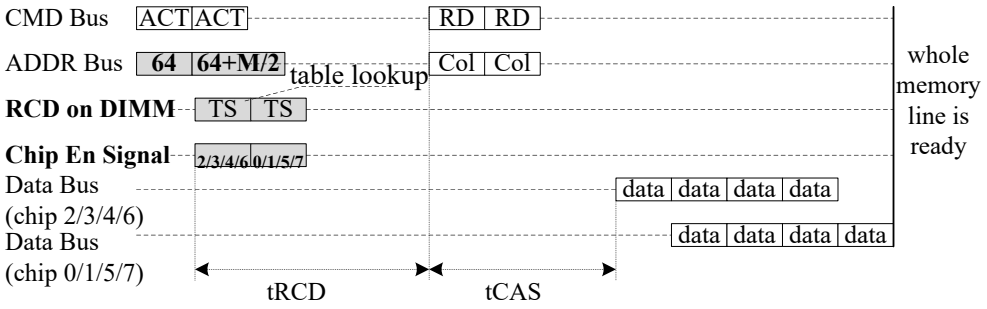


Figure 5.5: The scheduling details of DrMP-P.

By studying the tWR values at row segment level, we have found that the slowest row segments of different rows often fall within different chips. Figure 5.4 shows typical tWR values for two consecutive device rows. In the experiment, we generate a large number of chips and construct DIMMs from random chip selection. In the figure, the slowest row segments of row `i` and `i+K` are from chip 0 and chip 6 (with tWRs of 40 and 36), respectively. Based on this observation, we pair two consecutive device rows to construct a fast row and a slow row so that the average restore time can be effectively reduced, which helps to speed up precise computing.

DrMP-P first creates K row pairs so that pair `i` contains row `i` and row `i+K` (there are 2K rows in total and 0≤i<K). These two rows are K rows apart in physical address space and, as shown in Figure 5.2, next to each other on the device. Each chip j contributes two row segments to each row pair, referred to as LRS_j and HRS_j, respectively, as in Figure 5.4. Here, we have:

LRS_j Low device address row segment from chip j

HRS_j High device address row segment from chip j

Without pairing, row `i` always contains LRS segments while row `i+K` always contains HRS segments. With pairing, we re-distribute the 16 row segments in a pair such that a new row (composed of 8 row segments from 8 chips) is a mix of LRS and HRS segments. Forming a pair is simple: for two device row segments of a pair in a chip, DrMP-P assigns the faster one to row `i` and the slower one to row `i+K`. As a result, DrMP-P constructs one fast row, `i`, and one slow row, `i+K`, respectively.

For the example shown in Figure 5.4, row `i` is composed of HRS_0, HRS_1, LRS_2, LRS_3, LRS_4, HRS_5, LRS_6, and HRS_7. Row `i+K` consists of the complementary row segments for the pair. The tWR of each row is determined by the worst tWR of the composed segments, i.e., 24 and 40, respectively for these two constructed rows.

Bit flags. DrMP-P reuses the bit flags from DrMP-A. For row pair `i`, we attach (U_i, M_i, tWR_i) and $(U_{i+K}, M_{i+K}, tWR_{i+K})$ to the two rows, respectively. First, U_i and U_{i+K} are always set to "111" for precise computing. Second, M_i records the faster row segment from each chip — '0'/'1' indicate LRS/HRS. For the above example, M_i="11000101" so that $M_{i+K} = \sim M_i$ = "00111010". Lastly, DrMP-P sets the tWR of each physical row using the

largest tWR value from its component row segments.

Memory scheduling. Like DrMP-A, the flags are used by DrMP-P to schedule memory operations. Figure 5.5 illustrates the scheduling for DrMP-P. In this discussion, we use a closed page policy. The memory controller always fetch a row's DrMP bit flags before accessing the row (simlar to DrMP-A). If U_i is "111", then ACT/RD/PRE commands are sent to operate eight DRAM chips. The commands may operate on either LRS or HRS of a row pair. DrMP-P activates and accesses the LRS or HRS segments in a chip in two consecutive bus cycles as follows.

When sending ACT, DrMP-P supplies the row address `i`, which is used to index into the mapping vector table on the DIMM. In the example, this retrieves M_i = "11000101". With this mapping vector, chips 2/3/4/6 are activated first. The complementary others (chips 0/1/5/7) are activated in the next cycle. The activations are shown in bold fonts in Figure 5.5. Given that the two chip groups receive ACT and RD commands in two consecutive cycles, their output data have one memory cycle difference in the time of arrival at the processor (assuming two rows have the same tRAS).

While DrMP-P occupies two consecutive memory cycles for activation, it does not trigger tRRD constraint (i.e., the row to row delay, which restricts the number of ACT commands with a time window) as the total number of row segments activated is eight, the same as the baseline.

Extra timing. In DrMP-P, the RCD of each DIMM integrates a 256-entry mapping vector table that holds recently used mapping vectors, M_i. In addition, a one-bit enable wire is added from the RCD to each chip, as shown in Figure 5.4.

Compared to traditional memory scheduling, DrMP-P introduces two extra memory cycles on an access (modeled using CACTI [HP, 2008] as detailed in Section 3.8): (1) one cycle is used to search the mapping vector table to determine which chips to activate and access; (2) a second cycle is required due to the delayed access to the second chip group.

DrMP-P for precise computing. DrMP-P speeds up precise computing by adopting the new tWR timings when accessing memory rows. As a comparison, in the row pairing-oblivious baseline, row `i` has all LRS segments and row `i+K` has all HRS segments — their tWR values are 40 and 36, respectively. The average tWR is 38 assuming both rows are

accessed with same frequency. For the same access pattern, DrMP-P has an average tWR of 32 = (24+40)/2+2 including the 2-cycle extra access overhead.

5.2.4 DrMP-U = DrMP-A + DrMP-P

In DrMP-A and DrMP-P, a row pair is used either for approximate computing or precise computing, but not a mix of both. In this section, we propose DrMP-U, which combines DrMP-A and DrMP-P to fully exploit fine-grained differences in restore time of row segments.

DrMP-U relies on the fact that the slow logical row of a pair for DrMP-P still has several fast row segments. For example, row i+K in Figure 5.4 has two row segments with tWR values smaller than or equal to 19. If this row is used to save approximate data with usage flag "110", the important bits can be saved in fast segments, such that the row tWR can be reduced to 19, achieving a large performance improvement for approximate computing. In addition, the simple segment grouping strategy in DrMP-P is sub-optimal. Given that the tWR of row `i` is 24, it is unnecessary to take the faster row segment (LRS, tWR=16) from chip 2. Using a slower row segment (HRS, tWR=20) has no impact on the fast physical row. Yet, this choice improves the chance of a smaller tWR when using the slow row for approximate computing — the important bits can be saved in LRS of chip 2.

DrMP-U exploits this observation to construct two fast rows, instead of only one in DrMP-P. A row pair is created from two physical rows that are K rows apart, similar to DrMP-P. For the pair containing rows `i` and `i+K` ($0 \leq i < K$), DrMP-U uses row `i` to save precise data and row `i+K` to save approximate data. The pair bit flags (PU_i, PM_i, $PtWR_i$) combine the flags from the two rows, e.g., PU_i consists of two sub-flags U_i and U_{i+K}.

DrMP-U integrates DrMP-A and DrMP-P in one framework with the flags. When PU_i="000/000", the baseline is adopted for the row pair and DrMP is disabled (no approximation). When PU_i="111/111", the row pair is in the DrMP-P mode (precise-only computing). When PU_i="aaa/bbb" and "aaa" is neither 111 or 000, the mode is DrMP-A and "bbb" cannot be 111 or 000 (approximate-only computing). If PU_i is not one of these cases, then PU_i must be "111/bbb" where *bbb* can be 001, 010, 101, or 110, depending on what approximate data to save in row `i+K`. This mode is DrMP-U (hybrid approximate- and

precise-computing).

PM_i has two 8-bit bit vectors. The first vector assists the access of row `i` for precise computing, as discussed for DrMP-P. DrMP-U uses (1) the negation of the first bit vector; and (2) the second bit vector to access row `i+K` . The former determines the row segments to hold approximate data while the latter determines the subset of segments to hold important bits, as shown in Figure 5.6. $PtWR_i$ saves two tWR values for accessing row `i` and `i+K`, respectively. Both are fast accesses.

