Reducing DRAM Latency at Low Cost
by Exploiting Heterogeneity

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Donghyuk Lee

M.S., Electrical and Computer Engineering, Carnegie Mellon University
B.S., Electrical Engineering, Seoul National University

Carnegie Mellon University
Pittsburgh, PA

May 2016

Acknowledgement

The last five years at Carnegie Mellon University have been most exciting time of my life, thanks to all of the fantastic people that I have met and worked together with. First and foremost, I am grateful to my advisor, Prof. Onur Mutlu. He provided a great opportunity for me to join his research group as a Ph.D. student and provided great guidance to do not only all the works in this dissertation but also many other works. He taught me to think differently and thoroughly to determine real-world problems and to find better ways to solve these problems, leading to making an impact on the real world. Prof. Mutlu always supported me and encouraged me to improve all my works and my abilities. I am also very thankful to Prof. Mutlu for providing me with a great research environment. Under his support, providing all of the required resources, I could focus on my research and collaborate greatly with many fantastic people.

I would like to thank my thesis committee members, Prof. Todd Mowry, Prof. Kayvon Fatahalian, Prof. Shih-Lien Lu, and Prof. Mattan Erez for their time, efforts, and comments in bringing this dissertation to completion. Special thanks to Prof. Erez for his precise feedback on the entire dissertation. Thanks to Prof. Mowry for his guidance on my research, which started from my qualifying examination. Thanks to Prof. Lu for his interest in my research and valuable feedback. Thanks to Prof. Kayvon Fatahalian for his valuable comments on my research. I would also like to thank Prof. Rajeev Balasubramonian, Dr. Michael Kozuch and Konrad Lai for their interests and great feedback on my research.

The SAFARI group has been like my home and family. Without support from SAFARI members, this dissertation could not have been completed. Yoongu Kim has always been my good friend and mentor. I am really thankful to all his support over every step of my graduate school career. I mostly followed his mentoring, which made it possible to finish this dissertation. Thanks to Vivek Seshadri for his incisive insight and plentiful helps, which improved all works in this dissertation. His optimistic attitude on research and life impressed me. Thanks to Lavanya Subramanian for all her great support, and for many valuable discussions on research and life. Whenever I faced problems, she was always there and provided priceless suggestions. Thanks to Samira Khan for her kindness to listen to all of my problems and provide valuable suggestions. With her dedicated support, I was able to keep working and finish this dissertation. Thanks to Gennady Pekhimenko for his valuable comments on my ideas and suggestions on research directions. His enthusiasm on research and endless efforts to achieve each of his goals impressed me, which made me to put more efforts into my research. Thanks to Hongyi Xin for being a great office mate over four and half years. He taught me the basics of programming and provided valuable guidance on being a programmer. He made my graduate life more exciting and fruitful. Thanks to Saugata Ghose for his critiques. He provided great feedback and helped to improve my work. Thanks to Rachata Ausavarungnirun for his kindness to discuss anything on research and life. He always supported and encouraged me to be a better researcher. Thanks to Chris

Abstract

In modern systems, DRAM-based main memory is significantly slower than the processor. Consequently, processors spend a long time waiting to access data from main memory, making the long main memory access latency one of the most critical bottlenecks to achieving high system performance. Unfortunately, the latency of DRAM has remained almost constant in the past decade. This is mainly because DRAM has been optimized for cost-per-bit, rather than access latency. As a result, DRAM latency is not reducing with technology scaling, and continues to be an important performance bottleneck in modern and future systems.

This dissertation seeks to achieve low latency DRAM-based memory systems at low cost in three major directions. The key idea of these three major directions is to enable and exploit latency heterogeneity in DRAM architecture. First, based on the observation that long bitlines in DRAM are one of the dominant sources of DRAM latency, we propose a new DRAM architecture, Tiered-Latency DRAM (TL-DRAM), which divides the long bitline into two shorter segments using an isolation transistor, allowing one segment to be accessed with reduced latency. Second, we propose a fine-grained DRAM latency reduction mechanism, Adaptive-Latency DRAM, which optimizes DRAM latency for the common operating conditions for individual DRAM module. We observe that DRAM manufacturers incorporate a very large timing margin as a provision against the worst-case operating conditions, which is accessing the slowest cell across all DRAM products with the worst latency at the highest temperature, even though such a slowest cell and such an operating condition are rare. Our mechanism dynamically optimizes DRAM latency to the current operating condition of the accessed DRAM module, thereby reliably improving system performance. Third, we observe that cells closer to the peripheral logic can be much faster than cells farther from the peripheral logic (a phenomenon we call architectural variation). Based on this observation, we propose a new technique, Architectural-Variation-Aware DRAM (AVA-DRAM), which reduces DRAM latency at low cost, by profiling and identifying only the inherently slower regions in DRAM to dynamically determine the lowest latency DRAM can operate at without causing failures.

This dissertation provides a detailed analysis of DRAM latency by using both circuit-level simulation with a detailed DRAM model and FPGA-based profiling of real DRAM modules. Our latency analysis shows that our low latency DRAM mechanisms enable significant latency reductions, leading to large improvement in both system performance and energy efficiency across a variety of workloads in our evaluated systems, while ensuring reliable DRAM operation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1   Problem

Primarily due to its low cost-per-bit, DRAM has long been the choice substrate for architecting main memory systems. In fact, DRAM's cost-per-bit has been decreasing at a rapid rate as DRAM process technology scales to integrate ever more DRAM cells into the same die area. As a result, each successive generation of DRAM has enabled increasingly large-capacity main memory subsystems at low cost.

In stark contrast to the continued scaling of cost-per-bit, the *latency* DRAM has remained almost constant, During the same 11-year interval in which DRAM's cost-per-bit decreased by a factor of 16, DRAM latency (as measured by the t_{RCD} and t_{RC} timing constraints) decreased by only 30.5% and 26.3%, as shown in Figure 1.1. From the perspective of the processor, an access to DRAM takes hundreds of cycles – time during which a modern processor is likely stalled, waiting for DRAM [12, 76, 189, 191, 193, 194, 195]. Such wasted time, which is more than 50-60% of the execution time for many memory-intensive workloads [12, 189, 191, 194], leads to large performance degradations commonly referred to as the "memory wall" [283] or the "memory gap" [279].

However, the high latency of commodity DRAM chips is in fact a *deliberate* trade-off made by DRAM manufacturers. While process technology scaling has enabled DRAM designs with both lower cost-per-bit *and* lower latency [98], DRAM manufacturers have usually sacrificed the latency benefits of scaling in order to achieve even lower cost-per-bit. Hence, while low-latency DRAM chips exist [129, 173, 235], their higher cost-per-bit relegates them to
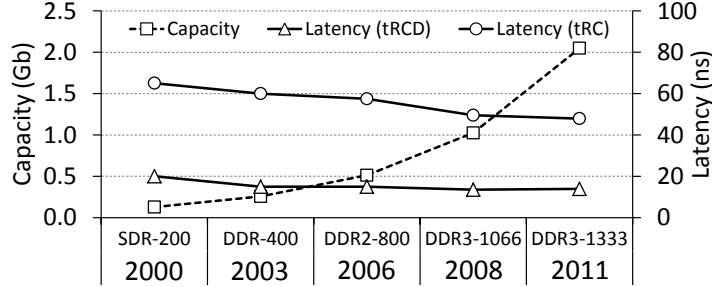
† We refer to the dominant DRAM chips during the period of time [27, 111].

FIGURE 1.1: DRAM Capacity & Latency Over Time [27, 111, 207, 232]

specialized applications such as high-end networking equipment that demand very low latency even at the expense of a very high cost [270].

Our main research objective is to enable a low latency DRAM-based memory system, in order to achieve high system performance,via simple and low cost DRAM architectures and memory control techniques.

1.2 Our Approach

Towards achieving our goal of a low latency DRAM-based memory system, our major approach is to enable or exploit *latency heterogeneity* in DRAM. Due to the large number of cells in DRAM, there already exists such latency heterogeneity (latency variation) due to multiple aspects. For example, *i)* DRAM cells in different locations in the same chip or across different chips have different latencies, and *ii)* DRAM has different latencies at different operating conditions (e.g., temperature). However, to provide a simplified interface to access DRAM, DRAM has been designed to improve only the *worst case latency*. One simple example is that DRAM uses *identical* timing parameters for accessing *all* DRAM cells from *any* DRAM chip at *all* operating conditions, even though the latency can be very different based on the location of each cell, the characteristics of different chips, or the characteristics of each cell at different operating conditions. Therefore, DRAM has been focused on either minimizing these latency variations in its architecture or hiding them. Unfortunately, the latency variation in DRAM is expected to increase with further DRAM cell scaling due to

worsening process variation at smaller process technology nodes. Therefore, enabling low DRAM latency is expected to be more difficult in the future.

Our approach for lowering DRAM latency is enabling or exploiting such latency heterogeneity by *i)* rearchitecting DRAM at low cost, *ii)* enabling fine-grained DRAM control to optimize DRAM latency for the common operating conditions, and *iii)* exploiting the latency variation inherent in the internal DRAM architecture.

1.2.1 Lowering DRAM Latency by Rearchitecting DRAM Bitline Architecture (Tiered-Latency DRAM)

DRAM has been developed to make its capacitor-based cell smaller for higher capacity. The small size of this capacitor necessitates the use of an auxiliary structure, called a *(local) sense amplifier*, to detect the small amount of charge held by the cell and amplify it to a full digital logic value. But, a local sense amplifier is approximately one hundred times larger than a cell [223]. To amortize its large size, each local sense amplifier is connected to many DRAM cells through a wire called a *bitline*. Every bitline has an associated parasitic capacitance whose value is proportional to the length of the bitline. Unfortunately, such parasitic capacitance slows down DRAM operation for two reasons. First, it increases the latency of the local sense amplifiers. When the parasitic capacitance is large, a cell cannot quickly create a voltage perturbation on the bitline that could be easily detected by the local sense amplifier. Second, it increases the latency of charging and precharging the bitlines. Although the cell and the bitline must be restored to their quiescent voltages during and after an access to a cell, such a procedure takes much longer when the parasitic capacitance is large. Due to the above reasons and a detailed analysis of the latency break-down [145], we conclude that long bitlines are the dominant source of DRAM latency.

The bitline length is a key design parameter that exposes the important trade-off between latency and die-size (cost). Short bitlines (few cells per bitline) constitute a small electrical load (parasitic capacitance), which leads to low latency. However, they require more local sense amplifiers for a given DRAM capacity, which leads to a large die-size. In contrast, long bitlines have high latency and a small die-size. As a result, neither of these two approaches

can optimize for both latency and cost-per-bit. The goal in this work is to design a new DRAM architecture to approximate the best of both worlds (i.e., low latency and low cost), based on the key observation that long bitlines are the dominant cause of DRAM latency.

To achieve the latency advantage of short bitlines and the cost advantage of long bitlines, we propose the *Tiered-Latency DRAM* (TL-DRAM) architecture, which divides the long bitline into two shorter segments using an *isolation transistor*: the *near segment* (connected directly to the local sense amplifier) and the *far segment* (connected to the local sense amplifier only when the isolation transistor is turned on). As a result, the near segment has much lower latency than the far segment. To maximize the latency benefits from this heterogeneous bitline architecture, we propose two mechanisms – *i)* using the near segment as a hardware-managed cache to the far segment, and *ii)* exposing the near segment to the operating system, which places latency-critical data to the near segment. We propose two new policies to manage the near segment cache that specifically exploit the asymmetric latency characteristics of TL-DRAM. Our most sophisticated cache management algorithm, Benefit-Based Caching (BBC) improves system performance by an average of 12.8% and reduces energy consumption by an average of 23.6% over a wide variety of data-intensive workloads.

In summary, TL-DRAM enables latency heterogeneity in DRAM by changing the internal DRAM architecture with low area cost, and provides mechanisms to maximize the latency benefits by enabling intelligent data placement.

1.2.2 Optimizing DRAM Latency to the Common Operating Conditions (Adaptive-Latency DRAM)

When a DRAM chip is accessed, it requires a certain amount of time before enough charge can move into the cell (or the bitline) for the data to be reliably stored (or retrieved). To guarantee this behavior, DRAM manufacturers impose a set of minimum latency restrictions on DRAM accesses, referred to as *timing parameters* [105]. Ideally, timing parameters should provide just enough time for a DRAM chip to operate correctly. In practice, however, DRAM manufacturers *pessimistically incorporate a very large margin* into their timing parameters to ensure correct operation under *the worst case conditions* due to two major concerns. First,

due to **process variation** [73, 147, 233], some outlier cells suffer from a larger delay than other cells, and require more time to be charged. Although every cell is designed to have a large capacitance (to hold more charge) and a small resistance (to facilitate the flow of charge), some deviant cells may not be implemented in such a manner. Second, due to **temperature dependence**, all cells suffer from a weaker charge-drive at high temperatures, and require more time to charge the bitline. Consequently, to accommodate the combined effect of process variation *and* temperature dependence (the worst case condition), existing timing parameters prescribed by the DRAM manufacturers are set to a very large value.

Our approach is to reduce the DRAM latency by reducing the additional *latency slack* in DRAM due to the pessimism on timing parameters. To study the potential for reducing timing parameters for each DRAM module, we characterize 115 DRAM modules from three manufacturers to expose the excessive margin that is built into their timing parameters. We make two observations. First, even at the highest temperature of 85℃, there is a high potential for reducing the latency of DRAM modules (21.1% on average for read and 34.4% for write operations). Second, we observe that at lower temperatures (e.g., 55℃) the potential for latency reduction is even greater (32.7% on average for read and 55.1% on average for write operations). As a result, we conclude that exploiting process variation and lower temperatures enable a significant potential to reduce DRAM latencies.

Based on our characterization, we propose Adaptive-Latency DRAM (AL-DRAM), a mechanism that dynamically optimizes the timing parameters for different modules at different temperatures. AL-DRAM exploits only the *additional charge slack* in the common-case compared to the worst-case, thereby maintaining the reliability of DRAM modules. We evaluate AL-DRAM on a real system [18, 19] and show show that AL-DRAM improves the performance of a wide variety of memory-intensive workloads by 14.0% (on average) without introducing any errors.

In summary, AL-DRAM enables lower DRAM latency while maintaining memory correctness and without requiring changes to the internal DRAM architecture or the DRAM interface (low cost).

1.2.3 Lowering DRAM Latency by Exploiting the Awareness of Internal DRAM Architecture (Architectural-Variation-Aware DRAM)

Modern DRAM consists of 2D cell arrays, each of which has its own accessing structure (e.g., wordline driver) and data sensing structure (e.g., local sense amplifier). We observe that there is variability across DRAM cells based on their locations in a DRAM cell array (*mat*). Some DRAM cells can be accessed faster than others due to their physical location. We refer to this variability in cells' access times, caused by the physical organization of DRAM, as *architectural variation*. Architectural variation arises from the difference in the distance between the cells and the peripheral logic that is used to access these cells. The wires connecting the cells to peripheral logic exhibit large resistance and large capacitance [143, 145]. Consequently, cells experience different RC delays based on their distance from the peripheral logic (e.g, accessing and sensing structures). Cells closer to the peripheral logic experience smaller delay and can be accessed faster than the cells located farther from the peripheral logic.

Architectural variation in latency is present in both vertical and horizontal directions in a mat: *i)* Each vertical *column of cells* in a mat is connected to a *local sense amplifier* and *ii)* each horizontal *row of cells* in a mat is connected to a *wordline driver*. Variations in the vertical and horizontal dimensions, together, divide the cell array into heterogeneous latency regions, where cells in some regions require larger latencies for reliable operation. This variation in latency has direct impact on the reliability of the cells. Reducing the latency *uniformly across all regions* in DRAM would improve performance, but can introduce failures in the *inherently slower* regions that have to be accessed longer for correct DRAM operation. We refer to these inherently slower regions of DRAM as *architecturally vulnerable regions*. We first experimentally demonstrate the existence of architectural variation in modern DRAM chips and identify the architecturally vulnerable regions. We then propose new mechanisms that leverage this variation to reduce DRAM latency while providing reliability at low cost.

Based on our experimental study that characterizes 96 DRAM modules by using our FPGA-based DRAM testing infrastructure, we show that *i)* modern DRAM chips exhibit

architectural latency variation in both row and column directions, and *ii)* architectural vulnerability gradually increases in the row direction within a mat and repeats the variability pattern in every mat. Then, we develop two new mechanisms that exploit the architecturally vulnerable regions to enable low DRAM latency with high reliability and at low cost (*Architectural-Variation-Aware DRAM (AVA-DRAM)*). The first mechanism, *AVA Profiling*, identifies the lowest possible latency that ensures reliable operation at low cost by periodically profiling *only* the architecturally vulnerable regions. To further reduce DRAM latency, the second mechanism, *AVA Shuffling*, distributes data from architecturally vulnerable regions to multiple ECC codewords to make it correctable by using ECC. AVA Profiling can dynamically reduce the latencies of read/write operations by 35.1%/57.8% at 55℃ while ensuring reliable operation at low cost. AVA Shuffling on average corrects 26% of total errors which are not correctable by conventional ECC, leading to further latency reduction for 24 DRAM modules out of 96 DRAM modules. We show that the combination of our techniques, AVA-DRAM, leads to a raw DRAM latency reduction of 40.0%/60.5% (read/write) and an overall system performance improvement of 14.7%/13.7%/13.8% (2-/4-/8-core) over a variety of workloads in our evaluated systems, while ensuring reliable operation.

AVA-DRAM is the first work that exposes and experimentally demonstrates the existence of architectural variation. We then propose two mechanisms that leverage architectural variation towards achieving high performance, energy efficiency, and reliability through dynamic profiling (AVA Profiling) and data shuffling (AVA Shuffling).

1.3   Thesis Statement

DRAM latency can be reduced by enabling and exploiting latency heterogeneity in DRAM architecture.

1.4 Contributions

This dissertation makes the following major contributions.

- This dissertation makes the observation that long internal wires (bitlines) are the dominant source of DRAM latency, and exposes the important trade-off between DRAM latency and area. Based on this, this dissertation proposes a new DRAM architecture, Tiered-Latency DRAM, which divides long bitline into fast and slow segments, enabling latency heterogeneity in DRAM. This dissertation quantitatively evaluates the latency, area, and power characteristics of Tiered-Latency DRAM through circuit simulations based on a publicly available 55nm DRAM process technology [223]. We show that the near segment latency (t_{RC}) for a 32-row near segment can be 49% lower than the modern DRAM standard latency.

- This dissertation describes two major ways of leveraging TL-DRAM: *i)* by using the near segment as a hardware-managed cache without exposing it to software, and *ii)* by exposing the near segment capacity to the OS and using hardware/software to map frequently accessed pages to the near segment. We propose two new policies to manage the near segment that specifically exploit the asymmetric latency characteristics of TL-DRAM. Our most sophisticated cache management algorithm, Benefit-Based Caching (BBC) improves system performance by an average of 12.8% and reduces energy consumption by an average of 23.6% over a wide variety of workloads.

- This dissertation provides a detailed analysis of why we can reduce DRAM timing parameters without sacrificing reliability in the common case. We show that the latency of a DRAM access depends on how quickly charge moves into or out of a cell. Compared to the worst-case cell operating at the worst-case temperature (85℃), a typical cell at a typical temperature allows much faster movement of charge, leading to shorter latency. This enables the opportunity to reduce timing parameters without introducing errors.

- This dissertation provides detailed DRAM profiling results (for 115 DRAM modules, comprised of 920 DRAM chips, from three manufacturers) by using an FPGA-based DRAM testing infrastructure, and exposes the large margin built into their timing parameters. In

particular, we identify four timing parameters that are the most critical during a DRAM access: t_{RCD}, t_{RAS}, t_{WR}, and t_{RP}. At 55℃, we demonstrate that the parameters can be reduced by an average of 17.3%, 37.7%, 54.8%, and 35.2% while still maintaining correctness.

- This dissertation proposes a practical mechanism, *Adaptive-Latency DRAM (AL-DRAM)*, to take advantage of the extra margin built into DRAM latency. The key idea is to dynamically adjust the DRAM timing parameters for each module based on its latency characteristics and temperature so that the timing parameters are dynamically optimized for the current operating condition and the current DRAM module. We show that the hardware cost of AL-DRAM is very modest, with no changes to DRAM. We evaluate AL-DRAM on a real system [18, 19] running real workloads by dynamically reconfiguring the timing parameters. AL-DRAM improves system performance by an average of 14.0% and a maximum of 20.5% over a wide variety of memory-intensive workloads, without incurring any errors.

- This dissertation exposes and experimentally demonstrates the phenomenon of *architectural latency variation* in DRAM cell arrays, i.e., that the access latency of a cell depends on its location in the DRAM array. This phenomenon causes certain regions of DRAM to be inherently more vulnerable to latency reduction than others based on their relative distance from the peripheral logic.

- This dissertation identifies the regions in DRAM that are most vulnerable to latency reduction based on the internal hierarchical organization of DRAM bitlines and wordline drivers. We call such regions as *architecturally vulnerable regions*. We experimentally demonstrate the existence of architecturally vulnerable regions in DRAM by testing and characterizing 96 real DRAM modules (768 DRAM chips).

- This dissertation develops two new mechanisms, called AVA Profiling and AVA Shuffling, that exploit architectural variation to improve performance and reliability of DRAM at low cost. AVA Profiling dynamically finds the lowest latency at which a DRAM chip can operate reliably. AVA Profiling can dynamically reduce the latencies of read/write operations

by 35.1%/57.8% at 55℃ while ensuring reliable operation at low cost. AVA Shuffling distributes data from architecturally vulnerable regions to multiple ECC codewords to make it correctable by using ECC. AVA Shuffling on average corrects 26% of total errors which are not correctable by conventional ECC. We show that the combination of our techniques, AVA-DRAM, leads to a raw DRAM latency reduction of 40.0%/60.5% (read/write) and an overall system performance improvement of 14.7%/13.7%/13.8% (2-/4-/8-core) over a wide variety of workloads in our evaluated systems, while ensuring reliable operation.

1.5 Dissertation Outline

This dissertation is organized into seven chapters. Chapter 2 presents background on memory system organization and DRAM organization. Chapter 3 discusses related prior work on techniques for reducing and tolerating DRAM latency. Chapter 4 presents the design of Tiered-Latency DRAM and mechanisms to leverage the Tiered-Latency DRAM substrate for reducing overall DRAM latency. Chapter 5 first presents the latency slack in DRAM standard timing parameters that are dictated by the worst case conditions (accessing the smallest cell in DRAM products at the worst case operating temperature). It then proposes and evaluates Adaptive-Latency DRAM that optimizes DRAM timing parameters for the common case (accessing the common cells in each DRAM module at current operating temperature). Chapter 6 first presents the observation of architectural variation in a DRAM cell array (mat). It then proposes and evaluates AVA-DRAM that leverages architectural variation for reducing DRAM latency at low cost, by periodically profiling only the worst latency regions at low cost and distributing data in the worst latency regions to multiple ECC codewords. Chapter 7 introduces system design guidelines for future memory systems that have heterogeneous latency. Finally, Chapter 8 presents our conclusions and future research directions that are enabled by this dissertation.

Chapter 2

Background

To understand the dominant sources of DRAM latency, we first provide the necessary background on DRAM organization and operation.

2.1 DRAM Organization

DRAM is organized in a hierarchical manner where each DRAM module consists of multiple chips, banks, mats, and subarrays. Figure 2.1 shows the hierarchical organization of a typical DRAM-based memory system, where the hierarchy consists of five levels (Figure 2.1a–2.1d).

Module. At the highest level of the hierarchy (shown in Figure 2.1a), a memory controller in a processor is connected to a DRAM *module* over a memory channel. The memory channel has a 64-bit data bus that is divided into eight 8-bit buses connected to eight chips in the DRAM module. These eight chips operate in lock step while accessing the DRAM module. For example, when there is a read request from the memory controller, each DRAM chip transfers 8 bytes of data in 8 data bursts over the dedicated 8-bit data bus connected to it, transferring a total of 64 bytes (cache line size) across all chips.

Chip. A DRAM *chip* (shown in Figure 2.1b) consists of *i)* multiple banks and *ii)* peripheral logic that is used to transfer data to the memory channel through the IO interface.

Bank. Each *bank* (shown in Figure 2.1c), is subdivided into multiple *mats*. In a bank, there are two global components that are used to access the mats: *i)* a *row decoder* that selects a row of cells *across* a subarray that consists of multiple mats and *ii) global sense amplifiers*

(A) DIMM (8 chips) (B) Chip (8 banks)

(C) Bank (D) Mat (512×512 cells)
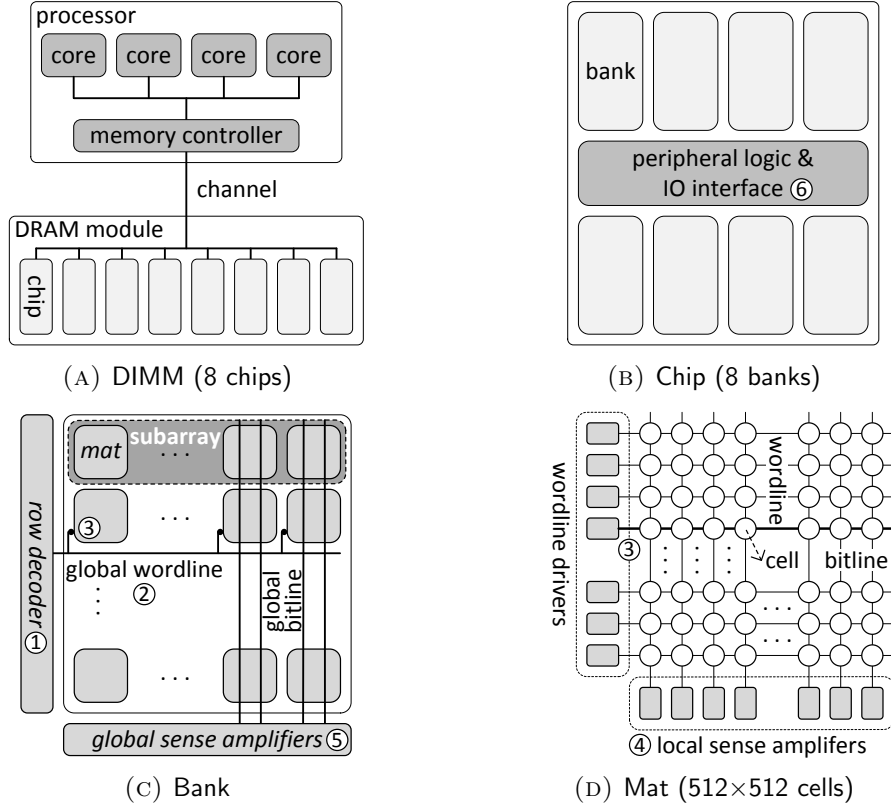
FIGURE 2.1: Hierarchical Organization of a DRAM System

that transfer a fraction of data from the row through the global bitlines, based on the column address.

Mat. Figure 2.1d shows the organization of a *mat* that consists of three components: *i)* a 2-D cell array in which the cells in each row are connected horizontally by a shared wire called the *wordline*, and the cells in each column are connected vertically by a wire called the *bitline*, *ii)* a column of wordline drivers that drive each wordline to appropriate voltage levels in order to activate a row during an access and *iii)* a row of *local sense amplifiers* that sense and latch data from the activated row.

Subarray. A row of mats in a bank forms a subarray, where cells in a *subarray* are accessed simultaneously, managed by wordlines connected to a global wordline, as shown in Figure 2.1c.

2.2 DRAM Cell Organization

As explained in Section 2.1, a DRAM subarray is a group of mats in the horizontal direction and each mat is a 2-D array of elementary units called *cells*. As shown in Figure 2.2a, a cell consists of two components: *i)* a capacitor that represents binary data in the form of stored electrical charge and *ii)* an access transistor that is switched on/off to connect/disconnect the capacitor to a *bitline*. As shown in Figure 2.2b, there are approximately 512 cells in the vertical direction (a "column" of cells), all of which share the same bitline. For each bitline, there is a *local sense amplifier* whose main purpose is to read from a cell by reliably detecting the very small amount of electrical charge stored in the cell. When writing to a cell, on the other hand, the local sense amplifier acts as an electrical driver and programs the cell by filling or depleting its stored charge.

(A) Cell   (B) Bitline & Local Sense Amplifier   (C) Simplified View
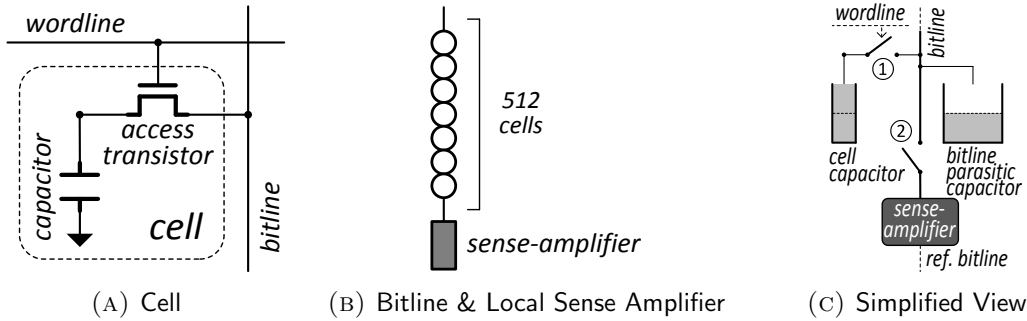
FIGURE 2.2: DRAM Elementary Components

Numerous bitlines (and their associated local sense amplifiers) are laid side-by-side in parallel to compose a subarray (Figures 2.1c and 2.1d). All cells in the horizontal direction (a "row" of cells) have their access transistors controlled by a *wordline*. When the wordline voltage is raised to V_{DD}, all cells of a row are connected to their respective bitlines and sensed in lockstep by the local sense amplifiers. This is why the set of all local sense amplifiers in a subarray is also called a *row buffer*. At any given time, at most one wordline in a subarray is ever raised (i.e., at most one cell per column is connected to the bitline) – otherwise, cells in the same column would corrupt each other's data.

Figure 2.2c depicts a simplified view of a cell as well as its bitline and local sense amplifier, in which electrical charge is represented in gray. Switch ① represents the access transistor controlled by the wordline, and switch ② represents the on/off state of the sense amplifier.

2.3   DRAM Access

As the timelines in Figure 2.3 show, a DRAM chip access can be broken down into three distinct *phases*: *i)* activation, *ii)* IO, and *iii)* precharging. Activation and precharging occur entirely within the subarray, whereas IO occurs in the peripheral logic and IO circuitry. All these operations consist of two levels of accesses through: *i)* global structures across all subarrays within a bank (global sense amplifiers, global wordlines and global bitlines) and *ii)* local structures within a mat (local sense amplifiers, local wordlines, and local bitlines). A DRAM access goes through multiple steps in the global-local hierarchy:
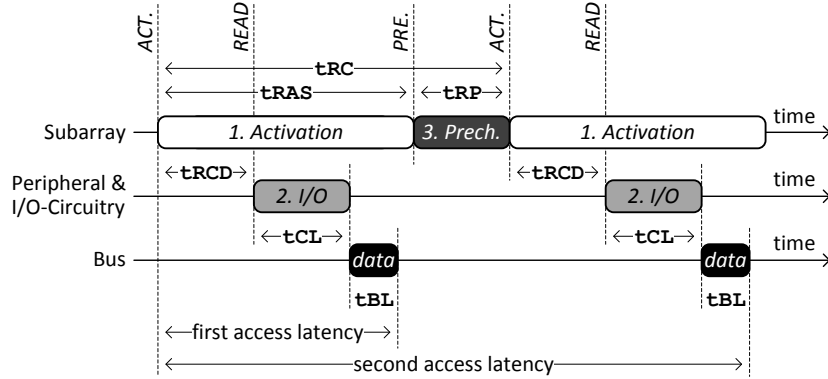
FIGURE 2.3: Three Phases of DRAM Access

- **Activation.** During the *activation* phase, the row decoder in a bank receives a row address to activate (① in Figure 2.1c), then, it first activates the corresponding global wordline in the bank (② in Figure 2.1c). The global wordline, in turn, activates the corresponding wordline driver in each mat of the subarray. The wordline driver in each mat activates the corresponding local wordline connecting the row of cells to the bitlines (③ in Figures 2.1c and 2.1d). Soon thereafter, the data in the row of cells is copied (detected) to the local sense amplifiers of that subarray (④ in Figure 2.1d).

14

- **IO.** During the *IO* phase, the local sense amplifiers transfer the data to the global sense amplifiers, through the global bitlines (⑤ in Figure 2.1c). Data from the global sense amplifiers is then sent to the memory channel through the IO interface of the chip (⑥ in Figure 2.1b). From there, the data leaves the DRAM chip and is sent to the processor over the memory channel. As Figure 2.3 shows, the IO phase's latency is overlapped with the latency of the activation phase.

- **Precharging.** During the *precharging* phase, the raised wordline in the subarray is lowered, disconnecting the row of cells from the bitlines. Also, the subarray's local sense amplifiers and bitlines are initialized (i.e., cleared of their data) to prepare for the next access to a new row of cells.

Three DRAM Commands. The DRAM controller (typically residing on the processor die) issues *commands* to the DRAM chip to initiate the three phases listed above. As shown in Figure 2.3, there are three commands, one for each phase. In their respective order, they are: `ACTIVATION` (ACT), `READ/WRITE`, and `PRECHARGE` (PRE). Among the commands, `ACTIVATION` and `PRECHARGE` are subarray-related commands since they directly operate on the subarray, whereas `READ` and `WRITE` are IO-related commands.

Timing Constraints. After the DRAM controller issues a command to initiate a phase, it must wait for a sufficient amount of time before issuing the next command. Such restrictions imposed between the issuing of commands are called *timing constraints*. DRAM timing constraints are visualized in Figure 2.3 and summarized in Table 2.1. Two of the most important timing constraints are t_{RCD} (*row-to-column delay*) and t_{RC} (*row-cycle time*). Every time a new row of cells is accessed, the subarray incurs t_{RCD} (15ns; `ACTIVATION` → `READ/WRITE`) to copy the row into the local sense amplifiers. On the other hand, when there are multiple accesses to different rows in the same subarray, an earlier access delays all later accesses by t_{RC} (52.5ns; `ACTIVATION` → `ACTIVATION`). This is because the subarray needs time to complete the activation phase (t_{RAS}) and the precharging phase (t_{RP}) for the earlier access, whose sum is defined as t_{RC} ($= t_{RAS} + t_{RP}$), as shown in Figure 2.3.

Phase	Commands	Name	Value
1	ACTIVATION → READ ACTIVATION → WRITE	t_{RCD}	15ns
	ACTIVATION → PRECHARGE	t_{RAS}	37.5ns
2	READ → $data$ WRITE → $data$	t_{CL} t_{CWL}	15ns 11.25ns
	$data\ burst$	t_{BL}	7.5ns
3	PRECHARGE → ACTIVATION	t_{RP}	15ns
1 & 3	ACTIVATION → ACTIVATION	t_{RC} $(t_{RAS} + t_{RP})$	52.5ns

TABLE 2.1: Timing Constraints (DDR3-1066) [232]

Access Latency. Figure 2.3 illustrates how the DRAM access latency can be decomposed into individual DRAM timing constraints. Specifically, the figure shows the latencies of two read accesses (to different rows in the same subarray) that are served one after the other. From the perspective of the first access, DRAM is "unloaded" (i.e., no prior timing constraints are in effect), so the DRAM controller immediately issues an ACTIVATION on its behalf. After waiting for t_{RCD}, the controller issues a READ, at which point the data leaves the subarray and incurs additional latencies of t_{CL} (peripherals and IO circuitry) and t_{BL} (bus) before it reaches the processor. Therefore, the latency of the first access is 37.5ns ($t_{RCD} + t_{CL} + t_{BL}$). On the other hand, the second access is delayed by the timing constraint that is in effect due to the first access (t_{RC}) and experiences a large "loaded" latency of 90ns ($t_{RC} + t_{RCD} + t_{CL} + t_{BL}$).

