ADAPTIVE AND POWER-AWARE FAULT TOLERANCE FOR FUTURE EXTREME-SCALE COMPUTING

by

Xiaolong Cui

Bachelor of Engineering

Xi'an Jiaotong University

2012

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

2016

ADAPTIVE AND POWER-AWARE FAULT TOLERANCE FOR FUTURE EXTREME-SCALE COMPUTING

Xiaolong Cui, PhD

University of Pittsburgh, 2016

As the demand for computing power continue to increase, both HPC community and Cloud service provides are building larger computing platforms to take advantage of the power and economies of scale. On the HPC side, several of the most powerful countries are competing for developing the next generation supercomputer–exascale computing machines to accelerate scientific discoveries, big data analytics, etc. On the Cloud side, the world's largest IT companies are all building large-scale datecenters, for both private usage and public services. However, aside from the benefits, several daunting challenges will appear when it comes to extreme-scale.

This thesis aims at simultaneously solving two major challenges, i.e., power consumption and fault tolerance, for future extreme-scale computing systems. We come up with a novel computational model, referred to as Lazy Shadowing, as an efficient and scalable approach to achieve high-levels of resilience, through forward progress, in extreme-scale, failure-prone computing environments. Two approaches have been proposed to implement this idea. Accordingly, detailed mathemetical models have been developed to quantify the system efficiency improving and energy saving of Lazy Shadowing.

In this work, I propose to continue the research in two aspects. Firstly, I propose to develop a MPI-based prototype to valdate the correctness of Lazy Shadowing in real environment. Using the prototype, I will run benchmarks and real applications to quantify its benefits compared to state-of-the-art approaches. In addition, I propose to further explore the potential of Lazy Shadowing and improve its efficiency. Based on the specific system con-

figuration and application QoS requirement, I will study the possibility of partial shadowing in three dimensions, i.e., time, space, and workload.

TABLE OF CONTENTS

1.0 INTRODUCTION

As a fundamental building block of modern computing systems, it is important for main memory to achieve short latency, high density, efficient energy and low cost, etc. Owing to the perfect balance on these aspects, dynamic random access memory (DRAM) has been the de facto standard in the past few decades, and is widely deployed in almost all systems, including servers, desktops, mobile devices and embedded systems. And, whereas a bunch alternative memories have been proposed, such as PCM [Lee et al., 2009a], STT-RAM [Kultursay et al., 2013] and 3D-XPoint [Patterson, 2015], none of them is strictly superior to DRAM, and thus DRAM is unlikely to be completely replaced. DRAM's great success is tremendously contributed by its continuous technology scaling to maintain performance growth and capacity enhancement. With the increasing stresses on capacity, bandwidth and performance from new high-performance applications and systems, DRAM must keep scaling down to meet the demands.

Unfortunately, scaling has become difficult due to the challenges to further shrink sizes of DRAM circuits and devices [Mandelman et al., 2002; Hong, 2010; Childers et al., 2015]. As a result, memory accesses tends to take longer time to finish [Son et al., 2014; Wang, 2015], cell behaviors are expected to be more statistical [Childers et al., 2015] and stored data is more vulnerable to disturbance and noise [Kim et al., 2014; Qureshi et al., 2015a; Kang et al., 2014]. Consequently, many cells will violate the timing standards, and DRAM chip yield and reliability will be degraded, which is undesirable for the already tight profit margin of DRAM manufacturing.

To mitigate the performance loss and yield degradation in further scaling, we need to expose the scaling effect to higher levels such that more effective approaches can be achieved with better trade-offs. Candidate solutions can be proposed from multiple levels, ranging from devices to architectures and even to applications. And, the efficient ones require the co-operation from different components, including DRAM chips, memory controllers, operating systems and processors, etc.

1.1 PROBLEM STATEMENT

Scaling-down has been the conventionally effective method to grow DRAM chip capacity, to reduce power consumption and to lower the per-bit cost. As scaling down, DRAM cells 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 applies lower supply voltage [Mukundan et al., 2013; Zhang et al., 2016], which further decreases per-cell charge and also increases 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 increase [Zhang et al., 2016; Mukundan et al., 2013], which makes it more difficult to sense data.

Moreover, scaling to smaller technology nodes grow the fabrication complexity and makes it more difficult to precisely control the dimensions of DRAM cells, which introduces more severe process variations (PV). Accordingly, DRAM cells are expected to show more statistical behaviors in a wide range; 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]. As a result, a large number of chips would fail the 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, which leads to 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 might cause data to be faulty.

To tackle the scaling issues, whereas significant researches have been put into retention and refresh [Hamamoto et al., 1998; Wong et al., 2008; Kim and Lee, 2009; 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., 2015b; Bhati et al., 2015b] and sensing [Son et al., 2013; D. Lee et al., 2013; Shin et al., 2014; Lee et al., 2015a], 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 explore the restoring operation in further scaling DRAM, and propose schemes to mitigate the induced issues like performance loss, yield degradation and other potential ones. Specifically, this thesis plans to answer the following questions: (i) how to solve the enlarged timing constraints utilizing the memory organization characteristics? (ii) can we integrate the restoring with other memory operations to find solutions? (ii) what kind of help can we get from operating system and the applications? Generally, this thesis tries to explore DRAM scaling from restoring perspective on different layers shown in Figure 1.1, covering hardware architecture ❶, operating system ❷ and applications ❸, etc.

1.2.1 Achieve Fast Rows via Reorganization (Completed)

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

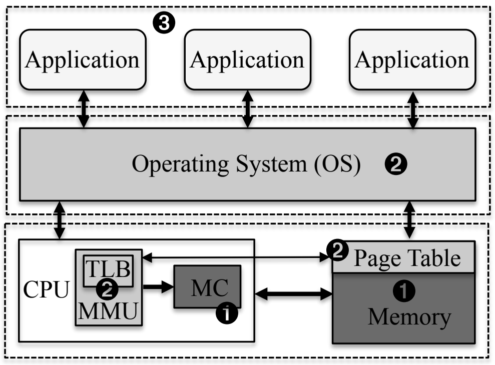


Figure 1.1: Overall view of thesis work. While placing a focus on memory structures, we explore restoring effects from other layers, including page translation from operating system and inherent behaviors of applications.

values. To preserve high chip yield, we should relax the timings to allow cells reliably finishing operations, which indicates a longer bank occupancy and thus degraded performance.

Fortunately, the variation characteristics also provide a large portion of good 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 compensate the performance loss. The exploration can be performed either on coarse chip-level or fine chunk-level, depending the overhead budget 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 timing is likely to offer very limited help. Accordingly, we choose to form logical rows by clustering similar fine-granularity chunks from different chips. As a result, more fast memory regions can be exposed. In addition, we extend the idea to assembly phase, where compatible chips are grouped together to form ranks, and hence bad chip contamination could be well controlled.