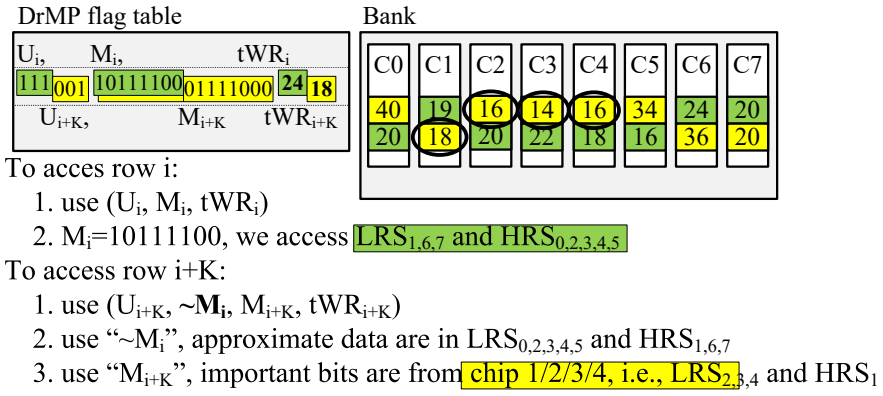


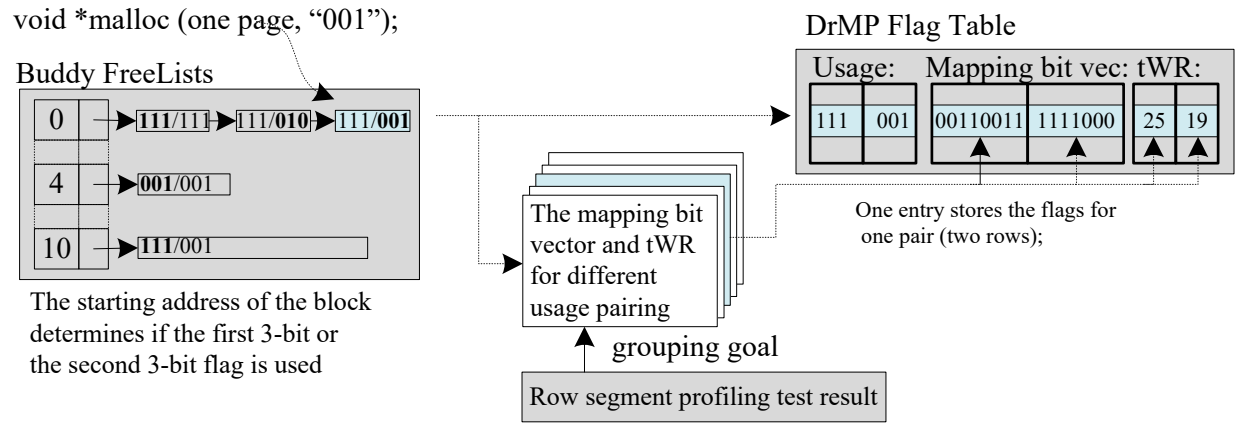
Figure 5.6: DrMP-U combines DrMP flags and uses two mapping vectors to enable approximate computing.

For the example in Figure 5.6, when accessing row `i+K` for approximate computing, DrMP-U uses "$\sim M_i$", i.e., 01000011, to find that the approximate data are saved in LRS_0, HRS_1, LRS_2, LRS_3, LRS_4, LRS_5, HRS_6, and HRS_7. Given that M_{i+K} is "01111000", the middle four segments save important bits in chips 1/2/3/4, i.e., HRS_1, LRS_2, LRS_3, and LRS_4.

Row segment grouping. To maximize the scheduling opportunity for approximate computing, DrMP-U needs to find a better row segment grouping solution.

Given a device pair, there are 16 row segments $\mathtt{S}_j$ ($0 \leq j \leq 15$). Let us assume S_0 ... S_7 are the LRS from chip 0 to chip 7; S_8 ... S_{15} are the HRS from chip 0 to chip 7; and their tWR values are $\mathtt{tWR}_j$ accordingly.

The *physical row segment grouping* problem is to find a valid partition of row segments S^{k0}... S^{k15} that satisfies:



(a) DrMP enhances MMU with usage flag (b) DrMP flag table is filled in based on usage flag

Figure 5.7: The OS assisted memory management for DrMP.

(1) S^{k0}... S^{k15} is a permutation of row segments S_j.

(2) S^{k0}... S^{k15} are divided into two groups: the first eight form the first new physical row while the next eight form the second row. The row segments in each group have their tWR values sorted, that is, we have $tWR^{k0} \leq tWR^{k1} \leq ... \leq tWR^{k7}$,

and $tWR^{k8} \leq tWR^{k9} \leq ... \leq tWR^{k15}$.

(3) The first physical row is a fast row, i.e., $tWR^{k7} <= tWR^{k15}$.

The optimization goal depends how the row pair is used. For DrMP-P, the goal is to minimize (tWR^{k7} + tWR^{k15}). For DrMP-U, as an example, we want support a mix of approximate and precise computing with the assumption that the first row is for precise computing while the second row is for approximate computing; and half of the data bits are important. The goal is then set to minimize (tWR^{k7} + tWR^{k11}).

After profiling to determine the best row segment tWR values, the OS performs an exhaustive search to find the best row segment grouping for different usage patterns. For a 4GB memory, it takes less than 20 seconds for one pattern, and less than 2 minutes for all five usage patterns. If chip manufacturers conduct post-fabrication test and regrouping, we will need a better heuristic. We leave this to future work.

5.2.5 Precision-aware Memory Management

DrMP-P and DrMP-U couple the usage of paired rows, which brings new constraints on memory allocation. For example, assume rows `i` and `i+K` ($0 \leq i < K$) are paired by DrMP-U such that rows `i` and `i+K` save precise and approximate data, respectively. Row `i`, after being reclaimed by the memory allocator, may not be allocated to store approximate data. This is because storing approximate data in row `i` needs a new M_i while row `i+K` needs the negation of the old M_i to determine data locations, which prevents loading another M_i.

We use Figure 5.7 to illustrate precision-aware memory allocation. For simplicity, we apply DrMP only to allocate *normal* user data. We do not allocate memory space for device drivers or DMA operations. Modern OSes, like Linux, adopt buddy allocation to allocate blocks of consecutive memory pages to user applications. Linux's buddy allocator maintains an array of 11 freelists that link free blocks of 2^l pages ($0 \leq i \leq 10$). A request asking for more than 2^{10} pages is served by multiple blocks from the freelist. Given a 2^l-paged block whose first page address is `x`, its **paired block** is defined as the 2^l-paged block whose first page address is either `x+P/2` (if $0 \leq x < P/2$) or `x-P/2` (if $P/2 \leq x < P$), here P is the total number of pages in the memory.

In DrMP, a memory request specifies not only the size but also the required precision:

void *malloc(int size, char UsageFlag);

Here, *UsageFlag* is a 3-bit flag as shown in Table 5.1. To service this request, DrMP adds a 6-bit usage flag "aaa/bbb" to each block in the freelist. The flag describes the usage of the block and its paired block. That is, a block whose starting address is in the first half of the memory uses "aaa" while its paired block uses "bbb". The bold font in the figure indicates the flag that is actually used by a free block.

When linking a 2^{10}-paged block to the freelist, the OS links its paired block at the same time. The usage flags are initialized for both blocks. DrMP saves the usage flag in the first byte of each free block and this flag is carried to smaller blocks when a large block is split, as shown in Figure 5.7. A valid usage flag is one of the following:

1. "000/000" indicates that the two blocks are used as they are in the baseline. DrMP is

disabled.

2. "111/111" indicates that the two blocks are paired and are used for precise computing only.
3. "111/`aaa`" indicates that the two blocks are paired and used for mixed precision computing. The block with the starting address in the low half of the memory is for precise computing and the paired block (with higher address) is for approximate computing. The flag `aaa` can be one of "001","010","101", and "110", depending on what approximate data to save in the block.
4. "`aaa`/`aaa`" (when `aaa` is neither "000" nor "111") indicates the two blocks are for approximate computing only. They are not paired. `aaa` is set similarly as above.

With DrMP, a memory allocation request is serviced by the memory allocator to provide a block of a matching size and usage flag. In Figure 5.7, the request for one page of "001"-type approximate data is satisfied by the third block in the #0-freelist. The buddy allocator without DrMP extension would return the first block instead.

The OS maintains a DrMP flag table to assist precision-aware restore scheduling. Given each row pair, the table keeps one entry that saves the usage flag, mapping bit vector, and tWR values for both rows, as shown in Figure 5.7.

The OS fills in the DrMP flag table when it updates a corresponding page table entry. The usage flags are extracted from the allocated memory block. Based on the usage flag, the OS loads the mapping bit vectors and tWR values from the grouping results (as described in Section 5.2).

Fragmentation optimization. A concern for memory allocation is that DrMP may increase system fragmentation. In Figure 5.7, a request for a 2^4-paged block for precise computing may not be satisfied even though there is a block with a matching size. The request triggers a bigger block to be split, creating additional small blocks in the system. We next discuss optimization to minimize fragmentation.

DrMP combines compatible usage flags such that the OS may return a block with a compatible (i.e., not exactly the same) usage flag. For the example in Figure 5.7(a), to satisfy the request for a 2^4-paged block with "010" flag, it is acceptable to return a block with

Table 5.2: Evaluated Applications

Application	Type	Input	Quality metric	Approx accesses (%)
kmeans	Machine Learning	Color image	Image difference	45.4%
blackscholes	Financial Analysis	Portfolio options	Avg. price error	6.3%
raytracer	3D Image Renderer	Light, texture, etc	Image diff	4.0%
sor	Scientific Computing	Grid pattern	Mean entry diff	79.5%
lu	Scientific Computing	Dense matrix	Mean entry diff	98.0%
smm	Scientific Computing	Dense matrix	Mean norm. diff	73.5%

"111/001" flag and its starting address is in the second half of the memory. We dynamically alter the usage flag of the block to "111/010". It is safe to do so because either "001" or "010" set the tWR to the fourth fastest row segments, and thus, they share the same mapping bit vectors and tWR values for the row pair.

In addition, DrMP can satisfy the request with "010" flag with a block of a "001" flag. Here, the OS returns a more reliable but slightly slower block.

5.2.6 Architecture Enhancements

Figure 5.8 shows an overview of the architectural enhancements. The light color shaded boxes indicate the enhancements to support DrMP-A and the dark color shaded boxes indicate the additional enhancements to support DrMP-P and DrMP-U.