2.4   DRAM Cell Operation: A Detailed Look

Subarray-related timing constraints (t_{RCD} and t_{RC}) constitute a significant portion of the unloaded and loaded DRAM access latencies: 40% of 37.5ns and 75% of 90ns, respectively. Since t_{RCD} and t_{RC} exist only to safeguard the timely operation of the underlying subarray, in order to understand why their values are so large, we must first understand how the subarray operates during the activation and the precharging phases. (As previously explained, the IO phase does not occur within the subarray and its latency is overlapped with the activation

phase.) Specifically, we show how the bitline plays a crucial role in both activation and precharging, such that it heavily influences both t_{RCD} and t_{RC}. As shown in Figure 2.4, a cell transitions through five different states during each access.
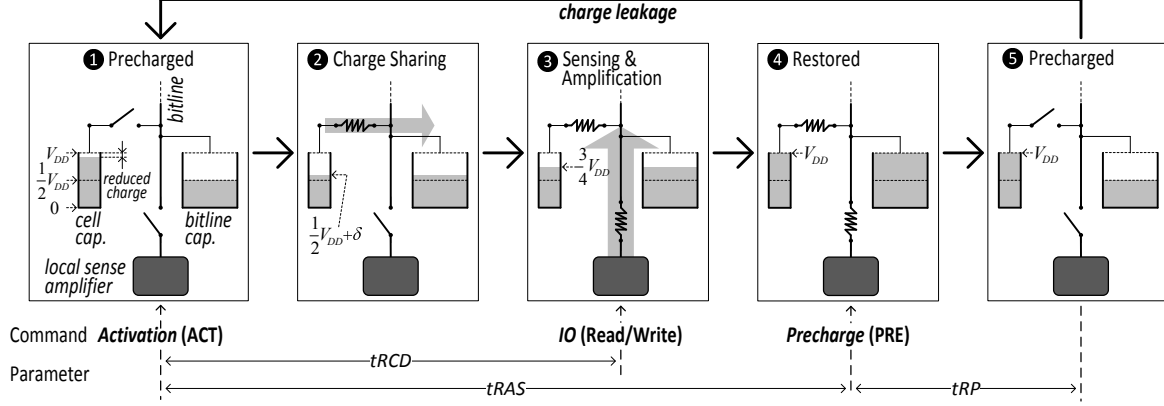
FIGURE 2.4: DRAM Operations, Commands and Parameters

- **Precharged (Initial) State.** In the first state (State ❶), which is called the *precharged* state, before it is accessed, the cell is initially "fully" charged, while the bitline is only halfway charged (i.e., the bitline voltage is maintained at $\frac{1}{2}V_{DD}$). In practice, the cell is usually not completely charged because of a phenomenon called *leakage*, wherein the cell capacitor loses charge over time.

- **Activation (Charge Sharing).** In order to access data from a cell, the DRAM controller issues a command called `ACTIVATION`. Upon receiving this command, DRAM increases the wordline voltage, thereby turning on access transistors connected to the wordline, leading to connecting their capacitors to the bitlines. Since the cell is at a higher voltage than the bitline, the charge in the cell capacitor (or the lack thereof) is slowly shared with the *bitline parasitic capacitor*, thereby perturbing the bitline voltage away from its quiescent value ($\frac{1}{2}V_{DD}$) in the positive (or negative) direction until their voltages are equalized at $\frac{1}{2}V_{DD} + \delta$ (or $\frac{1}{2}V_{DD} - \delta$). This is depicted in state ❷, which is called *charge sharing*. During charge sharing, note that the cell's charge is modified (i.e., data is lost) because it is shared with the bitline. But this is only temporary since the cell's charge is restored as part of the next step, as described below.

17

- **Activation (Sensing & Amplification).** After allowing sufficient time for the charge sharing to occur, the sense amplifier is turned on. Immediately, the sense amplifier "senses" (i.e., observes) the polarity of the perturbation on the bitline voltage. Then the sense amplifier "amplifies" the perturbation by injecting (or withdrawing) charge into (or from) both the cell capacitor and the bitline parasitic capacitor. After a latency of t_{RCD}, midway through amplification, enough charge has been injected (or withdrawn) such that the bitline voltage reaches a *threshold* state of $\frac{3}{4}V_{DD}$ (or $\frac{1}{4}V_{DD}$). At this point, data is considered to have been "copied" from the cell to the sense amplifier (State ❸). In other words, the bitline voltage is now close enough to V_{DD} (or *0*) for the sense amplifier to detect a binary data value of '1' (or '0') and transfer the data to the IO circuitry, allowing `READ` and `WRITE` commands to be issued. After completing the sense-amplification, the voltage of the bitline and the cell are fully amplified to V_{DD} or *0* (State ❹). Only at this point is the charge in the cell fully restored to its original value. The latency to reach this *restored* state (State ❹) is t_{RAS} (which is one component of t_{RC}). If there is a write operation, some additional time is required for the bitline and the cell to reach this state, which is expressed as a timing parameter called t_{WR}.

- **Precharging.** Before we can access data from a different cell connected to the same bitline, the sense amplifier must be taken back to the precharged state. This is done by issuing a `PRECHARGE` command. Upon receiving this command, DRAM first decreases the wordline voltage, thereby turning off access transistors and disconnecting the cell from the bitline. By doing so, the cell becomes decoupled from the bitline and is not affected by any future changes in the bitline voltage. Next, DRAM disables the sense amplifier and withdraws (or injects) charge from the bitline parasitic capacitor such that the bitline voltage reaches the quiescent value of $\frac{1}{2}V_{DD}$ (State ❺). Precharging is required to ensure that the next accessed cell can perturb the bitline voltage in either direction (towards V_{DD} or towards *0*). This would not be possible if the bitline is left unprecharged at V_{DD} or *0*. The time taken for the precharge operation is expressed as a timing parameter called t_{RP} (which is the other component of t_{RC}).

- **Idle State (between Accesses).** At state ❺, note that the cell is completely filled with charge. Subsequently, however, the cell slowly loses some of its charge until the next access (cycling back to state ❶). The length of time for which the cell can reliably hold its charge is called the cell's *retention time*. If the cell is not accessed for a long time, it may lose enough charge to invert its stored data, resulting in an error. To avoid data corruption, DRAM refreshes the charge in all of its cells at a regular interval, called the *refresh interval*. To avoid data corruption, modern DRAMs periodically access the cells to restore the lost amount of charge in all of its cells, called the *refresh*. This refresh operation happens at a regular interval, called the *refresh interval*. DRAM refresh operations waste energy and also degrade performance by delaying memory requests. This negative impacts are expected to increase in the future high capacity DRAMs. To mitigate the energy and performance impact of DRAM refresh, many works performed experimental studies to analyze DRAM refresh characteristics [119, 155, 217] and proposed hardware or software techniques [40, 156, 217].

Summary. Through our discussion, we have established a relationship between the timing constraints (t_{RCD}, t_{RAS}, t_{RP}, and t_{RC}) and the internal DRAM architecture and cell operations. To summarize, these timing constraints are determined by how quickly the bitline voltage can be driven – for t_{RCD}, from $\frac{1}{2}V_{DD}$ to $\frac{3}{4}V_{DD}$ (*threshold*); for t_{RAS}, from $\frac{1}{2}V_{DD}$ to V_{DD} (*restored*) and back again to $\frac{1}{2}V_{DD}$; for t_{RP}, from V_{DD} to $\frac{1}{2}V_{DD}$ *precharging*. In Chapter 4, we first show that the drivability of the bitline is determined by the bitline parasitic capacitance, whose value is a function of the bitline length, then proposes a new DRAM architecture that enables latency heterogeneity by changing the DRAM bitline architecture. In Chapter 5, we exploit the latency slack in the internal DRAM architecture to optimize the timing constraints for the common operation conditions. In Chapter 6, we exploit the latency variation inherited from the internal DRAM architecture to reduce DRAM latency.

Chapter 3

Related Prior Work

In this chapter, we describe prior proposals to reduce overall memory access latency in different areas, *i)* new DRAM architectures for lowering DRAM latency, *ii)* heterogeneous memory control techniques for reducing DRAM latency, *iii)* new DRAM architectures for enabling more parallelism, *iv)* memory scheduling to mitigate high DRAM latency, and *v)* other related prior works for mitigating high DRAM latency.

3.1 Low Latency DRAM Architecture

Prior works aim to reduce DRAM latency in three major directions. The first approach is changing/optimizing DRAM architecture to reduce DRAM latency. The second approach is integrating low latency SRAM cells in DRAM, which can be used for caching recently accessed data. The third approach enables latency variation in DRAM. We present more detail of these three approaches. The key distinguishing factor of the proposals in this dissertation is the fact that they are low cost, while most prior works have high cost.

DRAMs Optimized for Low Access Latency. Some specialized DRAMs provide shorter latency than commodity DRAM by reducing the number of cells-per-bitline in their cell array. Micron's RL-DRAM [173] and Fujitsu's FCRAM [235] have much shorter bitlines than commodity DRAM, enabling lower latency than conventional DRAM. Embedded DRAM [42, 117, 166, 197] is recently introduced to use DRAM as the last level cache in the processor. To this end, embedded DRAM uses very short bitlines, enabling both competitive latency and much larger capacity compared to conventional SRAM-based on-chip

caches. Unfortunately, reducing cells-per-bitline requires more sense amplifiers to integrate the same amount of storage in a DRAM chip, leading to high area cost. We estimate the effective area of using shorter bitlines by using CACTI-D [268]. Our estimation shows that using 128 cells-per-bitline increases area by 30% compared to using 512 cells-per-bitline (commonly used in modern DRAM chip) because doing so requires four times the sense amplifiers of conventional DRAM. Therefore, this approach results in significantly higher cost-per-bit than conventional DRAM. Compared to these approaches, all our proposals reduce DRAM latency *without* significant area overhead and thus achieve both high system performance and low implementation cost.

Cached DRAM – Integrating SRAM Cache in DRAM. Cached DRAM [69, 83, 86, 90, 199, 234, 296] integrates an SRAM cache in a DRAM chip. Due to the locality in memory accesses, many requests could hit recently accessed data that is temporarily stored in the on-DRAM cache. Cached DRAM serves such requests from the low latency SRAM cells, reducing overall memory access latency. However, such cached DRAM approaches have two major limitations. First, an SRAM cache incurs significant area overhead. Based on DRAM area analysis using CACTI-D [268], an SRAM-cached DRAM requires 145.3% additional area (leading to requiring 245.3% area of the baseline conventional DRAM) to integrate SRAM cells (6% of total DRAM capacity) in a DRAM chip. Second, transferring data between the DRAM array and the SRAM cache requires the use of relatively narrow global I/O buses within the DRAM chip, leading to high latency to move the data from the DRAM array into the SRAM cache (and vice versa).

Compared to this approach, our proposals reduce DRAM latency *without* significant area overhead and thus achieve both high system performance and low implementation cost.

Enabling Latency Heterogeneity in DRAM. Son et al. [252] proposed a low latency DRAM architecture in two major directions. First, it integrates both short bitline subarrays, which have low access latency, and long bitline subarrays, which have high access latency, within a DRAM chip. Second, it leverages the latency difference from the physical locations of banks. For example, banks near an IO interface can be accessed with lower latency than banks far from an IO interface. Therefore, this approach enables latency heterogeneity in

DRAM by enabling different bitline lengths in different locations. For example, accessing a subarray, which is near the IO interface and which consists of short bitlines, has the lowest access latency. This static partitioning of different latency regions limits the effectiveness of the approach. One shortcoming of the design is the high latency required to move data between the slow and fast regions. This approach provides largest benefit if latency critical data is statically allocated to the low latency regions (the low latency subarrays). Static identification of latency critical or hot data could be difficult.

Lu et al. [159] improves the heterogeneous subarray architecture (having both long bitline subarrays and short bitline subarrays) by introducing low-latency migration capability between slow and fast subarrays. In the open-bitline scheme [96], even bitlines are connected to the upper sense amplifiers and odd bitlines are connected to the lower sense amplifiers. This work introduces new DRAM cells to connect these two bitlines (one connected to the upper sense amplifier and the other to the lower sense amplifier), enabling data migration between slow and fast subarrays (e.g., long bitline subarrays and short bitline subarrays). However, this approach requires specialized migration cells, leading to higher manufacturing cost.

Our first approach (TL-DRAM) reduces DRAM latency by introducing latency heterogeneity *within* a subarray, which is different from the works of Son et al. and Lu et al., which enable heterogeneity *across* subarrays. As such, TL-DRAM can achieve very fast migration between the slow and fast regions of the same subarray.

Our two other approaches (AL-DRAM and AVA-DRAM) reduce DRAM latency by exploiting the existing latency slack in DRAM, which none of these prior works has leveraged. Therefore, our proposals can be combined with these prior works in a synergistic manner to achieve even lower DRAM latency than all individual proposals.

3.2 Reducing DRAM Latency by Enabling Heterogeneous Memory Control

Prior works optimize DRAM latency for the operating conditions by exploiting process variation [37] or memory access patterns [85, 247]. The key distinguishing factor of our proposals is the fact that they provide mechanisms to maintain *reliability* while reducing DRAM latency.

Heterogeneous Memory Control Based on Process and Voltage Variation. Chandrasekar et al. [37] evaluate the potential of relaxing some DRAM timing parameters to reduce DRAM latency. This work observes latency variations across DIMMs as well as for a DIMM at different operating temperatures. However, there is no explanation as to why this phenomenon exists and no clear mechanism to exploit it in this prior work.

Heterogeneous Memory Control Based on Memory Access Patterns. Shin et al. [247] show that DRAM leakage affects two DRAM timing parameters (t_{RCD}/t_{RAS}). As a result, recently-refreshed rows have more charge, and can be accessed with lower latency than DRAM standard. Based on this observation, they propose a mechanism to access recently refreshed rows with reduced latency. However, this approach has two limitations. First, this work focuses only on the latency variation between refreshes, resulting in relatively small performance gains. Second, it is not clear how the proposed mechanism affects DRAM reliability.

Compared to these prior works, our proposals identify and analyze the root cause of latency variation in detail, and provide mechanisms to maintain DRAM reliability while reducing DRAM latency.

Multi-Row Activation. Choi et al. [45] propose simultaneously activating multiple rows that have the same set of data. This leads to effectively higher amount of charge for the data, thereby reducing access latency to the rows. However, the drawback of this approach is that it potentially sacrifices DRAM capacity because multiple rows needs to have the same data to lower the latency to access the data.

Compared to this prior work, our proposals, AL-DRAM and AVA-DRAM, do not sacrifice capacity while reducing DRAM latency. This is because our proposals leverage the already-existing latency variation in DRAM.

3.3 Enabling More Parallelism in DRAM to Hide DRAM Latency

Prior works have focused on mitigating DRAM latency by enabling more parallelism in DRAM, with the goal of mitigating high DRAM latency. One major example is the innovation of having multiple DRAM "banks". Each bank has its own DRAM cells and access structures (e.g., row decoder and sense amplifiers). In a multi-bank DRAM architecture, data in different banks can be accessed simultaneously, reducing overall DRAM access latency. Unfortunately, increasing the number of banks requires additional area for additional access structures. There are two major approaches to enable more parallelism in DRAM at low cost, with the goal of hiding DRAM latency, *i)* enabling more accesses to occur in parallel, and *ii)* enabling accesses and refreshes to occur in parallel. The key distinguishing factor of the proposals in this dissertation is that they *directly* reduce DRAM latency instead of trying to overlap latency by exploiting parallelism.

Enabling Accesses to Different Subarrays in Parallel. Kim et al. [126] propose a new DRAM architecture (SALP) that enables the almost simultaneous access of many subarrays in a DRAM chip at low cost. The key observation behind this work is that each DRAM subarray (in a DRAM bank) has its own cell access structures. By decoupling global structures over subarrays in a bank, SALP enables each subarray to operate mostly in parallel, enabling more requests to be served in parallel (to be precise, in a pipelined manner). Half-DRAM [294] proposes a low-cost implementation of SALP.

Enabling More Ranks in a DRAM channel. Several prior works [11, 277, 298] propose partitioning a DRAM rank into multiple independent rank subsets that can be accessed in parallel [10]. All of these proposals reduce the frequency of row buffer conflicts and bank conflicts, thereby potentially improving DRAM access latency. However, these works

require more cycles to transfer data for a single access (due to the narrower bus allocated to each subset), which increases the latency of a single DRAM access.

Enabling Access and Refresh in Parallel. DRAM needs to periodically perform refresh operations to restore the lost charge in DRAM cells. During this refresh operation, DRAM cannot be accessed, causing delays in the servicing of requests. Chang et al. [40] propose a new mechanism that allows the servicing of data access requests and refresh operations in parallel by leveraging partial array refresh (which already exists in DRAM) and SALP. As a result, this work mitigates the negative impact of refresh to DRAM latency.

These prior works do *not* fundamentally reduce DRAM latency (instead, they exploit parallelism to tolerate the latency) and do *not* provide latency overlapping benefits when there are rank, bank, or subarray conflicts. In contrast, our proposals directly reduce DRAM latency even when there are rank, bank, or subarray conflicts. We believe that our proposals can be combined with these prior works in a synergistic manner to further hide memory access latency.

3.4 Memory Scheduling for Mitigating High DRAM Latency

To tolerate high DRAM latency, many memory scheduling techniques have been proposed in two major directions. The first approach is increasing more parallelism by exploiting DRAM interface. The second approach is prioritizing latency critical requests with the awareness of applications' characteristics. The key difference of our proposals from these prior approaches is that our proposals directly reduce raw DRAM latency, whereas memory scheduling can reduce queuing or DRAM produced overhead latencies.

Memory Scheduling for More Parallelism. Several prior works [137, 139, 182, 193, 229, 301] propose memory scheduling techniques that enable more parallelism. FR-FCFS (First-Ready First-Come First-Serve) [229, 301] prioritizes accesses to already activated row. Serving more requests from the already activated row amortizes the row activation and precharge latency over multiple requests. Parallelism-Aware Batch Scheduling

(PARBS) [182, 193] forms batches of requests and serves requests for a row together, maximizing parallelism in memory accesses. Lee et al. [139] proposes two memory scheduling mechanisms that maximize the bank-level parallelism (BLP). The first mechanism, BAPI (BLP-Aware Prefetch Issues), manages Miss Status Holding Registers (MSHRs) to issue prefetch requests to different banks for serving them in parallel. The second mechanism, BPMRI (BLP-Preserving Multi-core Request Issue), manages the memory controller's request buffers to issue requests from the same core together, reducing interference from other cores in multi-core systems. DRAM-aware writeback [137] maximizes the row-buffer locality by scheduling the write requests corresponding to the activated rows. The dirty-block indexing [238] reorganizes the dirty bits in on-chip cache to efficiently gather per-row dirty bit information, which helps to efficiently implement DRAM-aware scheduling mechanisms (e.g., DRAM-aware writeback).

Application-Aware Memory Scheduling. Several prior works [22, 25, 51, 63, 109, 124, 125, 181, 182, 183, 192, 193, 201, 258, 259, 260, 261, 272, 297] propose application-aware memory scheduling techniques that take into account the memory access characteristics of applications with the goal of tolerating high DRAM latency. The key idea is to prioritize latency-critical requests over other requests. PARBS [182, 193] batches the oldest requests from applications and ranks individual applications based on the number of outstanding requests from the application. Using this total rank order, PARBS prioritizes requests of applications that have low-memory-intensity. ATLAS [124] ranks individual applications based on the amount of long-term memory service each application receives, and prioritizes applications that receive low memory service. TCM (Thread cluster memory scheduling) [125] ranks individual applications by memory intensity such that low-memory-intensity applications are prioritized over high-memory-intensity applications. Prior works have exploited DRAM access scheduling in the controller (e.g. [22, 63, 84, 97, 192, 258, 259, 260, 261, 272, 297]), to mitigate inter-application interference, queueing latencies, and DRAM produced overheads in multi-core systems.

Compared to these prior works, our proposals directly reduce raw DRAM latency. Thus, our proposals are fundamentally different from these memory scheduling mechanisms. Our proposals can be also combined with these prior works to further reduce DRAM latency.

3.5 In-Memory Communication and Computation

Transferring significant amounts of data over the memory channel takes significant amount of time. Therefore, it can significantly delay other data transfers, leading to high DRAM latency. To mitigate the negative impacts of the memory channel contention, prior works have focused on reducing the memory channel contention by *i)* offloading bulk data movement to DRAM [41, 240, 241] or other parts of the memory system [146, 243], and *ii)* enabling in-memory computation [7, 8, 72, 74, 81, 130, 206, 239, 257, 293]. The key difference of our proposals from these prior works is that our proposals directly reduce raw DRAM latency. Our proposals can be also combined with these prior works.

Bulk In-Memory or Across-Memory Communication. Rowclone [240] proposes in-memory bulk data copy and initialization by enabling an extremely high bandwidth interface within a DRAM subarray. In a conventional DRAM-based memory system, to migrate a page data from the original row (source row) to the other row (destination row), the memory controller first reads the data from the source row, and writes the data back to the destination row, which takes very long time (high latency). Using direct wire connections between cells in a subarray (bitlines), Rowclone migrates data from a row to the other row directly with low latency. While Rowclone is promising to enable lower latency for page-granularity data transfer and initialization, Rowclone reduces latency only for page copy and initialization operations. Compared to this prior work, our proposals reduce DRAM latency in general, and thus can improve performance of most memory intensive applications.

LISA [41] enables fast in-memory bulk data movement across the subarrays in a DRAM bank by connecting adjacent sense amplifiers through bitlines. By placing a fast subarray (having short bitlines) and a slow subarray (having long bitlines) side by side and by enabling data migrations between these different latency subarrays, LISA enables latency heterogeneity

in a DRAM bank. Compared to this work that combines different-latency subarrays, our TL-DRAM substrate enables different-latency segments *within* a subarray, leading to a simpler and more efficient subarray architecture. Furthermore, AL-DRAM and AVA-DRAM do not change DRAM architecture, having more productivity compared to the prior work.

Prior works [146, 241] have focused on efficiently managing IO interfaces by either transferring data from different locations in a DRAM row in a gather-scatter manner [241] or enabling direct transfer through Dual-Port DRAM [146]. In this dissertation, we exploit latency heterogeneity in DRAM, reducing overall DRAM access latency. Thus, our proposals are fundamentally different from these works and can be combined with them to further reduce DRAM latency.

Bulk In-Memory Computation. Prior works have focused on offloading simple computations to the memory system, reducing the load on the memory channel (e.g., [7, 8, 72, 74, 81, 89, 130, 206, 239, 257, 293]). The key distinguishing factor of the proposals in this dissertation is the fact that they reduce raw DRAM latency while the most prior works in in-memory computation mainly reduce DRAM bandwidth requirement. Thus, our proposals can be combined with these prior works to further improve the memory system performance.

3.6 Enabling Heterogeneity in the System to Optimize System Design

Heterogeneity is a fundamental approach to approximate the benefits of multiple different components or technologies to enable the optimization of multiple metrics [185, 196]. Many recent works exploited heterogeneous designs in the system, including the memory system, for various purposes. These include works that combine multiple different memory technologies to achieve the benefits of multiple technologies [43, 134, 135, 149, 165, 167, 169, 214, 221, 224, 227, 291], works that combine multiple different interconnects to achieve both high-performance and energy-efficient interconnect designs [23, 24, 26, 77, 78, 79, 178, 184, 241], works that combine multiple different types of cores to achieve both high serial performance and high parallel throughput and high energy efficiency [20, 107, 108, 132, 263, 264], works

that combine multiple different types of execution paradigms within a single core to improve single-thread performance at high efficiency [71, 163], works that incorporate heterogeneity into thread/memory/interconnect scheduling, throttling and partitioning algorithms to optimize for multiple metrics [22, 25, 38, 51, 52, 53, 61, 65, 78, 79, 80, 109, 114, 125, 138, 164, 165, 183, 203, 204, 267, 272, 297], and works that exploit heterogeneous behavior due to process variation in the system to improve energy efficiency, performance or lifetime [28, 29, 30, 31, 32, 34, 119, 156]. In this dissertation, we exploit the notion of heterogeneity to reduce DRAM latency at low cost, an approach that is new.

3.7 Other Related Prior Works for Mitigating High DRAM Latency

There are many other methods proposed for reducing/hiding memory latency.

Caching and Paging Techniques. First, many prior works (e.g., [47, 99, 110, 118, 215, 218, 219, 220, 242, 244]) have proposed sophisticated cache management policies in the context of processor SRAM caches. These techniques are potentially applicable to our TL-DRAM to manage which rows get cached in the near segment. The TL-DRAM substrate also allows the hardware or the operating system to exploit the asymmetric latencies of the near and far segments in either hardware or software through intelligent page placement and migration techniques [36, 51, 183, 262, 274].

Aggressive Prefetching. Second, systems employ prefetching techniques to preload data from memory before it is needed [13, 35, 49, 50, 59, 62, 64, 65, 138, 188, 189, 191, 194, 195, 200, 208, 255], which may hide some memory access latencies. However, prefetching is not efficient for applications that have irregular access patterns, consumes memory system bandwidth, and increases interference in the memory system [62, 64, 65, 138].

Multithreading. Third, systems employ multithreading [249, 266]. Multithreading enables instruction-level parallelism, but, increases contention in the memory system [51, 62, 181, 193] and does not aid to increase single-core performance [107, 264].

Value Prediction. Fourth, systems can employ value prediction [154, 188, 190, 236, 269, 289, 290, 299] to predict results of instructions before loading required data from memory, which might hide some memory access latencies. However, incorrect value predictions require roll back, degrading performance, value prediction requires additional complexity, and many instructions are difficult to value-predict accurately.

Dynamic Instruction Reuse. Fifth, systems can employ dynamic instruction reuse techniques [16, 17, 21, 46, 48, 91, 251], which mostly reduces repeated computations and partially reduces the memory bandwidth consumption. However, these techniques do not reduce memory access latency.

Memory/Cache Compression. Sixth, systems employ memory/cache compression [6, 14, 15, 55, 58, 60, 209, 210, 211, 212, 245, 280, 295], which reduces the memory bandwidth consumption in the memory system and mitigates the memory channel contention. However, this approach does not reduce raw DRAM access latency.

Our proposals are largely orthogonal to all of these prior works and can be combined with these prior works to further improve system performance by reducing raw DRAM latency.

Chapter 4

Tiered-Latency DRAM: Lowering Latency by Modifying the Bitline Architecture

The capacity and cost-per-bit of DRAM have historically scaled to satisfy the needs of increasingly large and complex computer systems, while DRAM latency has remained almost constant, making memory latency the performance bottleneck in today's systems, as we demonstrated and discussed in Chapter 1. However, the high latency of commodity DRAM chips is in fact a *deliberate* trade-off made by DRAM manufacturers. While process technology scaling has enabled DRAM designs with both lower cost-per-bit *and* lower latency [98], DRAM manufacturers have usually sacrificed the latency benefits of scaling in order to achieve even lower cost-per-bit, as we explain below. Hence, while low-latency DRAM chips exist [129, 173, 235], their higher cost-per-bit relegates them to specialized applications such as high-end networking equipment that require very low latency even at a very high cost [270].

DRAM manufacturers trade latency for cost-per-bit by adjusting the length of these bitlines. Shorter bitlines (fewer cells connected to the bitline) constitute a smaller electrical load on the bitline, resulting in decreased latency, but require a larger number of sense amplifiers for a given DRAM capacity (Figure 4.1a), resulting in higher cost-per-bit. Longer bitlines (more cells connected to the bitline) require fewer sense amplifiers for a given DRAM

capacity (Figure 4.1b), reducing cost-per-bit, but impose a higher electrical load on the bitline, increasing latency. As a result, neither of these two approaches can optimize for both cost-per-bit and latency.
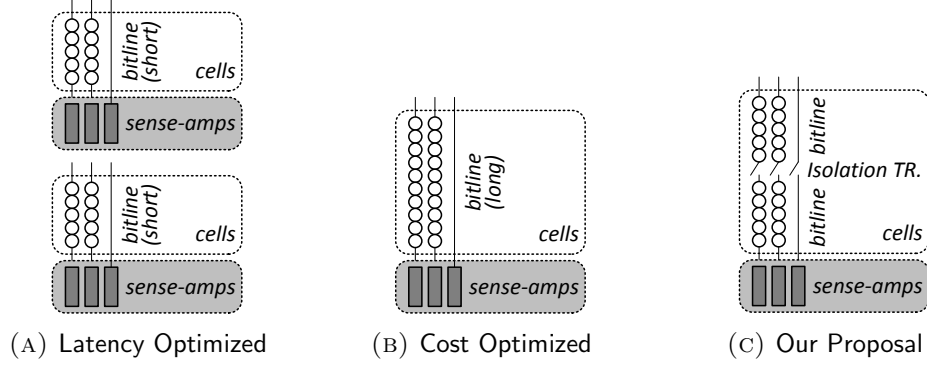
FIGURE 4.1: DRAM: Latency vs. Cost Optimized, Our Proposal

Our goal is to design a new DRAM architecture that provides low latency for the common case while still retaining a low cost-per-bit overall. Our proposal, which we call *Tiered-Latency DRAM*, is based on the key observation that *long bitlines are the dominant source of DRAM latency* [246].

Our key idea is to adopt long bitlines to achieve low cost-per-bit, while allowing their lengths to appear shorter in order to achieve low latency. In our mechanism (*Tiered-Latency DRAM*), each long bitline is split into two shorter segments using an *isolation transistor*, as shown in Figure 4.1c: the segment that is connected directly to the sense amplifier is called the *near segment*, whereas the other is called the *far segment*. To access a cell in the near segment, the isolation transistor is turned off, so that the cell and the sense amplifier see only the portion of the bitline corresponding to the near segment (i.e., reduced electrical load). Therefore, the near segment can be accessed quickly. On the other hand, to access a cell in the far segment, the isolation transistor is turned on to connect the entire length of the bitline to the sense amplifier. In this case, however, the cell and the sense amplifier see the full electrical load of the bitline in addition to the extra load of the isolation transistor. Therefore, the far segment can be accessed slowly.

4.1 Motivation: Short vs. Long Bitlines

The key parameter in the design of a DRAM subarray is the number of DRAM cells connected to each bitline (cells-per-bitline) – i.e., the number of DRAM rows in a subarray. This number directly affects the length of the bitline, which in turn affects both the access latency and the area of the subarray. As we describe in this section, the choice of the number of cells-per-bitline presents a crucial trade-off between the DRAM access latency and the DRAM die-size.

4.1.1 Latency Impact of Cells-per-Bitline

Every bitline has an associated parasitic capacitance whose value is proportional to the length of the bitline. This parasitic capacitance increases the subarray operation latencies: *i)* charge sharing, *ii)* sensing & amplification, and *iii)* precharging, which we discussed in Figure 2.4 in Chapter 2.

First, the bitline capacitance determines the bitline voltage after charge sharing. The larger the bitline capacitance, the closer its voltage will be to $\frac{1}{2}V_{DD}$ after charge sharing. Although this does not significantly impact the latency of charge sharing, this causes the sense amplifier to take longer to amplify the voltage to the final restored value (V_{DD} or *0*).

Second, in order to amplify the voltage perturbation on the bitline, the sense amplifier injects (or withdraws) charge into (or from) both the cell and the bitline. Since the sense amplifier can do so only at a fixed rate, the aggregate capacitance of the cell and the bitline determine how fast the bitline voltage reaches the *threshold* and the *restored* states (States ❸ and ❹ in Figure 2.4 in Chapter 2). A long bitline, which has a large parasitic capacitance, slows down the bitline voltage from reaching these states, thereby lengthening both t_{RCD} and t_{RAS}, respectively.

Third, to precharge the bitline, the sense amplifier drives the bitline voltage to the quiescent value of $\frac{1}{2}V_{DD}$. Again, a long bitline with a large capacitance is driven more slowly and hence has a large t_{RP}.

4.1.2 Die-Size Impact of Cells-per-Bitline

Since each cell on a bitline belongs to a row of cells (spanning horizontally across multiple bitlines), the number of cells-per-bitline in a subarray is equal to the number of rows-per-subarray. Therefore, for a DRAM chip with a given capacity (i.e., fixed total number of rows), one can either have many subarrays with short bitlines (Figure 4.1a) or few subarrays with long bitlines (Figure 4.1b). However, since each subarray requires its own set of sense amplifiers, the size of the DRAM chip increases along with the number of subarrays. As a result, for a given DRAM capacity, its die-size is inversely proportional to the number of cells-per-bitline (as a first-order approximation).

4.1.3 Trade-Off: Latency vs. Die-Size

From the above discussion, it is clear that a short bitline (fewer cells-per-bitline) has the benefit of lower subarray latency, but incurs a large die-size overhead due to additional sense amplifiers. On the other hand, a long bitline (more cells-per-bitline), as a result of the large bitline capacitance, incurs high subarray latency, but has the benefit of reduced die-size overhead. To study this trade-off quantitatively, we ran transistor-level circuit simulations based on a publicly available 55nm DRAM process technology [223]. Figure 4.2 shows the results of these simulations. Specifically, the figure shows the latency (t_{RCD} and t_{RC}) and the die-size for different values of cells-per-bitline. The figure clearly shows the above described trade-off between DRAM access latency and DRAM die-size.

As Figure 4.2 shows, existing DRAM architectures are either optimized for die-size (commodity DDR3 [179, 232]) and are thus low cost but high latency, or optimized for latency (RLDRAM [173], FCRAM [235]) and are thus low latency but high cost. **Our goal** is to design a DRAM architecture that achieves the best of both worlds – i.e., low access latency and low cost.