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. Following the idea, we profile workloads to identify hot pages, and then allocate them to fast memory regions to maximize performance gains.

1.2.2 Refresh-aware Partial Restore (Completed)

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, full charge is unnecessary if the access is close in time to the next refresh.

The observation motivates us to propose refresh-distance-aware partial restore. 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. The idea is implemented by partitioning one refresh window (usually 64ms) into multiple sub-windows, and set different restoring goals, i.e., different timings, for each.

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 access latency and overall performance, and hence we might achieve more truncation opportunities by upgrading refresh rate. However, blind upgrading introduces more refreshes, which not only prolongs memory unavailable period but also consumes more energy. As a compromise, we upgrade the refresh rate of selected bins, those were recently touched, and thus incur modest refreshing overhead to the system.

1.2.3 Explore Restoring in Extended Scenarios (Future)

Approximate Computing: 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 have intrinsic resilience to inaccuracy, which provides a good opportunity to tolerate the slow-to-restore cells in memory. By annotating the approximate data [Sampson et al., 2011, 2013; Miguel et al., 2014, 2015] which is tolerable to accuracy loss, we can shorten the restoring time of mapped memory locations. To guarantee the final output quality, we apply architectural techniques like error-correction and address remapping to avoid the faults of significant bits. Overall, we can achieve performance improvement and energy savings with a moderate sacrifice on output accuracy.

Information leakage: Sensitive information leakage through memory access patterns are rising concerns, and a bunch of works [Stefanov et al., 2013; Wang et al., 2014b; Shariee et al., 2015] have proposed to conceal the pattern. And, things become more challenging when we consider approximate computing and restoring variations. While the former provides a new kind of side channel that allows attackers to correlate approximate data to its location origin [Rahmati et al., 2015], the latter, even without approximate computing, is risky to leak more information on access patterns than prior cases [Wang et al., 2014b; Shariee et al., 2015], such as the possible physically mapped regions and the location correlation of accesses.

3D-stacked and Temperature: 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]. However, thermal management is still 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] 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., 2015a; 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 boost performance.

1.3 CONTRIBUTIONS

This thesis makes the following contributions:

- We perform pioneering study on DRAM restoring in deep sub-micron scaling. We built 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 achieve fast memory access; on system level, we maximizing performance improvement by allocating hot pages of the running workloads to fast regions.
- Going further, we integrate restoring variation characteristics with approximate computing to strike a good balance among performance, energy and accuracy. We then explore restoring issues in extended scenarios including information leakage and 3D-stacked memory.

1.4 OUTLINE

The rest of this proposal is organized as follow: Chapter 2 introduces the DRAM structures, operations and scaling issues. In Chapter 3, we build models to study restoring effects, and then propose a series of techniques to shorten restoring timing values. In Chapter 4, we explore the correlation between restoring and refresh, and seek the opportunities to early terminate restore operations. Further restoring explorations are discussed in Chapter 5. Chapter 6 and 7 lists the timeline and concludes the proposal, respectively.

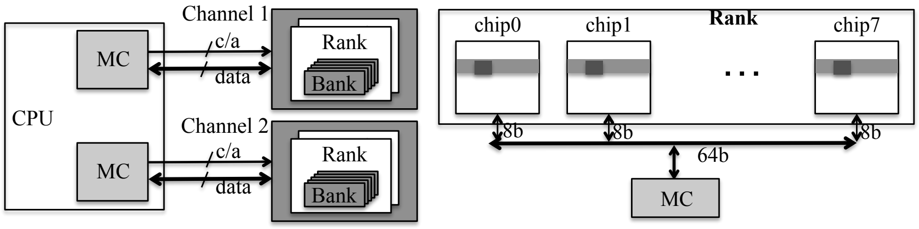
2.0 BACKGROUND

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, usually eight [1] , banks which operate in lockstep, i.e., all banks in a rank respond to a single command. 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.

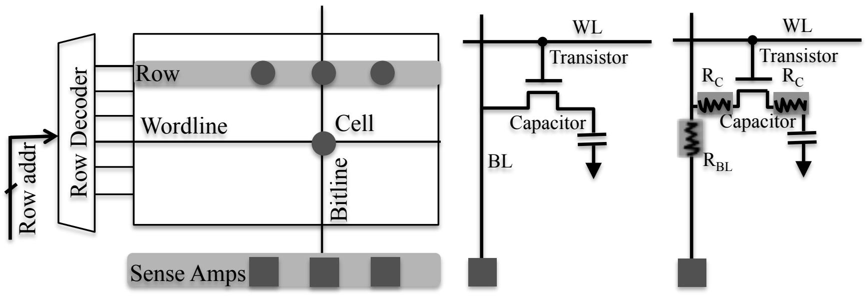
Physically, a DRAM rank is composed of multiple chips, inside which eight banks are deployed as cell arrays. The logical bank, as shown in Figure 2.1(a), is physically made up of the same numbered bank from all chips. For instance, `bank 0` of a rank contains `bank`

[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.



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

Figure 2.1: DRAM high-level structure.



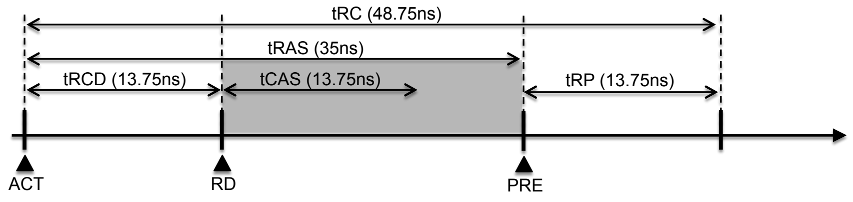
(a) Array **(b)** Cell **(c)** Circuit

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.

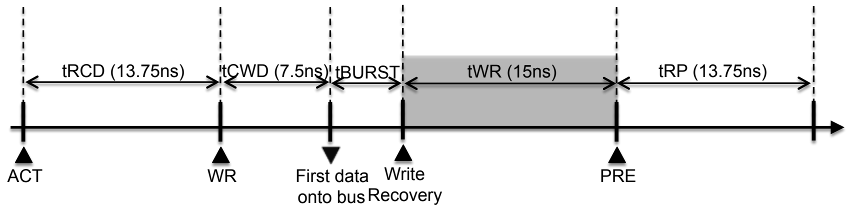
`0` [2] residing in all chips in the rank. Likewise, a DRAM row is dispersed across chips, as shown in Figure 2.1(b). In normal accesses to a rank, each chip provides 8 bits at a time simultaneously, which together satisfy the total data bus width of 64-bit. In addition, to amortize memory access overhead 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]. n is 8 for commodity DDR3, which translates into a granularity of 64B (64b$\times$8), the popular cache block size.