Space overhead. In the HDD, we save the row segment profiling information that marks the best tRAS and tWR for each row segment. The profile needs 6MB of storage for a 4GB main memory. The OS then applies the row segment grouping algorithm to get the mapping bit vectors and tWR/tRAS values for each different pairing pattern. Given that we have five patterns and each row is of 8KB, we need (8b+6b)×4GB/8KB×5 = 4.4MB HDD space for the mapping bit vectors. The DrMP flag table occupies 1.1MB (= 4GB/8KB×(3b+8b+6b)) space.

Given that the bit flags are for 8KB rows, they show good access locality, i.e., similar to

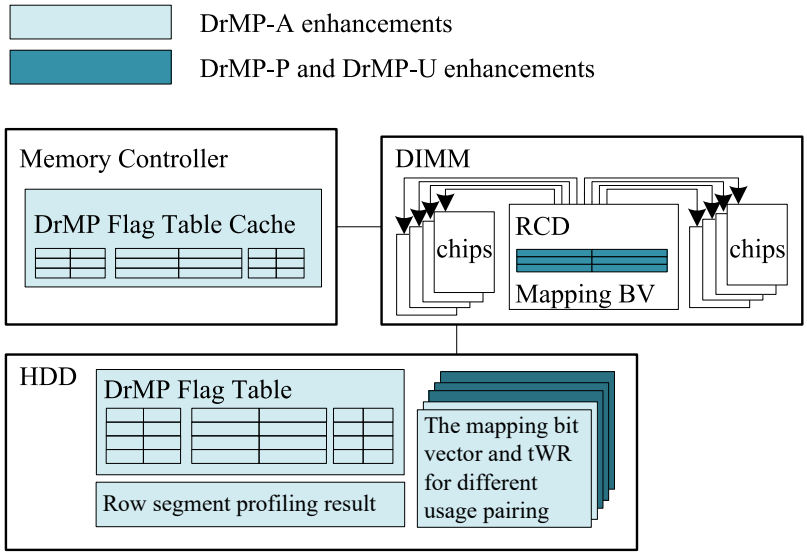


Figure 5.8: An overview of architectural enhancements.

the locality of a TLB buffer. We use a 512-entry on-chip DrMP flag table to buffer the most frequently entries, which requires about 2KB of space. To support DrMP-P and DrMP-U, we add a 256-entry mapping bit vector buffer in each DIMM, which occupies about 512B. The buffer is organized as a direct-mapped cache with tag fields maintained in the memory controller. Overall, the space overhead is modest.

Timing overhead. We used CACTI [HP, 2008] to model timing overhead. DrMP has minimal timing overhead for DrMP-A. It introduces two CPU cycles of extra latency to extract the DrMP flags and shuffle the data based on approximate usage flag. The overhead is added for both read and write accesses because the data need to be remapped between the device layout and logic layout.

DrMP introduces extra latency to pair rows for DrMP-P and DrMP-U. As described in DrMP-P, two extra memory cycles are needed — one cycle determines which chips to activate and the other is due to one cycle delay in sending the device command.

Energy overhead. By introducing chip enable wires, DrMP activates the same number of subarrays as the baseline. We use CACTI to model the flag cache in the memory controller and the map table buffer in the DIMM. The energy overhead is negligible as shown in the

experiments.

5.3 EXPERIMENTAL METHODOLOGY

Approximate computing is an emerging paradigm to trade off performance, energy and quality-of-service (QoS). To evaluate the effectiveness of DrMP on approximate computing, we adopted the two-phase methodology from past work [Miguel et al., 2014, 2016].

In the first phase, we used a Pin-based simulator to instrument programs annotated for approximate computing. The simulator tracks all loads and stores of integer and floating-point variables and, based on the memory map of weak cells (where timing exceeds reliable operation), injects faults into memory operands. The memory map was generated following the model of Zhang et al. [Zhang et al., 2015a], and the timing parameters were aligned with the industrial values [Kang et al., 2014; Son et al., 2014; Wang, 2015]. We integrated the usage flags in the memory map so that different row segment pairing strategies lead to different error rates at runtime. For this phase, we ran the benchmarks to completion and compared the final output with the one from the baseline run (i.e., with no restore errors).

In the second phase, we used a cycle accurate simulator, USIMM [Chatterjee et al., 2012], to compare performance and energy consumption. USIMM executed the instructions tracked in the first phase to make sure the two runs had the same instruction flow. We modeled a 4-core chip multiprocessor following past research [Shin et al., 2014; Nair et al., 2013a]. For the DRAM main memory, we used the Micron SDRAM DDR3 [Micron, 2009a]. Table 5.3 lists the configuration details.

5.3.1 Benchmarks

We selected a suite of benchmark programs that were used in the literature to evaluate approximate computing techniques. As shown in Table 5.2, the benchmarks are from different domains, including machine learning, financial analysis and scientific computing. In addition, we selected two memory intensive applications `libq` and `leslie` from SPEC CPU2006

Table 5.3: System Configuration

Processor	four 3.2Ghz cores; 128 ROB size
	Fetch width: 4, Retire width: 2, Pipeline depth: 10
Memory Controller	Bus frequency: 800 MHz
	Write queue capacity: 64
	Write queue high watermark: 40
	Write queue low watermark: 20
	Address mapping: rw:cl:rk:bk:ch:offset
	Page management policy: close-page with FRFCFS
DRAM	1 channel, 2 ranks/channel, 8 banks/rank,
	32K rows/bank, 8KB/row, 64B cache line
	tCK=1.25ns, width: x8
	tCAS(CL): 13.75ns, tRCD: 13.75ns, tRC: 48.75ns
	tCWD: 6.25ns (5 cycles), tBURST: 5.0ns (4 cycles)
	tRAS: 35ns, tRP: 13.75ns, tFAW: 24 cycles,
	tRRD: 5 cycles, tRFC: 208nCK, tREFI: 7.8μs

[SPEC, 2006]. These two applications always demand precise computing. They are used to form workloads with mixed precision demands.

For the evaluation, we manually annotated the benchmarks to identify the data that can be approximated. This approach is the same as past work [Miguel et al., 2015; Sampson et al., 2011]. Table 5.2 summarizes the percentage of memory accesses that access approximate data.

(a) Precise baseline (0%) **(b)** Approx-base-2 (5.97%) **(c)** DrMP-A-2 (1.98%)

(d) Approx-base-4 (2.16%) **(e)** DrMP-A-4 (0%)

Figure 5.9: Visual effects for approximated runs for `kmeans`.

5.3.2 Evaluation for Approximate Computing

To evaluate QoS, we compared the results from approximate execution to results from the baseline with precise execution and followed prior studies [Sampson et al., 2011; Miguel et al., 2014] to compute application-specific metrics. Traditionally, a flat threshold of 10% error

rate was set as the upper bound [Sampson et al., 2011; Esmaeilzadeh et al., 2012; Miguel et al., 2014; Wong et al., 2016]. However, this error rate often leads to a large deviation [Park et al., 2016; Lee et al., 2016]. For example, `blackscholes` is a financial analysis application from PARSEC 3.0. Its QoS metric is stock/option price difference. A 10% error for a $20 option leads to $2 difference, which is significant and generally unacceptable [Lee et al., 2016]. For this reason, we did not set a fixed percentage threshold. Our design goal is instead to minimize the error rate with a performance improvement.

5.4 RESULTS AND ANALYSIS

We studied and compared the following schemes.

— `Baseline`. This scheme mitigates PRT with fully relaxed restore timing, i.e., `tRAS`=42, `tWR`=25, and `tRCD`=15 [Zhang et al., 2016a]. The baseline adopts built-in spare rows and columns to rescue the worst set of cells. The same timing is applied to all chips.

— `PRT-Free`. This scheme assumes future DRAM chips are free from PRT, and thus use the current JEDEC timing, i.e., `tRAS`=28, `tWR`=12, and `tRCD`=11 [Micron, 2009a].

— `Approx-base-#`. This scheme is baseline approximation schemes without dedicated techniques to protect important bits. # is either 2 or 4, indicating whether the row tWR is set to the $2nd$ or $4th$ fastest row segment.

— `DrMP-A-#`. This scheme is DrMP-A, where rows are being utilized for approximate computing with important data bits being protected.

— `DrMP-P-#`. This scheme is DrMP-P. The paired rows are used to save precise data only.

— `DrMP-U-#`. This scheme is DrMP-U. Given one row pair, the row with the low address saves precise data, while the row with high address saves approximate data.

5.4.1 QoS of Approximate Computing

We first evaluated the effectiveness of our approximate computing strategy. Figure 5.10 compares the QoS of different schemes. `Approx-base-n` uses the tWR of the nth fastest row

segment. It is similar to `DrMP-A-n` except important bits in a row are not mapped. From the figure, we observe that mapping important bits greatly mitigates QoS degradation. For example, mapping reduces 100% QoS degradation of `lu` to 0.31% in `DrMP-4` when ensuring the reliability of four row segments in each row.