† RLDRAM bitline length is estimated from its latency and die-size [116, 173].
† The reference DRAM is 55nm 2GB DDR3 [223].
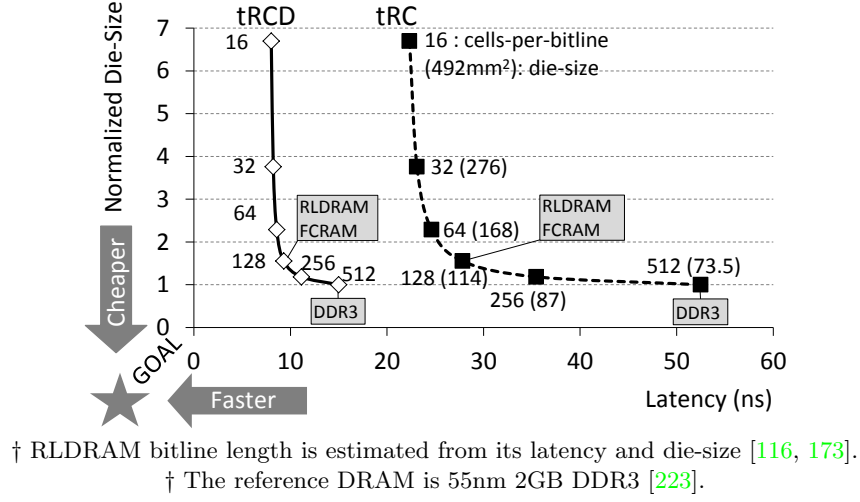
Figure 4.2: Bitline Length: Latency vs. Die-Size

4.2 Tiered-Latency DRAM (TL-DRAM)

To obtain both the latency advantages of short bitlines and the cost advantages of long bitlines, we propose the *Tiered-Latency DRAM* (TL-DRAM) architecture, as shown in Figure 4.3. The key idea of TL-DRAM is to introduce an *isolation transistor* that divides a long bitline into two segments: the *near segment*, connected directly to the sense-amplifier, and the *far segment*, connected through the isolation transistor. Unless otherwise stated, throughout the following discussion, we assume, without loss of generality, that the isolation transistor divides the bitline such that length of the near and far segments is 128 cells and 384 cells (=512-128), respectively. (Section 4.2.1 discusses the latency sensitivity to the segment lengths.)
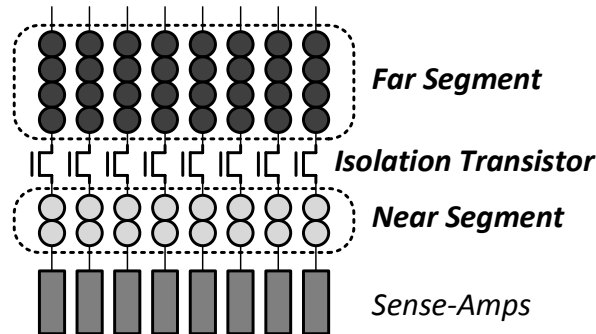
Figure 4.3: TL-DRAM: Near vs. Far Segments

4.2.1 Latency Analysis (Overview)

The primary role of the isolation transistor is to electrically decouple the two segments from each other. As a result, the effective bitline length (and also the effective bitline capacitance) as seen by the cell and sense-amplifier is changed. Correspondingly, the latency to access a cell is also changed – albeit differently depending on whether the cell is in the near or the far segment (Table 4.1), as will be explained next.

	Near Segment (128 cells)	Far Segment (384 cells)
t_{RCD}	**Reduced** (15ns $\rightarrow$ 9.3ns)	**Reduced** (15ns $\rightarrow$ 13.2ns)
t_{RC}	**Reduced** (52.5ns $\rightarrow$ 27.8ns)	Increased (52.5ns $\rightarrow$ 64.1ns)

TABLE 4.1: Segmented Bitline: Effect on Latency

Near Segment. When accessing a cell in the near segment, the isolation transistor is turned off, disconnecting the far segment (Figure 4.4a). Since the cell and the sense-amplifier see only the reduced bitline capacitance of the shortened near segment, they can drive the bitline voltage more easily. In other words, for the same amount of charge that the cell or the sense-amplifier injects into the bitline, the bitline voltage is higher for the shortened near segment compared to a long bitline. As a result, the bitline voltage reaches the *threshold* and *restored* states (Figure 2.4 in Chapter 2) more quickly, such that t_{RCD} and t_{RAS} for the near segment is significantly reduced. Similarly, the bitline can be precharged to $\frac{1}{2}V_{DD}$ more quickly, leading to a reduced t_{RP}. Since t_{RC} is defined as the sum of t_{RAS} and t_{RP} (Section 2.3 in Chapter 2), t_{RC} is reduced as well.

Far Segment. On the other hand, when accessing a cell in the far segment, the isolation transistor is turned on to connect the entire length of the bitline to the sense-amplifier. In this case, however, the isolation transistor acts like a resistor inserted between the two segments (Figure 4.4b).

When the sense-amplifier is turned on during activation (as explained in Section 2.4 in Chapter 2), it begins to drive charge onto the bitline. In commodity DRAM, this charge is

36

(A) Near Segment Access   (B) Far Segment Access
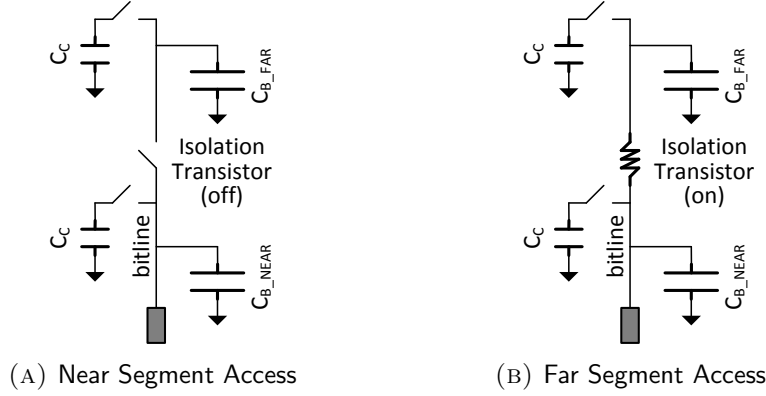
FIGURE 4.4: Circuit Model of Segmented Bitline

spread across the large capacitance of the entire long bitline. However, in TL-DRAM, the resistance of the isolation transistor limits how quickly charge flows to the far segment, such that the reduced capacitance of the shortened near segment is charged more quickly than that of the far segment. This has two key consequences. First, because the near segment capacitance is charged quickly, the near segment voltage rises more quickly than the bitline voltage in commodity DRAM. As a result, the near segment voltage more quickly reaches $\frac{3}{4}V_{DD}$ (*threshold* state, Section 2.4 in Chapter 2) and, correspondingly, the sense-amplifier more quickly detects the binary data value of '1' that was stored in the far segment cell. That is why t_{RCD} is lower in TL-DRAM than in commodity DRAM even for the far segment. Second, because the far segment capacitance is charged more slowly, it takes *longer* for the far segment voltage — and hence the cell voltage — to be *restored* to V_{DD} or *0*. Since t_{RAS} is the latency to reach the *restored* state (Section 2.4 in Chapter 2), t_{RAS} is increased for cells in the far segment. Similarly, during precharging, the far segment voltage reaches $\frac{1}{2}V_{DD}$ more slowly, for an increased t_{RP}. Since t_{RAS} and t_{RP} both increase, their sum t_{RC} also increases.

Sensitivity to Segment Length. The lengths of the two segments are determined by where the isolation transistor is placed on the bitline. Assuming that the number of cells per bitline is fixed at 512 cells, the near segment length can range from as short as a single cell to as long as 511 cells. Based on our circuit simulations, Figure 4.5a and Figure 4.5b plot the latencies of the near and far segments as a function of their length, respectively. For

reference, the rightmost bars in each figure are the latencies of an unsegmented long bitline whose length is 512 cells. From these figures, we draw three conclusions. First, the shorter the near segment, the lower its latencies (t_{RCD} and t_{RC}). This is expected since a shorter near segment has a lower effective bitline capacitance, allowing it to be driven to target voltages more quickly. Second, the longer the far segment, the lower the far segment's t_{RCD}. Recall from our previous discussion that the far segment's t_{RCD} depends on how quickly the near segment (not the far segment) can be driven. A longer far segment implies a shorter near segment (lower capacitance), and that is why t_{RCD} of the far segment decreases. Third, the shorter the far segment, the smaller its t_{RC}. The far segment's t_{RC} is determined by how quickly it reaches the full voltage (V_{DD} or 0). Regardless of the length of the far segment, the current that trickles into it through the isolation transistor does not change significantly. Therefore, a shorter far segment (lower capacitance) reaches the full voltage more quickly.

(A) Cell in Near Segment   (B) Cell in Far Segment
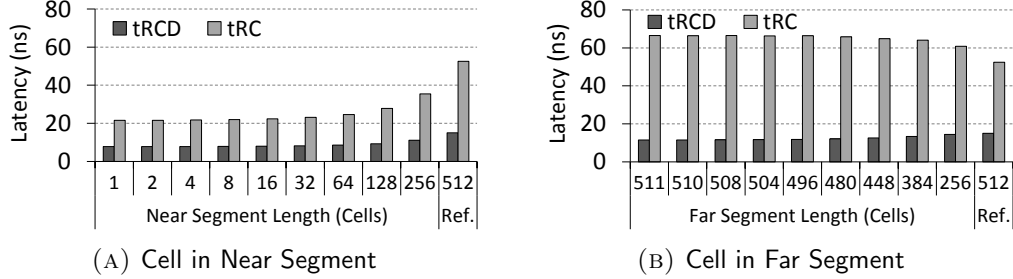
FIGURE 4.5: Latency Analysis

4.2.2 Latency Analysis (Circuit Evaluation)

We model TL-DRAM in detail using SPICE simulations. Simulation parameters are mostly derived from a publicly available 55nm DDR3 2Gb process technology file [223] which includes information such as cell and bitline capacitance and resistance, physical floorplanning, and transistor dimensions. Transistor device characteristics were derived from [198] and scaled to agree with [223].

Figure 4.6 and Figure 4.7 show the bitline voltages during activation and precharging respectively. The x-axis origin (time 0) in the two figures correspond to when the subarray

receives the `ACTIVATION` or the `PRECHARGE` command, respectively. In addition to the voltages of the segmented bitline (near and far segments), the figures also show the voltages of two unsegmented bitlines (short and long) for reference.

(A) Cell in Near Segment (128 cells)

(B) Cell in Far Segment (384 cells)
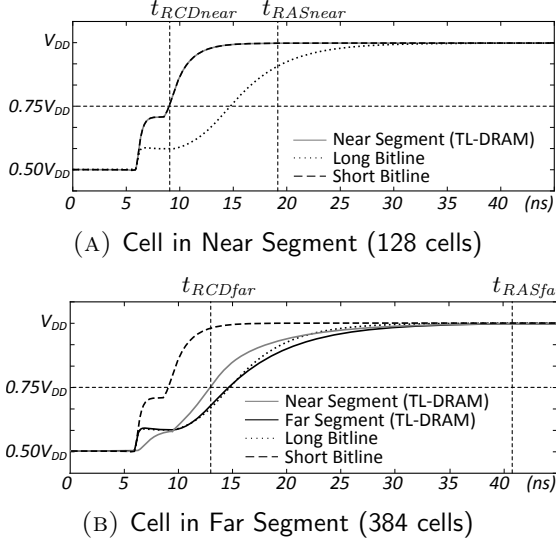
FIGURE 4.6: Activation: Bitline Voltage

(A) Cell in Near Segment

(B) Cell in Far Segment
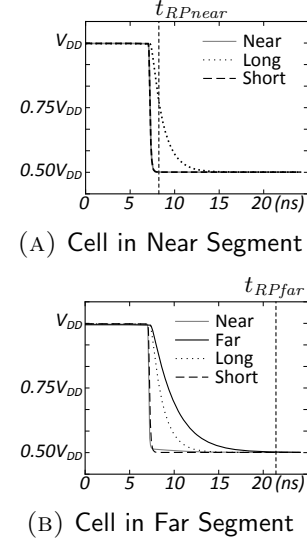
FIGURE 4.7: Precharging

Activation (Figure 4.6). First, during an access to a cell in the near segment (Figure 4.6a), the far segment is disconnected and is floating (hence its voltage is not shown). Due to the reduced bitline capacitance of the near segment, its voltage increases almost as quickly as the voltage of a short bitline (the two curves are overlapped) during both charge sharing and sensing & amplification. Since the near segment voltage reaches $\frac{3}{4}V_{DD}$ and V_{DD} (the *threshold* and *restored* states) quickly, its t_{RCD} and t_{RAS}, respectively, are significantly reduced compared to a long bitline. Second, during an access to a cell in the far segment (Figure 4.6b), we can indeed verify that the voltages of the near and the far segments increase at different rates due to the resistance of the isolation transistor, as previously explained. Compared to a long bitline, while the near segment voltage reaches $\frac{3}{4}V_{DD}$ more quickly, the far segment voltage reaches V_{DD} more slowly. As a result, t_{RCD} of the far segment is reduced while its t_{RAS} is increased.

Precharging (Figure 4.7). While precharging the bitline after accessing a cell in the near segment (Figure 4.7a), the near segment reaches $\frac{1}{2}V_{DD}$ quickly due to the smaller capacitance,

almost as quickly as the short bitline (the two curves are overlapped). On the other hand, precharging the bitline after accessing a cell in the far segment (Figure 4.7b) takes longer compared to the long bitline baseline. As a result, t_{RP} is reduced for the near segment and increased for the far segment.

4.2.3 Die-Size Analysis

Adding an isolation transistor to the bitline increases only the height of the subarray and not the width [115, 116, 151, 231]. Without the isolation transistor, the height of a baseline subarray is equal to the sum of height of the cells and the sense-amplifier. In the following analysis, we use values from the Rambus power model [223].[1] The sense amplifier and the isolation transistor are respectively 115.2x and 11.5x taller than an individual cell. For a subarray with 512 cells, the overhead of adding a single isolation transistor is $\frac{11.5}{115.2+512} =$ 1.83%.

Until now we have assumed that all cells of a DRAM row are connected to the same row of sense-amplifiers. However, in the open bitline scheme, the sense-amplifiers are twice as wide as an individual cell. Therefore, in practice, only every other bitline (cell) is connected to a bottom row of sense-amplifiers. The remaining bitlines are connected to the another row of sense-amplifiers of the vertically adjacent (top) subarray. This allows for tighter packing of DRAM cells within a subarray. As a result, each subarray requires two sets of isolation transistors as shown in Figure 4.8a. Therefore, the increase in subarray area due to the two isolation transistors is 3.66%. Once we include the area of the peripheral and I/O circuitry which does not change due to the addition of the isolation transistors, the resulting DRAM die-size area overhead is 3.15%.

We plot the near segment and the far segment in Figure 4.8b. The portion directly connected to either the top sense-amplifiers and the bottom sense-amplifiers form the near segment and the remained portion is the far segment. To access the far segment, the isolation transistors are turned on and activated an wordline of the far segment, which are similar to the conventional DRAM's row access. To access the near segment, isolation transistors are

---

[1]We expect the values to be of similar orders of magnitude for other designs.

(A) Isolation Transistor in Open Bitline Scheme (B) Far & Near Segment in Open Bitline Scheme
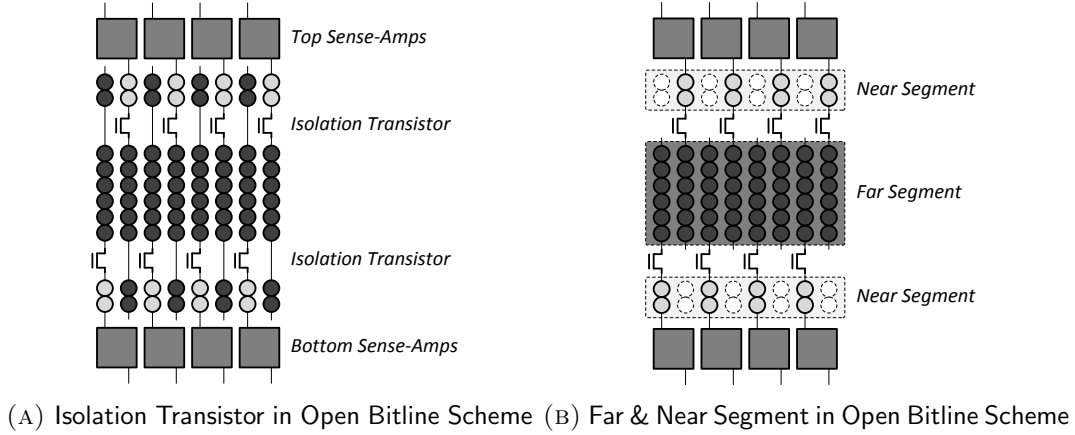
FIGURE 4.8: Tiered-Latency DRAM Integration in Open Bitline Scheme

turned off, leading to connecting only the two near segments are connected to sense-amplifiers (light grey portion in Figure 4.8b). In this case, half of the cells in the near segments can not be accessed since the dotted cells in Figure 4.8b are not connected to sense-amplifiers. As a result, in the open bitline scheme, half of the capacity in the near segment can not be available to use, leading to higher area overhead. For example, TL-DRAM with a near segment size of 32 rows loses 3.125% capacity, leading to 6.275% area overhead in total.

4.2.4 Enabling Inter-Segment Data Transfer

One way of exploiting TL-DRAM's asymmetric latencies is to use the near segment as a cache to the far segment. Compared to the far segment, the near segment is smaller and faster. Therefore, frequently-used or latency-critical data can benefit significantly from being placed in the near segment as opposed to the far segment. The problem lies in enabling an efficient way of transferring (copying or migrating) data between the two segments. Unfortunately, in existing DRAM chips, even to transfer data from one row to another row within the same subarray, the data must be read out externally to the DRAM controller and then written back to the chip. This wastes significant amounts of power and bandwidth on the memory bus (that connects the DRAM chip to the DRAM controller) and incurs large latency.

In TL-DRAM, data transfer between the two segments occurs entirely within DRAM without involving the external memory bus. TL-DRAM leverages the fact that the bitline

41

itself is essentially a bus that connects to all cells in both the near and far segments. As an example, let us assume that we are accessing a cell in the far segment (transfer source). When the bitline has reached the *restored* state (Sec 2.4), the data in the cell is fully copied onto the bitline. Normally, at this point, the DRAM controller would issue a `PRECHARGE` to clear the bitline voltage to $\frac{1}{2}V_{DD}$. Instead, TL-DRAM allows the DRAM controller to issue another `ACTIVATION`, this time to a cell in the near segment (transfer destination). Since the bitline is at a full voltage, the bitline drives the near segment cell so that data is copied into it. According to our simulations, writing into the destination cell takes about 4ns. In fact, the destination cell can be connected to the bitline even before the source cell has reached the *restored* state, thereby overlapping the copying latency with the t_{RAS} latency. More generally, the source and the destination can be any two cells connected to the same bitline, regardless of which segment they lie on.

4.3 Leveraging the TL-DRAM Substrate

One simple way of leveraging the TL-DRAM substrate is to use the near segment as a hardware-managed cache for the far segment. In this approach, the memory controller does not expose the near segment capacity to the operating system (OS). While this approach reduces the overall available memory capacity, it keeps both the hardware and the software design simple. Another alternative approach is to expose the near segment capacity to the OS. As we will describe in Section 4.3.2, effectively exploiting the TL-DRAM substrate using this alternate approach may slightly increase the complexity of the hardware or the software. We now describe our different mechanisms to leverage the TL-DRAM substrate.

4.3.1 Near Segment as an OS-Transparent Hardware-Managed Cache

We describe three different mechanisms that use the near segment as a hardware-managed cache to the far segment. In all three mechanisms, the memory controller tracks the rows in the far segment that are cached in the near segment (for each subarray). The three

mechanisms differ in 1) when they cache a far-segment row in the near segment, and 2) when they evict a row from the near segment.

Mechanism 1: Simple Caching (SC). Our first mechanism, *Simple Caching* (SC), utilizes the near segment as an LRU cache to the far segment. Under this mechanism, the DRAM controller categorizes a DRAM access into one of three cases: *i)* sense-amplifier hit: the corresponding row is already activated; *ii)* near segment hit: the row is already cached in the near segment; and *iii)* near segment miss: the row is not cached in the near segment. In the first case, sense-amplifier hit (alternatively, row-buffer hit), the access is served directly from the row-buffer. Meanwhile, the LRU-ordering of the rows cached in the near segment remains unaffected. In the second case, near segment hit, the DRAM controller quickly activates the row in the near segment, while also updating it as the MRU row. In the third case, near segment miss, the DRAM controller checks whether the LRU row (eviction candidate) in the near segment is dirty. If so, the LRU row must first be copied (or written back) to the far segment using the transfer mechanism described in Section 4.2.4. Otherwise, the DRAM controller directly activates the far segment row (that needs to be accessed) and copies (or caches) it into the near segment and updates it as the MRU row.

Mechanism 2: Wait-Minimized Caching (WMC). When two accesses to two different rows of a subarray arrive almost simultaneously, the first access delays the second access by a large amount, t_{RC}. Since the first access causes the second access to *wait* for a long time, we refer to the first access as a *wait-inducing access.* Assuming both rows are in the far segment, the latency at the subarray experienced by the second access is $t_{RCfar} + t_{RCDfar}$ (77.3ns). Such a large latency is mostly due to the wait caused by the first access, t_{RCfar} (64.1ns). Hence, it is important for the second access that the wait is minimized, which can be achieved by caching the *first* accessed data in the near segment. By doing so, the wait is significantly reduced from t_{RCfar} (64.1ns) to t_{RCnear} (27.8ns). In contrast, caching the *second* row is not as useful, since it yields only a small latency reduction from t_{RCDfar} (13.2ns) to $t_{RCDnear}$ (9.3ns).

Our second mechanism, *Wait-Minimized Caching* (WMC), caches only *wait-inducing rows.* These are rows that, while they are accessed, cause a large wait (t_{RCfar}) for the

next access to a different row. More specifically, a row in the far segment is classified as wait-inducing if the next access to a different row arrives while the row is still being activated. WMC operates similarly to our SC mechanism except for the following differences. First, WMC copies a row from the far segment to the near segment *only if the row is wait-inducing.* Second, instead of evicting the LRU row from the near segment, WMC evicts the *least-recently wait-inducing* row. Third, when a row is accessed from the near segment, it is updated as the *most-recently wait-inducing* row only if the access would have caused the next access to wait had the row been in the far segment. The memory controller is augmented with appropriate structures to keep track of the necessary information to identify wait-inducing rows (details omitted due to space constraints).

Mechanism 3: Benefit-Based Caching (BBC). Accessing a row in the near segment provides two benefits compared to accessing a row in the far segment: 1) reduced t_{RCD} (faster access) and 2) reduced t_{RC} (lower wait time for the subsequent access). Simple Caching (SC) and Wait-Minimized Caching (WMC) take into account only one of the two benefits. Our third mechanism, *Benefit-Based Caching* (BBC) explicitly takes into account both benefits of caching a row in the near segment. More specifically, the memory controller keeps track of a *benefit* value for each row in the near segment. When a near segment row is accessed, its benefit is incremented by the number of DRAM cycles saved due to reduced access latency and reduced wait time for the subsequent access. When a far-segment row is accessed, it is immediately promoted to the near segment, replacing the near-segment row with the least benefit. To prevent benefit values from becoming stale, on every eviction, the benefit for every row is halved. (Implementation details are omitted due to space constraints.)

4.3.2 Exposing Near Segment Capacity to the OS

Our second approach to leverage the TL-DRAM substrate is to expose the near segment capacity to the operating system. Note that simply replacing the conventional DRAM with our proposed TL-DRAM can potentially improve system performance due to the reduced $t_{RCDnear}$, t_{RCDfar}, and t_{RCnear}, while not reducing the available memory capacity. Although this mechanism incurs no additional complexity at the memory controller or the operating

system, we find that the overall performance improvement due to this mechanism is low. To better exploit the benefits of the low access latency of the near segment, frequently accessed pages should be mapped to the near segment. This can be done by the hardware or by the OS. To this end, we describe two different mechanisms.

Exclusive Cache. In this mechanism, we use the near segment as an *exclusive* cache to the rows in the far segment. The memory controller uses one of the three mechanisms proposed in Section 4.3.1 to determine caching and eviction candidates. To cache a particular row, the data of that row is *swapped* with the data of the row to be evicted from the near segment. For this purpose, each subarray requires a *dummy-row* (D-row). To swap the data of the to-be-cached row (C-row) and to-be-evicted row (E-row), the memory controller simply performs the following three migrations:

$$\text{C-row} \rightarrow \text{D-row} \qquad \text{E-row} \rightarrow \text{C-row} \qquad \text{D-row} \rightarrow \text{E-row}$$

The exclusive cache mechanism provides almost full main memory capacity (except one dummy row per subarray, $< 0.2\%$ loss in capacity) at the expense of two overheads. First, since row swapping changes the mappings of rows in both the near and the far segment, the memory controller must maintain the mapping of rows in both segments. Second, each swapping requires three migrations, which increases the latency of caching.

Profile-Based Page Mapping. In this mechanism, the OS controls the virtual-to-physical mapping to map frequently accessed pages to the near segment. The OS needs to be informed of the bits in the physical address that control the near-segment/far-segment mapping. Information about frequently accessed pages can either be obtained statically using compiler-based profiling, or dynamically using hardware-based profiling. In our evaluations, we show the potential performance improvement due to this mechanism using hardware-based profiling. Compared to the exclusive caching mechanism, this approach requires much lower hardware storage overhead.

4.4 Implementation Details & Further Analysis

4.4.1 Near Segment Row Decoder Wiring

To avoid the need for a large, monolithic row address decoder at each subarray, DRAM makes use of *predecoding*. Outside the subarray, the row address is divided into M sets of bits. Each set, N bits, is decoded into 2^N wires, referred to as $N : 2^N$ predecoding.[2] The input to each row's wordline driver is then simply the logical AND of the M wires that correspond to the row's address. This allows the per-subarray row decoding logic to be simple and compact, at the cost of increasing the wiring overhead associated with row decoding at each subarray. As a result, wiring overhead dominates the cost of row decoding.

The inter-segment data transfer mechanism described in Section 4.2.4 requires up to two rows to be activated in the same subarray at once, necessitating a second row decoder. However, since one of the two activated rows is always in the near segment, this second row decoder only needs to address rows in the near segment. For a near segment with 32 rows, a scheme that splits the 5 ($\log_2 32$) near segment address bits into 3-bit and 2-bit sets requires 12 additional wires to be routed to each subarray (3:8 predecoding + 2:4 predecoding). The corresponding die-size penalty is 0.33%, calculated based on the total die-size and wire-pitch derived from Vogelsang [223, 275].

4.4.2 Additional Storage in DRAM Controller

The memory controller requires additional storage to keep track of the rows that are cached in the near segment. In the inclusive caching mechanisms, each subarray contains a near segment of length N, serving as an N-way (fully-associative) cache of the far segment of length F. Therefore, each near-segment row requires a $\lceil \log_2 F \rceil$-bit tag. Hence, each subarray requires $N \lceil \log_2 F \rceil$ bits for tag storage, and a system with S subarrays requires $SN \lceil \log_2 F \rceil$ bits for tags. In the exclusive caching mechanism, row swapping can lead to any physical page within a subarray getting mapped to any row within the subarray. Hence, each row within the subarray requires the tag of the physical page whose data is stored in that row. Thus, each

---

[2]N may differ between sets.

row in a subarray requires a $\lceil \log_2(N + F) \rceil$-bit tag, and a system with S subarrays requires $S(N + F)\lceil \log_2(N + F) \rceil$ bits for tags. In the system configuration we evaluate (described in Section 5.6), $(N, F, S) = (32, 480, 256)$, the tag storage overhead is 9 KB for inclusive caching and 144 KB for exclusive caching.

SC and WMC additionally require $\log_2 N$ bits per near segment row for replacement information ($SN \log_2 N$ bits total), while our implementation of BBC uses an 8-bit benefit field per near segment row ($8SN$ bits total). For our evaluated system, these are overheads of 5 KB and 8 KB respectively.

4.4.3 Energy Consumption Analysis

The non-I/O power consumption of the DRAM device can be broken into three dominant components: *i)* raising and lowering the wordline during `ACTIVATION` and `PRECHARGE`, *ii)* driving the bitline during `ACTIVATION`, and *iii)* transferring data from the sense amplifiers to the peripherals. The first two of these components differ between a conventional DRAM and TL-DRAM, for two reasons:

Reduced Power Due to Reduced Bitline Capacitance in Near Segment. The energy required to restore a bitline is proportional to the bitline's capacitance. In TL-DRAM, the near segment has a lower capacitance than that of a conventional DRAM's bitline, resulting in decreased power consumption.

Additional Power Due to Isolation Transistors. The additional power required to control the isolation transistors when accessing the far segment is approximately twice that of raising the wordline, since raising the wordline requires driving one access transistor per bitline, while accessing the far segment requires driving two isolation transistors per bitline (Sec 4.2.3).

Using DRAM power models from Rambus and Micron [175, 223, 275], we estimate power consumption of TL-DRAM and conventional DRAM in Figure 4.9. Note that, while the near segment's power consumption increases with the near segment length, the far segment's power does not change as long as the total bitline length is constant. Our evaluations in Section 4.6 take these differences in power consumption into account.

(A) Power Consumption for ACTIVATE

(B) Power Consumption for PRECHARGE
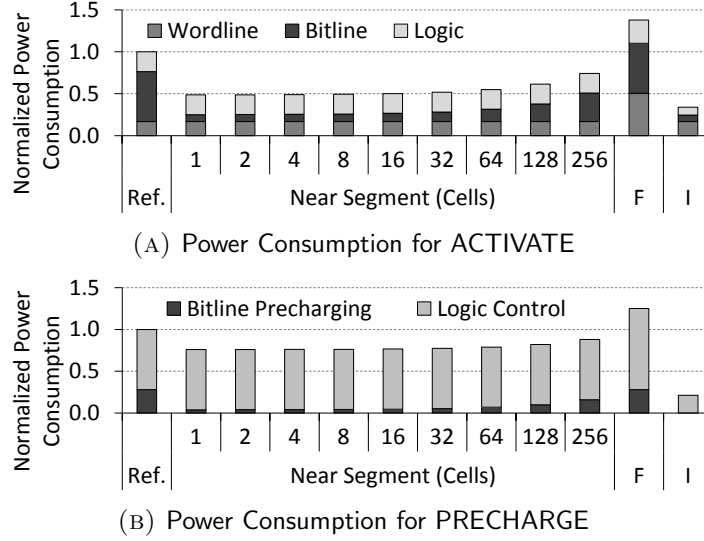
† Ref.: Long Bitline, F: Far Segment, I: Inter-Segment Data Transfer

FIGURE 4.9: Power Consumption vs. Bitline Length

4.4.4 More Tiers

Although we have described TL-DRAM with two segments per subarray, one can imagine adding more isolation transistors to divide a subarray into more tiers, each tier with its own latency and energy consumption characteristics. As a case study, we evaluated the latency characteristics of TL-DRAM with three tiers per subarray (near/middle/far with 32/224/256 cells). The normalized t_{RCD} and t_{RC} of the near/middle/far segments are 54.8%/70.7%/104.1% and 44.0%/77.8%/156.9%. While adding more tiers can enable more fine-grained caching and partitioning mechanisms, they come with the additional area cost of 3.15% per additional tier, additional energy consumption to control the multiple isolation transistors, and logic complexity in DRAM and the DRAM controller.

4.5 Evaluation Methodology

Simulators. We use a modified version of Ramulator [128], a fast cycle-accurate DRAM simulator that is available publicly [123, 127], and Ramulator releases an open-source implementation of TL-DRAM. We use Ramulator combined with a cycle-level x86 multi-core

simulator. We use Ramulator as part of a cycle-level in-house x86 multi-core simulator, whose front-end is based on Pin [162].

System Configuration. The evaluated system is configured as shown in Table 4.2. Unless otherwise stated, the evaluated TL-DRAM has a near segment size of 32 rows.

Processor	5.3 GHz, 3-wide issue, 8 MSHRs/core, 128-entry instruction window
Last-Level Cache	64B cache line, 16-way associative, 512kB private cache slice/core
Memory Controller	64/64-entry read/write request queue, row-interleaved mapping, closed-page policy, FR-FCFS scheduling [229]
DRAM	2GB DDR3-1066, 1/2/4 channel (@1-core/2-core/4-core), 1 rank/channel, 8 banks/rank, 32 subarrays/bank, 512 rows/bitline t_{RCD} (unsegmented): 15.0ns, t_{RC} (unsegmented): 52.5ns
TL-DRAM	32 rows/near segment, 480 rows/far segment t_{RCD} (near/far): 8.2/12.1ns, t_{RC} (near/far): 23.1/65.8ns

TABLE 4.2: Evaluated System Configuration

Parameters. DRAM latency is as calculated in Section 4.2.2. DRAM dynamic energy consumption is evaluated by associating an energy cost with each DRAM command, derived using the tools [175, 223, 275] and the methodology given in Section 4.4.3.

Benchmarks. We use 32 benchmarks from SPEC CPU2006, TPC [271], STREAM [4] and a *random* microbenchmark similar in behavior to GUPS [88]. We classify benchmarks whose performance is significantly affected by near segment size as *sensitive*, and all other benchmarks as *non-sensitive*. For single-core sensitivity studies, we report results that are averaged across all 32 benchmarks. We also present multi-programmed multi-core evaluations in Section 4.6.6. For each multi-core workload group, we report results averaged across 16 workloads, generated by randomly selecting from specific benchmark groups (`sensitive`, `high`, `low`, and `random`).

Simulation and Evaluation. We simulate each benchmark for 100 million instructions. For multi-core evaluations, we ensure that even the slowest core executes 100 million

instructions, while other cores continue to exert pressure on the memory subsystem. To measure performance, we use instruction throughput (IPC) for single-core systems and *weighted speedup* [250] for multi-core systems.

4.6 Results

4.6.1 Single-Core Results: Inclusive Cache

Figure 4.10 compares our proposed TL-DRAM based mechanisms (SC, WMC, and BBC) to the baseline with conventional DRAM. For each benchmark, the figure plots four metrics: *i)* performance improvement of the TL-DRAM based mechanisms compared to the baseline, *ii)* the number of misses per instruction in the last-level cache, *iii)* the fraction of accesses that are served at the row buffer, near segment and the far segment using each of the three proposed mechanisms, and *iv)* the power consumption of the TL-DRAM based mechanisms relative to the baseline. We draw three conclusions from the figure.
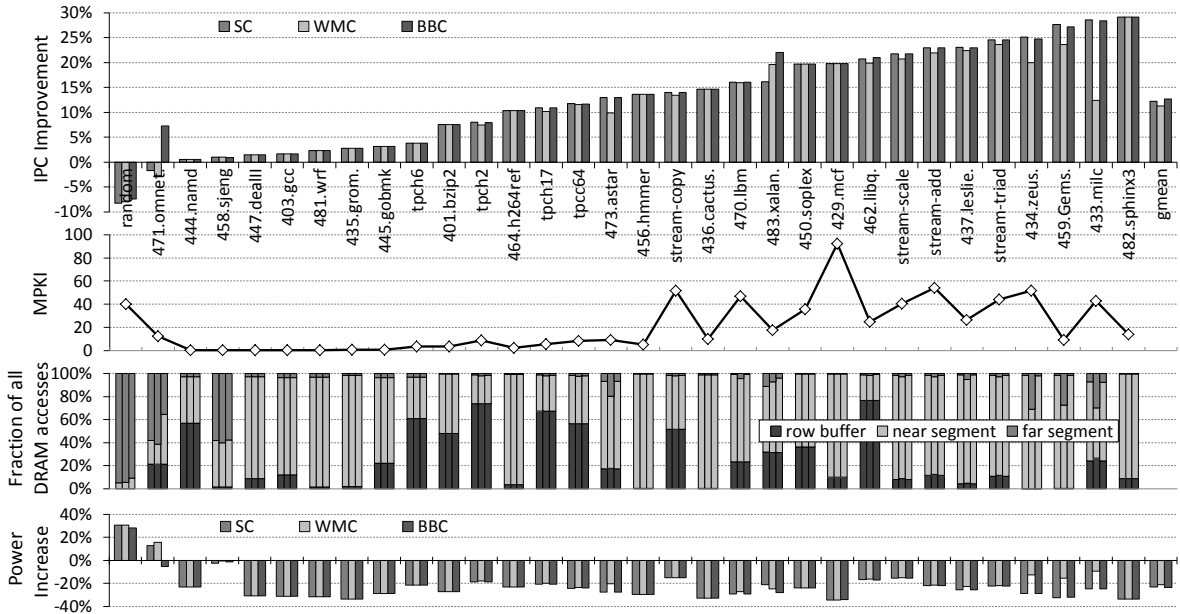
FIGURE 4.10: Single-core: IPC improvement, LLC MPKI, Fraction of accesses serviced at row buffer/near segment/far segment, Power consumption

First, for most benchmarks, all of our proposed mechanisms improve performance significantly compared to the baseline. On average, SC, WMC, and BBC improve performance by

12.3%, 11.3% and 12.8%, respectively, compared to the baseline. As expected, performance benefits are highest for benchmarks with high memory intensity (MPKI: Last-Level Cache Misses-Per-Kilo-Instruction) and high near-segment hit rate.