In more detailed level, DRAM cells are packed into 2D arrays, as Figure 2.2(b) shows, where each cell can be uniquely located by vertically bitline and horizontally 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 into the row buffer, which contains thousands of sense amplifiers to detect the voltage change on bitline.

[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) Read operation



(b) Write operation

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

2.2 DRAM OPERATIONS AND TIMING CONSTRAINTS

DRAM supports three types of accesses — read, write, and refresh. An on-chip memory controller (MC) 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. 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) 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 `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: refresh commands are issued by memory controller typically every 7.8μs to refresh a bin, which is composed of multiple rows. Upon receiving `REF`, DRAM device refresh the designated rows tracked by the internal counter. The refresh operation takes `tRFC` to complete, which proportionally depends on the number of rows in the bin. Whereas typically the whole memory rank is refreshed every 64ms, the vast majority cells can hold data for a much longer time [Liu et al., 2012a; Bhati et al., 2015b].

2.3 DRAM TECHNOLOGY SCALING

2.3.1 Scaling Issues

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., 2016], 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 sufficient 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

[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.

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 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.

2.3.2 Related Work on Restoring

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 `tWR` issues in further scaling DRAM, our paper [Zhang et al., 2015a] studied the variation behaviors and proposed to utilize chunk remapping to lower restoration durations. Afterwards, patents on `tWR` 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.

Whereas the `tWR` 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., 2015a], 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.

3.0 ACHIEVE FAST RESTORING 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 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: Modeling Parameters

tech node	μ_{bulk}	σ_{bulk}	μ_{tail}	σ_{tail}	ϕ	random weight
14nm [2]	2.048	0.247	3.283	0.0735	0.3	0.5

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

[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].

[2] Whereas we modeled both 20nm and 14nm nodes, here we only show the case of 14nm because of space limitation. Data and results on 20nm can be found in [Zhang et al., 2015a].

[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, then 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.

3.2 PROPOSED DESIGNS

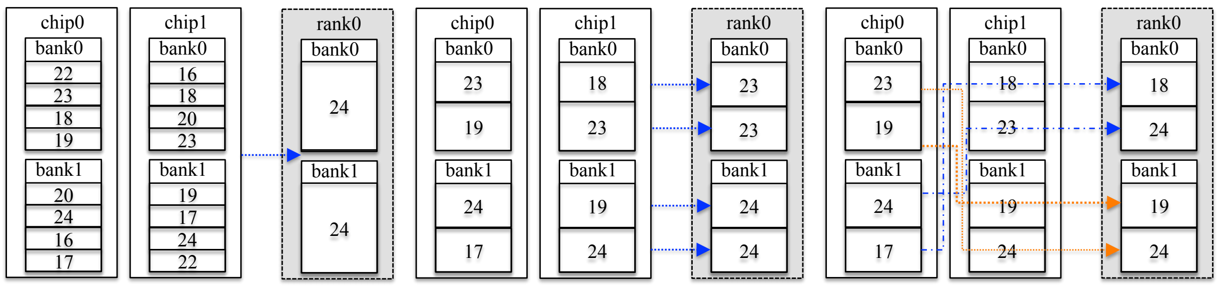
In this section, we elaborate the proposed designs. First, we discuss the post-fabrication schemes in terms of both coarse chip-level and fine chunk-level restoring management; then, we extend the schemes to assembly phase to deliberately form ranks by clustering compatible chips; and finally, the schemes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

3.2.1 Chip-specific Restoring Control

Conventionally, a single set of timing constraints is applied to the whole memory system, which totally ignores the existing variations and thus a too conservative setting. A simple enhancement can be made by exposing chip variations, i.e., setting different `tWR`s for different chips. As illustrated by Figure 3.1(a), 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.

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



(a) Chip-specific **(b)** Chunk-specific **(c)** Chunk-specific w/ remap

Figure 3.1: 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.

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 cheduling.

In Figure 3.1(b), one chunk consists of two rows. Since the first chunk has 23ns and 18ns `tWR`s 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 [3].

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 `tWR`s.

For the example in Figure 3.1(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

[3]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.

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 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 3.1(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.

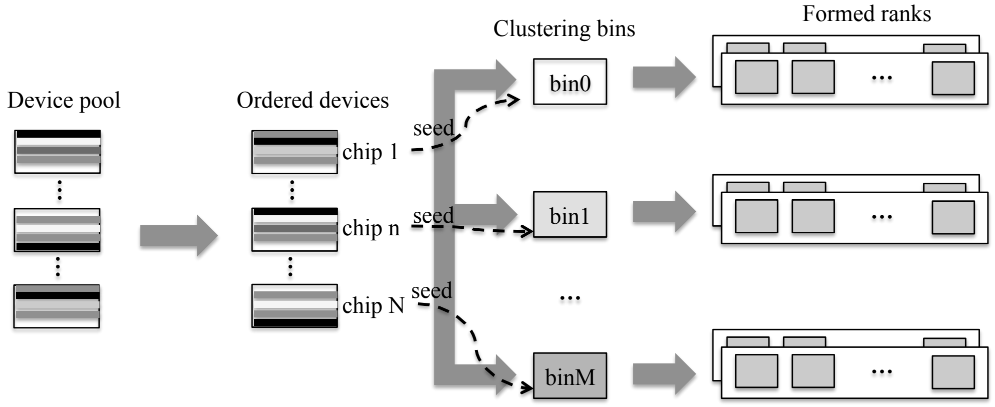


Figure 3.2: Rank construction consists of three steps — (1) chip sorting and seed chip selection; (2) distributing chips to bins; (3) constructing DRAM ranks using chips from each bin.

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 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.

3.2.5 Restore Time aware Page Allocation

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, the memory system can be more effective if fast chunks are assigned to service performance-critical pages. In this paper, 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].

In this paper, 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 [Son et al., 2013] 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 [Cong et al., 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 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.

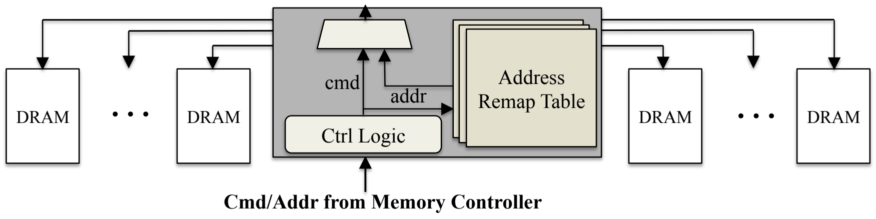


Figure 3.3: The on-DIMM architectural enhancement.

To enable chunk re-organization, we need one extra chunk remapping table as shown in Figure 3.3. Similar as HP's MC-DIMM [Ahn et al., 2009] and Mini-rank [Zheng et al., 2008], our design integrates a bridge chip on-DIMM, which remaps the physical address sent from the memory controller to device row addresses in each chip. For the chunk remapping table, each entry maps the corresponding DIMM-chunk to the chip-chunk at each chip.