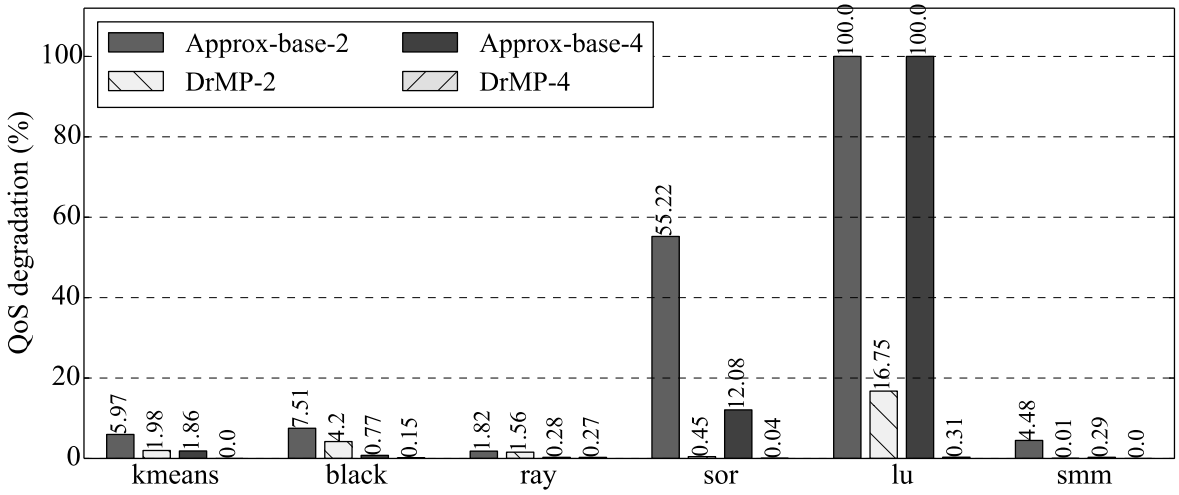


Figure 5.10: QoS degradation in different schemes.

We examined `kmeans` to check the visual effect of the output image, as shown in Figure 5.9. Figure 5.9(a) shows the image with no restore errors. We observe that Figure 5.9(b)-5.9(d) shows a color change — the gray tail of the eagle turns pinkish. In addition, Figure 5.9(b) and 5.9(d) shows visible noise. DrMP-A-4 (Figure 5.9(e)) has no visible change compared to the baseline. Therefore, it is important to reduce the error rate in approximate computing. The bit remapping in DrMP-A is effective in mitigating the QoS degradation.

5.4.2 Performance

Figure 5.11 reports the execution time of different schemes. The results are normalized to `Baseline`. In the figure, `Gmean` is the geometric mean of all workloads. For `DrMP-A` and `DrMP-U`, we compared the schemes when setting the row tWR to the $2nd$ and $4th$ fastest row segment for approximate computing. For `DrMP-P`, we studied two page allocation schemes — `DrMP-P-rand` is the baseline allocation that returns a random page; `DrMP-P-fast` returns

a random fast page first, i.e., it uses all fast pages before allocating slows ones. The latter is slightly worse than an OS-assisted profiling-based page allocation scheme that sorts the available pages according to their restore time and allocates pages based on their access frequency.

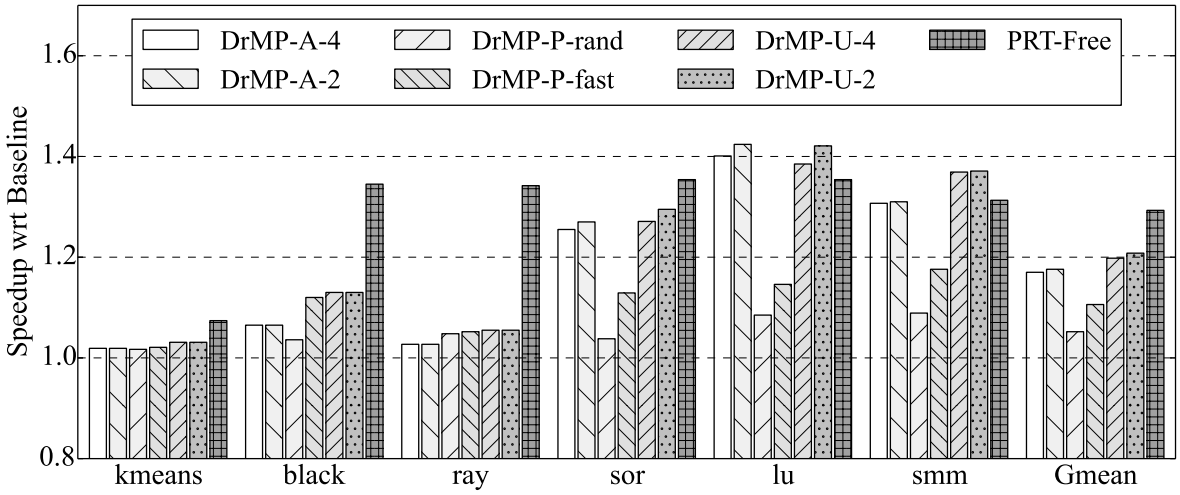


Figure 5.11: Performance comparison.

On average, `DrMP-A-4`, `DrMP-P-fast`, and `DrMP-U-4` have a 17%, 10.2% and 19.8% improvement over the baseline, respectively. Not unexpected, random page mapping in `DrMP-P-rand` lowers the speedup to 5.2%. The difference between `DrMP-A-2` and `DrMP-A-4` (similarly, `DrMP-U-2` and `DrMP-U-4`) is usually small. For applications that have dominant approximate data accesses, such as `lu` and `smm`, DrMP-U does better than `PRT-Free`. This improvement happens because many rows that save approximate data have a tWR faster than the standard 15ns tWR in `PRT-Free`. DrMP-U achieves lower improvement for applications that are less memory intensive with fewer approximate accesses, such as `kmeans` and `raytracer`. Given the moderate difference of performance and the notable contrast in QoS as reported in Figure 5.10, we use the $4th$ fastest row segment in subsequent sections.

5.4.3 Timing Values

Table 5.4 compares the average restore timing in different schemes. `PRT-Free` has the best timing, while `Baseline` has the worst. `Baseline`, `PRT-Free`, `DrMP-A-n` do not use row

pairing, and thus, they have one set of average values. The tRAS of DrMP-A is even better than `PRT-Free` because the timing is aggressively reduced, which introduces restore errors in some row segments. `DrMP-A-2` has better timing than `DrMP-A-4` as it reduces the tWR more aggressively.

Table 5.4: Restore Timing Value of Each Row Pair

Scheme	Low Address Row			High Address Row		
	tRAS	tWR	tRCD	tRAS	tWR	tRCD
	(memory cycles)					
Baseline	42	25	15	same as left		
PRT-Free	28	12	11	same as left		
DrMP-A-2	20	13	15	same as left		
DrMP-A-4	22	15	15	same as left		
DrMP-P	30	20	15	40	24	15
DrMP-U-2	30	20	15	19	12	15
DrMP-U-4	33	21	15	20	13	15

In DrMP-P, each row pair has a fast row and a slow row. The average of slow rows in DrMP-P is close to `Baseline` as the slowest row segments remain in the memory. The average of approximate rows in `DrMP-U-n` is close to `DrMP-A-n`, indicating that `DrMP-U` makes better use of device rows. `DrMP-P` and `DrMP-U` report similar timings for low address rows because `DrMP-U` exploits mainly the fast row segments in the slow rows.

5.4.4 Energy Consumption

Figure 5.12 reports memory energy consumption in terms of background (`bg`), active/precharge (`act/pre`), read/write (`rd/wr`) and refresh (`ref`). We followed the Micron power equations and parameters [Micron, 2007, 2009a]. We used CACTI to model the DrMP flag cache in the memory controller and in the DIMM. We observed that, by improving application performance, the DrMP schemes reduce the background energy the most. Overall, `DrMP-U-4` achieves 15% energy consumption reduction, which is within 7% gap of `PRT-Free`. The

primary contributor is the reduced background energy because of the shortened execution time. Read/write power/energy is also optimized with the reduction in the restore time.

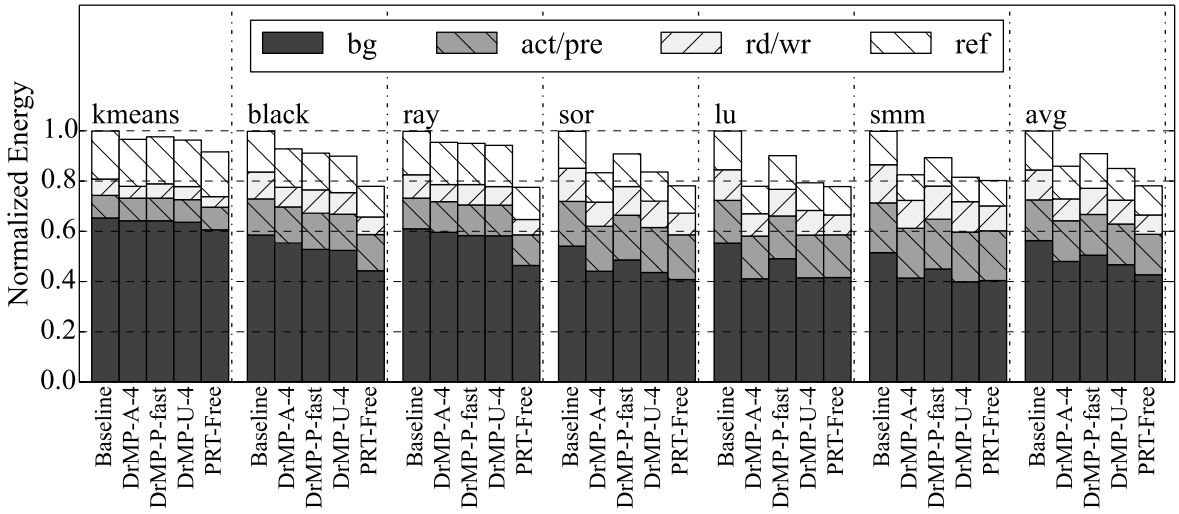


Figure 5.12: Energy comparison.