Second, all three of our proposed mechanisms perform similarly for most benchmarks. However, there are a few benchmarks where WMC significantly degrades performance compared to SC. This is because WMC only caches wait-inducing rows, ignoring rows with high reuse that cause few conflicts. BBC, which takes both reuse and wait into account, outperforms both SC and WMC. For example, BBC significantly improves performance compared to SC ($> 8\%$ for *omnetpp*, $> 5\%$ for *xalancbmk*) by reducing wait and compared to WMC ($> 9\%$ for *omnetpp*, $> 2\%$ for *xalancbmk*) by providing more reuse.

Third, BBC degrades performance only for the microbenchmark *random*. This benchmark has high memory intensity (MPKI = 40) and very low reuse (near-segment hit rate $< 10\%$). These two factors together result in frequent bank conflicts in the far segment. As a result, most requests experience the full penalty of the far segment's longer t_{RC}. We analyze this microbenchmark in detail in Section 4.6.3.

Power Analysis. As described in Section 4.4.3, power consumption is significantly lower for near-segment accesses, but higher for far-segment accesses, compared to accesses in a conventional DRAM. As a result, TL-DRAM achieves significant power savings when most accesses are to the near segment. The bottom-most plot of Figure 4.10 compares the power consumption of our TL-DRAM-based mechanisms to that of the baseline. Our mechanisms produce significant power savings (23.6% for BBC vs. baseline) on average, since over 90% of accesses hit in the row buffer or near segment for most benchmarks.

4.6.2 Effect of Near Segment Length: Inclusive Cache

The number of rows in each near segment presents a trade-off, as increasing the near segment's size increases its capacity but also increases its access latency. Figure 4.11 shows the average performance improvement of our proposed mechanisms over the baseline as we vary the near segment size. As expected, performance initially improves as the number of rows in the near segment is increased due to increased near segment capacity. However, increasing the number

of rows per near segment beyond 32 reduces the performance benefits due to the increased near segment access latency.
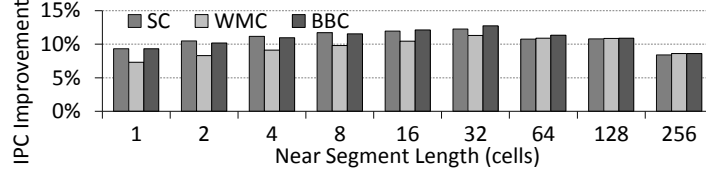
FIGURE 4.11: Varying Near Segment Capacity (Inclusive Cache)

In Figure 4.12, we categorize benchmarks in four groups based on the performance characteristics with varying the near segment length. We first divide into two groups based on the sensitivity of the performance to the near segment length. Some benchmarks increase performance (IPC) with increasing near segment length (especially, for relatively short near segments, e.g., 1 – 64 rows in the near segment out of 512 rows in a subarray), which we call `sensitive` group (e.g., *mcf* and *xalancbmk*). This is because these benchmarks have relatively large working set (the number of DRAM rows which need to be accessed in a short period of the execution time) compared to the near segment capacity. Therefore, these benchmarks show less performance at short near segment length (e.g., 1 or 2 rows for each near segment over 512 rows in a subarray) because the working set of each benchmark does not fit into the near segment capacity, requiring frequent data migration between the near and far segments. Increasing the near segment length leads to preserve more rows in the near segment, resulting in better performance.
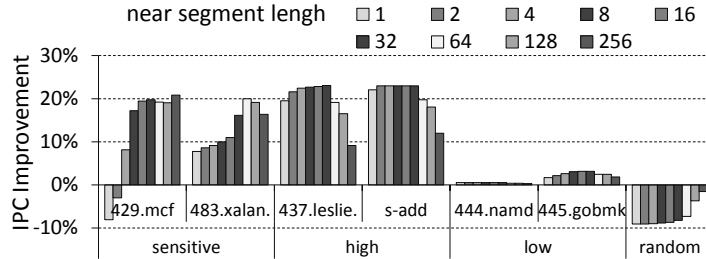
FIGURE 4.12: Four Application Groups Based on Performance Characteristics

For the other benchmarks, the performance does not much changes with increasing the near segment length. There are two cases. One case is that the working set of benchmarks is much larger than the near segment capacity. We only have one application in this case, *random* benchmark (`random` group). The other case is that the working set of benchmarks is much smaller than the near segment capacity (e.g, 1 or 2 rows in the near segment) and thus fits into the near segment capacity. We subdivide into these benchmarks (having small working set) into two groups based on the performance improvement with TL-DRAM – `high` and `low`. We observe that benchmarks in the `high` group are memory-intensive (high MPKI), while benchmarks in the `low` group are less memory-intensive (low MPKI).

For most benchmarks (except for *random* and *mcf*), increasing the number of rows per near segment beyond 64 reduces the performance benefits due to the increased near segment access latency, which is similar to the trend of Figure 4.11.

Based on these observations, we conclude that benchmarks have different performance characteristics with different latency and capacity of the near segment. We extend this application's characteristic to multi-program workloads in Section 4.6.6.

4.6.3   Effect of Far Segment Latency: Inclusive Cache

As we describe in Section 4.2.1, the t_{RCD} of the far segment is *lower* than the t_{RCD} of conventional DRAM. Therefore, even if most of the accesses are to the far segment, if the accesses are sufficiently far apart such that t_{RC} is *not* a bottleneck, TL-DRAM can still improve performance. We study this effect using our *random* microbenchmark (similar to GUPS [88]), which has very little data reuse and hence usually accesses the far segment. Figure 4.13 shows the performance improvement of our proposed mechanisms compared to the baseline with varying memory intensity (MPKI) of *random*. As the figure shows, when the memory intensity is low (< 2), the reduced t_{RCD} of the far segment dominates, and our mechanisms improve performance by up to 5%. However, further increasing the intensity of *random* makes t_{RC} the main bottleneck as evidenced by the degraded performance due to our proposed mechanisms.
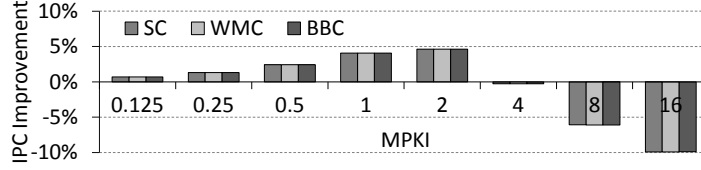
FIGURE 4.13: TL-DRAM Performance on Benchmark *random*

4.6.4 Sensitivity to Channel Count

For 1-core systems with 1, 2 and 4 memory channels, TL-DRAM provides 12.8%, 13.8% and 14.2% performance improvement compared to the baseline. The performance improvement of TL-DRAM increases with increasing number of channels. This is because, with more channels, the negative impact of channel conflicts reduces and bank access latency becomes the primary bottleneck. Therefore, TL-DRAM, which reduces the average bank access latency, provides better performance with more channels.

4.6.5 Sensitivity to CPU Frequency and DRAM Data Rate

We analyze the sensitivity to CPU frequency and DRAM data rate. Figure 4.14a shows the average system performance improvement of the single core workloads with different CPU frequencies from 2.7GHz to 5.3GHz. At the higher CPU frequencies, CPU generates memory requests more frequently, leading to providing more benefits with our mechanisms. Compared to 11.1% performance improvement at 2.7GHz CPU frequency, our mechanism provides 12.7% performance improvement.

Figure 4.14b shows the average system performance improvement of the single core workloads with different DRAM data rate. At higher DRAM data rate, the DRAM channel takes less time to transfer a cacheline, leading to reducing the memory channel conflicts. As expected, compared to 12.7% performance improvement on DDR3-1066 (533MHz DRAM clock frequency and 1066Mbps per-pin data rate), DDR3-2166 (1066MHz DRAM clock frequency and 2166Mbps per-pin data rate) provides much more performance improvement (19.4% the average performance improvement).

(A) Performance vs. CPU Frequency

(B) Performance vs. DRAM Data Rate
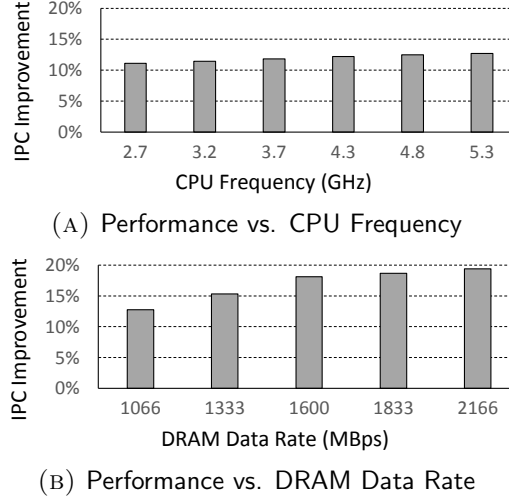
FIGURE 4.14: Sensitivity to CPU Frequency and DRAM Data Rate

Based on these sensitivity analyses, we conclude that *i)* our mechanisms provides significant performance improvement across different CPU frequencies and different DRAM data rates, and *ii)* our mechanisms provides more performance improvement at higher performance systems (e.g., systems that have higher CPU frequencies and memory systems that have higher data rate).

4.6.6 Multi-Core Results: Inclusive Cache

Figure 4.15 shows the system performance improvement with our proposed mechanisms compared to the baseline. We build dual-core workloads by combining two benchmarks from the four benchmark categories – `sensitive`, `high`, `low`, and `random` – as we defined in Section 4.6.2. We plot results in two separate figures, *i)* workloads without including `random` benchmark in Figure 4.15a, and *ii)* workloads with including `random` benchmark in Figure 4.15b.

We make three major observations from Figure 4.15a. First, when a workload includes at least one benchmark that is sensitive to the near segment capacity (left three cases in Figure 4.15a), the improvement in weighted speedup increases with increasing near segment capacity. Second, when neither benchmark is sensitive to the near segment capacity, the weighted speedup improvement is less sensitive to the near segment capacity (`High-Low` group

(A) Workloads without Random Benchmark

(B) Workloads with Random Benchmark
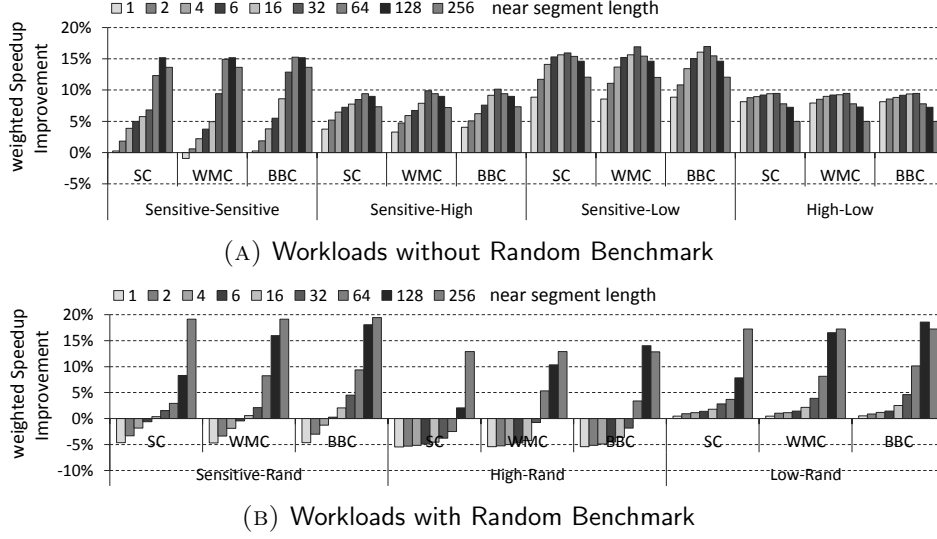
FIGURE 4.15: System Performance: 2-Core, Inclusive Cache

in Figure 4.15a). Third, for most workloads, the weighted speedup improvement decreases at very large near segment capacity (e.g, 128 or 256 rows per near segment) due to the increased near segment access latency.

Figure 4.15b shows the weighted speedup for workloads, which consists of a `random` benchmark and a benchmark from other categories. We make two major observations. First, the improvement in weighted speedup increases with increasing near segment capacity, which is similar to our observation from the `sensitive-sensitive` workloads in Figure 4.15a. Second, almost all workload categories and near segment capacities, BBC performs comparably to or significantly better than SC, emphasizing the advantages of our benefit-based near segment management policy.

As shown in Figure 4.16, on average, BBC improves weighted speedup by 12.3% and reduces power consumption by 26.4% compared to the baseline. We observe similar trends on 4-core systems, where BBC improves weighted speedup by 11.0% and reduces power consumption by 28.6% on average.
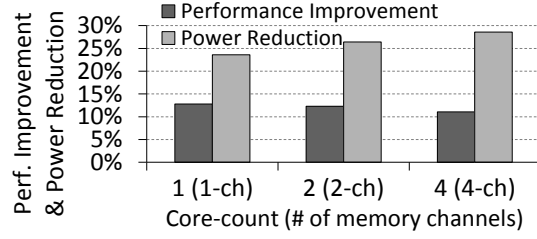
FIGURE 4.16: Inclusive Cache Analysis (BBC)

4.6.7 Exclusive Cache

Figure 4.17 shows the performance improvement of the TL-DRAM with the exclusive caching mechanism (Section 4.3.2) and 32 rows in the near segment over the baseline. For 1-/2-/4-core systems, TL-DRAM with exclusive caching improves performance by 7.9%/8.2%/9.9% and reduces power consumption by 9.4%/11.8%/14.3%. The performance improvement due to exclusive caching is lower compared to that of inclusive caching due to the increased caching latency, as explained in Section 4.3.2. Unlike inclusive caching, where BBC outperforms SC and WMC, WMC performs the best for exclusive caching. This is because unlike BBC and SC that cache any row that is accessed in the far segment, WMC caches a row only if it is wait-inducing. As a result, WMC reduces bank unavailability due to the increased caching latency.
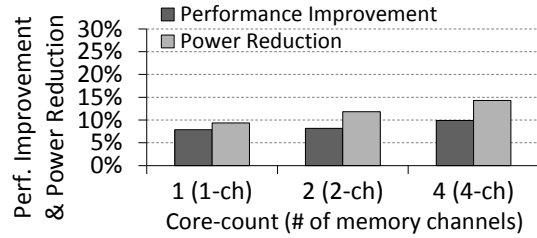
FIGURE 4.17: Exclusive Cache Analysis (WMC)

4.6.8 Profile-Based Page Mapping

Figure 4.18 shows the performance improvement of TL-DRAM with profile-based page mapping over the baseline which uses normal DRAM. The evaluated memory subsystem has 64 rows in the near segment. Therefore, the top 64 most frequently accessed rows in

57

each subarray, as determined by a profiling run, are allocated in the near segment. For 1-/2-/4-core systems, TL-DRAM with profile-based page mapping improves performance by 8.9%/11.6%/7.2% and reduces power consumption by 19.2%/24.8%/21.4%. These results indicate a significant potential for such a profiling based mapping mechanism. We leave a more rigorous evaluation of such a profiling based mapping mechanism to future work.
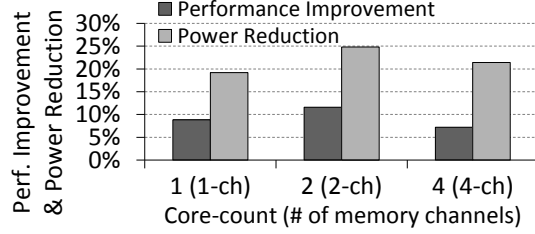
FIGURE 4.18: Profile-Based Page Mapping

4.7 Summary

Existing DRAM architectures present a trade-off between cost-per-bit and access latency. One can either achieve low cost-per-bit using long bitlines or low access latency using short bitlines, but not both. In this chapter, we introduce Tiered-Latency DRAM (TL-DRAM), a DRAM architecture that provides both low latency and low cost-per-bit. The key idea behind TL-DRAM is to segment a long bitline using an isolation transistor, creating a segment of rows with low access latency while keeping cost-per-bit on par with commodity DRAM.

We present mechanisms that take advantage of our TL-DRAM substrate by using its low-latency segment as a hardware-managed cache. Our most sophisticated cache management algorithm, Benefit-Based Caching (BBC), selects rows to cache that maximize access latency savings. Our evaluation results show that our proposed techniques significantly improve both system performance and energy efficiency across a variety of systems and workloads.

We conclude that TL-DRAM provides a promising low-latency and low-cost substrate for building main memories, on top of which existing and new caching and page allocation mechanisms can be implemented to provide even higher performance and higher energy efficiency.

Chapter 5

Adaptive-Latency DRAM: Optimizing DRAM Latency to the Common Operating Conditions

When a DRAM chip is accessed, it requires a certain amount of time before enough charge can move into the cell (or the bitline) for the data to be reliably stored (or retrieved). To guarantee this behavior, DRAM manufacturers impose a set of minimum latency restrictions on DRAM accesses, called *timing parameters*, as we demonstrated in Chapter 2. Ideally, timing parameters should provide just enough time for a DRAM chip to operate correctly. In practice, however, DRAM manufacturers *pessimistically incorporate a very large margin* into their timing parameters to ensure correct operation under *worst-case* conditions. This is because of two major concerns. First, due to *process variation*, some outlier cells suffer from a larger RC-delay than other cells, and require more time to be charged. For example, an outlier cell could have a very narrow connection (i.e., contact) to the bitline, which constricts the flow of charge and increases the RC-delay [147]. Second, due to *temperature dependence*, all cells suffer from a weaker charge-drive at high temperatures, and require more time to charge the bitline. DRAM cells are intrinsically leaky, and lose some of their charge even when they are not being accessed. At high temperatures, this leakage is accelerated exponentially [119,

155, 180, 228, 288], leaving a cell with less charge to drive the bitline when the cell is accessed — increasing the time it takes for the bitline to be charged.

Consequently, timing parameters prescribed by the DRAM manufacturers are dictated by the *worst-case cells* (the slowest cells) operating under the *worst-case conditions* (the highest temperature of 85℃ [105]). Such pessimism on the part of the DRAM manufacturers is motivated by their desire to *i)* increase chip yield and *ii)* reduce chip testing time. The manufacturers, in turn, are driven by the extremely cost-sensitive nature of the DRAM market, which encourages them to adopt pessimistic timing parameters rather than to *i)* discard chips with the slowest cells or *ii)* test chips at lower temperatures. Ultimately, the burden of pessimism is passed on to the end-users, who are forced to endure much greater latencies than what is actually needed for reliable operation under common-case conditions.

We first characterize 115 DRAM modules from three manufacturers to expose the excessive margin that is built into their timing parameters. Using an FPGA-based testing platform [119, 122, 155], we then demonstrate that DRAM timing parameters can be shortened to reduce DRAM latency without sacrificing any observed degree of DRAM reliability. We are able to reduce latency by taking advantage of the two large gaps between the worst-case and the "common-case." First, most DRAM chips are *not* exposed to the worst-case temperature of 85℃: according to previous studies [66, 67, 157] and our own measurements (Section 5.2.2), the ambient temperature around a DRAM chip is typically less than 55℃. Second, most DRAM chips do *not* contain the worst-case cell with the largest latency: the slowest cell for a typical chip is still faster than that of the worst-case chip (Section 5.5).

Based on our characterization, we propose Adaptive-Latency DRAM (AL-DRAM), a mechanism that dynamically optimizes the timing parameters for different modules at different temperatures. AL-DRAM exploits the *additional charge slack* present in the common-case compared to the worst-case, thereby preserving the level of reliability (at least as high as the worst-case) provided by DRAM manufacturers. We evaluate AL-DRAM on a real system [18, 19] that allows us to dynamically reconfigure the timing parameters at runtime. We show that AL-DRAM improves the performance of a wide variety of memory-intensive workloads by 14.0% (on average) without introducing errors. Therefore, we conclude that

60

AL-DRAM improves system performance while maintaining memory correctness and without requiring changes to DRAM chips or the DRAM interface.

5.1 Charge & Latency Interdependence

As we explained, the operation of a DRAM cell is governed by two important concepts: *i)* the quantity of charge and *ii)* the latency it takes to move charge. These two concepts are closely related to each other — one cannot be adjusted without affecting the other. To establish a more quantitative relationship between charge and latency, Figure 5.1 presents the voltage of a cell and its bitline as they cycle through the precharged state, charge-sharing state, sense-amplification state, restored state, and back to the precharged state (Section 2.1).[1] This curve is typical in DRAM operation, as also shown in prior works [75, 115, 145, 252]. The timeline starts with an `ACTIVATION` at 0 ns and ends with the completion of `PRECHARGE` at 48.75 ns. From the figure, we identify three specific periods in time when the voltage changes slowly: *i)* start of sense-amplification (part ①), *ii)* end of sense-amplification (part ②), and *iii)* end of precharging (part ③). Since charge is correlated with voltage, these three periods are when the charge also moves slowly. In the following, we provide three observations explaining why these three periods can be shortened for typical cells at typical temperatures — offering the best opportunity for shortening the timing parameters.

Observation 1. At the start of the sense-amplification phase, the higher the bitline voltage, the quicker the sense-amplifier is jump-started. Just as the amplification phase starts, the sense-amplifier detects the bitline voltage that was increased in the previous charge-sharing phase (by the cell donating its charge to the bitline). The sense-amplifier then begins to inject more charge into the bitline to increase the voltage even further — triggering a positive-feedback loop where the bitline voltage increases more quickly as the bitline voltage becomes higher. This is shown in Figure 5.1 where the bitline voltage ramps up faster and faster during the initial part of the amplification phase. Importantly, if the bitline has a higher

---

[1]Using 55nm DRAM parameters [223, 275], we simulate the voltage and current of the DRAM cells, sense-amplifiers, and bitline equalizers (for precharging the bitline). To be technology-independent, we model the DRAM circuitry using NMOS and PMOS transistors that obey the well-known MOSFET equation for current-voltage (SPICE) [226]. We do not model secondary effects.
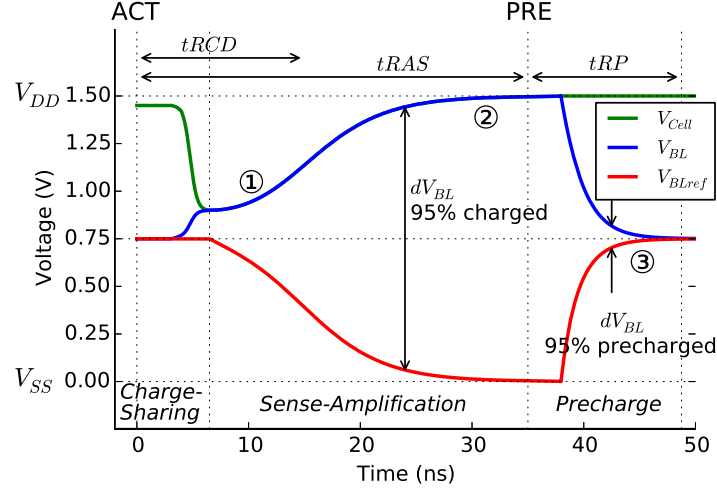
FIGURE 5.1: Phases of DRAM Voltage Levels

voltage to begin with (at the start of sense-amplification), then the positive-feedback is able to set in more quickly. Such a high initial voltage is comfortably achieved by typical cells at typical temperatures because they donate a large amount of charge to the bitline during the charge-sharing phase (as they have a large amount of charge). As a result, they are able to reach states ❸ and ❹ (Figure 2.4 in Chapter 2) more quickly, creating the opportunity to shorten t_{RCD} and t_{RAS}.

Observation 2. At the end of the sense-amplification phase, nearly half the time (42%) is spent on injecting the last 5% of the charge into the cell. Thanks to the positive-feedback, the middle part of the amplification phase (part between ① and ② in Figure 5.1) is able to increase the bitline voltage quickly. However, during the later part of amplification (part ② in Figure 5.1), the RC-delay becomes much more dominant, which prevents the bitline voltage from increasing as quickly. In fact, it takes a significant amount of extra delay for the bitline voltage to reach V_{DD} (Figure 5.1) that is required to *fully* charge the cell. However, for typical cells at typical temperatures, such an extra delay may not be needed — the cells could already be injected with *enough* charge for them to comfortably share with the bitline when they are next accessed. This allows us to shorten the later part of the amplification phase, creating the opportunity to shorten t_{RAS} and t_{WR}.

Observation 3. At the end of the precharging phase, nearly half the time (45%) is spent on extracting the last 5% of the charge from the bitline. Similar to the amplification phase, the later part of the precharging phase is also dominated by the RC-delay, which causes the bitline voltage to decrease slowly to $\frac{1}{2}V_{DD}$ (part ③ in Figure 5.1). If we decide to incur less than the full delay required for the bitline voltage to reach exactly $\frac{1}{2}V_{DD}$, it could lead to two different outcomes depending on which cell we access next. First, if we access the *same* cell again, then the higher voltage left on the bitline works in our favor. This is because the cell — which is filled with charge — would have increased the bitline voltage anyway during the charge-sharing phase. Second, if we access a *different* cell connected to the same bitline, then the higher voltage left on the bitline may work as a handicap. Specifically, this happens only when the cell is devoid of any charge (e.g., storing a data of '0'). For such a cell, its charge-sharing phase operates in the opposite direction, where the cell steals some charge away from the bitline to decrease the bitline voltage. Subsequently, the voltage is "amplified" to 0 instead of V_{DD}. Nevertheless, typical cells at typical temperatures are capable of comfortably overcoming the handicap — thanks to their large capacitance, the cells are able to steal a large amount of charge from the bitline. As a result, this creates the opportunity to shorten t_{RP}.

5.2   Charge Gap: Common-Case vs. Worst-Case

Based on the three observations, we understand that *timing parameters can be shortened if the cells have enough charge.* Importantly, we showed that such a criterion is easily satisfied for typical cells at typical temperatures. In this section, we explain what it means for a cell to be "typical" and why it has more charge at "typical" temperatures. Specifically, we examine two physical phenomena that critically impact a DRAM cell's ability to receive and retain charge: *i)* process variation and *ii)* temperature dependence.

5.2.1 Process Variation: Cells Are Not Created Equal

Process variation is a well-known phenomenon that introduces deviations between a chip's intended design and its actual implementation [73, 147, 233]. DRAM cells are affected by process variation in two major aspects: *i)* cell capacitance and *ii)* cell resistance. Although every cell is designed to have a large capacitance (to hold more charge) and a small resistance (to facilitate the flow of charge), some deviant cells may not be manufactured in such a manner [82, 113, 119, 121, 150, 155, 156]. In Figure 5.2a, we illustrate the impact of process variation using two different cells: one is a typical cell conforming to design (left column) and the other is the worst-case cell deviating the most from design (right column).

(A) Existing DRAM   (B) Our Proposal (Adaptive-Latency DRAM)
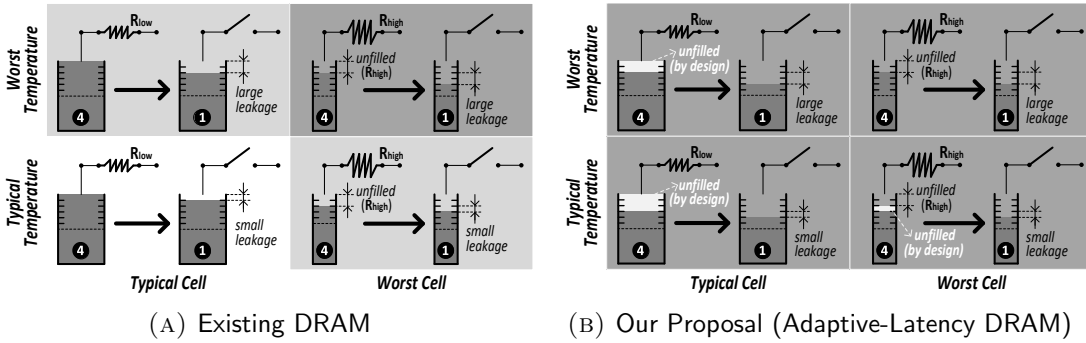
FIGURE 5.2: Effect of Reduced Latency: Typical vs. Worst
(Darker Background means Less Reliable)

As we see from Figure 5.2a, the worst-case cell contains less charge than the typical cell in state ❹ (Restored state, as was shown in Figure 2.4 in Chapter 2). This is because of two reasons. First, due to its *large resistance*, the worst-case cell cannot allow charge to flow inside quickly. Second, due to its *small capacitance*, the worst-case cell cannot store much charge even when it is full. To accommodate such a worst-case cell, existing timing parameters are set to a large value. However, *worst-case cells are relatively rare.* When we analyzed 115 DRAM modules, the overwhelming majority of them had significantly more charge than what is necessary for correct operation (Section 5.5 will provide more details).

5.2.2 Temperature Dependence: Hot Cells Are Leakier

Temperature dependence is a well-known phenomenon in which cells leak charge at almost double the rate for every 10℃ increase in temperature [119, 155, 180, 228, 288]. In Figure 5.2a, we illustrate the impact of temperature dependence using two cells at two different temperatures: *i)* typical temperature (55℃, bottom row), and *ii)* the worst-case temperature (85℃, top row) supported by DRAM standards.

As we see from the figure, both typical and worst-case cells leak charge at a faster rate at the worst-case temperature. Therefore, not only does the worst-case cell have less charge to begin with, but it is left with *even less* charge because it leaks charge at a faster rate (top-right in Figure 5.2a). To accommodate the combined effect of process variation *and* temperature dependence, existing timing parameters are set to a very large value. However, *most systems do not operate at 85℃* [66, 67, 157].[2] We measured the DRAM ambient temperature in a server cluster running a memory-intensive benchmark, and found that the temperature *never* exceeds 34℃ — as well as never changing by more than 0.1℃ per second. We show this in Figure 5.3, which plots the temperature for a 24-hour period (left) and also zooms in on a 2-hour period (right). In addition, we repeated the measurement on a desktop system that is not as well cooled as the server cluster. As Figure 5.4 shows, even when the CPU was utilized at 100% and the DRAM bandwidth was utilized at 40%, the DRAM ambient temperature never exceeded 50℃. Other works [66, 67, 157] report similar results, as explained in detail in Footnote 2. From this, we conclude that the majority of DRAM modules are likely to operate at temperatures that are much lower than 85℃, which slows down the charge leakage by an order of magnitude or more than at the worst-case temperature.

---

[2] Figure 22 in [67] and Figure 3 in [157] show that the maximum temperature of DRAM chips at the highest CPU utilization is 60–65℃. While some prior works claim a maximum DRAM temperature over 80℃ [300], each DRAM module in their system dissipates 15W of power. This is very aggressive nowadays — modern DRAM modules typically dissipate around 2–6W (see Figure 8 of [87], 2-rank configuration same as the DRAM module configuration of [300]). We believe that continued voltage scaling and increased energy efficiency of DRAM have helped reduce the power consumption of the DRAM module. While old DDR1/DDR2 use 1.8–3.0V power supplies, newer DDR3/DDR4 use only 1.2–1.5V. In addition, newer DRAMs adopt more power saving techniques (i.e., temperature compensated self refresh, power down modes [100, 174]) that were previously used only by Low-Power DRAM (LPDDR). Furthermore, many previous works [54, 153, 153, 157, 300] propose hardware/software mechanisms to maintain a low DRAM temperature and energy.
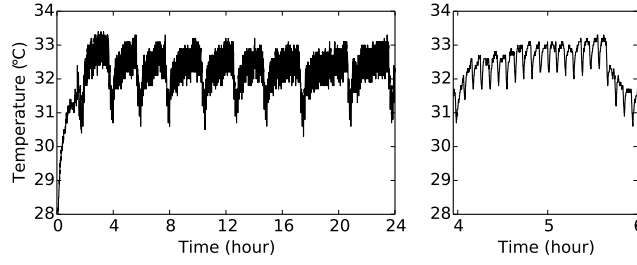
FIGURE 5.3: DRAM Temperature in a Server Cluster
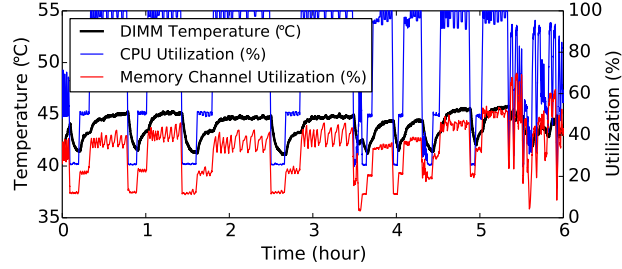
FIGURE 5.4: DRAM Temperature in a Desktop System

5.2.3 Reliable Operation with Shortened Timing

As explained in Section 5.1, the amount of charge in state ❶ (i.e., the precharged state in Figure 2.4 in Chapter 2) plays a critical role in whether the correct data is retrieved from a cell. That is why the worst-case condition for correctness is the top-right of Figure 5.2a, which shows the least amount of charge stored in the worst-case cell at the worst-case temperature in state ❶. However, DRAM manufacturers provide reliability guarantees even for such worst-case conditions. In other words, the amount of charge at the worst-case condition is still greater than what is required for correctness.

If we were to shorten the timing parameters, we would also be reducing the charge stored in the cells. It is important to note, however, that we are proposing to exploit *only* the *additional slack* (in terms of charge) compared to the worst-case. This allows us to provide as strong of a reliability guarantee as the worst-case.

In Figure 5.2b, we illustrate the impact of shortening the timing parameters in three of the four different cases: two different cells at two different temperatures. The lightened portions inside the cells represent the amount of charge that we are giving up by using the

shortened timing parameters. Note that we are not giving up any charge for the worst-case cell at the worst-case temperature. Although the other three cells are not fully charged in state ❹, when they eventually reach state ❶, they are left with a similar amount of charge as the worst-case (top-right). This is because these cells are capable of either holding more charge (typical cell, left column) or holding their charge longer (typical temperature, bottom row). Therefore, optimizing the timing parameters (based on the amount of existing slack) provides the opportunity to reduce overall DRAM latency while still maintaining the reliability guarantees provided by the DRAM manufacturers.

In Section 5.5, we present the results from our characterization study where we quantify the slack in 115 DRAM modules. Before we do so, we first propose our mechanism for identifying and enforcing the shortened timing parameters.