3.4 EXPERIMENTAL METHODOLOGY

3.4.1 Configuration

To evaluate the effectiveness of our designs, we compared them to traditional repair solutions using an in-house chip-multiprocessor system simulator. We modeled a quad-core system with the parameters shown in Table 4.3.

We used VARIUS to generate 90 chips, and then form ranks in different fashion discussed in Section 3.2.4. 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].

Table 3.2: 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

3.4.2 Workloads

We used SPEC CPU2006 and simulated 1 billion instructions after skipping the warm-up phase of each benchmark. Based on MPKI, the applications are classified into three categories (Spec-High/Spec-Med/Spec-Low).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, following existing 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 [Jacob et al., 2007] to mitigate performance and yield loss. In our experiments, we set the spare density as high as 16 spare rows per 512-row block, which resides in the aggressive spectrum [Koren and Krishna, 2010; Kirihata et al., 1996]. 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, 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 Execution Time

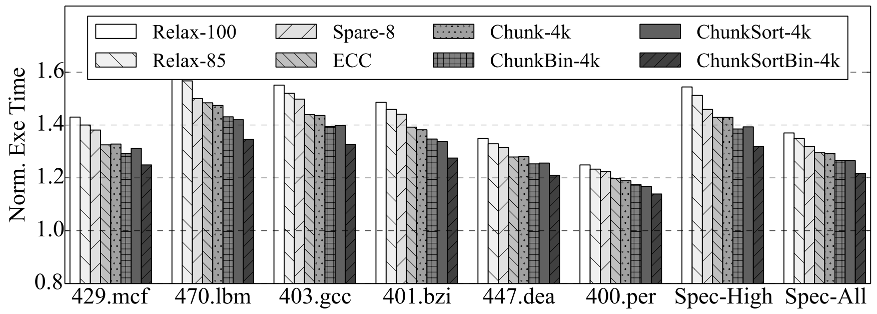
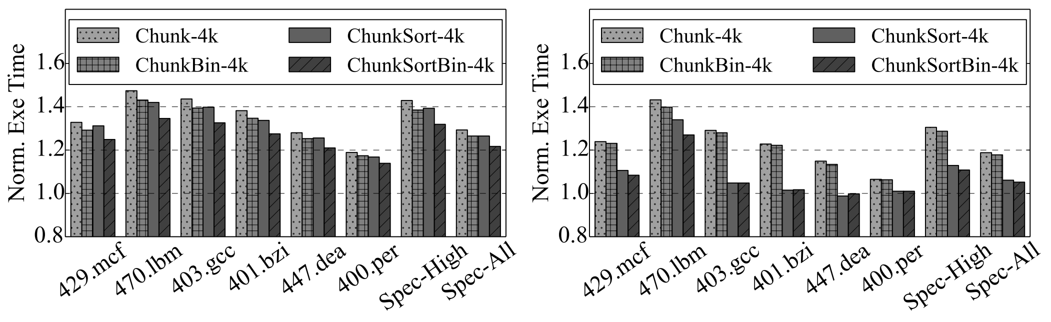


Figure 3.4: The execution time comparison of different schemes under random page allocation policy. Representative applications and the geometric means for highly memory-intensive (Spec-High) and all applications (Spec-All) are presented here.

From Figure 3.4, 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 40 cycles, which significantly hurts performance. On average, `Relax-100` and `Relax-85` prolong the execution time by 37.0% and 34.9%, respectively. Highly memory-intensive applications tend to have large degradation (i.e., over 40%). (2) Adding spare rows helps to mitigate performance losses: `Spare-8` is 31.9% 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 less than 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,



(a) With random page allocation **(b)** With restore time aware page allocation

Figure 3.5: The execution time comparison of different schemes at 14nm technology node.

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 37% in `Relax-100` to 26.5%, and 4% better than `Chunk-4k` for `Spec-High`.

In addition, restore time aware rank construction helps to reduce tWR — `ChunkBin-4k` is 3% better than `Chunk-4k` while `ChunkSortBin-4k` is 4.8% 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.

3.5.2 Page Allocation Effects

Figure 3.5 compare the results using random and restore-time-aware page allocation schemes. 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 15% improvement over random allocation. 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. Nevertheless, those benchmarks with large footprint and relatively uniform access pattern, e.g., 470.*lbm*, can still achieve distinct benefits.

3.5.3 Further Studies

The effectiveness of conventional Sparing technique strongly depends on the sparing levels being used; the proposed chunk-based schemes depends on the chunk granularity. Hence, we conducted sensitivity studies on these two key parameters. And the experimental results show that diminishing returns using more spares because of increasingly more slow cells. As expected, the study varying number of chunks shows that higher improvement can be achieved with increasing storage and latency overheads.

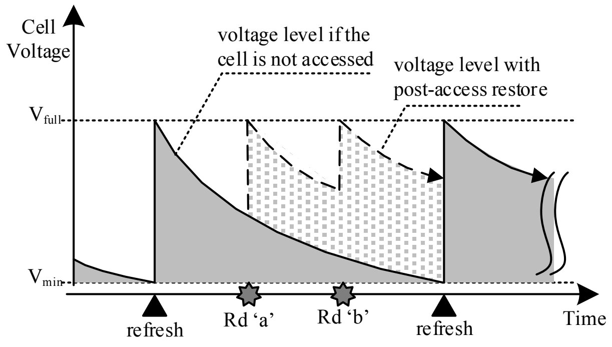
3.6 CONCLUSION

In this work, we studied DRAM 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 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.

4.0 SHORTEN RESTORING USING REFRESH

4.1 MOTIVATION OF PARTIAL RESTORE

Due to intrinsic leakage, the voltage level of a DRAM cell reduces monotonically after a full restore. The solid curve in Figure 4.1 illustrates the voltage decay of an untouched cell (i.e., not accessed) within one refresh window, commonly 64ms. Stored data is safe as long as the voltage remains above V$_{min}$ (0.73V$_{dd}$ [1] here) before the next refresh. If a read or write access occurs, the post-access restore operation fully charges the cell, 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.



Cell Voltage
voltage level if the cell is not accessed
voltage level with post-access restore
V$_{full}$
V$_{min}$
refresh
Rd 'a'
Rd 'b'
refresh
Time

Figure 4.1: 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 terminated when this target voltage level is reached.