5.4.5 System Overhead

In DrMP, the 512-entry flag cache in the memory controller is organized as 4-way set associative cache. On average, it has 97.8% hit rate. The CACTI simulation shows that the cache has 0.35ns access latency, 0.02mm^2 area, 1.5mW standby leakage power and 5.45pJ energy per access. The DIMM mapping table is organized as a 256-entry direct-mapped cache with tags maintained in the memory controller. It has an average 97.9% hit rate. The CACTI simulation shows that this structure has 0.22ns access time, 0.016mm^2 area, 1.14mW standby leakage and 2.96pJ energy per access.

Frequently used flag cache entries are captured in the L2 cache and loaded to the flag cache and mapping table on misses of these structures. The performance overhead is less than 1% in our simulation.

Due to limitations of our evaluation framework, we did not evaluate the overhead of the buddy memory allocation. We expect it to be low due to its invocation, i.e., to allocate blocks of consecutive pages to processes, is less frequent per process.

5.4.6 Design Space Exploration

5.4.6.1 Server integration. We studied using `DrMP` in a server environment that has more applications for precise computing. In this study, 2 cores run applications from Table 5.3 and the remaining 2 cores run `libq` and `leslie` from SPEC [SPEC, 2006]. This workload mix leads to a larger percentage of precise memory accesses. From Figure 5.13, `DrMP-P` achieved higher speedups while `DrMP-A` had lower improvement. `DrMP-A` and `DrMP-P-fast` have 14.8% and 12.6% improvements, changing from 17.0% and 10.6% improvements in Figure 5.11, respectively. The decrease in `DrMP-A` is caused by an increasing number of precise accesses, and the increase in `DrMP-P` is caused by touching more fast precise rows. Fortunately, `DrMP-U` reaches a balance to maintain similar speedup values, show it is a general scheme.

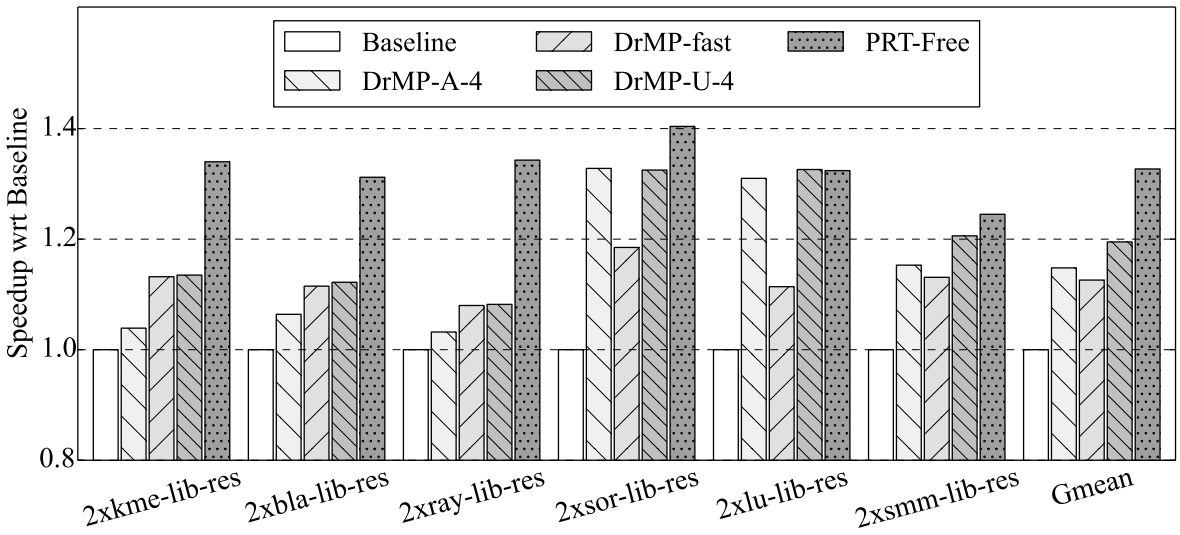


Figure 5.13: Performance comparison of server workload mix.

5.4.6.2 Integration of the state-of-the-art. *Restore truncation* (RT) [Zhang et al., 2016a] is a recent PRT mitigation approach. It partially restores memory cells to a low voltage level, depending on the distance of an access to the next row refresh. Since DrMP exploits error resilience through approximate computing, these two designs are orthogonal. Figure 5.14 reports the results when both schemes are adopted (RT+DrMP). From the figure,

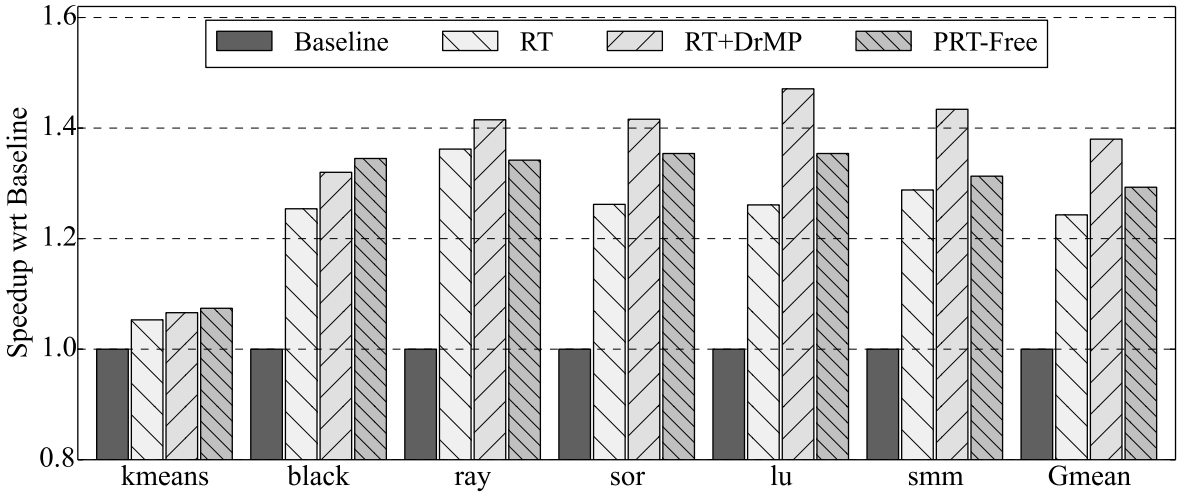


Figure 5.14: Performance comparison of DrMP and RT [Zhang et al., 2016a].

RT+DrMP achieves a larger performance improvement over RT: on average, RT+DrMP is 13.7% better than RT, and 8.7% better than `PRT-Free`.

5.5 CONCLUSION

This chapter proposes DrMP, a novel fine-grained precision-aware DRAM restore scheduling approach, to mitigate PRT. The approach exploits process variations (PVs) within and across DRAM rows to save data with mixed precision. There are three variants of the approach: DrMP-A, DrMP-P, and DrMP-U. DrMP-A supports approximate computing by mapping important data bits to fast row segments to reduce restore time for improved performance at a low application error rate. DrMP-P pairs memory rows together to reduce the average restore time for precise computing. DrMP-U combines DrMP-A and DrMP-P to better trade performance, energy consumption, and computation precision. Our experimental results show that, on average, DrMP achieves 20% performance improvement and 15% energy reduction over a precision-oblivious baseline. Further, DrMP achieves an error rate less than 1% at the application level for a suite of benchmarks, including applications

that exhibit unacceptable error rates under simple approximation that does not differentiate the importance of different bits.

6.0 CONCLUSIONS AND FUTURE WORK

6.1 SUMMARY

DRAM technology scaling has reached a threshold where physical limitations exert unprecedented hurdles on cell behaviors. Without dedicated mitigations, memory is expected to suffer from serious performance loss, yield degradation and reliability decrease, going against the demands of system designs and applications. Among the induced problems, prolonged restore timing (PRT) has been a long time neglected issue, and it is likely to impose great constraints to the scaling advancement. As a result, this thesis explores DRAM further scaling from restoring perspective.

Reduced cell dimensions and worsening process variation (PV) cause increasingly slow access and significantly more outliers falling beyond the JEDEC specifications. To quantify the PV effects, we built circuit level SPICE models and performed Monte Carlo simulations to capture the restore distribution parameters. With the obtained parameters, we then faithfully generated billions of cells, which are organized into arrays, banks, chips, and ranks following the standard DRAM organizations. On basis of the studies, we examined the high-level organizations, detailed cell behaviors, and slow cell portions to come up with different techniques to overcome the restore issues.

From high organization level, to alleviate the influences on performance and yield, we propose to go beyond the conventional worse-case determined techniques, and instead manage the timing constraints per chip. As further improvement, we chop the chip banks into finer chunks, and thus more fast regions can be exposed to upper level. In addition, we devise the extra chunk remapping and dedicated rank formation to restrict the impacts of slow cells. However, the exposed fast regions can not be fully utilized for restoring oblivious

page allocation; accordingly, to maximize performance gains, we moved forward to profile the pages of the workloads, and deliberately allocate the hot pages to fast parts.