5.3   Adaptive-Latency DRAM

Our mechanism, Adaptive-Latency DRAM (AL-DRAM), allows the memory controller to exploit the latency variation across DRAM modules at different operating temperatures by using customized (aggressive) timing parameters for each DRAM module/temperature combination. Our mechanism consists of two steps: *i) identification* of the best timing parameters for each DRAM module/temperature, and *ii) enforcement*, wherein the memory controller dynamically extracts each DRAM module's operating temperature and uses the best timing parameters for each DRAM module/temperature combination.

5.3.1   Identifying the Best Timing Parameters

Identifying the best timing parameters for each DRAM module at different temperatures is the more challenging of the two steps. We propose that DRAM manufacturers identify the best timing parameters at different temperatures for each DRAM chip during the testing phase and provide that information along with the DRAM module in the form of a simple table. Since our proposal only involves changing four timing parameters (t_{RCD}, t_{RAS}, t_{WR}, and t_{RP}), the size of the table for, e.g., five, different temperature points is small and such a

table can potentially be embedded in the Serial Presence Detect circuitry (a ROM present in each DRAM module [104]). We expect this approach to have low cost as DRAM manufacturers already have an elaborate testing mechanism to identify faulty DRAM modules. An alternative approach to perform this profiling step is to do it at the end-user system using an online testing while the system is running. We leave the exploitation of such an online testing mechanism to future work.

5.3.2 Enforcing Dynamic Timing Parameters

Dynamically enforcing the best timing parameters at the memory controller is fairly straightforward. The memory controller populates a hardware table with the best timing parameters for different temperatures for all the DRAM modules connected to the controller. The memory controller divides time into regular intervals (e.g., 256 ms). At the beginning of each interval, it extracts the temperature of each DRAM module. It then employs the best timing parameters corresponding to the temperature of each DRAM module for all accesses during the remainder of the interval.

This approach should work well in practice as temperature does not change very frequently in real systems — our measurements on real server and desktop systems indicate that temperature changes at the rate of at most 0.1℃ per second. In addition, existing DRAM designs such as LPDDR3 [103], and the recently announced DDR4 [102] already have in-DRAM temperature sensors to minimize self-refresh energy. By accessing the temperature value of in-DRAM temperature sensors during auto-refresh (usually performed every 7.8us), our mechanism monitors DRAM temperature without any performance penalty and frequently enough to detect even drastic temperature changes.

In Section 5.6, we evaluate the benefits of AL-DRAM on a real system equipped with a memory controller whose timing parameters can be dynamically changed. Our evaluation shows that AL-DRAM can significantly improve system performance for a wide variety of applications.

5.4 DRAM Latency Profiling Methodology

In this section, we describe our FPGA-based DRAM profiling (i.e., testing) infrastructure and the methodology we use to characterize DRAM modules for variation in access latency.

5.4.1 Profiling Infrastructure

To analyze the characteristics of latency variation in DRAM, we have built an FPGA-based testing platform [37, 39, 119, 120, 122, 155, 217] using Xilinx ML605 boards [285] connected to a host machine over a PCI-e bus [284], as shown in Figure 5.5. The FPGA has a DDR3 DRAM memory controller [286] (Figure 5.5b). We customize the memory controller so that we can change DRAM timing parameters from the host machine at runtime. To test DRAM modules at different operating temperatures, we built a heat chamber consisting of a heat encloser, a heater, and a temperature controller with a thermo-couple sensor (Figure 5.5a). During the test, the temperature within the heat chamber is maintained within $\pm 0.5°C$ of the target temperature. In all, we present results for 115 DRAM modules that are produced by three different DRAM manufacturers during the last 5 years (from 2010 to 2014). The majority of these are DDR3-1600 SO-DIMMs with standard timing parameters: $t_{RCD} = 13.75$ ns, $t_{RAS} = 35$ ns, $t_{WR} = 15$ ns, and $t_{RP} = 13.75$ ns.

(A) Full System   (B) FPGA Board
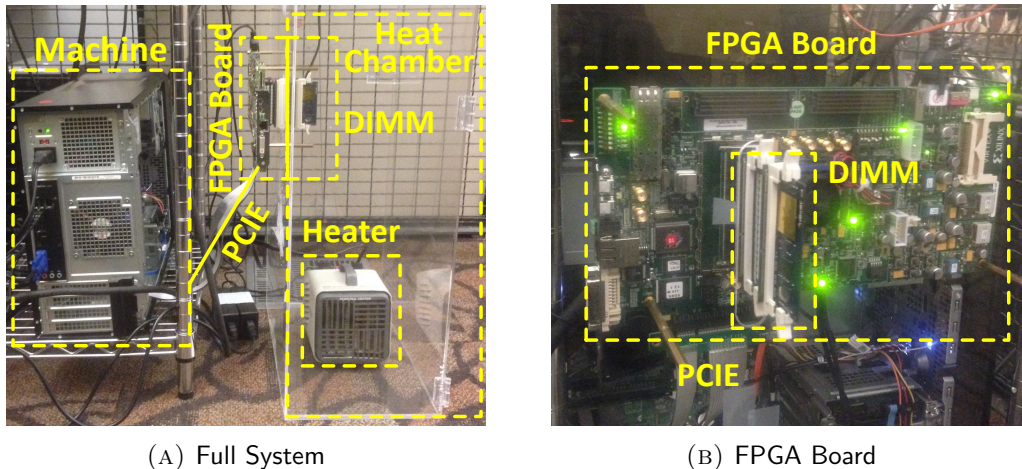
FIGURE 5.5: FPGA-Based DRAM Test Infrastructure

5.4.2 Profiling Mechanism

The goal of our profiling experiments is to determine the amount by which different timing parameters can be reduced without inducing any errors. For this purpose, we devise two types of tests: one for testing read operations and another for write operations. The read test aims to analyze reductions in t_{RCD}, t_{RAS}, and t_{RP}, whereas the write test aims to analyze reductions in t_{WR}. Both types of tests are carefully designed to induce a reasonable amount of coupling-noise between circuit components (i.e., cells, bitlines, and wordlines), as well as to generate power-noise. We now describe the steps involved in each of these tests in detail. Each test involves a test address, a test data pattern, and a target set of timing parameters.

Read Test. For a read test, we first write the test data pattern to the target address with the standard DRAM timing parameters. We follow it with a read to the target address with the *target*, smaller, timing parameters. This step could potentially corrupt the data if the cells involved in the operation are not restored with enough charge or if the corresponding bitlines are not precharged sufficiently. To test for data corruption, we wait for the refresh interval such that DRAM cells have the smallest charge due to charge leakage over time. Then, we perform another read operation to the target address. If the latter read data matches the originally-written test data, the test is considered successful — i.e., for the corresponding address, the timing parameters can be reduced to the target timing parameters. Otherwise, the test is considered a failure and the system logs the list of errors.

Write Test. For a write test, we first write the inverted test data pattern to the target address with the standard DRAM timing parameters. We then write the test data pattern with the target timing parameters. The original write (inverted test pattern) is to ensure that the test write actually flips the values on the bitlines, thereby making the t_{WR} timing parameter relevant. After the test write, we wait for the refresh interval and perform a verification read. We check if the verification succeeded and log the list of errors, if any.

Coupling and Power-Noise Effects. We carefully design the tests to be close to the worst-case in coupling and power-noise (to stress the reliability of our proposal). From the standpoint of a single row, each test *i)* writes/reads data to/from the row, and then *ii)* reads

70

data from the row for verification (access – wait – verify). However, from the standpoint of the entire DRAM chip, we test different rows in an overlapped manner, staggered by t_{RC} (access, access, $\cdots$, access, verify, verify, $\cdots$, verify). We perform this test sequence twice with two different orders of row addresses (increasing/decreasing). Due to this overlapped manner of testing multiple rows, the tests exhibit the following effects.

- *Bitline-to-Cell Coupling:* Non-activated cells and bitlines are coupled with each other in as many as half of the cells per bitline (i.e., 256 times a 512-cell bitline).

- *Wordline-to-Wordline Coupling:* Each wordline is coupled with adjacent wordlines in as many columns as in a row (i.e., 256 times for a 4k-cell row and 64-bit per access).

- *Power-Noise:* Power noise is close to the worst-case due to the maximum number of allowed row activations per time interval (i.e., activating a row every t_{RC}).

Data Patterns. We repeat both tests for all DRAM addresses with eight different checkered data patterns [273] (0000, 0011, 0101, 1001, 0110, 1010, 1100, 1111). To test the bitline precharge effect, we use exclusive-OR data patterns (i.e., 0101 vs. 1010) for adjacent rows. Therefore, bitlines are amplified in the opposite direction from the previous row activation.

Iterating Tests. We repeat tests with a large number of reduced target timing parameters. We accumulate the erroneous cells from all the test patterns. Since cells of the same DRAM row are activated and precharged together, we run through all the DRAM addresses in the column-major order to reduce interference between tests of the cells in the same row. We perform 10 iterations of each test before deciding whether it is free from errors.

as we described, we carefully modeled the worst-case test scenarios by *i)* using minimum value of timing parameters for both read and write test, *ii)* generating couplings and power-noise as much as possible, and *iii)* using multiple data patterns. These test scenarios are similar to the most error prone test patterns related to DRAM timing parameters in Memtest [2], while our test scenarios could not cover all test patterns that can be used by memtest and DRAM companies. Therefore, we expect that using our test scenarios can uncover most of the DRAM errors which are related to reducing DRAM timing parameters.

71

5.5 DRAM Latency Profiling Results and Analysis

In this section, we present and analyze the results of our profiling experiments. We first present the effect of reducing individual timing parameters on a single representative module (Section 5.5.1). We then present the results of reducing multiple timing parameters simultaneously (Section 5.5.2), analyze the timing slack present (Section 5.5.3), and quantify timing parameter reductions (Section 5.5.4) using this representative module. Finally, we summarize our profiling results for all 115 modules (Section 5.5.5).

5.5.1 Effect of Reducing Individual Timing Parameters

As discussed in Section 5.2, DRAM access latency is tightly coupled with the amount of charge in the accessed cell. To verify this on real DRAM chips, we need to adjust the amount of charge intentionally. Unfortunately, there is no way to quantitatively change the charge stored in DRAM cells. Instead, we indirectly adjust the charge amount by enabling the leakage of DRAM cells. Note that DRAM cells have less charge in two cases: *i)* at longer refresh intervals and *ii)* at higher temperatures. Using these characteristics, we design aggressive test environments for analyzing the effect of reducing each timing parameter by: *i)* sweeping the refresh interval from the DRAM standard (64 ms) to very long refresh intervals (up to 512 ms) at the highest temperature in the DRAM standard (85℃), and *ii)* sweeping the temperature at a very long refresh interval (512 ms).

Sweeping the Refresh Interval: Figure 5.6a shows the number of errors when we reduce each of the four timing parameters (t_{RCD}, t_{RAS}, t_{WR}, and t_{RP}) at different refresh intervals while the temperature is kept at 85℃. X-axis in the figure represents the value of the tested timing parameter in linear scale, and Y-axis represents the error count in logarithm scale (except for the zero label point which represents *no* error. We use this hybrid logarithm scale in Y-axis for all later figures as well. We make three key observations from the figure. First, it is possible to reduce the timing parameters significantly without incurring errors. In particular, at the standard refresh interval of 64 ms, the four timing parameters can be reduced by (3.75 ns, 15 ns, 10 ns, 3.75 ns). Second, as the refresh interval increases, the

number of errors starts to increase and the range of timing parameters for error-free operation decreases. In fact, increasing the refresh interval beyond 256 ms results in errors even with the standard timing parameters. Third, reducing t_{RCD} or t_{RP} below a certain point results in a drastic increase in the number of errors. This is because doing so causes a functional failure of DRAM — e.g., reducing t_{RCD} below 10 ns does not give DRAM enough time to even activate the wordline completely, resulting in a malfunction of the entire row.

(A) Sweeping the Refresh Interval (Temperature: 85℃)

(B) Sweeping the Temperature (Refresh Interval: 512ms)
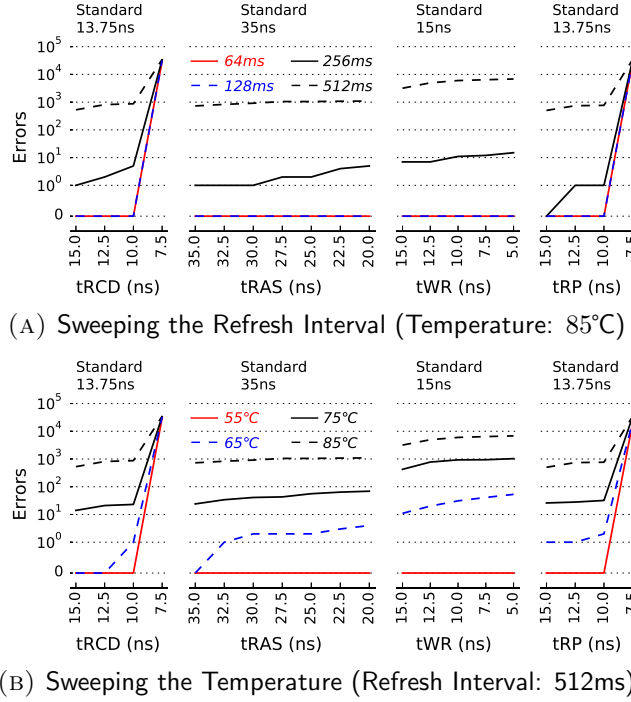
FIGURE 5.6: Effect of Varying Each Timing Parameter on Error Count

Sweeping the Temperature: Figure 5.6b shows the number of errors with reduced timing parameters at different temperatures (55℃ to 85℃) while the refresh interval is kept at 512 ms. Note that the refresh interval is 8 times as long as its standard value of 64 ms, meaning that the number of errors shown in the figure are much larger than what they should be for commodity DRAM. From the figure, we find that the number of errors decreases at lower temperatures, similar to how it decreases at lower refresh intervals in the previous experiment. Therefore, we conclude that there is a significant opportunity to reduce timing parameters at lower temperatures.

73

5.5.2 Effect of Reducing Multiple Timing Parameters

The results of the previous experiment showed that there is a significant opportunity for reducing each timing parameter individually. However, we hypothesize that reducing one timing parameter may also decrease the opportunity to reduce another timing parameter simultaneously. For example, if we reduce the value of t_{RAS}, the charge on a cell may not be fully restored by the end of the sense-amplification process. This may decrease the opportunity to reduce t_{RP}, as the bitlines may have to be fully precharged to ensure that it can be reliably perturbed by the partially charged cell.

To study the potential for reducing multiple timing parameters simultaneously, we run our tests with all possible combinations of timing parameters in the following range: t_{RCD} (12.5–10 ns), t_{RAS} (35–20 ns), t_{WR} (15–5 ns), t_{RP} (12.5–10 ns). For each combination, we also vary the temperature from 55℃ to 85℃ and the refresh interval from 64 ms to 960 ms.

Figure 5.7 presents the number of errors for the read and write tests for multiple such combinations at 85℃. Our results validate our hypothesis. For example, although it was possible to *individually* reduce t_{RAS} to 20 ns at 85℃ and 200 ms refresh interval, t_{RAS} can only be reduced to 32.5 ns if both t_{RCD} and t_{RP} are *also* reduced to 10 ns. In this section, we only present the results for specific combinations to show the effect of reducing multiple timing parameters clearly. In Section 5.5.4, we will present the test results with all timing parameter combinations and resulting potential reductions.

(A) Read   (B) Write
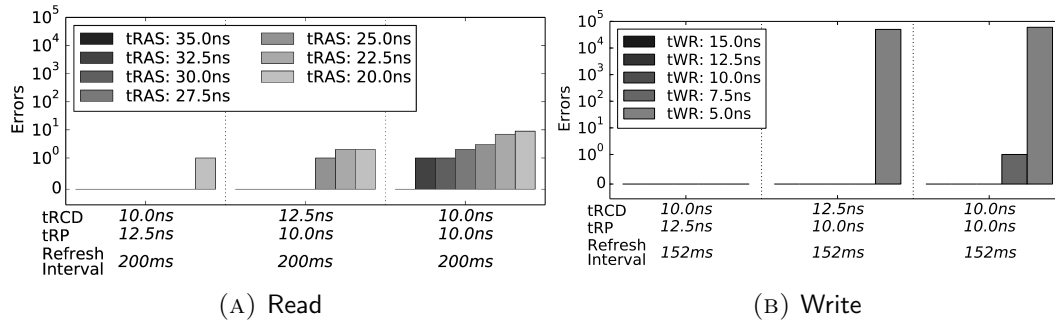
FIGURE 5.7: Error Counts When Varying Multiple Timing Parameters Simultaneously

5.5.3 Effect of Temperature on Timing Slack

As discussed in Section 5.2, the length of the timing parameters is dictated by the weakest cell with the shortest retention time at 85℃. However, ensuring enough charge is stored in such a cell may not be enough to guarantee reliable operation — this is because the cell could be affected by other failure mechanisms that are difficult to foresee. Just to name one example, some DRAM cells are known to suffer from a phenomenon known as *variable retention time* (VRT) [119, 155], in which their retention time could change unpredictably between short and long values. As a counter-measure against such failures, DRAM manufacturers provide a built-in *safety-margin* in retention time, also referred to as *a guard-band* [9, 119, 276]. This way, DRAM manufacturers are able to guarantee that even the weakest cell is insured against various other modes of failure.

In order to quantify the safety-margin, we sweep the refresh interval from 64 ms to 960 ms. The safety-margin incorporated by the DRAM manufacturers is the difference between the highest refresh interval that exhibits no errors at 85℃ and the standard refresh interval (64 ms). Figure 5.8 plots the number of errors (in gray scale) for varying refresh intervals for different temperatures (55℃ to 85℃). For each temperature, we also vary t_{RAS} (Figure 5.8a) and t_{WR} (Figure 5.8b). A box at (x, y) represents the number of errors at a refresh interval y and t_{RAS}/t_{WR} of x. A white box indicates no errors, and a darker box indicates a large number of errors. The red line separates the error-free region from the region with at least one error.

We make several observations from the figure. First, as expected, for each temperature, increasing the refresh interval (i.e., going down on the y axis) leads to more errors. Second, for a given refresh interval and temperature, reducing the t_{RAS}/t_{WR} timing parameters (i.e., going right on the x axis for each temperature) also increases the number of errors. Third, at 85℃, the highest refresh interval for error-free operation is 192 ms for the read test and 144 ms for the write test — this implies a safety-margin of 128 ms for reads and 80 ms for writes. Fourth, the slack on the retention time increases with decreasing temperature, because retention time increases with decreasing temperature (Section 5.2). As shown in the
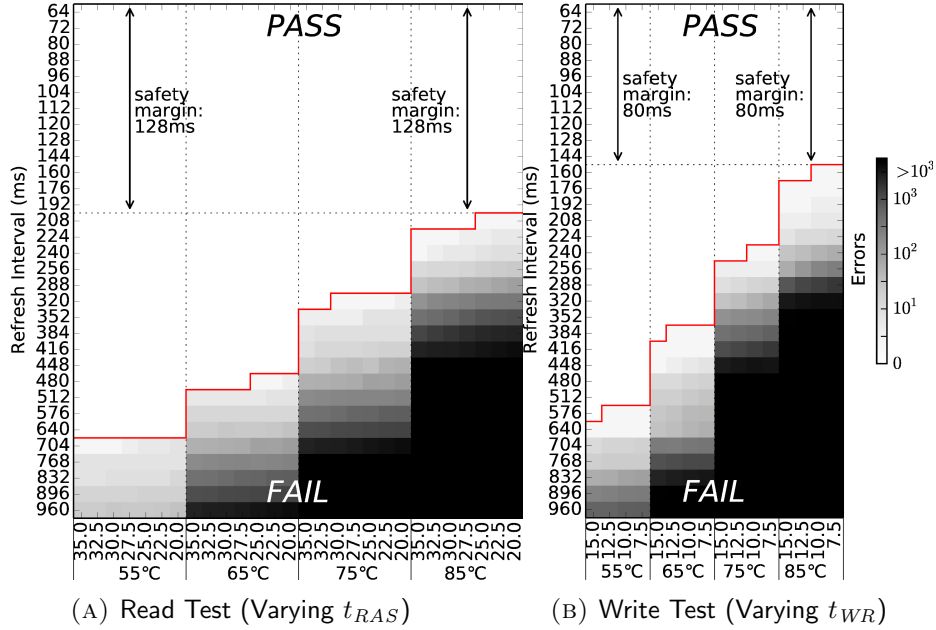
FIGURE 5.8: Error Counts When Varying Temperature, Refresh Interval, and t_{RAS}/t_{WR} (t_{RCD}/t_{RP}: 12.5 ns)

figure, at 55℃, the margin on the retention time is 576 ms for reads and 448 ms for writes — these values are at least 4× higher than their safety-margins (at 85℃). In summary, there is significant room for reducing DRAM latency at lower temperatures while still ensuring reliable DRAM operation.

5.5.4 Potential Timing Parameter Reductions While Maintaining the Safety-Margin

So far, we have discussed the effect of reducing timing parameters both individually and simultaneously. We also have studied the safety-margin and the effect of the operating temperature on the slack in timing parameters. In this section, we study the possible timing parameter reductions of a DRAM module while maintaining the safety-margin.

We first measure the safety-margin of a DRAM module by sweeping the refresh interval at the worst operating temperature (85℃), using the standard timing parameters. Figure 5.9a plots the maximum refresh intervals of each bank and each chip in a DRAM module for both read and write operations. We make several observations. First, the maximum refresh

intervals of both read and write operations are much larger than the DRAM standard (208 ms for the read and 160 ms for the write operations vs. the 64 ms standard). Second, for the smaller architectural units (banks and chips in the DRAM module), some of them operate without incurring errors even at much higher refresh intervals than others (as high as 352 ms for the read and 256 ms for the write test). This is because the error-free retention time is determined by the worst single cell in each architectural component (i.e., bank/chip/module).

(A) Maximum Error-Free Refresh Interval at 85℃ (Bank/Chip/Module)

(B) Read Latency (Refresh Interval: 200 ms)

(C) Write Latency (Refresh Interval: 152 ms)
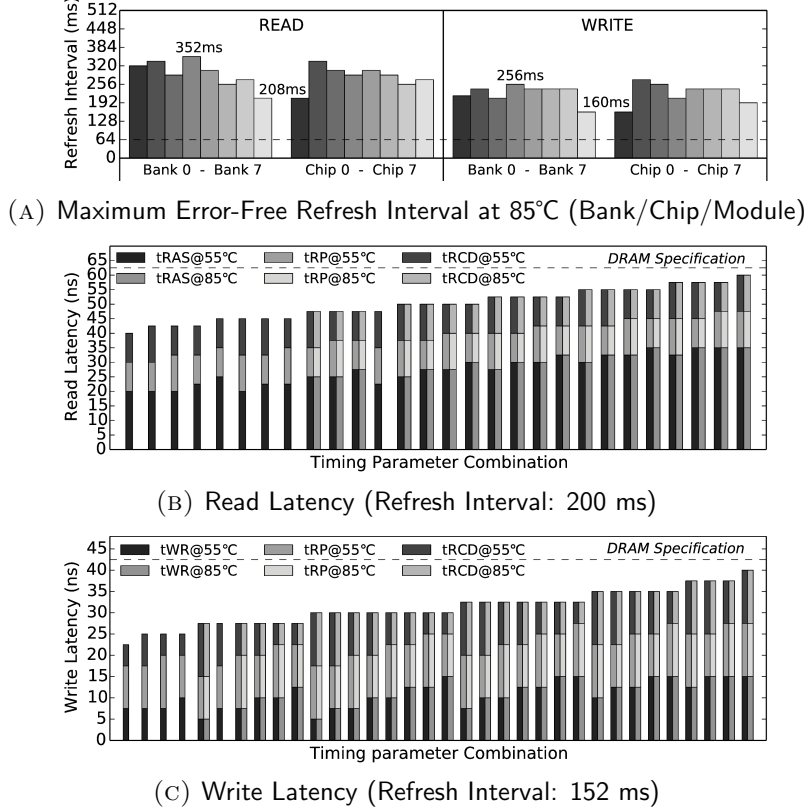
FIGURE 5.9: Latency Reductions While Maintaining the Safety-Margin

Based on this experiment, we define the *safe refresh interval* for a DRAM module as the maximum refresh interval that leads to no errors, minus an additional margin of 8 ms, which is the increment at which we sweep the refresh interval. The safe refresh interval for the read and write tests are 200 ms and 152 ms, respectively. We then use the safe refresh intervals to run the tests with all possible combinations of timing parameters. For each combination, we run our tests at two temperatures: 85℃ and 55℃.

77

Figure 5.9b plots the error-free timing parameter combinations (t_{RCD}, t_{RAS}, and t_{RP}) in the read test. For each combination, there are two stacked bars — the left bar for the test at 55℃ and the right bar for the test at 85℃. Missing bars indicate that the test (with that timing parameter combination at that temperature) causes errors. Figure 5.9c plots same data for the write test (t_{RCD}, t_{WR}, and t_{RP}).

We make three observations. First, even at the highest temperature of 85℃, the DRAM module reliably operates with reduced timing parameters (24% for read, and 35% for write operations). Second, at the lower temperature of 55℃, the potential latency reduction is even higher (36% for read, and 47% for write operations). These latency reductions are possible *while* maintaining the safety-margin of the DRAM module. From these two observations, we conclude that there is significant opportunity to reduce DRAM timing parameters *without compromising reliability.* Third, multiple different combinations of the timing parameters can form the same overall value of the timing parameters. For example, three different combinations of (t_{RAS}, t_{RP}, and t_{RCD}) show the same overall value of 42.5 ns. This might enable further optimization for the most critical timing parameter at runtime. We leave the exploitation of such a fine-grained optimization to future work.

5.5.5 Effect of Process Variation on Timing Slack

So far, we have discussed the effect of temperature and the potential to reduce various timing parameters at different temperatures for a single DRAM module. The same trends and observations also hold true for all of the other modules. In this section, we analyze the effect of process variation by studying the results of our profiling experiments on 115 DRAM modules. We also present results for intra-chip/inter-chip process variations by studying the process variation across different banks/chips within each DRAM module.

Figure 5.10a plots the highest refresh interval that leads to correct operation across all cells at 85℃ in *each DRAM module* for the read test (Figure 5.10b for the write test). Our key observation is that although there exist a few modules which *just* meet the timing parameters (with a low safety-margin), a vast majority of the modules very comfortably meet the standard timing parameters (with a high safety-margin). This indicates that a majority

of the DRAM modules have significantly higher safety-margins than the worst-case module *even at the highest-acceptable operating temperature of 85℃.*

(A) Read Retention Time (DIMM)

(B) Write Retention Time (DIMM)

(C) Read Retention Time (Bank)

(D) Write Retention Time (Bank)

(E) Read Retention Time (Chip)

(F) Write Retention Time (Chip)
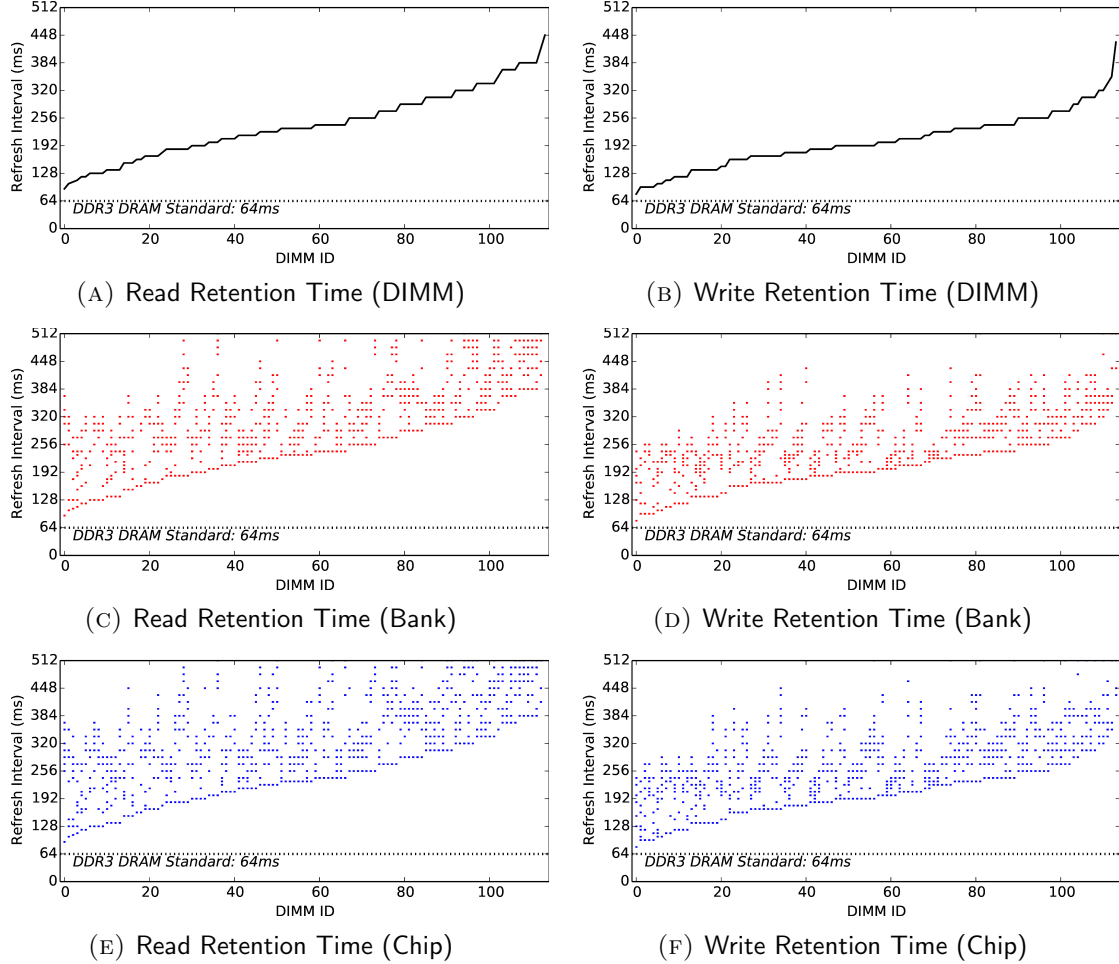
FIGURE 5.10: Retention Time of Multiple DIMMs

We do the same analysis of the cell retention time in banks and chips. The red dots in Figure 5.10c show the highest refresh interval that leads to correct operation across all cells within *each bank* for all eight banks. The blue dots in Figure 5.10e show the highest refresh interval that leads to correct operation across all cells within *each chip* for all DRAM chips in a DRAM module (usually eight chips per each DRAM module). Figures 5.10d and 5.10f plot the same data for the write test.

We make two key observations. First, the effect of process variation is even higher for banks and chips within the same DRAM module, explained by the large spread of the

red and blue dots in each DRAM module. We provide the average value of the retention time difference (highest retention time − lowest retention time of banks/chips in a DRAM module) across 115 DRAM modules. For read test, the average value is 134.8ms/160.7ms for banks/chips in a DRAM module, respectively. For write test, the average value is 111.2ms/138.64ms for banks/chips in a DRAM module, respectively. Considering that the average values are close to twice of the standard refresh interval (64ms), the effect of process variation across banks/chips are significant. This indicates that a majority of the DRAM banks/chips have significantly higher safety-margins than the worst-case bank/chip.

Since banks within a DRAM module can be accessed independently with different timing parameters, one can potentially imagine a mechanism that more aggressively reduces timing parameters at a bank granularity and not just the DRAM module granularity. In some memory architectures [298] that provide an individual control channel to each chip in a DRAM module, one can imagine a mechanism that each DRAM chip in a DRAM module has different timing parameters, which reduces DRAM latency more aggressively. We leave these for future work.

To study the potential of reducing timing parameters for each DRAM module, we sweep all possible combinations of timing parameters ($t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$) for all the DRAM modules at both the highest acceptable operating temperature (85℃) and a more typical operating temperature (55℃). We then determine the acceptable DRAM timing parameters for each DRAM module for both temperatures while maintaining its safety-margin of each DRAM module.

Figures 5.11a and 5.11b show the results of this experiment for the read test and write test respectively. The y-axis plots the sum of the relevant timing parameters (t_{RCD}, t_{RAS}, and t_{RP} for the read test and t_{RCD}, t_{WR}, and t_{RP} for the write test). The solid black line shows the latency sum of the standard timing parameters (DDR3 DRAM specification). The dotted red line and the dotted blue line show the most acceptable latency parameters for each DRAM module at 85℃ and 55℃, respectively. The solid red line and blue line show the average acceptable latency across all DRAM modules.
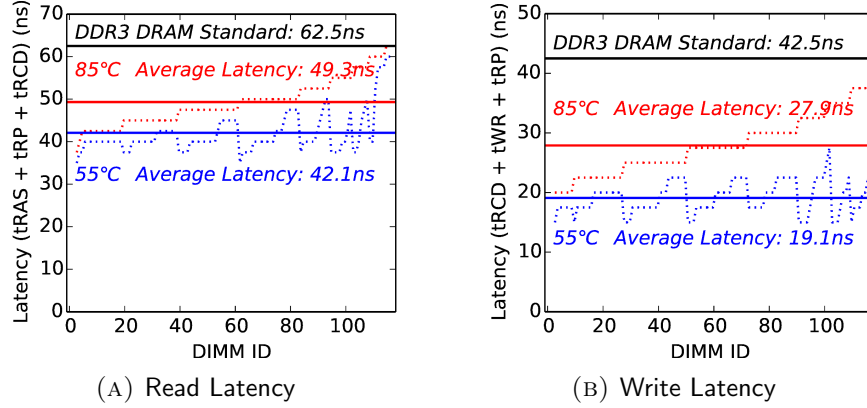
FIGURE 5.11: Latency Analysis of Multiple DRAM modules

We make two observations. First, even at the highest temperature of 85℃, DRAM modules have a high potential of reducing their access latency: 21.1% on average for read, and 34.4% on average for write operations. This is a direct result of the possible reductions in timing parameters $t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$— 15.6%/20.4%/20.6%/28.5% on average across all the DRAM modules. In this dissertation, we present only the *average* potential reduction for each timing parameter. We provide detailed characterization of each DRAM module online at the SAFARI Research Group website [144].

As a result, we conclude that process variation and lower temperatures enable a significant potential to reduce DRAM access latencies. Second, we observe that at lower temperatures (e.g., 55℃) the potential for latency reduction is even greater (32.7% on average for read, and 55.1% on average for write operations), where the corresponding reduction in timing parameters $t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$ are 17.3%/37.7%/54.8%/35.2% on average across all the DRAM modules.

Given that DRAM latency is a significant bottleneck for system performance, this reduction in timing parameters will directly translate to improvement in overall system performance (as we will show in Section 5.6.2).

5.5.6   Analysis of the Repeatability of Cell Failures

In order to adopt a mechanism that exploits the DRAM latency variation in real systems, it is important to understand whether the latency-margin related errors remain consistent over time. For this purpose, we specifically measured how many cells consistently experience errors under different evaluation conditions. We perform this consistency test for five different scenarios listed in Table 5.1. For each scenario, we choose a refresh interval for which there are at least 1000 errors. We then repeat each test for 10 iterations and report the percentage of cells that appear in all 10 iterations.