The cell charging curve starts with a deep slope and flattens when approaching V$_{full}$ [Lee

[1] Value is calculated on basis of charge sharing expression [Kurinec and Iniewski, 2013] and offset noise [Hong et al., 2002], details can be found in [Zhang et al., 2016].

et al., 2015b; Demone, 2011], as demonstrated in SPICE modeling. Hence, reducing target voltage can drastically shorten restore time. For example, SPICE modeling indicates that restoring a cell's charge to 0.89 V_{dd} rather than 0.975 V_{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

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

sub-window	Distance to next refresh			Target restore	`tRAS`	`tWR`	`tRCD`
	64ms-row	128ms-row	256 ms-row	voltage (V_{dd})	(DRAM cycles) [2]		
1st	[64ms, 48ms)	[128ms, 96ms)	[256ms, 192ms)	0.975	42	25	15 [3]
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` truncates a long restore operation according to the time distance to its next refresh. The refresh window is partitioned into multiple sub-windows, each of which provides a set of timing parameter 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 paper. The smaller the timing values are, the larger 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 has diminishing benefits as shown in our experiments.

[2]Timing values are gotten from SPICE modeling, more details can be found in [Zhang et al., 2016].
[3]The studies focus on the relationship between restore and retention. Consequently, unrelated timing values, such as `tRCD`, are unchanged.

As illustrated by Figure 4.2, when servicing a read or write access, `RT-next` calculates 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.

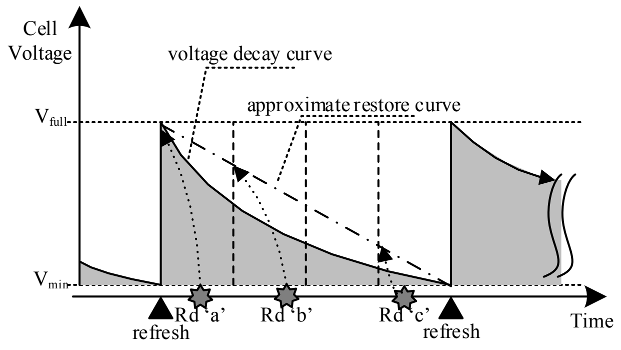


Figure 4.2: `RT-next` safely truncates restore operation. Exponential curve has been verified from SPICE modeling, and linear line shows the conservative restoring goals.

`RT-next` in multi-rate refresh. [4] Applying `RT-next` in a multi-rate refresh environment works similarly 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.

As shown in Figure 4.3, 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

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. We next present `RT-select`, a scheme that upgrades refresh rate for more truncation opportunities.

[4]Retention time is modeled following [Kim and Lee, 2009; Liu et al., 2012a; Nair et al., 2013b], and leaky cells are randomly distributed based on prior works [Liu et al., 2012a; Bhati et al., 2015b; Wang et al., 2014a]

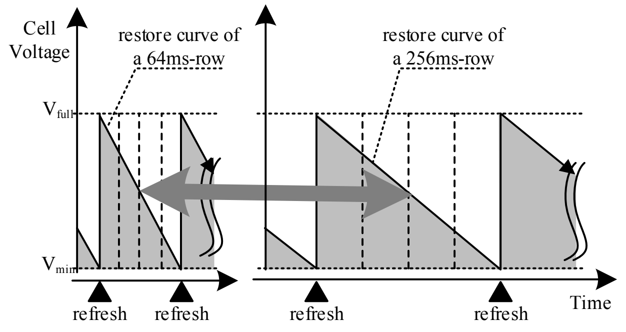


Figure 4.3: Restoring voltage according to linear line simplifies timing control in multi-rate refresh — a 64ms-row and a 256ms-row share the same timing values in each correspond sub-window.

Figure 4.4 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 4.4(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 4.4(b)).

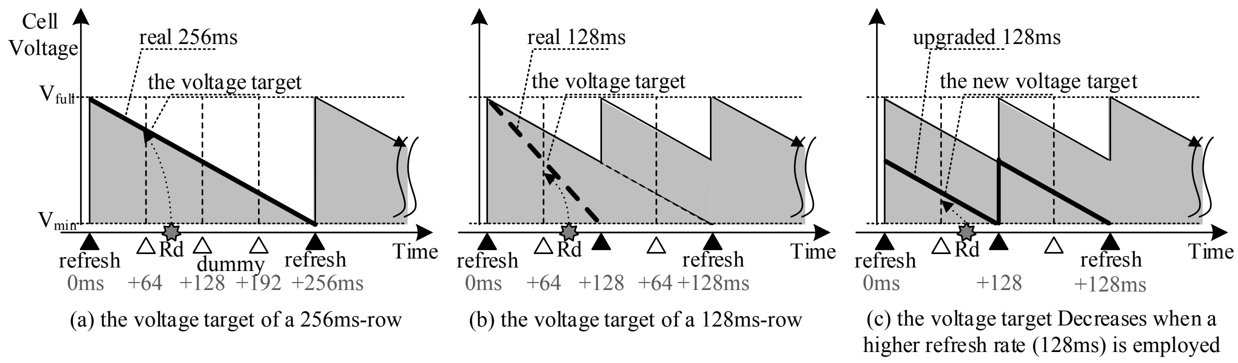


Figure 4.4: The voltage target can be reduced if a strong row is refreshed at higher rate.

Refreshing a 256ms-row at 128ms rate exposes more truncation benefits, as shown in Figure 4.4(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 4.4(c), rather than the dashed thick line from a real 128ms-row, as shown in 4.4(b). In particular, 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 4.4. 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.

4.3 ARCHITECTURAL ENHANCEMENTS

To enable `RT-next` and `RT-select`, we enhance the memory controller, as shown in Figure 4.5. 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].

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 4.2, 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/8192), 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.

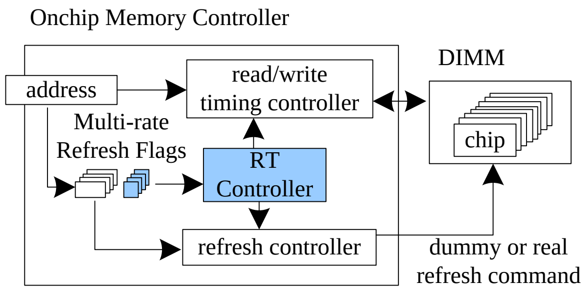


Figure 4.5: The RT architecture (the shaded boxes are added components).

When upgrading the refresh rate of a refresh bin, we update the rate flag according to the last column in Table 4.2. 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 further refresh rounds indicated by the two least significant bits (LSB) of rate flag.

Table 4.2: 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 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 is minimal, as shown in Section 4.5.2.

4.4 EXPERIMENTAL METHODOLOGY

4.4.1 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 4.3, similar to those in [Nair et al., 2013a; Shin et al., 2014]. The DRAM timing constraints follow Micron DDR3 SDRAM data sheet [MicronTech, 2009]. 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 works better in multicore systems [Liu et al., 2012b].