Examining the cell structure and behaviors, we find that restoring can seek help from the correlated refresh operations, which periodically fully charge the cells. We propose to perform partial restore withe respect to the distance to next refresh: the closer to next refresh, the less needed charge and thus the earlier the restore operation can be terminated. For ease of implementation, we divide the refresh window into four sub ones, and apply an separate set of timings for each. Moreover, compared to refresh, restore contributes more critically to the overall performance, and hence we optimize the partial restore with refresh rate upgrading. More frequent refresh helps to lower the restoring objectives, but at a risk of raising energy consumption. As a compromise, we selectively upgrade recent touched rows only to balance restore benefits and refresh overheads.

While PRT issue can be greatly mitigated with chunk remapping and partial restore, the gains can be even higher when application characteristics are being taken into account. By applying approximate computing, we devised further scaling DRAMs to provide both reliable, i.e., precise, and non-reliable, i.e., approximate, data regions; correspondingly, we analyzed the source codes of applications to annotate variables that can be approximated, i.e., non-critical. With the help from hardware enhancements and operating system changes, the non-critical data bits can be dedicatedly mapped to non-reliable regions to avoid severe accuracy losses on final output. As a result, we can achieve a tradeoff between performance/energy and accuracy.

Detailed experimental results show that with the proposed techniques, both performance and energy can be greatly improved to alleviate the slow restore issue. Our comprehensive techniques to solve restore issues, together with the prior arts on refresh optimization and sensing reductions, are capable of enabling DRAM to scale further without significant obstacles.

6.2 FUTURE RESEARCH DIRECTIONS

Whereas a comprehensive set of techniques have been discussed to overcome restore scaling issues, it is still open to have deeper understandings of the problems and propose more effective designs from new perspectives. This section describes potential research opportunities to further solve PRT issue.

6.2.1 Solve Restoring from Reliability Perspective

DRAM reliability has become a major concern for servers in data centers, as reported in recent in-field studies [Schroeder and Gibson, 2006; Sridharan and Liberty, 2012]. This has motivated the wide adoption of chipkill protection over traditional SECDED (sing error correction double error detection) protection. To further improve the correction capability and reduce storage overheads, a bunch of smart designs have been proposed, e.g., V-ECC [Yoon and Erez, 2010], LOT-ECC [Udipi et al., 2012] and Bamboo [Kim et al., 2015].

Whereas correction codes are mainly targeting at permanent faults (e.g., broken address decoder and damaged data pins) and transient faults (e.g., cosmic rays), long restore cells can be also treated as stuck bits to be covered by correction codes, similar to [Schechter et al., 2010; Nair et al., 2013b; Qureshi et al., 2015]. However, the issue is that there is a too large portion of long restore cells, and thus it requires complex coding/decoding logics and high storage overheads. To avoid the induced impacts, one workable solution can be applying ECC on top of our proposed mitigation techniques. Apparently, the low portion of slow cells can now be well covered by ECC, and thus the restore timing constraints can be further reduced. It is also possible to combine ECC with the aforementioned techniques to keep the same benefits but with smaller overheads. In either way, innovative ECC designs can be proposed or adapted in the restore scenario.

6.2.2 Study Security Issues of Restoring Variation

Modern computing systems suffer from increasing concerns on privacy, security and trust issues, with timing channel attacks as a representative example. And recently, the concern on

timing channel attack has moved from shared caches [Percival, 2005; Wang and Lee, 2007; Liu and Lee, 2013] and on-chip networks [Wang and Suh, 2012; h. Wassel et al., 2013] to shared main memory [Stefanov et al., 2013; Wang et al., 2014b; Shariee et al., 2015]. Memory access pattern can leak a significant amount of sensitive information through statistical inference [Stefanov et al., 2013].

As a remedy, ORAM [Stefanov et al., 2013] was proposed to conceal a client's access pattern to remote storage by continuously shuffling and re-encrypting data as they are accessed. Wang et al. [2014b] proposed temporal partitioning (TP) to isolate thread accesses to hide access pattern. As an improvement, Fixed Service (FS) policies were studied by [Shariee et al., 2015] to reshape memory access behaviors without much performance degradation.

Compared to the simple case of a single set of timings for the whole memory system, restoring variations in further scaling DRAM are likely to leak more information. For instance, various memory access speeds to different memory regions may expose the footprint to malicious users. And things can be much worse with the adoption of NUMA-aware page allocation and approximate computing. The former easily leak the frequently accessed data, and the latter correlates data to its location origin [Rahmati et al., 2015].

In addition, simply borrowing the schemes in [Wang et al., 2014b; Shariee et al., 2015] would introduce higher overhead because of the much longer worst-case restoring timings. As a result, it is necessary to integrate information leakage, restoring issues, page allocation and approximate computing to come out a workable solution with acceptable performance loss and safety guarantee.

6.2.3 Explore Restoring in 3D Stacked DRAM

Recent advances in die stacking techniques enables efficient integration of logic and memory dies in a single package, with a concrete example of Hybrid Memory Cube [Consortium, 2015]. HMC is especially promising for its innovative architecture that stacks multiple memory dies atop of the bottom logic die, and adopts packetized serial link interface to transfer data and requests [Zhang et al., 2015b]. With the superior high bandwidth, low latency and packet-based interface, lots of work have proposed to move computation units inside the logic die

[Balasubramonian et al., 2014; Ahn et al., 2015].

However, thermal management becomes a big issue in stacked memories [Loi et al., 2006; Eckert et al., 2014], and the deployment of bottom computation logics like simple cores [Ahn et al., 2015] and even GPU [Zhang et al., 2014a] further worsens the issue. Besides, temperature variations exist among vertical dies [Khurshid and Lipasti, 2013]. It is known that DRAM is sensitive to temperature changes, including refresh [Lee et al., 2015; Mukundan et al., 2013] and restoring time [Son et al., 2014; Kang et al., 2014]. Therefore, it is worthwhile to explore restoring time in stacked memories, and utilize the temperature characteristics to dig more opportunities to mitigate restore and to boost performance.

BIBLIOGRAPHY

The SAP HANA database. https://www.sap.com/products/hana.html.

Agrawal, A., Ansari A., and Torrellas J. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules. In new roman.fd New Roman.fd International Symposium on High Performance Computer Architecture (HPCA), pages 84–95, 2014.

Ahn, J., Jouppi N., Kozyrakis C., Leverich J., and Schreiber R. Future scaling of processor-memory interfaces. In The International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 1–12, 2009.

Ahn, J., Yoo S., Mutlu O., and Choi K. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 336–348, 2015.

Ahn, J. H., Leverich J., Schreiber R. S., and Jouppi N. P. Multicore DIMM: an energy efficient memory module with independently controlled DRAMs. IEEE Computer Architecture Letters (CAL), 2008.

Arnab, R., Jayakumar H., Sutar S., and Raghunathan V. Quality-aware data allocation in approximate DRAM. In IEEE Conference on Automation Science and Engineering (CASES), pages 89–98, 2015.

Ayoub, R., Nath R., and Rosing T. S. CoMETC: Coordinated management of energy/thermal/cooling in servers. ACM Transactions on Design Automation of Electronic Systems (TODAES), 19(1):1–28, 2013.

Balasubramonian, R., Chang J., Manning T., Moreno J. H., Murphy R., Nair R., and Swanson S. Near-data processing: Insights from a MICRO-46 workshop. IEEE Micro, pages 36–42, 2014.

Bhati, I., Chang M., Chishti Z., Lu S., and Jacob B. DRAM refresh mechanisms, penalties, and trade-offs. IEEE Transactions on Computers (TC), 64, 2015a.

Bhati, I., Chishti Z., Lu S.-L., and Jacob B. Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 235–246, 2015b.

Bhattacharjee, A. and Martonosi M. Thread criticality predictors for dynamic performance,

power and resource management in chip multiprocessors. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 290–301, 2009.

Chandrasekar, K., Goossens S., Weis C., Koedam M., Akesson B., Wehn N., and Goossens K. Exploiting expendable process-margins in DRAMs for run-time performance optimization. In IEEE Conference on Design, Automation and Test in Europe (DATE), pages 1–6, 2014.

Chatterjee, N., Balasubramonian R., Shevgoor M., Pugsley S. H., Udipi A. N., Shafiee A., Sudan K., Awathi M., and Chishti Z. USIMM: the utah simulated memory module. Technical report, Univ of Utah, 2012.

Chen, X. and Peh L.-S. Leakage power modeling and optimization in interconnection networks. In IEEE International Symposium on Low Power Electronics and Design (ISLPED), pages 90–95, 2003.

Choi, J., Shin W., Jang J., Suh J., Kwon Y., Moon Y., and Kim L.-S. Multiple clone row DRAM: a low latency and area optimized DRAM. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 223–234, 2015.

Consortium, Hybrid Memory Cube. Hybrid memory cube specification 2.0. http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf, 2015.

D. Lee, Y. Kim, Seshadri V., Liu J., Subramanian L., and Mutlu O. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In International Symposium on High Performance Computer Architecture (HPCA), pages 615–626, 2013.

Demone, P. High speed DRAM architecture with uniform access latency. United States Patent, US8045413 B2, Oct 25, 2011.

Diamos, G., Sengupta S., Catanzaro B., Chrzanowski M., Coates A., Elsen E., Engel J., Hannun A., and Satheesh S. Persistent RNNs: Stashing recurrent weights on-chip. pages 2024–2033, 2016.

DoE, . Exacale computing initiative review. Technical report, ASCAC, Department of Energy, August 2015.

Eckert, Y., Jayasena N., and Loh G. H. Thermal feasibility of die-stacked processing in memory. In Workshop on Near-Data Processing (WoNDP), 2014.