As summarized in Table 5.1, the first four scenarios show that a very high fraction (more than 95%) of the erroneous cells consistently experience an error over multiple iterations of the same test. Even though the overlap ratio is high, it is not 100%, which means that the characteristic of some cells may be changing over time. We believe that these effects could be related to the VRT phenomenon (Section 5.5.3). For the fifth scenario (where parameters are set separately for reads and writes), the repeatability of errors is the lowest at 71%. We hypothesize that this is the result of different power-noise conditions (between activation and write) for these two different operations. This is why the read and write operations need to be profiled separately, since they are likely to sensitize errors in different sets of cells.

Scenario	Overlap (%)
10 iterations of the same test	96.94
10 iterations of eight different data patterns	96.01
10 iterations of eight timing-parameter combinations	98.99
10 iterations at two different temperatures (85 vs. 65℃)	96.18
10 iterations of two different test types (read vs. write)	71.59

TABLE 5.1: Repeatability and Consistency of Erroneous Cells

5.6   Real-System Evaluation

We evaluate AL-DRAM on a real system, whose detailed configuration is listed in Table 5.2. We chose this system for its AMD processor, which offers dynamic software-based control

over DRAM timing parameters at runtime [18, 19]. We paired the processor with one or more DRAM modules that have ECC (error-correction code) support. For the purpose of minimizing performance variation, we disabled several optimization features of the system (e.g., dynamic core frequency scaling, adaptive DRAM row policy, and prefetching).

System	Dell PowerEdge R415 [56]
Processor	AMD Opteron 4386 (8 cores, 3.1GHz, 8MB LLC) [18, 19]
Main Memory	6×4GB ECC UDIMM (Single-/Dual-Rank) DDR3-1600 (800MHz clock rate, 1.25ns cycle time) Default ($t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$): 13.75/35.0/15.0/13.75ns Reduced ($t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$): **10.0/23.75/10.0/11.25ns**
Storage	128GB SSD (random read/write speed: 97K/90K IOPS)
Operating System	Linux 3.11.0-19-generic

TABLE 5.2: Evaluated System Configuration

5.6.1 Tuning the Timing Parameters

First, we evaluate the possible latency reduction in DRAM modules without losing any data integrity. We stress the system with memory intensive workloads (99.1% CPU utilization with STREAM [4, 181] running in all eight cores) while reducing the timing parameters. The minimum values of the timing parameters that do not introduce any errors at 55℃ or less define the maximum acceptable reduction in latency in the system. Table 5.2 shows that the potential reduction is 27%/32%/33%/18% for $t_{RCD}/t_{RAS}/t_{WR}/t_{RP}$, respectively. During the evaluation, the observed DRAM temperature range is 30℃–39℃ (always less than 55℃, even at a very high CPU and memory utilization). Therefore, we need only one set of DRAM timing parameters for our real system evaluation.

5.6.2 Performance Improvement

Figure 5.12 shows the performance improvement of reducing the timing parameters in the memory system with one rank and one memory channel. We run a variety of different applications (SPEC, STREAM [4, 181], PARSEC [3], Memcached [1], Apache [5], and GUPS [88]) in two different configurations. The first one (Config. 1) runs only one thread and, the second

one (Config. 2) runs multiple applications/threads (as described in the figure). We run each configuration 30 times (only SPEC is executed 3 times due to the large execution time), and present the average performance improvement across all the runs and their standard deviation as an error bar. Based on the last-level cache misses per kilo instructions (MPKI), we categorize our applications into memory-intensive or non-intensive groups, and report the geometric mean performance improvement across all applications from each group. We draw three key conclusions from Figure 5.12. First, AL-DRAM provides significant performance improvement over the baseline (as high as 20.5% for the very memory-bandwidth-intensive STREAM applications [4, 181]). Second, when the memory system is under higher pressure with multi-core/multi-threaded applications, we observe significantly higher performance (than in the single-core case) across all applications from our workload pool. Third, as expected, memory-intensive applications benefit more in performance than non-memory-intensive workloads (14.0% vs. 2.9% on average). We conclude that by reducing the DRAM timing parameters using our approach, we can speed up a system by 10.5% (on average across all 35 workloads on the multi-core/multi-thread configuration).
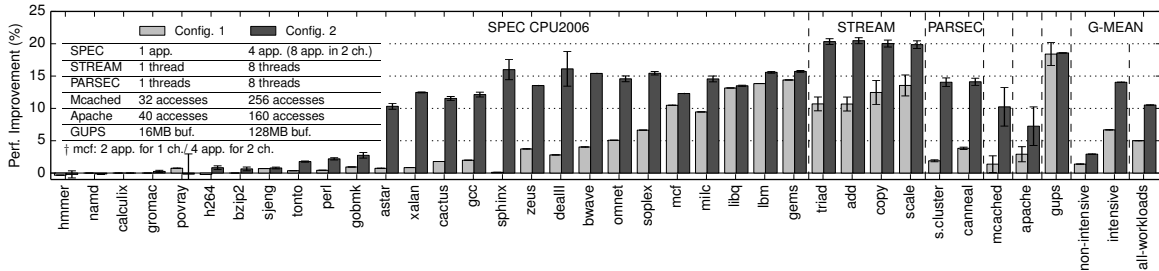
FIGURE 5.12: Real-System Performance Improvement with AL-DRAM (Each Error Bar Shows the Standard Deviation across Multiple Runs)

5.6.3 Reliability of Reduced Timing Parameters

By reducing the timing parameters, we are also stripping away the excessive margin in DRAM's electrical charge. The remaining margin should be enough for DRAM to achieve correctness by overcoming process variation and temperature dependence (as we discussed in Section 5.2.3). To verify the correctness of our experiments, we ran our workloads for 33

days non-stop, and examined their and the system's correctness with reduced timing parameters. Using the reduced timing parameters over the course of 33 days, our real system was able to execute 35 different workloads in both single-core and multi-core configurations (see Figure 5.12) while preserving correctness and being error-free.

Note that these results do *not* absolutely guarantee that no errors can be introduced by reducing the timing parameters. Our real-system experiments are limited in their statistical significance, since they involve a small sample population (six ECC DRAM modules) over a relatively short test duration (33 days). However, DRAM manufacturers *already have the necessary testing methodology to guarantee reliable operation with reduced timing parameters that are appropriately chosen.* Existing industrial-grade methodology for measuring and ensuring reliability (at different timing parameters) is typically based on millions of hours of aggregate test time, which is clearly beyond the scope of this work but is also clearly doable by DRAM manufacturers. Thus, we believe that we have demonstrated a proof-of-concept which shows that DRAM latency can be reduced at little-to-no apparent impact on DRAM reliability.

5.6.4 Sensitivity Analysis

Number of Channels, and Ranks: We analyze the impact of increasing the number of ranks and channels on the performance achieved with AL-DRAM in Figure 5.13. Note that adding ranks enables more memory parallelism while keeping the total memory bandwidth constant, but adding channels also increases the total memory bandwidth. The 2-channel systems we evaluate are overprovisioned for the workloads we evaluate as our workloads exert little pressure on memory in such systems. We make two major observations. First, AL-DRAM significantly improves system performance even on highly-provisioned systems with large numbers of channels, and ranks: 10.6%/5.2%/2.9% on average across our memory-intensive/multi-core workloads in a 2-rank 1-channel/1-rank 2-channel/2-rank 2-channel system, respectively. Second, the benefits AL-DRAM are higher when the system is more memory parallelism- and bandwidth-constrained, as expected, because memory latency becomes a bigger bottleneck when limited parallelism/bandwidth causes contention for memory. As

on-chip computational power continues to increase at a much faster rate than the amount of off-chip memory bandwidth [97, 230] due to the limited pin count, and as future applications become increasingly data-intensive, it is very likely that future systems will be increasingly memory bandwidth constrained [186]. We conclude that AL-DRAM will likely become more effective on such future systems.
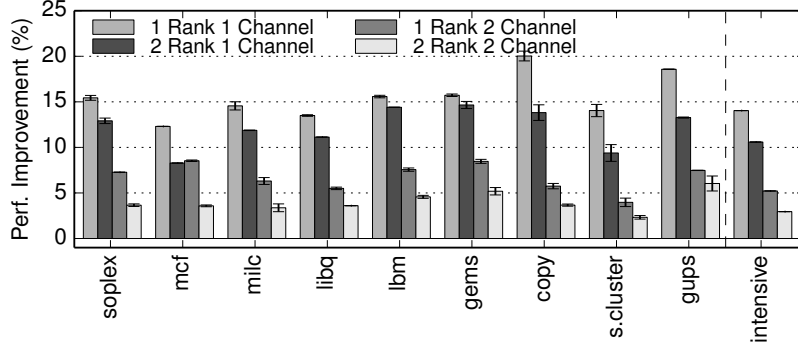
FIGURE 5.13: AL-DRAM Performance Improvement on a Real System with Different Rank and Channel Configurations

Heterogeneous Workloads: We evaluate the performance impact of AL-DRAM for heterogeneous workloads generated by combining two different applications from those listed in Figure 5.13. We observe that AL-DRAM provides significant weighted speedup improvement over the baseline system: 7.1% on average in a 1-rank 2-channel system.

Row Policy: Performance improvement can depend on the row buffer management policy used in the memory system. We simulate our mechanism, AL-DRAM, to analyze the sensitivity of our results to two policies (as our tested system does not have any flexibility to change row policy): open-row & closed-row [124]. We use a modified version of Ramulator [128], a fast cycle-accurate DRAM simulator that is available publicly [123, 127], and Ramulator releases an open-source implementation of AL-DRAM. We use Ramulator combined with a cycle-level x86 multi-core simulator. We use Ramulator as part of a cycle-level in-house x86 multi-core simulator, whose front-end is based on Pin [162].

We evaluate 51 multi-core workloads (randomly-selected from SPEC CPU2006, STREAM, TPC, and GUPS) in a 1-rank 2-channel system. Our evaluation shows that AL-DRAM provides similar performance improvements for both policies (11.4%/11.0% improvement for open/closed row policies over the baseline system).

Energy Efficiency: By reducing the timing parameters and overall execution time, AL-DRAM improves energy efficiency. Unfortunately, we do not have an infrastructure to measure the DRAM energy consumption from the tested real system. Instead, we estimate the DRAM energy consumption of AL-DRAM using a DRAM energy calculator [175]. When using 4GByte modules with DDR3-1600, AL-DRAM reduces DRAM power consumption by 5.8% (in a current specification, I_{DD1} [232]). The major energy reduction is attributed to the reduction in row activation time. We leave the DRAM energy consumption measurement in real systems to future work.

5.7   Summary

The standard DRAM timing constraints are grossly overprovisioned to ensure correct operation for the cell with the lowest retention time at the highest acceptable operating temperature. We make the observation that *i)* a significant majority of DRAM modules do *not* exhibit the worst case behavior and that *ii)* most systems operate at a temperature much lower than the highest acceptable operating temperature, enabling the opportunity to significantly reduce the timing constraints.

Based on these observation, in this chapter, we introduce Adaptive-Latency DRAM (AL-DRAM), a simple and effective mechanism for dynamically optimizing the DRAM timing parameters for the current operating condition without introducing any errors. AL-DRAM dynamically measures the operating temperature of each DRAM module and employs timing constraints optimized for *that DRAM module at that temperature.* Results of our latency profiling experiments on 115 modern DRAM modules show that our approach can significantly reduce four major DRAM timing constraints by 17.3%/37.7%/54.8%/35.2 averaged across all 115 DRAM modules tested. This reduction in latency translates to an average

14% improvement in overall system performance across a wide variety of memory-intensive applications run on a real multi-core system.

We conclude that AL-DRAM is a simple and effective mechanism, which exploits the large margin present in the standard DRAM timing constraints, to reduce DRAM latency.

Chapter 6

AVA-DRAM:

Lowering DRAM Latency by

Exploiting Architecture Variation

In this chapter, we introduce new techniques to improve DRAM latency by taking advantage of the variation in cell latencies. We observe that there is variability in DRAM cells based on their location, which has not been exposed or leveraged by any previous works. Some DRAM cells can be accessed faster than others depending on their physical location. We refer to this variability in cells' access times, caused by the physical organization of DRAM, as *architectural variation*.

Architectural variation arises from the difference in the distance between the cells and the peripheral logic that is used to access these cells (Figure 6.1). The wires connecting the cells to peripheral logic exhibit large resistance and large capacitance [143, 145]. Consequently, cells experience different RC delays based on their distances from the peripheral logic. Cells logic closer to peripheral experience smaller delay and can be accessed faster than the cells located farther from the peripheral logic.

Architectural variation in latency is present in both vertical and horizontal directions in a 2D DRAM cell array (called a mat): *i)* each vertical *column of cells* is connected to a component called a *sense amplifier* and *ii)* each horizontal *row of cells* of a mat is connected

---

We provide detailed characterization of each DRAM module online at the SAFARI Research Group website [142].
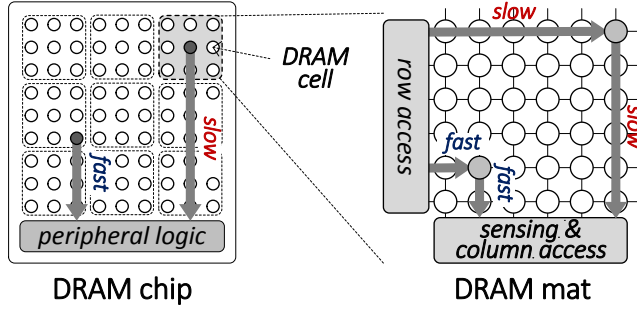
FIGURE 6.1: Architectural Variation in a DRAM Chip

to a *wordline driver*. Variations in the vertical and horizontal dimensions, together, divide the cell array into heterogeneous latency regions, where cells in some regions require larger access latencies for reliable operation. This variation in latency has direct impact on the reliability of the cells. Reducing the latency *uniformly across all regions* in DRAM would improve performance, but can introduce failures in the *inherently slower* regions that have to be accessed longer for correct DRAM operation. We refer to these inherently slower regions of DRAM as *architecturally vulnerable regions*.

The goal of this work is twofold. First, to experimentally demonstrate the existence of architectural variation in modern DRAM chips and identify the architecturally vulnerable regions. Second, to develop new mechanisms that leverage this variation to reduce DRAM latency while providing reliability at low cost.

Identifying architecturally vulnerable regions. Towards achieving our goals, we first identify the architecturally vulnerable regions of DRAM. Doing so is not a trivial task due to two major challenges. First, *identifying architecturally vulnerable regions requires a detailed knowledge of DRAM internals.* Modern DRAM cells are organized in a hierarchical manner, where cells are subdivided into multiple mats and these mats are organized as a matrix (Figure 6.1). Due to this hierarchical organization, we show that the vulnerability of cells *does* not necessarily increase linearly with increasing row and column addresses, but depends on *i)* the location of the cell in that mat and *ii)* the location of the mat in the chip.

Second, *identifying architecturally vulnerable regions is difficult due to the current DRAM interface that does not expose how data corresponding to an address is mapped inside of*

DRAM. Even though certain regions in DRAM might be architecturally vulnerable, internal scrambling and remapping of rows and columns scatters and distributes that region all over the address space [120]. In this work, we provide a detailed analysis on how to identify the vulnerable regions despite the limitations posed by DRAM interface.

To understand the architectural variation of latency in modern DRAM chips, we use an FPGA-based DRAM testing infrastructure to characterize 96 DRAM modules. Our experimental study shows that *i)* Modern DRAM chips exhibit architectural latency variation in both row and column directions, *ii)* Architectural vulnerability gradually increases in the row direction within a mat and repeats the variability pattern in every mat. *iii)* Some columns are more vulnerable than others based on the internal hierarchical design of the specific DRAM chip.

Reducing DRAM latency by exploiting the knowledge of architecturally vulnerable regions. We develop two new mechanisms that exploit the architecturally vulnerable regions to enable low DRAM latency with high reliability and at low cost. First, we propose to reduce the DRAM latency at runtime, by identifying the lowest possible latency that still ensures reliable operation. To this end, we develop an online DRAM testing mechanism, called *AVA Profiling*. The key idea is to periodically test *only* the architecturally vulnerable regions to find the minimum possible DRAM latency (for reliable operation), as these regions would exhibit failures earlier than others when reducing the latency and therefore, would indicate the latency boundary where further reduction in latency would hurt reliability.

Second, to reduce the DRAM latency even further beyond the point of reliable operation, we propose a mechanism to reduce multi-bit failures while operating at a lower latency, called *AVA Shuffling*. The key idea is to leverage the error characteristics in architecturally vulnerable regions to remap or shuffle data such that the failing bits get spread over multiple ECC code words and become correctable by ECC.

6.1 Architectural Variation

In this work, we show that DRAM access latency varies based on the location of the cells in the DRAM hierarchy. Intuitively, transferring data from the cells near the IO interfaces incurs less time than transferring data from the cells farther away from the IO interfaces. We refer to this variability in the cell latency caused by the physical organization of DRAM as *architectural variation*.

Properties of Architectural Variation. Architectural variation has specific characteristics that clearly distinguish it from other known types of variation observed in DRAM cells (e.g., process variation and temperature dependency [37, 143]). Architectural variation in DRAM has the following four characteristics.

- **Predetermined at design time.** Architectural variation depends on the internal DRAM design. As a result, it is predetermined at *design time*. This is unlike other types of variation, (e.g., process variation and temperature induced variation [37, 143]), which depend on the manufacturing process after design.

- **Static distribution.** The distribution of architectural variation is static. For example, a cell closer to the sense amplifier is *always* faster than a cell that is farther away from the sense amplifier, assuming there are no other sources of variation (e.g., process variation). On the other hand, prior works show that variability due to process variation follows a random distribution [37, 143].

- **Constant.** Architectural variation depends on the physical organization, which remains constant over time. Therefore, it is different from other types of variation that change over time (e.g., variable retention time [119, 121, 155], wear-out due to aging [92, 148, 171, 177, 237, 253, 254, 265, 278]).

- **Similarity in DRAMs with the same design.** DRAMs that share the same internal design and organization exhibit similar architectural variation, unlike process variation that manifests significantly differently in different DRAM chips with the same design.[1]

Architectural Variation in the DRAM Hierarchy. As we mentioned earlier, architectural variation arises from the difference in distance between the DRAM cells and the peripheral logic that is used to access the cells. As DRAM is internally organized as a multi-level hierarchy (in the form of chips, banks and ranks), architectural variation exists at multiple levels.

In this work, we focus on the architectural variation within and across mats, as they are the smallest units in DRAM and it is inherently difficult for the manufacturers to minimize this variation by dividing the mats into even smaller units. Furthermore, such architectural variation in mats is becoming only worse with technology scaling. That is mainly because integrating more cells within a DRAM chip requires increasing the length of the wordline and bitline to amortize the area of wordline drivers and sense amplifiers across more cells. In a majority of today's DRAMs, a mat consists of 512×512 cells, while more recently, DRAM with mats consisting of 1024×1024 cells has been introduced [152].

The goal of this work is to *i)* experimentally demonstrate, characterize, and understand the architectural variation in modern DRAM chips and *ii)* leverage this variation and our understanding of it to reduce DRAM latency at low cost in a reliable way. Unfortunately, detecting the architecturally vulnerable regions is not trivial and depends on two factors *i)* how bitline and wordline drivers are organized internally and *ii)* how data from a cell is accessed through the DRAM interface. In order to define and understand the architectural variation in modern DRAM chips, we investigate three major research questions related to the impact of DRAM organization, interface, and operating conditions on architectural variation and provide answers to these questions in the following sections (Section 6.1.1–6.1.3).

---

[1]To increase yield, modern DRAM integrates redundant rows and columns in its banks (usually 1 to 3% of total cells). When uncovering faulty cells during manufacturing time, those faulty rows and columns are remapped to the redundant rows and columns (row/column repair). Since remapped rows and columns are different in different DRAM chips, we expect to observe small variations in architectural variation of different chips which share the same design.

6.1.1 Impact of DRAM Organization

The first question we answer is: *how does the DRAM organization affect the architecturally vulnerable regions?* Based on a detailed understanding of DRAM internal organization, we provide hypotheses on the characteristics of architectural variation and systematic methodologies to identify these characteristics in modern DRAM chips. Using these methodologies, we experimentally validate our hypotheses with the results we provide in Section 6.3.

Effect of Row Organization on Architectural Variation. As discussed in Chapter 2, a mat consists of a 2D array of DRAM cells along with peripheral logic needed to access this data. In the vertical direction, DRAM cells, connected through a bitline, share a local sense amplifier (typically, 512 cells [275]). As a result, variation in access latency gradually increases as the distance of a row from the local sense amplifier increases (due to the longer latency of propagation delay through the bitline). This variation can be exposed by overclocking the DRAM by using smaller values for DRAM timing parameters. Cells in the rows closer to the local sense amplifiers can be accessed faster, so they exhibit no failures due to overclocking. On the contrary, cells located farther away from the sense amplifier need longer time to access, and might start failing when smaller values are used for the timing parameters. As a result, accessing rows in ascending order starting from the row closest to the sense amplifiers should exhibit gradual increase in failures due to architectural variation, as shown in Figure 6.2a. In this figure, darker colors indicate slower cells, which are more vulnerable to failures, if we reduce access latency.

(A) Conceptual Bitline   (B) Open Bitline Scheme
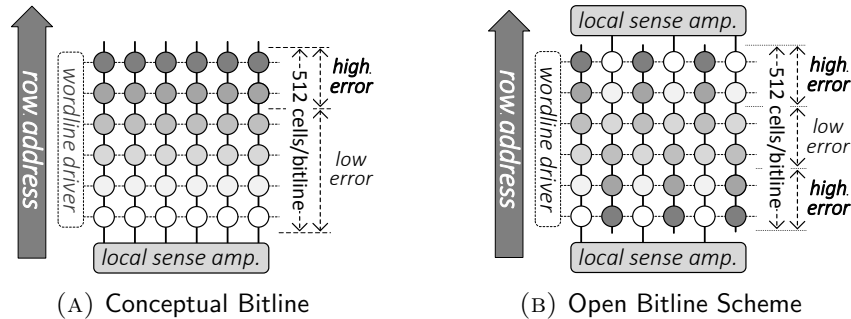
FIGURE 6.2: Architectural Variation due to Row Organization

In the open-bitline scheme [96], alternate bitlines within a mat are connected to two different rows of sense amplifiers (the top and bottom of the mat), as shown in Figure 6.2b. In this scheme, even cells and odd cells in a row located at the edge of the mat exhibit very different distances from their corresponding sense amplifiers, leading to different access latencies. On the other hand, cells in the middle of a mat have similar distance from both the top and bottom sense amplifiers, exhibiting similar latencies. Due to this organization, we observe that there are more failures in rows located on both ends of a mat, but there is a gradual decrease in failures for rows in the middle of the mat.

Based on these, we define two characteristics of vulnerable regions across the rows when we reduce DRAM latency uniformly. First, **the number of failures would gradually increase with increased distance from the sense amplifiers**. Second, **this gradual increase in failures would periodically repeat in every mat (every 512 rows)**. We experimentally demonstrate these characteristics in Section 6.3.1.

Effect of Column Organization on Architectural Variation. As we discussed in Section 2.1, the wordline drivers in DRAM are organized in a hierarchical manner, where a global wordline is connected to *all* mats within a row and then a local wordline driver activates a *single* row within a mat. This *hierarchical wordline organization* leads to latency variation at two levels. First, a wordline in a mat located closer to the global wordline driver starts activating a row earlier than a mat located farther away from the global wordline driver (*architectural variation due to the global wordline*). Second, within a mat, a cell closer to the local wordline driver gets activated faster than a cell farther away from the local wordline driver (*architectural variation in local wordline*). Therefore, columns that have the same distance from the local wordline driver, but located in two different mats will have different latency characteristics (Figure 6.3, where darker color indicates slower cells, which are more vulnerable to failures, if we reduce access latency).

Based on these, we define two characteristics of vulnerable regions across columns when we reduce DRAM latency uniformly. First, **although some columns are more vulnerable than others, the number of failures would *not* gradually increase with ascending column numbers**. Second, **the failure characteristics observed with ascending**
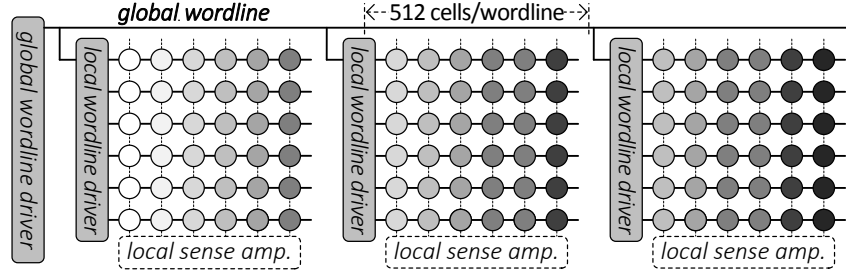
FIGURE 6.3: Architectural Variation in Column Organization

column numbers would be similar for all rows. We experimentally demonstrate these characteristics in Section 6.3.2.

6.1.2 Impact of the Row/Column Interface

Our second question is: *how does the row/column interface affect the ability to identify the architecturally vulnerable regions in DRAM?* Unfortunately, identifying architecturally vulnerable regions becomes challenging due to a limited understanding of how data corresponding to an address is mapped inside DRAM. While it is possible to identify vulnerable regions based on location, exposing and exploiting such information through the row/column DRAM addressing interface is challenging due to two reasons.

Row Interface (Row Address Mapping). DRAM manufacturers internally scramble the row addresses in DRAM making the address known to the system different from the actual physical address [273]. As a result, consecutive row addresses issued by the memory controller, can be mapped to entirely different regions of DRAM. Unfortunately, the mapping of the row addresses is not *exposed* to the system and varies across products from different generations and manufacturers. In the previous section we showed that if latency is reduced, accessing rows in mats in ascending row numbers would exhibit gradual increase in failures. Unfortunately, due to row remapping, accessing rows in ascending order of addresses known to memory controller will exhibit irregular and scattered failure characteristics.

Column Interface (Column Address Mapping). In the current column interface, the column addresses issued by the memory controller do not access consecutive columns in a mat, making it challenging to identify the vulnerable regions in a wordline. When a column

96

address is issued, 64-bit of data from a row is transferred with global bitlines (typically, 64-bit width [275]). Then, this data is transferred in eight 8-bit bursts over the IO channel as shown in Figure 6.4. However, data transferred with each column address comes from different mats, making it impossible to always access *consecutive* physical columns in a mat by simply *increasing* the column address.

(A) Data Mapping

(B) Data Burst (Data Out)
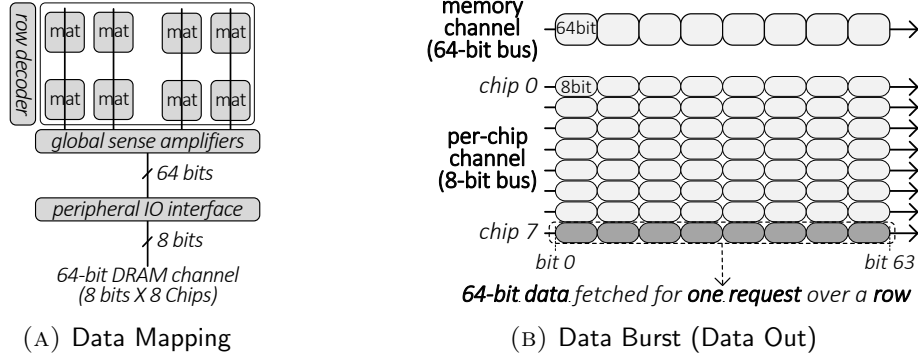
FIGURE 6.4: Accessing Multiple Mats in a Data Burst

In this work, we provide alternate ways to identify architecturally vulnerable DRAM regions using the current row/column interface in DRAM. We describe the key ideas of our methods below. Section 6.3.3 and 6.3.4 provide the detailed analysis with experimental validation of our methods in real DRAM chips.

- *Inferring vulnerable rows from per-row failure count.* In order to identify the gradual increase in architecture variability with increasing row addresses in mats (in terms of internal DRAM physical address), we try to reverse engineer the row mapping in DRAM. We hypothesize the mapping for one mat and then verify that mapping in other DRAM mats in different chips that share the same design. The key idea is to correlate the number of failures to the physical location of the row. For example, the most vulnerable row would be the one with the most failures and hence should be located at the edge of the mat. Section 6.3.3 provides experimental validation of our method.

- *Inferring vulnerable columns from per-bit failure count in the IO channel.* As we explained earlier, a column access transfers 64 bits of data from a chip over the IO channel. These 64 bits come from 64 bitlines that are distributed over different mats across the entire

row. Our key idea to identify the vulnerable bitlines in the column direction is to examine each bit in the 64-bit data burst. We expect that due to architectural variation, some bits in a 64-bit burst that are mapped to slower bitlines than others, are more vulnerable than other bits. In Section 6.3.4, we experimentally identify the location of bits in bursts that consistently exhibit more failures, validating the existence of architectural variation in columns.

6.1.3   Impact of Operating Conditions

The third question we seek to answer is: *Does architectural variation in latency show similar characteristics at different operating conditions?* DRAM cells get affected by temperature and refresh interval. Increasing the temperature or refresh interval increases the leakage in cells, making them more vulnerable to failure. However, as all the cells in DRAM get similarly affected by changes in operating condition, we observe that the trends due to architectural variation remain similar at different temperatures and refresh intervals, even though the absolute number of failures may change. We provide the detailed experimental analysis of architectural variation at different operating conditions in Section 6.3.5.

6.2   DRAM Testing Methodology

In this section, we describe our FPGA-based DRAM testing infrastructure and the testing methodology we use for our experimental studies in Section 6.3.

FPGA-Based DRAM Testing Infrastructure. We build infrastructure similar to that used for AL-DRAM in Chapter 5 and previous work [37, 39, 119, 120, 122, 155, 217]. Our infrastructure provides the ability to: *i)* generate test patterns with flexible DRAM timing parameters, *ii)* provide an interface from a host machine to the FPGA test infrastructure, and *iii)* maintain a stable DRAM operating temperature during experiments. We use a Xilinx ML605 board [285] that includes an FPGA-based memory controller which is connected to a DDR3 SODIMM socket (Figure 6.5b). We designed the memory controller [286] with the flexibility to change DRAM parameters. We connect this FPGA board to the host machine

through a PCI-e interface [284], as shown in Figure 6.5a. We manage the FPGA board from the host machine and preserve the test results in the host machine's storage. In order to maintain a stable operating temperature for the DRAM modules, during our experiments, we place the FPGA board in a heat chamber that consists of a temperature controller, a temperature sensor, and a heater which enables us to test at different temperatures (Figure 6.5a).

(A) Full System   (B) FPGA Board
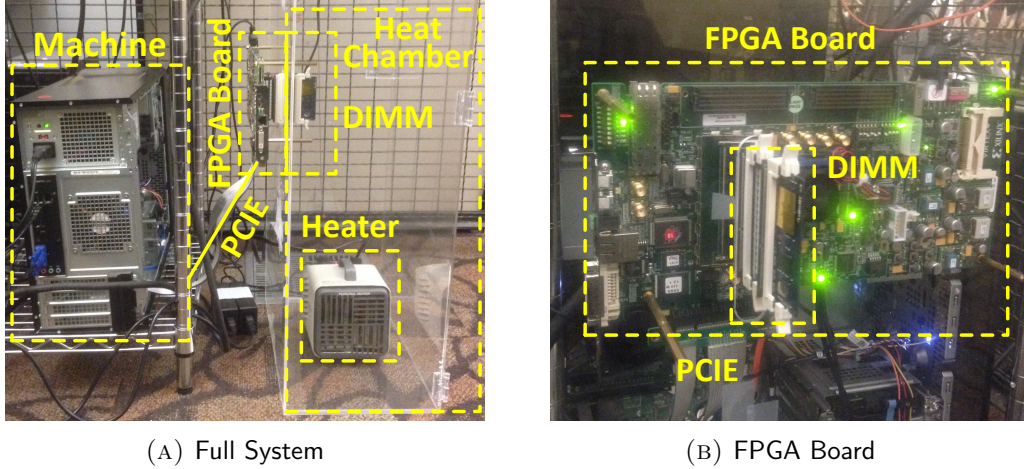
FIGURE 6.5: FPGA-Based DRAM Test Infrastructure

Profiling Methodology. The major purpose of our experiments is to characterize architectural variation in latency. We would like to *i)* determine the characteristics of failures when we reduce timing parameters beyond the error-free operation regions, and *ii)* observe any correlation between the error characteristics and the internal architecture/organization of the DRAM modules. To this end, we analyze the error characteristics of DRAM by lowering DRAM timing parameters below the values specified for error-free operation.

An experiment consists of three major steps: *i) writing background data*, *ii) changing timing parameters*, and *iii) verifying cell content*. In Step 1, we write a certain data pattern to the entire DRAM module with standard DRAM timing parameters, ensuring that correct (the intended) data is written into all cells. In Step 2, we access DRAM with the changed timing parameters, and wait for the *refresh interval* such that DRAM cells have the smallest charge due to charge leakage over time. In Step 3, we verify the content of the DRAM cells

after the timing parameters are changed. To pass verification, a DRAM cell must maintain its data value until the next refresh operation. If the data read in Step 3 does not match the data written in Step 1, we log the addresses corresponding to the failure and the order of bits in the failed address.

Data Patterns. In order to exercise worst-case latency behavior, we use a row stripe pattern, wherein a test pattern is written in odd rows and an inverted test pattern is written in even rows [273]. This pattern drives the bitlines in opposite directions when accessing adjacent rows. The patterns we have used in our tests are 0000, 0101, 0011, and 1001. We perform the test twice per pattern, once with the test data pattern and once with the inverted version of the test data pattern, in order to test every cell in charged (e.g., data 1) and non-charged states (e.g., data 0). We report the sum of failures from these two cases for each test. We perform 10 iterations of the same test for each DRAM module to make sure the errors are consistent.

We evaluate three DRAM timing parameters, t_{RCD}, t_{RAS}, and t_{RP}. For each timing parameter, our evaluations start from the standard DRAM timing parameters (13.75/35.0/13.75ns for $t_{RCD}/t_{RAS}/t_{RP}$, respectively) [176] and reduce the timing parameters to the lowest values that our DRAM infrastructure allows (5ns for t_{RCD} and t_{RAS}, and t_{RCD} + 10ns for t_{RAS}). We use 96 DRAM modules (DDR3-1600 [176]), comprising 768 DRAM chips, from three DRAM manufacturers for our experiments.

6.3 DRAM Test Results and Analysis

In this section, we present the results of our profiling studies that demonstrate the presence of architectural variation in both the vertical (bitline) and horizontal (wordline) directions. We *i)* show the existence of architectural variation in Sections 6.3.1 and 6.3.2, *ii)* analyze the impact of the row and column interface in Sections 6.3.3 and 6.3.4, and *iii)* characterize the impact of operating conditions on architectural variation in Section 6.3.5. We then provide a summary of our analysis on architectural latency variation across 96 DRAM modules (Section 6.3.6).

6.3.1 Architectural Variation in Bitlines

As we explain in Section 6.1.1, we expect different error characteristics for cells connected to a bitline, depending on the distance from the local sense amplifiers. To demonstrate the existence of architectural variation in a bitline, we design a test pattern that sweeps the row address.