Table 4.3: 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 watermarks: 40/20 Address mapping: rw:cl:rk:bk:ch:offset Page management policy: close-page with FRFCFS
DRAM	2channels, 1rank/channel, 8banks/rank, 64K rows/bank, 8KB/row, 64B cache line tCK=1.25ns, width: x8

4.4.2 Workloads

For evaluation, we use workloads from the Memory Scheduling Championship [JWAC-3, 2012], which covers 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.

4.5 RESULTS

We evaluated our proposed RT 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 figure.

4.5.1 Impact on Performance

Figure 4.6 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.

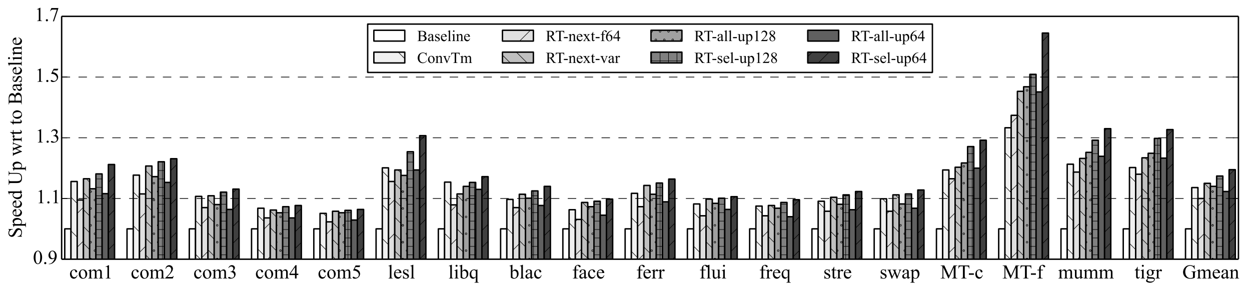


Figure 4.6: Performance comparison of different schemes. `Baseline` and `ConvTm` adopts projected and conventional timings, respectively; `NoRefresh` assumes no refresh activity in `Baseline`; `RT-next-XX` truncates restore based on its distance to next fresh, and `f64` and `var` supports fixed 64ms refresh rate and multiple rates, respectively; `RT-all-upYY` aggressively upgrades refresh bins with rates lower than `YY` to `YY`; finally, `RT-sel-upYY` are optimized schemes to balance the restore benefits and refresh overhead.

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.

4.5.2 Energy Consumption

Figure 4.7 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 results are averaged over workloads within each suite. We used the Micron power equations [MicronTech, 2007], and the parameters from vendor data sheets [MicronTech, 2009] 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 8192 bins are organized as 3KB direct mapped cache with 8B line size. We used CACTI5.3 [Cacti, 2009] 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`.

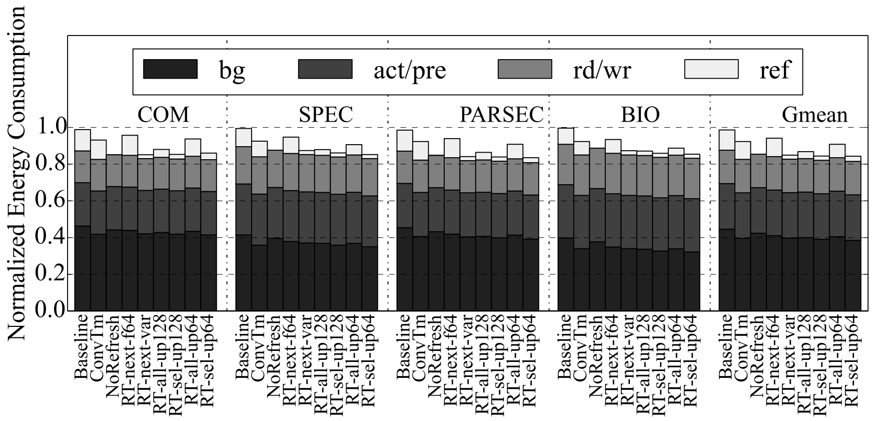


Figure 4.7: Comparison of memory system energy.

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 much more refresh energy than

RT-all-up128/-up64, respectively. RT-sel-up64 saves 17% energy compared to Baseline, and consumes slightly lower energy than NoRerefresh due to decreased execution time. And, as expected, RT-sel- refresh schemes is more energy efficient than RT-all- refresh peers.

4.5.3 Comparison against the State-of-the-art

Figure 4.8 compares RT with three related schemes in the literature.

- 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].

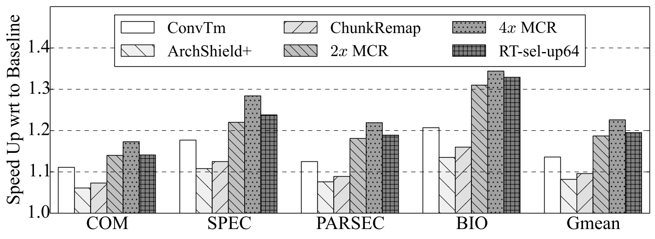


Figure 4.8: Comparison against the state-of-the-art.

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. $4x$ MCR outperforms RT-sel-up64 by a modest percentage, and RT-sel-up64 works better than $2x$ MCR.

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.

Table 4.4: 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.

4.5.4 Further Studies

To further evaluate the effectiveness of RT, we compare against several *ideal* schemes, including refresh-free scheme, conventional timings and best interval timings, etc. The results show that RT schemes defeat all of those schemes and the gap to the most ideal scheme of *best interval without refresh* is within 3%.

In addition, we evaluate the performance sensitivity by varying configurations including chip density, refresh granularity, refresh sub-window and page management policy. The results positively demonstrate the robustness of the achieved performance.

4.6 CONCLUSION

In this paper, we studied the restoring issues in further scaling DRAM, identified partial restore opportunity and proposed two restore truncation (RT) schemes to exploit the opportunities with different tradeoffs. Our experimental results showed that, on average, RT improves performance by 19.5% and reduces energy consumption by 17%.

5.0 FURTHER EXPLORATIONS OF RESTORING

This chapter will briefly discuss further explorations of restoring, as future work. We'll extend the restoring topic to approximate computing, memory information leakage and 3D stacked memory.

5.1 COMBINE RESTORING WITH APPROXIMATE COMPUTING

5.1.1 Introduction on Approximate Computing

Energy and power are increasing concerns in nowadays computer systems, ranging from mobile devices to data centers; and much energy is spent on guaranteeing correctness [Sampson et al., 2011]. Nevertheless, many modern applications have intrinsic tolerance to inaccuracy [d. Kruijf et al., 2010; Sampson et al., 2011]. For instance, lots of problems have no perfect answer in domains like machine learning, computer vision and sensor data analysis, and hence the adopted solutions rely on heuristic approach; and, large-scale data analytics cares more about aggregate trends rather than the correctness of individual data elements. Apparently, these applications provide good opportunities to explore energy-accuracy tradeoff.