Esmaeilzadeh, H., Sampson A., Ceze L., and Burger D. Neural acceleration for general-purpose approximate programs. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 449–460, 2012.

Ghosh, M. and Lee H.-H. S. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 134–145, 2010.

Guo, Q., Strauss K., Ceze L., and Malvar H. S. High-density image storage using approximate memory cells. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 413–426, 2016.

Wassel, h. , Gao Y., Oberg J. K., Huffmire T., Kastner R., Chong F. T., and Sherwood T. SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 583–594, 2013.

Hamamoto, T., Sugiura S., and Sawada S. On the retention time distribution of dynamic random access memory (DRAM). IEEE Transactions on Electron Devices (TED), 45(6): 1300–1309, 1998.

Hong, S., Kim S., Wee J.-K., and Lee S. Low-voltage DRAM sensing scheme with offset-cancellation sense amplifier. IEEE Journal of Solid-State Circuits (JSSC), 37(10), 2002.

HP, . CACTI 5.3: An integrated cache timing, power and area model. http://www.hpl.hp.com/research/cacti/, 2008.

Hu, C. C. Modern semiconductor devices for integrated circuits. Prentic Hall, 2009.

Hynix, . 2Gb DDR3 SDRAM data sheet. Technical report, Hynix Semiconductor, 2010.

Iwai, H. Roadmap for 22nm and beyond. Microelectronic Engineering, 86(7-9):1520–1528, 2009.

J. Cong, B. Liu, W. Jiang and Zou Y. Automatic memory partitioning and scheduling for throughput and power optimization. ACM Transactions on Design Automation of Electronic Systems (TODAES), 16(2):1–25, March 2011.

Jacob, B., Ng S., and Wang D. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann, 2007.

JEDEC, . Double data rate (DDR) SDRAM specification, 2000.

JEDEC, . DDR2 SDRAM specification, 2009a.

JEDEC, . DDR3 SDRAM specification, 2009b.

JEDEC, . Definition of the SSTE32882 registering clock driver, December 2009c.

JEDEC, . DDR4 SDRAM, 2012.

Jeon, H., Kim Y.-B., and Choi M. Standby leakage power reduction technique for nanoscale CMOS VLSI systems. IEEE Transactions on Instrumentation and Measurement (TIM), 59(5), 2010.

Jevdjic, D., Loh G. H., Kaynak C., and Falsafi B. Unison cache: A scalable and effective die-stacked DRAM cache. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 25–37, 2014.

Jiang, B., Sudhama C., Khamankar R., Kim J., and Lee J. C. Effects of nonlinear storage capacitor on DRAM read/write. IEEE Electron Device Letters (EDL), 15(4), 1994.

Kang, U., Yu H.s. , Park C., Zheng H., Halbert J., Bains K., Jang S., and Choi J. S. Co-architecting controllers and DRAM to enhance DRAM process scaling. In The Memory Forum, 2014.

Karnik, T., Borkar S., and De V. Statistical design for variation tolerance: Key to continued Moore's law. In IEEE International Conference on Integrated Circuit Design and Technology (ICICDT), pages 175–176, 2004.

Kaseridis, D., Stuecheli J., and John L. K. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 24–35, 2011.

Keeth, B., Baker R. J., Johnson B., and Lin F. DRAM circuit design: fundamental and high-speed topics. Wiley-IEEE Press, 2007.

Khan, S., Lee D., Kim Y., Alameldeen A. R., Wilkerson C., and Mutlu O. The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study. In ACM SIGMETRICS Conference (SIGMETRICS), pages 519–532, 2014.

Khudia, D. S, Zamirai B., Samadi M., and Mahlke S. Rumba: An online quality management system for approximate computing. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 554–566, 2015.

Khurshid, M. J. and Lipasti M. Data compression for thermal mitigation in the hybrid memory cube. In IEEE International Conference on Computer Design (ICCD), pages 185–192, 2013.

Kim, J., Sullivan M., and Erez M. Bamboo ecc: Strong, safe, and flexible codes for reliable computer memory. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 101–112, 2015.

Kim, K. and Lee J. A new investigation of data retention time in truly nanoscaled DRAMs. IEEE Electron Device Letters (EDL), 30(8):846–848, 2009.

Kim, Y., Daly R., Kim J., Fallin C., Lee J. H., Lee D., Wilkerson C., Lai K., and Mutlu O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 361–372, 2014.

Kirihata, T., Watanabe Y., Wong H., DeBrosse J. K., Yoshida M., Kato D., Fujii S., Wordeman M. R., Poechmueller P., Parke S. A., and Asso Y. Fault-tolerant designs for 256 Mb DRAM. IEEE Journal of Solid-State Circuits (JSSC), 31(4):558–566, April 1996.

Koren, I. and Krishna C. M. Fault-tolerant systems. Morgan Kaufmann, 2010.

Kultursay, E., Kandemir M., Sivasubramaniam A., and Mutlu O. Evaluating STT-RAM as an energy-efficient main memory alternative. In International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 256–267, 2013.

Kurinec, S. K. and Iniewski K. Nanoscale semiconductor memories: Technology and applications (devices, circuits, and systems). CRC Press, 2013.

Lee, B. C., Ipek E., Mutlu O., and Burger D. Architecting phase change memory as a scalable DRAM alternative. In International Symposium on High Performance Computer Architecture (HPCA), pages 2–13, 2009a.

Lee, C. J., Narasiman V., Mutlu O., and Patt Y. Improving memory Bank-Level Parallelism in the presence of prefetching. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 327–336, 2009b.

Lee, D., Choi J., Kim J., Noh S. H., Min S. L., Cho Y., and Kim C. S. LRUF: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Transactions on Computers (TC), 50(12):1352–1361, 2001.

Lee, D., Kim Y., Pekhimenko G., Khan S., Seshadri V., Chang K., and Mutlu O. Adaptive-latency DRAM: Optimizing DRAM timings for the common-case. In International Symposium on High Performance Computer Architecture (HPCA), pages 489–501, 2015.

Lee, S., Boroujerdian B., John L. K., and Gerstlauer A. Synthesis of quality configurable systems. 2016.

Lindholm, E., Nickolls J., Oberman S., and Montrym J. NVIDIA tesla: A unified graphics and computing architecture. IEEE Micro, 28(2):1520–1528, 2008.

Liu, F. and Lee R. Security testing of a secure cache design. In Workshop on Hardware and Architectural Support for Security and Privacy (HASP), 2013.

Liu, J., Jaiyen B., Veras R., and Mutlu O. RAIDR: Retention-aware intelligent DRAM refresh. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 1–12, 2012a.

Liu, J., Jaiyen B., Kim Y., Wilkerson C., and Mutlu O. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 60–71, 2013.

Liu, L., Cui Z., Xing M., Bao Y., Chen M., and Wu C. A software memory partition approach for eliminating bank-level interference in multicore systems. In International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 367–376, 2012b.

Liu, S., Pattabiraman K., Moscibroda T., and Zorn B. G. Flikker: saving DRAM refresh-power through critical data partitioning. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 213–224, 2011.

Loi, G. L., Agrawal B., Srivastava N., Lin S., Sherwood T., and Banerjee K. A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy. In ACM Annual Design Automation Conference (DAC), pages 991–996, 2006.

Lucas, J., Alvarez-Mesa M., Andersch M., and Juurlink B. Sparkk: Quality-scalable approximate storage in DRAM. In The memory forum, pages 1–6, 2014.

Mahajan, D., Park J., Amaro E., Sharma H., Yazdanbakhsh A., Kim J. K., and Esmaeilzadeh H. TABLA: A unified template-based framework for accelerating statistical machine learning. In International Symposium on High Performance Computer Architecture (HPCA), pages 14–26, 2016.

Micron, . TN-46-12: Mobile dram power-saving features/calculations. Technical report, Micron Technology, 2001.

Micron, . TN-41-01: Calculating memory system power for DDR3. Technical report, Micron Technology, 2007.

Micron, . TN-46-14: Hardware tips for point-to-point system design: Termination, layout, and routing. Technical report, Micron Technology, June 2008.

Micron, . 4Gb DDR3 SDRAM - MT41J512M8. Technical report, Micron Technology, 2009a.

Micron, . TN-04-55: DRAM module form factors. Technical report, Micron Technology, March 2009b.

Miguel, J. S., Badr M., and Jerger N. E. Load value approximation. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 127–139, 2014.

Miguel, J. S., Albericio J., Moshovos A., and Jerger N. E. Doppelganger: A cache for approximate computing. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 50–61, 2015.

Miguel, J. S., Albericio J., Jerger N. E., and Jaleel A. The bunker cache for spatio-value approximation. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1–12, 2016.

MSC, . (2012) memory scheduling championship (msc). http://www.cs.utah.edu/~rajeev/jwac12/, 2012.

Mukundan, J., Hunter H., Kim K.h. , Stuecheli J., and Martinez J. F. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 48–59, 2013.

Mutlu, O. and Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 63–74, 2008.

Nair, P. J., Chou C.-C., and Qureshi M. K. A case for refresh pausing in DRAM memory systems. In International Symposium on High Performance Computer Architecture (HPCA), pages 627–638, 2013a.

Nair, P. J., Kim D.-H., and Qureshi M. K. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 72–83, 2013b.

Ozturk, O. and Kandemir M. ILP-based energy minimization techniques for banked memories. ACM Transactions on Design Automation of Electronic Systems (TODAES), 13(2): 1–40, July 2008.