Per-Row Error Count with Row Address Sweeping. Figure 6.6 plots the error count for three values of a DRAM timing parameter, t_{RP} (whose standard value is 13.75ns), with a refresh interval of 256 ms (greater than the normal 64 ms refresh interval to emphasize the effects of access latency) and an ambient temperature of 85℃. We tested all rows (and 16 columns) in a DRAM module and plot the number of erroneous accesses for every modulo 512 rows.[2] We accumulate errors every modulo 512 rows because, *i)* each bitline is connected to 512 cells, and *ii)* our expectation that the architectural variation pattern will repeat every 512 cells (we provide each row's error count in Figure 6.7b). We draw two key observations. First, reducing a timing parameter below its standard value induces errors, and reducing it further induces more errors. At a t_{RP} of 10.0ns (3.75ns reduction from the standard value), the number of errors is small, while at a t_{RP} of 5.0ns, we observe a large number of errors. Second, we observe error count variation across rows only at 7.5ns (from 0 to more than 3500 in Figure 6.6c), while most errors are *randomly* distributed at 10.0ns (Figure 6.6b) and most rows show very high error counts at 5.0ns (Figure 6.6d).

Periodicity in Per-Row Error Count. To understand these trends better, we sort the error counts in every row address modulo 512, with a t_{RP} of 7.5ns (data in Figure 6.6c), as shown in Figure 6.7a. We then apply this sorting to every group of 512 rows (e.g., first row to 512th row as the first group, 513th row to 1024th row as the second group, and so on). We plot the error counts of individual rows in Figure 6.7b. The reason why we do this sorting is because we expect periodicity in error counts, which is not shown in Figure 6.6c. Figures 6.7a and 6.7b show similar characteristics: error increases periodically across 512

---

[2]Even though there are redundant cells (rows), DRAM does not allow direct access to redundant cells. Therefore, we can only access a 512×512 cell mat (2^n data chunk). Figure 6.6 plots the number of erroneous requests in every 512 cell chunk.
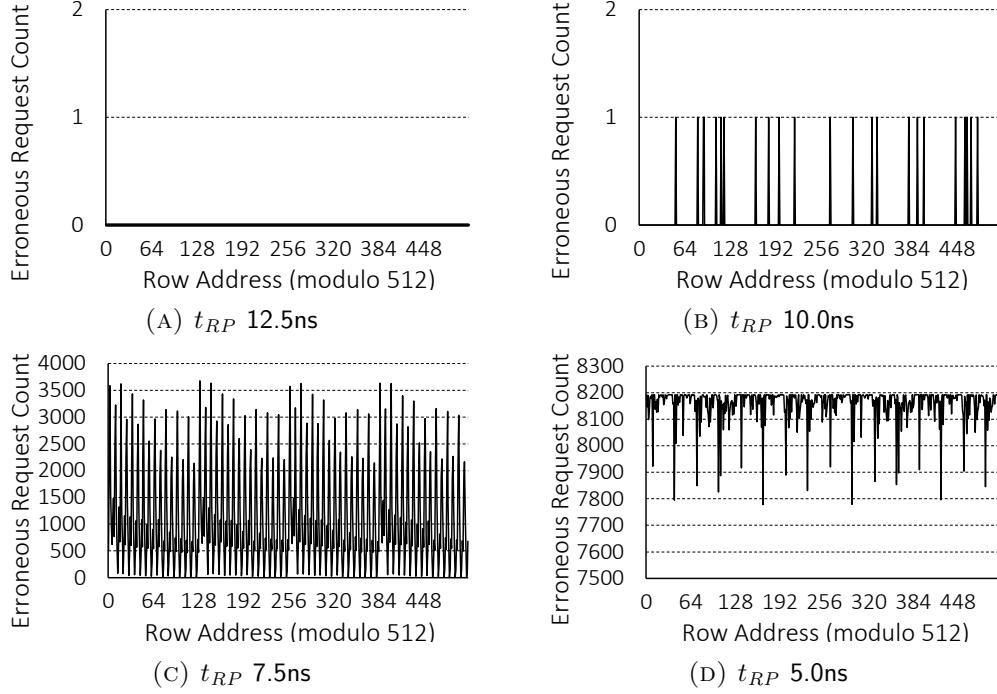
FIGURE 6.6: Erroneous Request Count when Sweeping Row Addresses with Reduced t_{RP} Timing Parameter

rows. Therefore, we conclude that *error count shows periodicity with row address*, confirming our expectation that there is predictable architectural variation in the latency of cells across a bitline.

6.3.2 Architectural Variation in Wordlines

As we explained in Section 6.1.1, we expect architectural variation across cells in a wordline, depending on the distance from the wordline driver. To confirm the existence of architectural variation across a wordline, we use a similar evaluation methodology as the one used in Section 6.3.1, except that *i)* we sweep the column address instead of the row address, *ii)* merge errors in the same column across multiple rows (128 columns in a row). In order to minimize the impact of variation across a bitline and focus on variation across a wordline, as shown in Section 6.3.1, we test columns in only 16 rows.

Per-Column Error Count with Column Address Sweeping. Figure 6.8 provides results with two t_{RP} values (10ns and 7.5ns). Similar to the evaluation with sweeping row
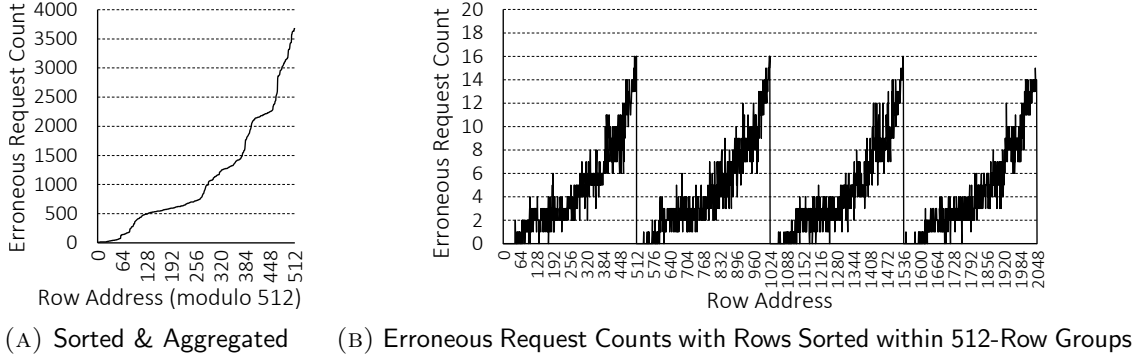
FIGURE 6.7: Periodicity in Error Request Count (t_{RP} 7.5ns)

addresses, we see that the number of errors is small and the distribution is random when t_{RP} is reduced by a small amount, as shown in Figure 6.8a. However, the number of errors is large when t_{RP} is reduced significantly, as shown in Figure 6.8b. We observe variations in error counts across different column addresses at a t_{RP} of 7.5ns. Besides other variations, there is a large jump near the 48th column and a dip in error count near the 96th column, as shown in Figure 6.8b.

To understand these, we separately plot each row's error count, which displays different patterns. We provide two such types of patterns (from multiple rows) in Figures 6.8c and 6.8d. In one such type, shown in Figure 6.8c, the error count drastically increases at around the 80th column and drops at around the 96th column (There are other types of patterns with similar shapes but with the jumps/drops happening at different locations). In the type of pattern shown in Figure 6.8d, the error count drastically increases at 96th column and stays high. We attempt to correlate such behavior with the internal organization of DRAM.

Figure 6.9 shows an illustration of how the precharge control signal flows across mats. The timing parameter t_{RP} dictates how long the memory controller should wait after it issues a precharge command before it issues the next command. When a precharge command is issued, the precharge signal propagates to the local sense amplifiers in each mat, leading to propagation delay (higher for sense amplifiers that are farther away). To mitigate this variation in the delay of the precharge control signal, DRAM uses two signals, *i)* main precharge

103

(A) t_{RP} 10ns & Aggregated

(B) t_{RP} 7.5ns & Aggregated

(C) t_{RP} 7.5ns & Case 1

(D) t_{RP} 7.5ns & Case 2
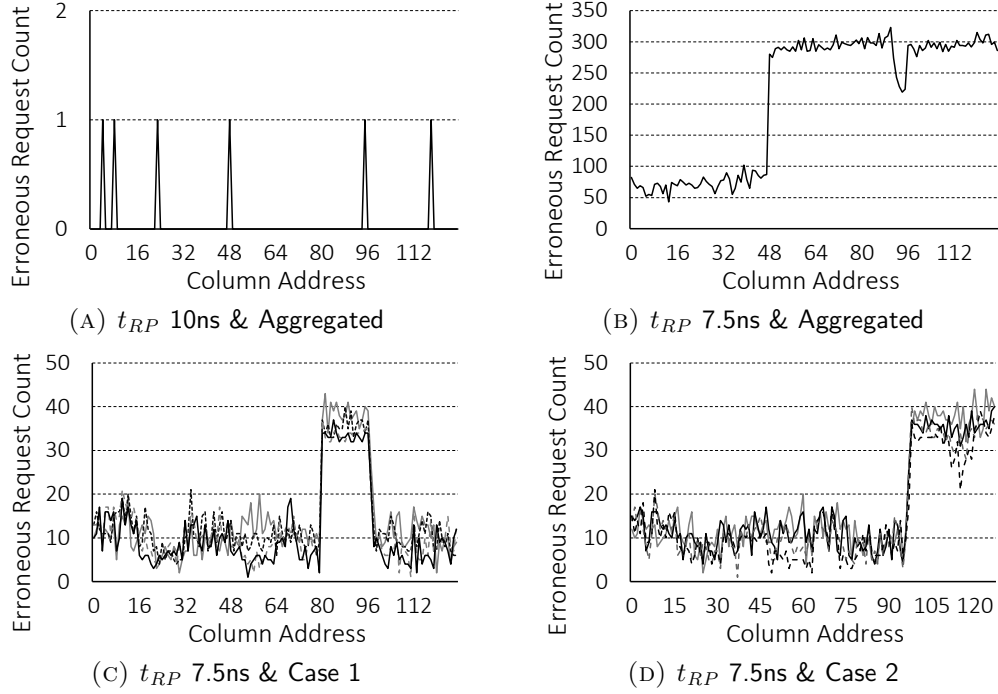
FIGURE 6.8: Erroneous Request Count when Sweeping Column Addresses with Reduced t_{RP} Timing Parameter

signal – propagating from left to right, and *ii)* sub precharge signal – that directly reaches the right and propagates from right to left.

The main and sub precharge signals arrive at different times at the different mats due to parasitic capacitance on the propagation path. The main precharge signal is delayed by α per mat going from left to right, while the sub precharge signal is delayed by β per mat $\alpha > \beta$, since the sub precharge signal does not have any load going from left to right. However, after that, the sub precharge signal exhibits a delay of α per mat when propagating through mats from right to left. The sense amplifiers in a mat respond to the faster one of the two precharge signals. For instance, in the illustration in Figure 6.9, mat 3 receives the precharge signal the last. Hence, accesses to it would exhibit more errors than accesses to other mats if t_{RP} is reduced. Such control signal delays result in the kind of jumps in errors at particular column addresses we see in real DRAM chips (e.g., Figures 6.8b, 6.8c, 6.8d). We conclude that error count varies across columns, based on the column's distance from the wordline and control signal drivers. While such control signal delays explain why such jumps occur,

104
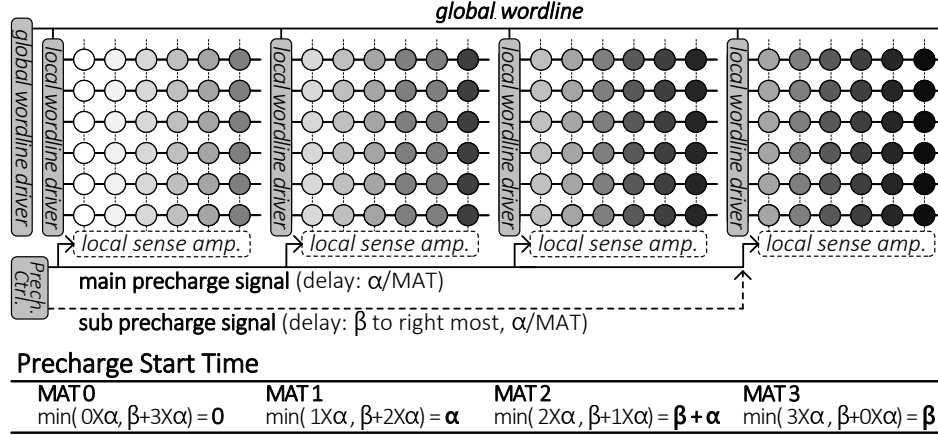
FIGURE 6.9: Architectural Variation due to Precharge Control

knowledge of the exact location of mats and how they are connected to the control signals is necessary to tie back the control signal propagation to the specific column addresses at which the jumps occur.

6.3.3 Effect of the Row Interface

As shown in Figure 6.6c, the error count across a bitline does not linearly increase with increasing *DRAM-external row address* – the address issued by the memory controller over the memory channel, while we observe periodicity when rows are sorted by error count, in Section 6.3.1. This is mainly because the DRAM-external row address is *not* directly mapped to the internal row address in a DRAM mat [120, 155]. Without information on this mapping, it is difficult to tie the error count periodicity to specific external row addresses. In this subsection, we estimate the most-likely mapping between the DRAM-external row address and the DRAM-internal row address (*estimated row mapping*) based on the observed error count. We then analyze the similarity of the estimated row address mapping across multiple DRAM modules manufactured by the same DRAM company (in the same time frame).

Methodology for Estimating Row Address Mapping. We explain our estimation methodology using a simple example shown in Figure 6.10, which has a 3-bit row address

(eight rows per mat). Figure 6.10a shows the DRAM-internal row address in both decimal and binary, increasing in the order of distance between the row and the local sense amplifier.

(A) Internal Address
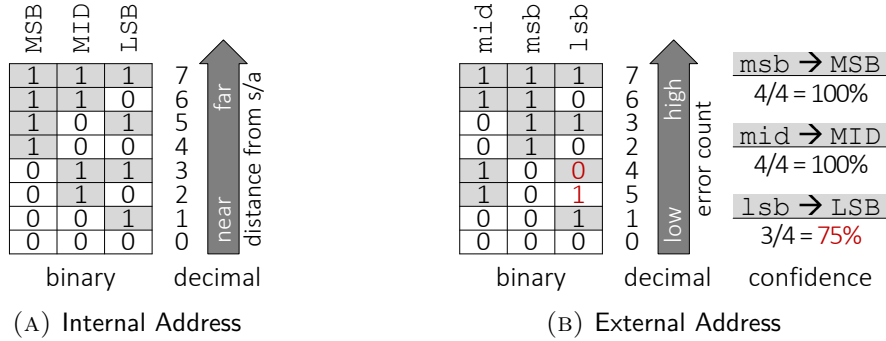
(B) External Address

FIGURE 6.10: DRAM-External Address vs. DRAM-Internal Address

Figure 6.10b shows DRAM-external row addresses which are *ranked based on the error counts.* As observed, the order is not the same as the DRAM-internal address order in Figure 6.10a. To determine the estimated external to internal row mapping, we try to find the bit values across bit positions that display the largest similarity. For instance, MSB in the internal address has four consecutive "1"s and "0"s (in order from largest to smallest rows). When comparing this with the external address, we see that the middle bit of the external address matches this order exactly (100% confidence). We compare the external and internal address bits and identify which bit positions in the external address map to which bit positions in the internal address. The estimated mapping (in the logical address) is indicated by dark boxes when the expected bit is "1" and light boxes when the expected bit is "0". There are cases when this mapping does not match with the actual external address (indicated in red).

Estimated Row Address Mapping in Real DRAM modules. We perform such an external to internal address mapping comparison and mapping exercise on eight DRAM modules manufactured by the same company in a similar time frame. Figure 6.11 shows the average confidence level of the estimated row mapping along with the standard deviation over the eight chips. We make three observations. First, all DRAM modules show the same estimated row mapping (with fairly high confidence) for at least the five most significant bits.

This result shows that DRAM modules manufactured by the same company at the same time have similar architectural variation. Second, the confidence level is almost always less than 100%. This is because process variation introduces perturbations besides architectural variation, which can change the ranking of rows (determined based on error counts). Third, the confidence level drops gradually from MSB to LSB. This is also due to the impact of process variation. The noise from process variation and row repair can change row ranking and grouping by error count. Address bits closer to MSB tend to divide rows into groups at a larger granularity than address bits closer to LSB. Therefore, the higher order bits show higher confidence. Based on these observations, we conclude that DRAMs that have the same design display similar error characteristics due to architectural latency variation.
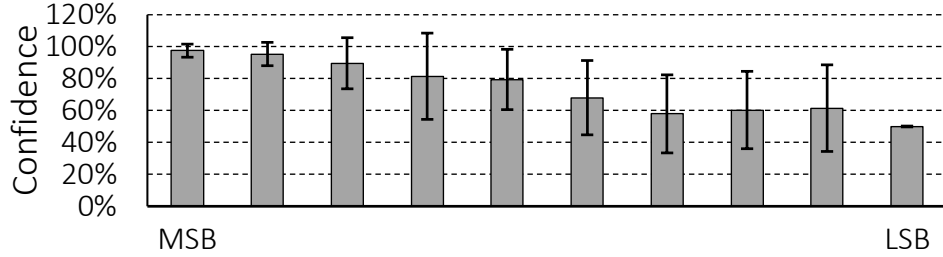
FIGURE 6.11: Confidence in Estimated Row Mapping for Each Bit

DRAM modules from two companies show this similarity of the estimated row address mapping. However, extracting the estimated row address mapping by this mechanism does not work for DRAM modules from a company. We expect that DRAMs from the company may have a symmetric organization in terms of external addresses. For example, MSB = 0 rows are distributed evenly over a mat. We leave the correlation between external addresses and internal organization for the DRAM modules from the company to future work.

6.3.4   Effect of the Column Interface

Another way to observe the error characteristics in the wordline organization is using the *mapping between the global sense amplifier and the IO channel*. As we explained, global sense amplifiers in a DRAM chip concurrently read 64-bit data from different locations of a row, leading to variation in errors. Figure 6.12 plots errors in 64-bit data-out (as shown in

Figure 6.4) in the IO channel (For example, first eight bits (bit 0 – 7) are the first burst of data transfer). We provide two conclusions. First, there is large variation in the amount of errors in the IO channel. For example, more than 26K errors happen in the third bit while no errors in the first bit of the IO channel. Second, the error characteristics of eight DRAM chips show similar trends. Section 6.4.2 uses these observations to develop a new error correction mechanism.
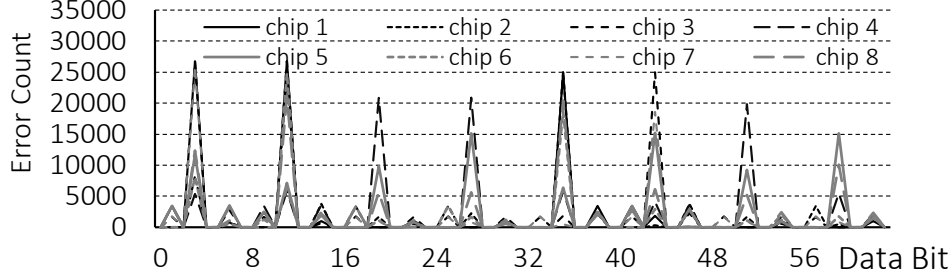
FIGURE 6.12: Error Count in Bit Positions

6.3.5 Effect of Operating Conditions

Figure 6.13 shows the error count sensitivity to the refresh interval and the operating temperature by using the same method as row sweeping (accumulating errors in modulo 512 rows as done in Section 6.3.1). We make three observations. First, neither the refresh interval nor temperature changes the overall trends of architectural variation (e.g., variability characteristics in different row addresses remain the same, though the absolute number of errors changes). Second, reducing the refresh interval or the ambient temperate leads to fewer errors.[3] Third, the variability in cells is much more sensitive to the ambient temperature than the refresh interval. When changing the refresh interval, the total error count does not change drastically (exhibits only 15% decrease in error count with 4X reduction in refresh interval). On the other hand, changing the ambient temperature has a large impact on the total error count (90% decrease in the total error count with 45℃ change in temperature). This is due

---

[3]We observe this trend over most of timing parameters. One possible exception against our observations is the tWR timing parameter which is worse at low temperatures due to increased cell contact resistance. However, this effect is known to happen below 25℃. In the temperature range we evaluate (45℃ to 85℃), the effect of cell leakage (more pronounced at high temperatures) is dominant, leading to more errors at higher temperatures. We will incorporate this discussion in the paper.

the fact that frequent refreshes impact only the cells and make them faster, whereas reducing temperature makes not only the cells but also the peripheral circuits faster. Based on these observations, we conclude that temperature or refresh does not change the trends in variability of the cells, however they impact the total number of failures in vulnerable regions at different rates.

(A) Varying Retention Time

(B) Varying Temperature
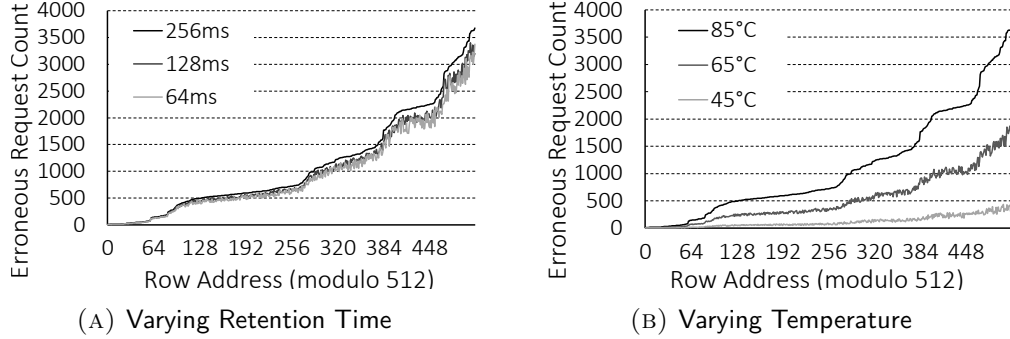
FIGURE 6.13: Architectural Variation vs. Operating Conditions

6.3.6 Summary Results of 96 DRAM modules

In this work, we profile 96 DRAM modules with 768 chips from three different manufacturers to characterize the architectural variation in modern DRAM chips. We observe similar trends and characteristics in modules from the same generation, though the absolute number of failures are different. In Table 6.1, we summarize how many DRAM modules we observe architectural variation out of the test DRAM modules from three major DRAM vendors.

	t_{RCD} (Observed/Tested)	t_{RP} (Observed/Tested)
Vendor A	27/30 (90.0%)	26/30 (86.7%)
Vendor B	16/30 (53.3%)	4/30 (13.3%)
Vendor C	14/36 (38.9%)	27/36 (75.0%)
Total	72/96 (75.0%)	

TABLE 6.1: DRAM Modules, Observed Architectural Variation

We make two observations from this table. First, most of the tested DRAM modules exhibit architectural variation (72 out of 96 modules, 75%). Second, we did not observe architectural variation in 24 DRAM modules. However, we believe that this is due to the

109

limitation in our infrastructure that can only reduce timing parameters at a coarser granularity (step of 2.5 ns). As a result, sometimes it is possible to miss the timing where architectural variation is clearly visible and enter a region where latency is low enough to make all cells fail. We believe that in real machines where state-of-the-art DRAM uses much lower clock period (e.g., DDR3-2133: 0.94ns), architectural variation will be prevalent.

Thus, we have experimentally demonstrated that architectural variation is prevalent across a large number of DRAM modules and our observations hold true in most of the modules. We conclude that most modern DRAM chips are amenable to reducing latency by exploiting architectural variation.

6.4 Mechanisms to Reduce Latency

So far, we have described the phenomenon of architectural variation, studied the reasons behind it, and provided the experimental results demonstrating the presence and characteristics of architectural variation. In this section, we focus on leveraging architectural variation to achieve low DRAM latency while maintaining reliability. Specifically, we propose two techniques, *i)* architectural-variation-aware online DRAM profiling (AVA Profiling) to determine how much DRAM latency can be safely reduced while still achieving failure-free operation and *ii)* architectural-variation-aware data shuffling (AVA Shuffling) to avoid uncorrectable failures (due to lower latency) in systems with ECC.

6.4.1 Architectural Variation Aware Online Latency Profiling

Previous works (including our second mechanism, AL-DRAM, in Chapter 5) observe that the standard DRAM timing parameter values are determined based on the worst-case impact of process variation and leverage this observation to reduce overall DRAM latency during normal operating conditions [37, 143]. For instance, AL-DRAM (in Chapter 5) observed large process and temperature dependency among different DRAM cells' access latencies. As a result, DRAM timing parameters can be lowered using the slowest cells in each DRAM module at current operating temperatures instead of using standard DRAM timing parameters. These

works have two shortcomings. They *i)* assume DRAM manufacturers would determine reliable timing parameters for multiple operating conditions, *ii)* do not take into account DRAM latency changes over time due to aging and wear out.

One solution to both shortcomings is to profile DRAM characteristics online. However, this solution has large performance overhead [68, 222, 248]. Our work develops a *dynamic* and *low cost* DRAM profiling technique that leverages architectural variation in DRAM. We call this technique *Architectural Variation Aware Online DRAM Profiling* (or simply *AVA Profiling*). The key idea is to *i)* separate errors into two categories, those caused by architectural variation and those caused by process variation, and then *ii)* employ different error mitigation techniques for these two categories.

Architectural Variation vs. Process Variation. The error characteristics from *i)* process variation and *ii)* architectural variation are very different. First, the errors caused by process variation are usually randomly distributed over the entire DRAM chip [37, 143]. Figure 6.14a illustrates random errors caused by process variation. As darker cells hold less charge, leading to higher access latency, these darker cells start failing earlier than the lighter cells. Because these errors are random, they often affect only individual DRAM cells around them. In this case, existing ECC mechanisms such as SECDED, can detect and recover these random errors, leading to better DRAM reliability.

(A) Process Variation   (B) Architectural Variation   (C) Process+Architectural Variation
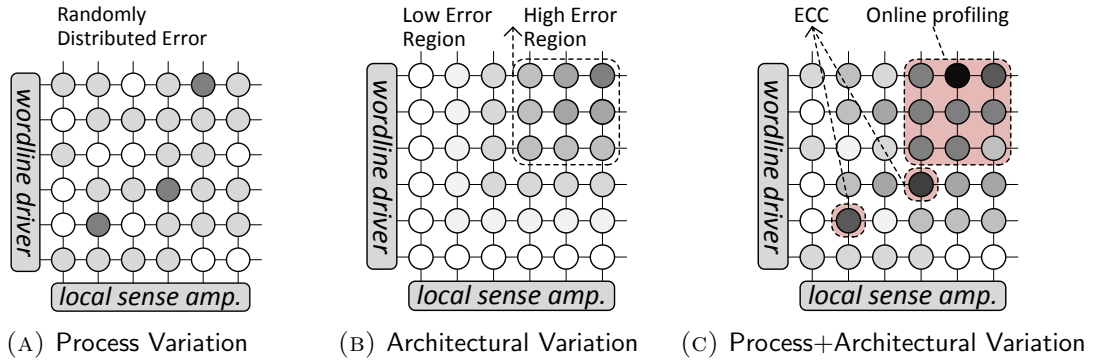
FIGURE 6.14: Latency Variation in a Mat (Darker: higher latency)

The errors caused by *architectural variation* are concentrated in specific regions. When considering only the impact of architectural variation, as shown in Figure 6.14b, cells that are farther away from the row driver and local sense amplifier have less charge and hence higher access latency. Therefore, a mat can be divided into a high error region (the rightmost and topmost region in a mat) and a low error region (the other regions besides the high error region). When timing parameters are aggressively reduced, the first set of errors tend to occur in the high error region. Furthermore, these errors tend to be multi-bit errors because high error cells are clustered in a small region of a mat. These multi-bit errors exceed the error correction capabilities of a simple ECC (e.g., SECDED) and require a complicated ECC which usually incurs high area and latency overhead. To avoid these undesirable multi-bit errors, we periodically profile the high error regions alone, which incurs much less overhead than profiling the entire DRAM, and tune timing parameters appropriately based on this profile.

AVA Profiling Mechanism. Our AVA Profiling mechanism combines ECC with online profiling in a synergistic manner, with the goal of reducing DRAM latency while maintaining high reliability. Figure 6.15 illustrates reliability reduction due to lowering DRAM timing parameters from the standard value (right most) to the lowest value (left most). Darker regions represent more failures (less reliability). Conventional ECC techniques divide the reliability spectrum into three regions: *i)* error free region (❶), *ii)* correctable error region (❷), and *iii)* uncorrectable error region (❸). ECC seeks to maintain a system in the error free or correctable error regions. However, determining whether a system is in the correctable region or not is not straightforward. We propose to achieve this by using online profiling that is aware of architectural variation.

Due to architectural variation, there is a specific region that requires the highest access latency in DRAM. We rely on DRAM manufacturers to provide information on the location of this slowest region. The AVA-profiling-based memory system, then, uses this slowest region to perform online latency profiling (we call it the *latency test region*). Note that the actual data is not stored in this region. The AVA profiler *periodically* accesses this latency test region and determines the smallest value of DRAM timing parameters required for reliable operation
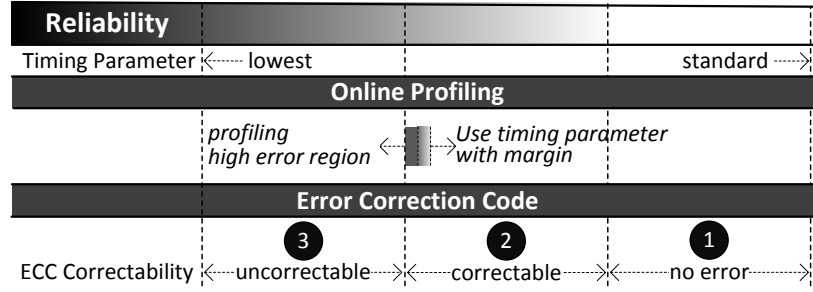
FIGURE 6.15: Architectural Variation Aware Online Profiling

in this latency test region (e.g., not causing multi-bit errors). The system then adds a small margin to the timing parameters obtained from this profiling (e.g., one clock cycle increase) to determine the timing parameters for the other regions (*data region*). Our mechanism profiles only the slowest region that is most impacted by architectural variation, thereby incurring low profiling overhead, while achieving low DRAM latency *and* high reliability.

To implement AVA Profiling, the latency test regions (e.g., a row per mat) need to be identified and avoided when repairing/remapping. Simply avoiding repair/remapping for the latency test regions (the most vulnerable regions) can avoid complexity associated with repair/remapping. Since these latency test regions are reserved only for test and not for data storage, the overhead in doing so is low. While we show the correlation between error count and row/column addresses to demonstrate the existence of architectural variation in Section 6.3, such correlation is not always necessary to implement the proposed mechanisms. Our mechanism only needs to know the addresses of the most vulnerable regions. DRAM companies are very likely to know the most vulnerable regions (based on their knowledge of the design) of their DRAM products. Such vulnerable regions can be assigned as test regions, which are reserved only for test and are not used for general data storage. During the repair process, such test regions are avoided, preventing repair/remapping issues.

In order to profile the test region, many possible mechanisms can be employed. For example, a simple and low overhead mechanism is to integrate the online test into the DRAM refresh operation. Such a mechanism can keep the implementation simple, while having minimal effect on memory system performance.

AVA Profiling with Other Latency Variations in DRAM. So far, we explained our mechanism to cover two types of latency variation in DRAM chips, process variation and architectural variation. However, there can be other latency variations in DRAM, e.g., PVT variation and VRT (variable retention time). We have designed our mechanisms with careful consideration of these variations as well by adopting two error mitigation techniques (ECC and online profiling) together.

As explained, we divide the DRAM failures into two categories: *i)* localized failures (caused by architectural variation and chip-to-chip process variation) and *ii)* random failures (caused by cell-to-cell process variation and variable retention time (VRT)), and exploit two different error mitigation techniques to tackle these two categories of failures, i.e., online profiling for localized failures and ECC for random failures.

Since the physical dimension of a mat is very small (e.g., 1415.6 um^2 for a $6F^2$ 30nm technology 512 cell $\times$ 512 cell mat), we assume that the effect of voltage and temperature variation might be similar across a mat. Chip-to-chip process variation should be the same across each mat. The cell-to-cell process variation and VRT effects can be covered by ECC.

Optimizing Performance and Reliability. Compared to a server system that uses ECC only for uncertain errors (e.g., alpha particle errors), systems that employ our mechanism for better performance might have higher failure rate. In such systems that need high reliability, AVA Profiling can be adapted to suit the system's higher reliability needs. For example, when determining timing parameters, AVA Profiling can consider both the profiled lower latency bound in test regions and the frequency of ECC correctable errors, keeping ECC correctable errors as low as the required threshold.

6.4.2   Architectural Variation Aware Shuffling

Our second approach focuses on leveraging architectural variation to mitigate uncorrectable errors in memory systems with ECC. As we observe in Section 6.3.4, when data is read out of a memory channel, data in specific locations tends to fail more frequently. This happens because data is delivered from locations which are distributed across a wordline. Due to the architectural variation in wordline and control signals, it takes longer to access cells in

114

specific locations compared to cells in other locations, leading to multi-bit errors in memory systems with ECC. Figure 6.16a shows the effect of architectural variation in systems with ECC. Data in the darker grey regions (*high-error bit*) tends to be more error-prone than data in the lighter grey regions. Unfortunately, these high-error bits are concentrated in a similar location across different chips and hence, they are part of the same data-transfer burst.

(A) Conventional Mapping   (B) Proposed Mapping
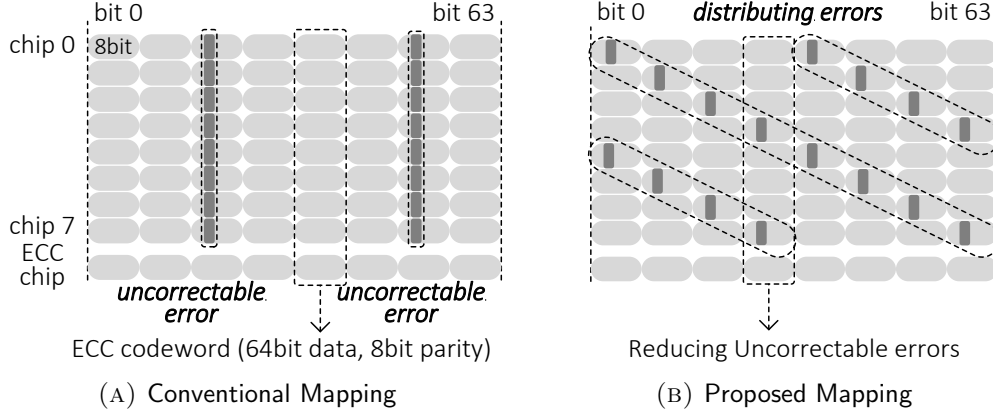
FIGURE 6.16: Architectural Variation Aware Data Shuffling

We tackle this problem and mitigate potential uncorrectable errors by leveraging awareness of architectural variation. Our key idea is *to distribute the high-error bits across different ECC code words*. We call this mechanism *architectural-variation-aware data shuffling (AVA Shuffling)*. There are three potential ways in which such a shuffling mechanism can be implemented. One way to achieve this is by using eight DRAM chips that have different data-out mappings. This would require changing the DRAM chip internally, when it is being manufactured. In this case, since the data mapping is changed internally in the DRAM chips to shuffle the high-error bits across different ECC code words, the address decoding mechanism for reads and writes can remain identical across DRAM chips. The second way is to shuffle the address mapping of DRAM chips within a DRAM module. We achieve this by connecting the address bus bits in a different order for different DRAM chips in a DRAM module, leading to different column addresses being provided by different DRAM chips. Using these two mechanisms, we can achieve data shuffling in the data output from DRAM, as shown

in Figure 6.16b. The third possible mechanism is to shuffle the data from a cache line access in the memory controller such that the high-error bits are distributed across different code words. The advantage of implementing the shuffling in the memory controller is that the shuffling mechanism can be changed. However, this would work only if the memory controller performs error correction after receiving all the data for a cache line request, as performance could be degraded from waiting for all data to reach the memory controller.