Approximate computing necessitates the collaboration of different layers spanning circuits, architectures and algorithms. On program level, we should annotate the approximate data, which is non-critical and able to tolerate inexactness; on instruction level, the system should distinguish approximate and precise instructions and then take use of different strategies to execute; the fundamental energy savings rely on hardware techniques, including voltage reduction, floating point rounding and refresh rate reduction, etc.

To quantify the energy-accuracy tradeoff, we need to measure the output quality of approximate execution and should ensure that the quality loss is sustainable. The qualify-of-service (QoS) metrics are per-application specific, and are measured by comparing the final outputs of approximate execution against those of precise one. For instance, if the outputs are images, then the QoS can be evaluated as the average difference of per-RGB value [Sampson et al., 2011] between the executions.

5.1.2 Related Work

Prior works on approximate computing performed explorations from both hardware [Chakrapani et al., 2006; Narayanan et al., 2010; d. Kruijf et al., 2010; Liu et al., 2011; Esmaeilzadeh et al., 2012a,b; Sampson et al., 2013; Miguel et al., 2014, 2015] and software [Baek and Chilimbi, 2010; Sampson et al., 2011; Hoffmann et al., 2011; Achour and Rinard, 2015]. Among them, there is significant research on storing approximate data more efficiently, which is also the focus of our tentative exploration. Flikker [Liu et al., 2011] refreshes approximate data at lower rates to save DRAM refresh energy, which is recently extended by [Lucas et al., 2014; Arnab et al., 2015]. Esmaeilzadeh et al. [2012a] proposed to apply dual voltage to SRAM array to balance energy and accuracy; Drowsy caches [Flautner et al., 2002] reduces the supply voltage to save power; to improve PCM's lifetime and performance, [Sampson et al., 2013] proposed to reduce write precision and reuse failed cells.

5.1.3 General Ideas

According to the observations of our prior studies [Zhang et al., 2015a], whereas the worst-case `tWR` of the whole memory suffers from significant increase, a large portion of cells have `tWR` within the specification range. Such variation provides opportunity for applying approximate computing to achieve performance and performance improvements: `tWR` can be aggressively reduced to low value(s) as long as we can guarantee the correctness of precise data, and control the error impact of approximate data.

Program annotation can be performed by inserting assembly code, which will be identified by Pintool. Pintool simulates the real instruction execution, and when stepping into annotated approximate region, it injects errors using the underlying DRAM bit mask. As prior arts [Sampson et al., 2011; Miguel et al., 2014], we will control the annotation to let the program run to the end, and compare the approximate results against precise ones to report QoS. Simultaneously, Pintool outputs memory accesses, which are then fed into conventional memory simulator to collect performance and energy values. The most challenging part lies on the tradeoff between QoS and performance achievement: fine-grained and aggressive `tWR` control benefits memory performance, but is likely to introduce too many errors. We will take some measurements to constraint the errors, candidate solutions can be dedicated allocation, location correction or memory bit remapping, etc.

5.2 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.

5.3 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 is 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] 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., 2015a; 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 boost performance.

6.0 TIMELINE OF PROPOSED WORK

Table 6.1: Timeline of Proposed Work.

Date	**Content**	**Deliverable results**
Jan - Feb	Explore restoring in approximate computing in Section 5.1 of Chapter 5	Pin-based framework for restoring approximation
Mar - May	Integrate restoring with information leakage in Section 5.2 of Chapter 5	Experimental data of memory performance and security
Jun - Sep	Study restoring in Hybrid Memory Cube (HMC) in Section 5.3 of Chapter 5	Modified simulator of temperature effect of restoring in HMC
Jul - Oct	Thesis writing	Thesis ready for defense
Oct - Dec	Thesis revising	Completed thesis

The proposed works will be undertaken as shown in the Table 6.1. I will start the effort with task (1) to develop Pin-based framework for approximate computing of restoring. This task involves program annotation, chip generation, QoS evaluation, and conventional performance simulation, etc. While multiple complicated subtasks are there, this task has been partially finished, and will not take much time to complete. Afterwards, I'll move to task (2) to study information leakage in restoring scenario, and this task is partially on basis of the previous approximation work. With the completion of task (2), the overall goal of exploring DRAM restoring in application level have been reached, and then I'll start the study restoring's temperature effect in HMC, i.e., task (3). The general infrastructure can be borrowed from my previous HMC work [Zhang et al., 2015b]. This task might be performed concurrently with other jobs, and thus might take more time to finish. At the end of task (3), the holistic exploration of DRAM restoring is considered finished, and thus I'll summarize all the tasks into my final thesis.

7.0 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, which goes against the demanding of system designs and applications. Among the induced problems, restoring has been an 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 cause increasingly slow access and significantly more outliers falling beyond the specifications. To alleviate the influences on performance and yield, we propose to manage the timing constraints at fine chunk granularity, 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 move forward to profile the pages of the workloads, and deliberately allocate the hot pages to fast parts.

In addition, restoring can seek help from the correlated refresh operation, 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 the earlier the restore operation can be terminated. For ease of implementation, we divide the refresh window into 4 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.

As the most fundamental building block of computer systems, DRAM prolonged restoring operation will ultimately affect upper application level, and thus we further explore in extended scenarios, including approximate computing, information leakage and 3D stacked memory. With our full-stack exploration, we believe the scaling issues can be greatly alleviated.

BIBLIOGRAPHY

Achour, S. and Rinard M. C. Approximate computation with outlier detection in Topaz. In *OOPLSA*, pages 711–730, 2015.

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 *HPCA*, pages 84–95, 2014.

Ahn, J., Yoo S., Mutlu O., and Choi K. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *ISCA*, pages 336–348, 2015.

Ahn, J. H., Jouppi N. P., Kozyrakis C., Leverich J., and Schreiber R. S. Future scaling of processor-memory interfaces. In *SC*, pages 1–12, 2009.

Arnab, R., Jayakumar H., Sutar S., and Raghunathan V. Quality-aware data allocation in approximate DRAM. In *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.

Baek, W. and Chilimbi T. M. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, pages 198–209, 2010.

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. *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 *ISCA*, pages 235–246, 2015b.

Bhattacharjee, A. and Martonosi M. Thread criticality predictors for dynamic performance, power and resource management in chip multiprocessors. In *ISCA*, pages 290–301, 2009.

Cacti, . CACTI 5.3: An integrated cache timing, power and area model. `http://www.hpl.hp.com/research/cacti/`, 2009.

Chakrapani, L. N., Akgul B. E. S., Cheemalavagu S., Korkmaz P., Palem K. V., and Seshasayee B. Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology. In *DATE*, pages 1110–1115, 2006.

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 *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.

Childers, B. R., Yang J., and Zhang Y. Achieving yield, density and performance effective DRAM at extreme technology sizes. In *MEMSYS*, pages 78–84, 2015.