Park, J., Amaro E., Mahajan D., Thwaites B., and Esmailzadeh H. AXGAMES: Towards crowdsourcing quality target determination in approximate computing. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 623–636, 2016.

Patterson, D. Past and future of hardware and architecture. In ACM Symposium on Operating Systems Principles (SOSP), pages 1–10, 2015.

Patterson, D. A. and Hennessy J. L. Computer organization and design: The hardware / software interface. Morgan Kaufmann, 2008.

Percival, C. Cache missing for fun and profit. In DSDCan, 2005.

Qureshi, M. K., Kim D.-H., Khan S., Nair P. J., and Mutlu O. AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems. In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 427–437, 2015.

Rahman, M., Childers B. R., and Cho S. COMeT+: Continuous online memory testing with multi-threading extension. IEEE Transactions on Computers (TC), 63(Issue 7):1668 – 1681, 2014.

Rahmati, A., Hicks M., Holcomb D. E., and Fu K. Probable cause: the deanonymizing effects of approximate DRAM. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 604–615, 2015.

Ramos, L. E., Gorbatov E., and Bianchini R. Page placement in hybrid memory systems. pages 85–95, 2011.

Ryan, J. F. and Calhoun B. H. Minimizing offset for latching voltage-mode sense amplifiers for sub-threshold operation. In International Symposium on Quality Electronic Design (ISQED), pages 127–132, 2008.

Sampson, A., Dietl W., Fortuna E., Gnanapragasam D., Ceze L., and Grossman D. EnerJ: approximate data types for safe and general low-power computation. In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 164–174, 2011.

Sampson, A., Nelson J., Strauss K., and Ceze L. Approximate storage in solid-state memories. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 25–36, 2013.

Samsung, . 20nm 8Gb DDR4 DRAM. http://www.samsung.com/global/business/semiconductor/minisite/Greenmemory/news-event/in-the-news/detail?contentsSeq=13782&contsClassCd=G.

Sarangi, S. R., Greskamp B., Teodorescu R., Nakano J., Tiwari A., and Torrellas J. VARIUS: A model of process variation and resulting timing errors for microarchitects. IEEE Transactions on Semiconductor Manufacturing (T-SM), 21(1):3–13, Feb 2008.

Schechter, S., Loh G. H., Strauss K., and Burger D. Use ECP, not ECC, for hard failures in resistive memories. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 141–152, 2010.

Schmoll, F., Heinig A., Marwedel P., and Engel M. Improving the fault resilience of an h.264 decoder using static analysis methods. In ACM Transactions on Embedded Computing Systems (TECS), volume 13, 2013.

Schroeder, B. and Gibson G. A. A large-scale study of failures in high-performance computing systems. In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 249–258, 2006.

Seshadri, V., Kim Y., Fallin C., Lee D., Ausavarungnirun R., Pekhimenko G., Luo Y., Mutlu O., Gibbons P. B., Kozuch M. A., and Mowry T. C. RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 185–197, 2013.

Shariee, A., Gundu A., Shevgoor M., Balasubramonian R., and Tiwari M. Avoiding information leakage in the memory controller with fixed service policies. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 89–101, 2015.

Shin, W., Yang J., Choi J., and Kim L.-S. NUAT: A non-uniform access time memory controller. In International Symposium on High Performance Computer Architecture (HPCA), pages 464–475, 2014.

Sinha, S., Yeric G., Chandra V., Cline B., and Can Y. Exploiting sub-20nm FinFET design with predictive technology models. In ACM Annual Design Automation Conference (DAC), pages 283–288, 2012.

Son, J., Kang U., Park C., and Seo S. Memory device with relaxed timing parameter according to temperature, operating method thereof, and memory controller and memory system using the memory device. United States Patent, US20140359242 A1, Dec 04, 2014.

Son, Y. H., Seongil O., Ro Y., Lee J. W., and Ahn J. H. Reducing memory access latency with asymmetric DRAM bank organizations. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 380–391, 2013.

SPEC, . SPEC CPU2006. http://www.spec.org/, 2006.

Sridharan, V. and Liberty D. A study of DRAM failures in the field. In The International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 1–11, 2012.

Sridharan, V., Stearley J., DeBardeleben N., Blanchard S., and Gurumurthi S. Feng shui of supercomputer memory positional effects in DRAM and SRAM faults. In The International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 1–11, 2013.

Stefanov, E., Dijk E.v. , Shi E., Fletcher C., Ren l. , Yu X., and Devadas S. Path ORAM: an extremely simple oblivious RAM protocol. In ACM SIGSAC conference on Computer & communications security (CCS), pages 229–310, 2013.

Stuecheli, J., Kaseridis D., Hunter H. C., and John L. K. Elastic refresh: Techniques to mitigate refresh penalties in high density memory. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 375–384, 2010.

Su, Chin-Lung, Yeh Yi-Ting, and Wu Chengwen . An integrated ECC and redundancy repair scheme for memory reliability enhancement. In IEEE International Symposium on

Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pages 81–89, 2005.

Udipi, A. N., Muralimanohar N., Balsubramonian R., Davis A., and Jouppi N. P. LOT-ECC: localized and tiered reliability mechanisms for commodity memory systems. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 285–296, 2012.

Uyemura, J. P. CMOS logic circuit design. Kluwer Academic Publisher, 2002.

Goor, A.J.van de and Tlili I.B.S. March tests for word-oriented memories. In IEEE Conference on Design, Automation and Test in Europe (DATE), pages 501–509, 1998.

Venkatesan, R.K., Herr S., and Rotenberg E. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM. In International Symposium on High Performance Computer Architecture (HPCA), pages 157–167, 2006.

Vijayarahavan, T., Eckert Y., Loh G. H., and others . Design and analysis of an APU for exascale computing. In International Symposium on High Performance Computer Architecture (HPCA), pages 1–12, 2017.

Vogelsang, T. Understanding the energy consumption of dynamic random access memories. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 363–374, 2010.

Wang, D. Backward compatible dynamic random access memory device and method of testing therefor. United States Patent, US9123441 B1, Sep 1, 2015.

Wang, J., Dong X., and Xie Y. Proactive DRAM: A DRAM-initiated retention management scheme. In IEEE International Conference on Computer Design (ICCD), 2014a.

Wang, Y. and Suh G. E. Efficient timing channel protection for on-chip networks. In IEEE/ACM International Symposium on Networks-on-Chip (NOCS), pages 142–151, 2012.

Wang, Y., Ferraiuolo A., and Suh G. E. Timing channel protection for a shared memory controller. In International Symposium on High Performance Computer Architecture (HPCA), pages 225–236, 2014b.

Wang, Y., Han Y., Wang C., Li H., and Li X. RADAR: A case for retention-aware DRAM assembly and repair in future FGR DRAM memory. In ACM Annual Design Automation Conference (DAC), pages 1–6, 2015.

Wang, Z. and Lee R. New cache designs for thwarting software cache-based side channel attacks. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 494–505, 2007.

Wilkes, M. V. The memory gap and the future of high performance memories. ACM SIGARCH Computer Architecture News (CAN), 29(1), 2001.

Wong, D., Kim N. S., and Annavaram M. Approximating warps with intra-warp operand value similarity. In International Symposium on High Performance Computer Architecture (HPCA), pages 176–187, 2016.

Wong, K., Parries P. C., Wang G., and Iyer S. S. Analysis of retention time distribution of embedded DRAM - a new method to characterize across-chip threshold voltage variation. In IEEE International Test Conference (ITC), pages 1–7, 2008.

Yoon, D. H. and Erez M. Virtualized and flexible ECC for main memory. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 397–408, 2010.

Yoon, D. Y., Jeong M. K., and Erez M. Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 295–306, 2011.

Zhang, D., Jayasena N., Lyashevsky A., Greathouse J. L., Xu L., and Ignatowski M. TOP-PIM: throughput-oriented programmable processing in memory. In ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), pages 85–98, 2014a.

Zhang, T., Chen K., Xu C., Sun G., Wang T., and Xie Y. Half-DRAM: a high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. In ACM/IEEE International Symposium on Computer Architecture (ISCA), pages 349–360, 2014b.

Zhang, X., Zhang Y., Childers B. R., and Yang J. Exploiting DRAM restore time variations in deep sub-micron scaling. In IEEE Conference on Design, Automation and Test in Europe (DATE), pages 477–482, 2015a.

Zhang, X., Zhang Y., and Yang J. DLB: Dynamic lane borrowing for improving bandwidth and performance in hybrid memory cube. In IEEE International Conference on Computer Design (ICCD), pages 133–140, 2015b.

Zhang, X., Zhang Y., Childers B. R., and Yang J. Restore truncation for performance improvement for future DRAM systems. In International Symposium on High Performance Computer Architecture (HPCA), pages 543–554, 2016a.

Zhang, X., Zhang Y., Childers B. R., and Yang J. Exploiting DRAM restore time variations in deep sub-micron scaling. In ACM International Symposium on Memory Systems (MEMSYS), pages 322–324, 2016b.

Zhang, X., Zhang Y., Childers B. R., and Yang J. Restore truncation for performance improvement for future DRAM systems. In International Conference on Parallel Architectures and Compilation Techniques (PACT). to be published, 2017a.

Zhang, X., Zhang Y., Childers B. R., and Yang J. On the restore time variations of future dram memory. ACM Transactions on Design Automation of Electronic Systems (TODAES), 22(2):1–24, 2017b.

Zheng, H., Lin J., Zhang Z., Gorbatov E., David H., and Zhu Z. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 210–221, 2008.