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 *ISCA*, pages 223–234, 2015.

Cong, J., Jiang W., Liu B., 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, Mar 2011.

Consortium, . Hybrid memory cube specification 2.0. http://www.jedec.org, 2015.

Kruijf, M.d. , Nomura S., and Sankaralingam K. Relax: an architectural framework for software recovery of hardware faults. In *ISCA*, pages 497–508, 2010.

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 *HPCA*, pages 615–626, 2013.

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

Eckert, Y., Jayasena N., and Loh G. H. Thermal feasibility of die-stacked processing in memory. In *WoNDP*, 2014.

Esmaeilzadeh, H., Sampson A., Ceze L., and Burger D. Architecture support for disciplined approximate programming. In *ASPLOS*, pages 301–312, 2012a.

Esmaeilzadeh, H., Sampson A., Ceze L., and Burger D. Neural acceleration for general-purpose approximate programs. In *MICRO*, pages 449–460, 2012b.

Flautner, K., Kim N. S., Martin S., Blaauw D., and Mudge T. Drowsy caches: simple techniques for reducing leakage power. In *ISCA*, pages 148–157, 2002.

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 *MICRO*, pages 134–145, 2010.

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 *ISCA*, pages 583–594, 2013.

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

Hoffmann, H., Sidiroglous S., Carbin M., Misailovic S., Agarwal A., and Rinard M. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, pages 199–212, 2011.

Hong, S. Memory technology trend and future challenges. In *IEDM*, pages 12.4.1–12.4.4, 2010.

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

Hynix, . 2gb ddr3 sdram data sheet. Hynix Semiconductor, 2010.

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, . DDR4 SDRAM, 2012.

JWAC-3, . MSC workloads. http://www.cs.utah.edu/~rajeev/jwac12/, 2012.

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 *ICICDT*, pages 175–176, 2004.

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 *SIGMETRICS*, pages 519–532, 2014.

Khurshid, M. J. and Lipasti M. Data compression for thermal mitigation in the hybrid memory cube. In *ICCD*, pages 185–192, 2013.

Kim, K. and Lee J. A new investigation of data retention time in truly nanoscaled DRAMs. *IEEE Electron Device Lett.*, 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 *ISCA*, pages 361–372, 2014.

Kirihata, T., Watanabe Y., Wong H., DeBrosse J., Yoshida M., Katoh D., Fujii S., Wordeman M., Poechmueller P., Parke S., and Asao Y. Fault-tolerant designs for 256Mb DRAM. *IEEE J. Solid-State Circuits*, 31(4):558–566, 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 *ISPASS*, pages 256–267, 2013.

Kurinec, S. K. and Iniewski K. *Nanoscale semiconductor memories: Technology and applications.* CRC Press, 2013.

Lee, B. C., Ipek E., Mutlu O., and Burger D. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 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 *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 Trans. Comput.*, 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 timing for the common-case: Optimizing dram timing for the common-case. In *HPCA*, pages 489–501, 2015a.

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 *HPCA*, pages 489–501, 2015b.

Liu, F. and Lee R. Security testing of a secure cache design. In *HASP*, 2013.

Liu, J., Jaiyen B., Veras R., and Mutlu O. RAIDR: Retention-aware intelligent DRAM refresh. In *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 *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 inerference in multicore systems. In *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 *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 *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.

Mandelman, J. A., Dennard R. H., Bronner G. B., DeBrosse J. K., Divakaruni R., Li Y., and Radens C. J. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM Journal of Research and Development*, 46(Issues 2-3):187–212, 2002.

MicronTech, . Calculating memory system power for DDR3. Micron Technology, 2007.

MicronTech, . 4Gb DDR3 SDRAM - MT41J512M8. Micron Technology, 2009.

Miguel, J. S., Badr M., and Jerger N. E. Load value approximation. In *MICRO*, pages 127–139, 2014.

Miguel, J. S., Albericio J., Moshovos A., and Jerger N. E. Doppelganger: A cache for approximate computing. In *MICRO*, 2015.

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 *ISCA*, pages 48–59, 2013.

Mutlu, O. and Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *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 *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 *ISCA*, pages 72–83, 2013b.

Narayanan, S., Sartori J., Kumar R., and Jones D. L. Scalable stochastic processors. In *DATE*, pages 335–338, 2010.

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, Jul 2008.

Patterson, D. Past and future of hardware and architecture. In *SOSP*, 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 *DSN*, pages 427–437, 2015a.

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 *DSN*, pages 427–437, 2015b.

Rahmati, A., Hicks M., Holcomb D. E., and Fu K. Probable cause: the deanonymizing effects of approximate DRAM. In *ISCA*, pages 604–615, 2015.

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

Ryan, J. F. and Calhoun B. H. Minimizing offset for latching voltage-mode sense amplifiers for sub-threshold operation. In *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 *PLDI*, pages 164–174, 2011.

Sampson, A., Nelson J., Strauss K., and Ceze L. Approximate storage in solid-state memories. In *MICRO*, pages 25–36, 2013.

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 Trans. Semicond. Manuf*, 21(1): 3–13, Feb 2008.

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 *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 *MICRO*, 2015.

Shin, W., Yang J., Choi J., and Kim L.-S. NUAT: A non-uniform access time memory controller. In *HPCA*, pages 464–475, 2014.

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, 12 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 *ISCA*, pages 380–391, 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 *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 *MICRO*, pages 375–384, 2010.

Su, C., Yeh Y., and Wu C. An integrated ECC and redundancy repair scheme for memory reliability enhancement. In *DFT*, pages 81–89, 2005.

Vogelsang, T. Understanding the energy consumption of dynamic random access memories. In *MICRO*, pages 363–374, 2010.

Wang, D. Backward compatible dynamic random access memory device and method of testing therefor. United States Patent, 9 2015.

Wang, J., Dong X., and Xie Y. Proactive DRAM: A DRAM-initiated retention management scheme. In *ICCD*, pages 22–27, 2014a.

Wang, Y. and Suh G. E. Efficient timing channel protection for on-chip networks. In *NOCS*, pages 142–151, 2012.

Wang, Y., Ferraiuolo A., and Suh G. E. Timing channel protection for a shared memory controller. In *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 *DAC*, pages 1–6, 2015.

Wang, Z. and Lee R. New cache designs for thwarting software cache-based side channel attacks. In *ISCA*, pages 494–505, 2007.

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 *ITC*, pages 1–7, 2008.

Yoon, D. Y., Jeong M. K., and Erez M. Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput. In *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 *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 *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 *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 *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 *HPCA*. to be published, 2016.

Zheng, H., Lin J., Zhang Z., Gorbatov E., David H., and Zhu Z. Mini-Rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*, pages 210–221, 2008.