Interleaving data (or bit) over multiple ECC codeword is not new, but, the awareness of the most vulnerable regions enables more robustness over a bit interleaving scheme that is not aware of the most vulnerable regions. AVA Shuffling, for instance, interleaves data to distribute the most vulnerable regions over different ECC codewords, leveraging knowledge of architectural variation. Awareness of architectural variation can enable other efficient shuffling mechanisms too, for example, using different data (burst) shuffling orders in different chips in a DRAM module.

Figure 6.17 shows the fraction of correctable errors using SECDED with/without AVA Shuffling. The Y-axis represents the total percentage of errors with lower DRAM timing parameters, and the X-axis represents 33 (randomly selected) DRAM modules.[4] We make two observations. First, our mechanism corrects 26% of the errors which are not correctable by using *only* conventional ECC, leading to further latency reduction for 24 DRAM modules out of 96 DRAM modules. Second, both error correction mechanisms (ECC with/without AVA Shuffling) show correlation in the error correction rate. For example, when ECC without AVA Shuffling shows high correction rate, ECC with AVA Shuffling also shows high error correction rate.

6.4.3 AVA Profiling/Shuffling vs. Having Stronger ECC

The goal of our proposal is *not* replacing ECC mechanisms, but, in contrast, complementing existing ECC mechanisms to achieve better performance and reliability.

---

[4]The operating conditions where selected to make sure that there are actually errors, so that ECC is useful.
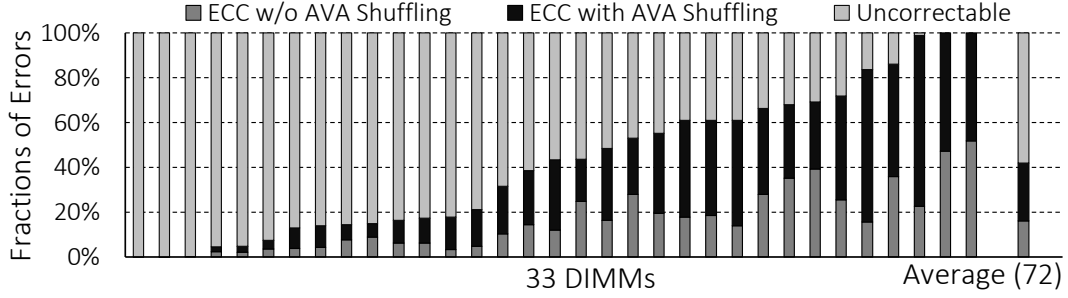
FIGURE 6.17: ECC with/without AVA Shuffling

First, having ECC alone (regardless of ECC strength) is not enough to guarantee correct operation with maximum latency reduction, since it is not possible to determine the smallest value for each timing parameter without profiling. AVA Profiling can do so, enabling maximum latency reduction while leveraging ECC support to correct random failures. Second, AVA Shuffling enables greater reliability with any ECC mechanism by distributing possible errors over different ECC codewords. Third, our work opens up new research opportunities to exploit architectural variation in combination with different ECC schemes. For example, variable-strength ECC can exploit awareness of architectural variation by adjusting ECC strength based on error probability indications/predictions from architectural variation.

6.4.4 DRAM Latency & Performance Analysis

DRAM Latency Profiling. We profile 96 DRAM modules, comprising 768 DRAM chips, for potential latency reduction. We use the same test methodology, described in Section 6.2, which is also similar to the profiling methodology of Chapter 5 and a previous work [37]. We measure the latency reduction of four timing parameters (t_{RCD}, t_{RAS}, t_{RP}, and t_{WR}).

Figure 6.18 shows the average latency reduction for DRAM read and write operations with three mechanisms — AL-DRAM (our second mechanism in Chapter 5), AVA Profiling, and AVA Profiling with Shuffling — as the sum of the corresponding timing parameters. We compare these mechanisms at two operating temperatures, 55℃ and 85℃. AL-DRAM mechanism can reduce the latency for read/write operations by 33.0% and 55.2% at 55℃, and 21.3% and 34.3% at 85℃, respectively. To integrating ECC (SECDED), our architectural

variation aware online profiling mechanism further reduces the corresponding latencies by 35.1% and 57.8% at 55℃, and 34.8% and 57.5% at 85℃, respectively. Using AVA Shuffling on top of AVA Profiling enables more latency reduction (by 1.8% on average). We conclude that our proposed mechanisms are able to achieve better latency reduction compared to AL-DRAM at different temperatures. This is mainly because ECC (and also ECC with AVA Shuffling) can correct many single-bit errors in a ECC codeword.

(A) READ ($t_{RAS} - t_{RP} - t_{RCD}$) (B) WRITE ($t_{WR} - t_{RP} - t_{RCD}$)
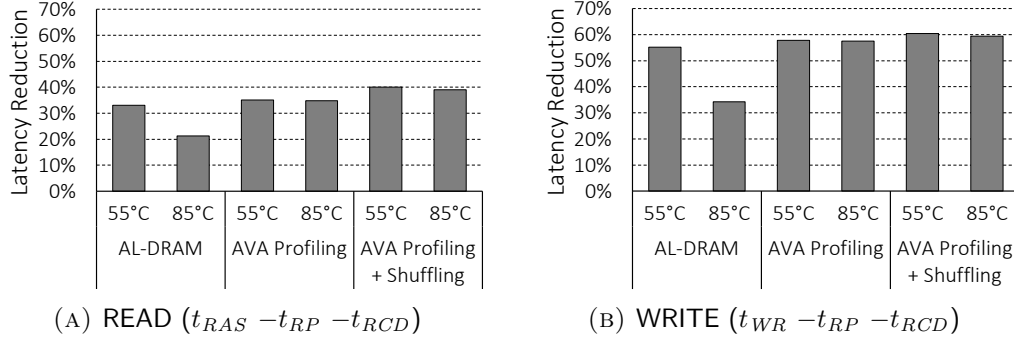
FIGURE 6.18: Read and Write Latency Reduction

Figure 6.19 shows average latency reduction in each timing parameter, t_{RCD}, t_{RAS}, t_{RP}, and t_{WR}. We compare AVA-DRAM to AL-DRAM at two operating temperatures, 55℃ and 85℃. We observe similar trends with these timing parameters as with the read/write latency results. In this dissertation, we present only the *average* potential reduction for each timing parameter. We provide detailed characterization of each DRAM module online at the SAFARI Research Group website [142].

While our sample size (96 DIMMs and 768 DRAM chips) may not be large enough to identify the exact amount of architectural variation and possible latency/error reduction across all DRAM designs, we strongly believe that our results are consistent enough to support our hypothesis of the existence of architectural variation. As shown in Section 6.3.6, 75% of modules show significant amount of architectural variation which we leverage to enable latency and error reduction.

Performance Evaluation. We evaluate the performance improvement of using our AVA Profiling mechanism. We use a modified version of Ramulator [128], a fast, cycle-accurate
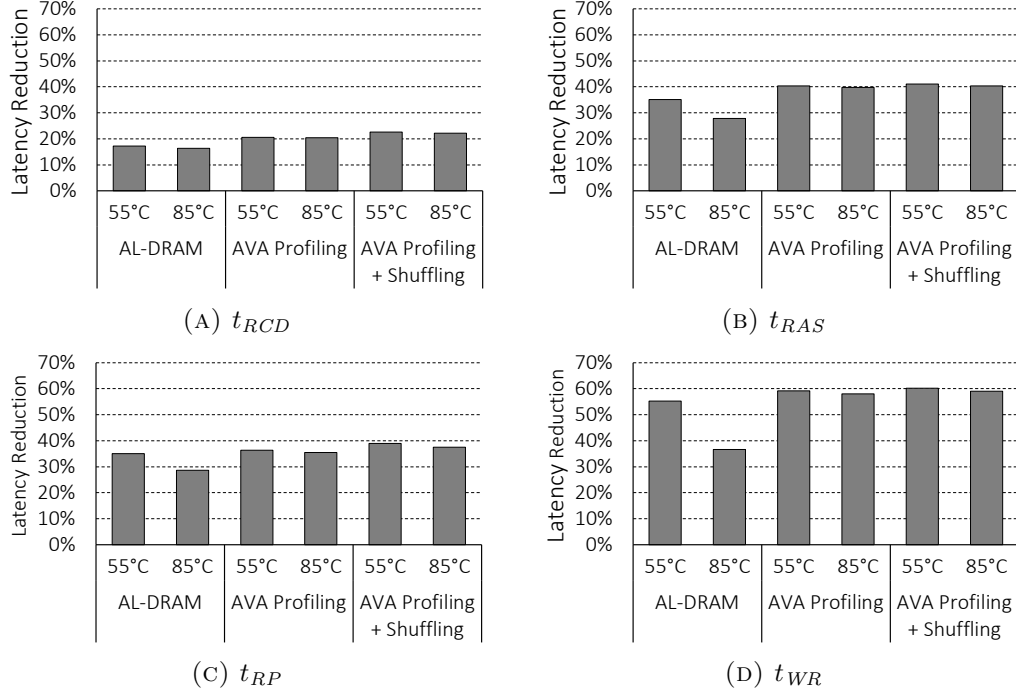
FIGURE 6.19: Latency Reduction for each Timing Parameter

DRAM simulator that is publicly available [127]. We use Ramulator combined with a cycle-level x86 multi-core simulator. Table 6.2 shows the system configuration we model. We built a cycle-accurate memory model with detailed timing parameters. We accurately modeled the latency reduction in our evaluations. Our baseline system has original standard timings (e.g., tRCD 13.75ns for DDR3-1600) and we use reduced timing parameters (e.g., tRCD 11.25ns) to evaluate our mechanisms.

For workloads, we use Pinpoints tool [162, 205] to collect traces. We use 32 benchmarks from SPEC CPU2006 [256], stream [4], TPC [271] and GUPS [88], each of which is used for a single-core workload. We construct 32 two-, four-, and eight-core workloads – a total of 96 multi-core workload (randomly selected from the 32 benchmarks). We measure single-core performance using instructions per cycle (IPC) and multi-core performance using weighted speedup [70, 250] metric. We simulate 100 million instructions after caches are warmed up.

119

Component	Parameters
Processor	8 cores, 3.2GHz, 3-wide issue, 8 MSHRs/core, 128-entry inst. window
Last-level cache	64B cache-line, 16-way associative, 512KB private cache-slice per core
Memory controller	64/64-entry read/write queues/controller, FR-FCFS scheduler
Memory system	DDR3-1600 [101], 2 channels, 2 ranks-per-channel

TABLE 6.2: Configuration of Simulated Systems

Figure 6.20 shows the performance improvement with AVA Profiling and AVA Shuffling. We draw two major conclusions. First, AVA Profiling provides significant performance improvement over the baseline DRAM (9.2%/14.7%/13.7%/13.8% performance improvement in single-/two-/four-/eight-core systems, respectively). This improvement is mainly due to the reduction in DRAM latency. Second, using AVA Profiling and AVA Shuffling together provides even better performance improvements (by 0.5% on average) due to additional latency reductions with AVA Shuffling. We achieve this performance while maintaining the DRAM reliability by dynamically monitoring and optimizing DRAM latency (AVA Profiling).
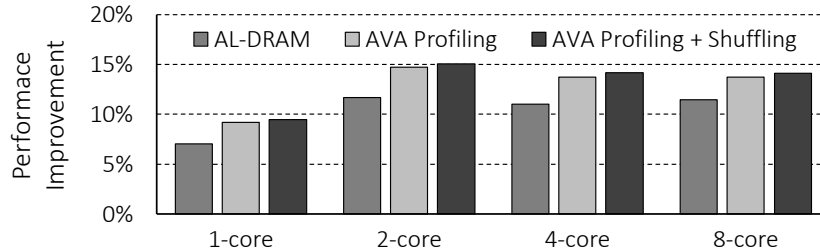
FIGURE 6.20: Performance Improvement with AVA-DRAM

We also observe that AL-DRAM provides good performance improvement (as high as 11.5% in two-core system, as also shown in Chapter 5). As we discussed in Section 6.4.1, AL-DRAM has two major shortcomings, which our mechanism overcomes by performing low cost online profiling by exploiting architectural variation. Considering that there are significant parts of DRAM suffer from aging or post-packaging failures [92, 148, 171, 237, 253, 254], which AL-DRAM cannot handle, we conclude that our mechanisms provide both better performance and reliability at low cost over AL-DRAM.

120

6.5 Summary

In this chapter, we provide the first experimental demonstration and characterization of the variation in memory access latency due the internal organization of DRAM, which we call *architectural variation*. We find that there is significant variation in the access latency of DRAM cells required for reliable operation, depending on where cells reside within the DRAM chip. We show that cell latency is especially affected by how close the cell is to the peripheral structures that are used to access the cell. We show the existence and characteristics of architectural variation by the profiling of 96 modern DRAM memory modules from three major manufacturers.

Based on the extensive understanding of architectural variation, we develop AVA-DRAM, which consists of two new methods to reduce DRAM latency reliably at low cost. AVA Profiling is the first technique that can dynamically find and use the lowest latency at which DRAM can operate reliably. AVA Shuffling further reduces latency by lowering it to a point that causes multi-bit errors, while ensuring reliable operation by shuffling data such that these errors become correctable by ECC. We demonstrate that AVA-DRAM can greatly reduce DRAM read/write latency, leading to a significant system performance improvement on a variety of workloads and system configurations.

We conclude that exploiting architectural latency variation inherent in DRAM using our new techniques provides a promising, reliable, and low-cost way of significantly reducing DRAM latency.

Chapter 7

System Design Guidelines
for Heterogeneous Memory Systems

To integrate low-latency DRAM architectures proposed in this dissertation in future systems, one required modification is to expose DRAM latency configurations and operating conditions (e.g., DRAM operating temperature, and latency variation in different DRAM chips and different areas in a DRAM chip) to the system (e.g., processor). We call this information *configuration information.* To this end, providing an efficient interface for transferring such configuration information from DRAM to the system is one of the key features that is required. In this chapter, we provide guidelines to seamlessly integrate such an interface within future systems. In the following sections, we first explain the required information for enabling various heterogeneous-latency DRAMs proposed in this dissertation, and then describe various options to construct an efficient interface between the DRAM and the system. Note that other works also pointed at the necessity of constructing such interfaces [187].

7.1 Required Information for Enabling Heterogeneous-Latency DRAM

In Chapters 4, 5, and 6, we proposed three memory systems that enable heterogeneous latency in DRAM. Each of these proposals has different configuration information required

for enabling low-latency DRAM architectures. In this Section, we describe examples for the configuration information.

7.1.1   Information for Tiered-Latency DRAM

Tiered-Latency DRAM divides a subarray into two different latency segments, the near segment (fast segment) and the far segment (slow segment). To exploit the Tiered-Latency DRAM based memory system, the processor needs to know following configuration information.

- **Subarray Organization.** As we explained in Section 2.1, a DRAM bank is divided into multiple subarrays. The processor should be made aware of the subarrays. For example, 4GByte DDR3 DRAM [172] consists of 8 banks, each of which has 64K rows (each of which is mapped to a value of 16-bit row addresses). Assuming that each subarray consists of 512 rows, each bank has 128 subarrays. Therefore, the 16-bit row address can be divided into two portions, *i)* address bits for selecting subarrays (7 bits *subarray address*) and *ii)* address bits for selecting rows in a subarray (9 bits *row address in a subarray*). This information should be exposed to the processor.

The easiest way is to integrate the subarray address and the row address (in a subarray) separately into the DRAM specification. For example, similar to the conventional DRAM specifications that have bank addresses and row addresses separately, the Tiered-Latency DRAM specification might have bank addresses, *subarray* address, and row address. Alternatively, Tiered-Latency DRAM might provide the subarray configuration directly to the processor. A simple example data format that contains the subarray configuration is a bit vector, each bit of which represents whether the corresponding row address bit is *i)* subarray address or *ii)* row address in a subarray. For example, a bank consists of 32 rows in total (16 bits) and is subdivided into 64 subarrays (7 bits), each of which consists of 512 rows (9 bits). Then, the 16-bit row address can be divided into 7-bit subarray address and 9-bit row address in each subarray. TL-DRAM can provide this information to the processor as a 16-bit vector of 1111111000000000b where "1" represents the corresponding row

address bit is a part of *subarray address* and "0" represents the corresponding row address bit is a part of *row address in a subarray.*

- **Near Segment and Far Segment Organization.** Tiered-Latency DRAM then subdivides a subarray into the near segment and the far segment. For example, 512 rows in a subarray can be divided into 32 rows in the near segment and 480 rows in the far segment. In this organization, 512 rows in a subarray consist of 32 32-row groups, and one of the 32-row groups can be the near segment.

  There can be many data formats to transfer this configuration information from TL-DRAM to the processor (specifically, the memory controller in the processor). One simple example is to provide the address range of the near segment in a 512-row subarray from TL-DRAM to the memory controller. For the case of 32-row near segment out of 512-row subarray (total 9-bit row address), the near segment can be assigned the range of row addresses from `000000000b` to `000001111b` (initial 32 rows). The amount of data to be transferred to the memory controller in this case is only 18 bits, which can be transferred over a specialized interface.

- **Operating Mode of Tiered-Latency DRAM.** Tiered-Latency DRAM is a substrate that can be leveraged in many ways. Since Tiered-Latency DRAM can support multiple modes (e.g., using the near segment as hardware-managed exclusive cache to the far segment, using the near segment as hardware-managed inclusive cache to the far segment, profile-based page mapping to the near segment, and so on), the system is required to select a current operation mode. This can be integrated to *Mode Register Set*, which already exists in the conventional DRAM [172].

7.1.2   Information for Adaptive-Latency DRAM

As we explained in Chapter 5, Adaptive-Latency DRAM leverages both DRAM operating conditions (e.g., DRAM operating temperature) and process variation in DRAM to optimize DRAM timing parameters. To this end, the following information needs to be exposed to the processor.

- **DRAM Operating Conditions.** We show that DRAM can be accessed with different timing parameters in different operating conditions. There can be many operating conditions, for example, operating temperature (which is leveraged by AL-DRAM), power consumption, voltage level, and so on. These are pieces of information that are different in different DRAM chips. To leverage the information for better performance, the information needs to be exposed to the processor. For example, in an AL-DRAM-based heterogeneous memory system, the processor needs to know the *operating temperature* of each DRAM module. The amount of information to be transferred depends on the resolution of temperature and frequency of transfer. For example, 10-bit temperature information can provide 0.1 degree resolution in the range of temperature, $0 - 100$℃.

- **DRAM Timing Parameters for each Operating Condition.** To leverage process variation, each DRAM module has a set of timing parameters for each operating condition (e.g., operating temperature) and requires transfer of the timing parameters to the processor. Each timing parameter can be represented using several bits (usually fewer than 10 bits) since most timing parameter values are less than 1024. A request of 64 bytes can transfer 512 bits in total, which are enough to transfer 51 timing parameters per request.

7.1.3 Information for AVA-DRAM

As we explained in Chapter 6, AVA-DRAM leverages the subarray organization to reduce DRAM latency while maintaining DRAM reliability. The key observation in AVA-DRAM is that the distance between cells and peripheral logic leads to variation in latency. The required information to leverage AVA-DRAM is as follows.

- **Inherently Slower Cells in a DRAM Mat.** AVA-DRAM leverages the existence of the inherently slower cells in a DRAM mat. A naive way to expose this information is by making the order of addresses in line with the distance from peripheral logic. This works for row addresses, each of which maps to a row in a bank. However, the hierarchical organization of the column access path might not be easy to represent the full organization with simple information. Instead, we propose that DRAM provides to the processor the

exact address range of the inherently slower cell region. In this approach, DRAM provides multiple sets of row and column addresses which show the worst latency in a subarray. For this, the required amount of data for a set is a 9-bit row address and a 10-bit column address.

7.2 Interface to Heterogeneous-Latency DRAM

To transfer the required *configuration information* from DRAM to the memory controller, we consider multiple constraints: *i)* additional overhead for integrating this information into the interface, *ii)* performance impact for transferring data, and *iii)* security issues related to exposing internal DRAM organization to the processor. We briefly describe each as follows.

Low Area Overhead. A naive way to transfer the data is to integrate a new set of I/O interfaces (e.g., specific wire connections and corresponding peripheral logic) for the configuration information. However, this approach might significantly increase the DRAM chip area and energy consumption by increasing the number of wire connections between DRAM modules and the system. An alternative way is to use the existing data bus in the memory channel. To this end, we might need to add *i)* specialized commands for accessing the configuration information and *ii)* peripheral logic to support the new commands. This approach does not require any additional physical wires in the memory channel. However, since it is not possible to access data for executing instructions during the transfer of the configuration information of DRAM operating conditions and organization, this approach might lead to performance degradation. Next, we discuss the performance impact of transferring the configuration information from the DRAM to the system using the existing memory channel.

Low Performance Impact. As we described, using the existing data bus for transferring the configuration information consumes memory channel bandwidth periodically, leading to performance degradation. There are two factors to determine how much the performance impact is. The first factor is how much data is needed to be transferred. Considering that the amount of data for configuration is very modest, we expect that the performance degradation might not be significant.

The second factor is how frequently the configuration information needs to be transferred. There are two categories of the configuration information based on the frequency of transferring data, *i)* static configuration information and *ii)* dynamic operating conditions. The organization of DRAM (e.g., subarray organization, the near and far segment organization, and so forth) does not change over time. Therefore, the static configuration information needs to be transferred *only once* at boot time. However, since the dynamic operating conditions change over time, DRAM needs to update the processor with its new operating conditions. The frequency of the updates depends on how frequently the operating conditions change. As we described in Section 5.2.2, since the operating temperatures in real systems do *not* change drastically (e.g., less than 0.1℃ per second even in the worst case), we expect that the performance degradation is even less for this type of information transfer. For AVA-DRAM, we have not answered the question of how frequently the latency parameters should be updated to avoid potential errors. However, it might be in the order of few hours, few days and few months, which leads to small performance impact. We leave this question of "how frequently should the latency parameters be updated" to future work.

Security Issues. While exposing the internal organization of DRAM is the easiest way for leveraging heterogeneous-latency DRAM architectures, it can potentially lead to security issues. Kim et al. [122] showed that accessing a row frequently enough before refresh can lead to errors in its adjacent rows. Exposing the internal DRAM organization and address mapping might lead to even more security holes. To address this issue, a possible alternative design is that *i)* the processor accesses DRAM with a standard and simplified address interface, and *ii)* DRAM maps the issued addresses to the physical rows and columns with address *scrambling*. In this design, the processor can access different latency regions without exposing DRAM organization to outside of DRAM, maintaining the existing security of the memory system.

7.3   Summary

In this chapter, we describe the required configuration information that needs to be communicated between the DRAM and the system to leverage heterogeneous-latency DRAM

architectures proposed in this dissertation. We provide the various considerations that need to be taken into account to implement the interfaces to communicate the configuration information. We hope that these discussions help in the design of the system-DRAM interface to take advantage of heterogeneous-latency DRAM.

Chapter 8

Conclusions and Future Research Directions

In this dissertation, we present three techniques to lower DRAM latency at low cost. These techniques enable or exploit heterogeneity in DRAM. They are *i)* Tiered-Latency DRAM, which enables heterogeneous bitlines at low cost by dividing the long bitline into two different latency segments, *ii)* Adaptive-Latency DRAM, which optimizes DRAM latency for the common operating conditions, and *iii)* AVA-DRAM, which lowers DRAM latency by exploiting architectural variation in the internals of DRAM organization.

We first present a new DRAM architecture, Tiered-Latency DRAM (TL-DRAM), that provides both low latency and low cost-per-bit in Chapter 4. Our key observation is that existing DRAM architectures present a trade-off between cost-per-bit and access latency. One can either achieve low cost-per-bit using long bitlines or low access latency using short bitlines, but not both. Our key idea to leverage this trade-off for lowering DRAM latency is to segment a long bitline using an isolation transistor, creating a segment of rows with low access latency while keeping cost-per-bit on par with commodity DRAM. We present mechanisms that take advantage of our TL-DRAM substrate by using its low-latency segment as a hardware-managed cache. Our most sophisticated cache management algorithm, Benefit-Based Caching (BBC), selects rows to cache that maximize access latency savings. We show that our proposed techniques significantly improve system performance by 12.8% and reduce energy consumption by 23.6% across a variety of systems and workloads.

While TL-DRAM reduces DRAM latency significantly, it requires changing existing DRAM architecture, which might limit the applicability of the proposed techniques (even though the changes are low cost, i.e, only 3% additional DRAM area).

Towards achieving the goal of lowering DRAM latency without changing the DRAM architecture, we approach to leverage *latency slack* that exists in modern DRAM. To this end, we build an FPGA-based DRAM test infrastructure that can profile DRAM characteristics, which *i)* can flexibly change DRAM timing parameters, *ii)* apply specific data and access patterns, and *iii)* maintain the operating conditions (e.g., ambient temperature). Based on characterizations of 115 DRAM modules, we introduce two new mechanisms to reduce DRAM latency without changing DRAM architecture.

We first propose Adaptive-Latency DRAM (AL-DRAM), a simple and effective mechanism for dynamically tailoring the DRAM timing parameters for the current operating condition without introducing any errors in Chapter 5. The standard DRAM timing constraints are grossly overprovisioned to ensure correct operation for the cell with the lowest retention time at the highest acceptable operating temperature. We make the observation that a significant majority of DRAM modules do *not* exhibit the worst case behavior and that most systems operate at a temperature much lower than the highest acceptable operating temperature, enabling the opportunity to significantly reduce the timing constraints. Based on these observations, AL-DRAM dynamically measures the operating temperature of each DRAM module and employs timing constraints optimized for *that DRAM module at that temperature*. Results of our latency profiling experiments on 115 modern DRAM modules show that our approach can significantly reduce four major DRAM timing constraints by 17.3%/37.7%/54.8%/35.2 averaged across all 115 DRAM modules tested. This reduction in latency translates to an average 14% improvement in overall system performance across a wide variety of memory-intensive applications run on a real multi-core system.

While AL-DRAM is a simple and effective mechanism to reduce DRAM latency, it has two overheads that may limit its use. First, it requires DRAM manufacturers would determine reliable timing parameters for multiple operating conditions, increasing test cost during the

manufacturing time. Second, it do not take into account DRAM latency changes over time due to aging and wear out.

To reduce DRAM latency on a more dynamic and achievable manner, we leverage our experimental demonstration and characterization of the variation in memory access latency due the internal organization of DRAM, which we call *architectural variation*, in Chapter 6. We find that there is significant variation in the access latency of DRAM cells required for reliable operation, depending on *where* cells reside within a DRAM chip. We show that cell latency is especially affected by how close the cell is to the peripheral structures that are used to access the cell.

Building upon an extensive understanding of architectural variation developed by the characterization of 96 modern DRAM modules from three major manufacturers, we develop AVA-DRAM, which leverages architectural variation, which consists of two new mechanisms to reliably reduce DRAM latency at low cost. AVA Profiling is the first technique that can dynamically find and use the lowest latency at which DRAM can operate reliably. AVA Shuffling further reduces latency by lowering it to a point that causes multi-bit errors, while ensuring reliable operation by shuffling data such that these errors become correctable by ECC. We demonstrate that AVA-DRAM can greatly reduce DRAM read/write latency (by 40.0%/60.5%, respectively), leading to a significant system performance improvement (by 14.7%/13.7%/13.8% on 2-/4-/8-core system, respectively) on a variety of workloads.

We conclude that our three mechanisms provide promising low-latency low-cost DRAM designs. We have shown that each of them significantly improves overall system performance. We believe that both our new characterization of DRAM latency (across a large number of real DRAM chips) and the mechanisms we have developed on the basis of our analysis of DRAM architecture and our experimental DRAM characterization enable the development of other mechanisms for improving DRAM latency and perhaps reliability.

8.1 Future Research Directions

We describe potential research directions to further reduce DRAM latency and mitigate the negative impact of high DRAM latency in the next sections.

8.1.1 Optimizing Timing Parameters in 3D-Stacked DRAM

3D-stacked DRAMs [93, 94, 95, 106, 140, 158] are likely to provide high-bandwidth and energy-efficient memory systems in the future. Similar to commodity DRAM, it is possible to optimize latency in 3D-stacked DRAM for the common case. To this end, the first step is investigating the operating conditions to determine the worst-case and common-case conditions in 3D-stacked DRAM. Then, the next step is developing mechanisms to reduce access latency for the common-case.

In addition to the possible latency reduction for the common-case, we expect that 3D-stacked DRAM might exhibit much higher reduction across the stack due to two major reasons. First, integrating DRAM chips on top of each other dissipates heat to the adjacent layer, which creates a temperature gradient across the layers. As a result, 3D-stacked DRAMs might exhibit much higher variations in operating temperature within DRAM. Second, the power delivery network of 3D-stacked DRAM drives power from bottom to top, creating a heterogeneous voltage supply across the layers. As variation in voltage and temperature directly impact the latency of DRAM access, these two variations might enable more opportunities to reduce access latency in 3D-stacked DRAM.

Considering the variations present across different layers of 3D-stacked DRAM, we believe that investigating these variations at different operating conditions leads to devising new 3D-stacked DRAM architectures that enable higher latency reduction, and developing mechanisms that adaptively optimize access latency for each layer.

8.1.2 Optimizing Refresh Operations for the Common-Case

Retention time of a DRAM cell depends on the data stored in the neighboring cells, referred to as a phenomenon called *data pattern dependence*. Depending on the data content stored in

DRAM, some rows can exhibit much higher retention time than others [119, 155]. In today's systems, this heterogeneity in retention time is not exploited to reduce refresh operations. All DRAM rows are refreshed at the same rate determined by the worst-case data patterns tested during the manufacturing time. Typical program contents from different applications may not exhibit these worst-case patterns frequently and may not require to be refreshed at the nominal rate. We can leverage this longer retention time of the common-case data patterns in programs to reduce the refresh overhead.

We believe that investigating the retention time in real DRAM chips with different program content using our FPGA-based testing infrastructure and analyzing the possible reduction in refresh rate for the common case would enable us to develop dynamic refresh mechanisms that minimize refresh operations in different regions of memory depending on the current program content.

8.1.3   System Design for Heterogeneous-Latency DRAM

We have shown in this dissertation that future low-latency DRAM architectures enable faster access by either *i)* creating a smaller faster region within DRAM or by *ii)* exploiting the variations present in different regions of memory at different operating conditions. Both of these approaches enable a heterogeneous-latency DRAM design, where some regions provide faster access to data. In order to maximize the benefits of such a heterogeneous latency DRAM, it is necessary to design appropriate interfaces and software-hardware collaborative mechanisms across the system stack.

We believe that investigating the following two approaches leads to designing an end-to-end system leveraging these heterogeneous DRAMs. First approach is designing a system that takes advantage of the faster segments to maximize the latency benefits. We can maximize the use of faster segment by allocating frequently used or more critical pages to this region. Exposing the latency variations to the memory controller and system software and allocating critical pages appropriately can provide better system performance. These mechanisms introduce heterogeneous-latency aware mapping and partitioning mechanisms that effectively leverage the additional memory latency tiers.

Second approach is investigating system-level mechanisms to leverage the operating conditions that result in faster DRAM access. For example, hot spots in DRAM can increase the access time of cells belonging to that region. A hardware-software collaborative technique can map pages in a way that accesses are spread out to different regions. This mechanism can ensure that DRAM can be operated at the lowest possible latency by avoiding hot spots. Similarly, a system can be designed to avoid the worst-case pattern by using intelligent coding to map that data to the common-case pattern, thereby avoiding the worst-case refresh rates.

We conclude and hope that this dissertation, with the analysis & characterization of many DRAM chips it provides and the new low-latency low-cost DRAM architectures it introduces, enables many new ideas in DRAM design for low latency and high reliability.

8.1.4 New Interfaces for Heterogeneous Main Memory

As we discussed in Chapter 7, heterogeneous main memory should have a specialized interface to expose its organization (e.g., subarray organization, fast/slow regions, and so forth) and operating conditions (e.g., temperature) to the processor. Fundamentally, the heterogeneous main memory system has four pieces of information that needs to be transferred through memory channel: *i)* command and address, *ii)* data for write, *iii)* data for read, and *iv)* the configuration information. We believe that this dissertation opens up a new research direction to investigate the new DRAM interfaces for future heterogeneous main memory systems.

The key question is how the wire connections in the memory channel should be organized. There may be three major approaches at a high level, One extreme approach is providing a separate and dedicated wire connection for each of the four information. The other extreme approach is transferring all information through one unified wire connection. For example, the conventional DDR interface transfers data for read and write through the same wire connection. Alternatively, the two extreme approaches can be integrated into a flexible memory channel, which can change the role of wire connections based on the operating conditions (e.g., operating temperature, strength of voltage supply, and so on) and the memory characteristics of the current workloads. We leave this research question, what is the most efficient interface for future heterogeneous main memory, to future work.

8.1.5 Reducing Latency of Emerging Memory Technologies

Due to difficulties in DRAM scaling, several new technologies are being heavily investigated as potential alternatives to DRAM that can replace or augment DRAM as main memory [112, 136, 161, 169, 196, 213, 227]. These technologies include Phase Change Memory (PCM) [57, 134, 135, 136, 168, 216, 221, 225, 282, 292], Spin-Transfer Torque Magnetic Memory (STT-MRAM) [131, 149, 168], Resistive RAM [281] or memristors [44, 202], and Conductive Bridging Memory (CB-RAM) [133]. Most of these technologies are at least as slow as DRAM, and in many cases slower, and therefore they likely exacerbate the main memory latency bottleneck tackled in this dissertation. We believe the techniques developed in this dissertation can inspire similar or related approaches to reduce latency in such emerging main memory technologies. We also believe the techniques developed in this dissertation can be adapted to NAND Flash memory technology [28, 29, 30, 31, 32, 33, 34, 160, 164, 170], to reduce the latency of flash memory chips. In particular, exploiting heterogeneity in both emerging and existing technologies seems promising beyond DRAM. We leave the exploration of this exciting and promising direction to future works and dissertations.

8.2 Final Summary

We conclude and hope that this dissertation, with the analysis & characterization of many DRAM chips it provides and the new low-latency low-cost DRAM architectures it introduces, enables many new ideas in DRAM design for low latency and high reliability.

Bibliography

[1] Memcached. http://memcached.org/.

[2] Memtest86+ v4.20. http://www.memtest.org.

[3] PARSEC Benchmarks. http://parsec.cs.princeton.edu/.

[4] STREAM Benchmark. http://www.streambench.org/.

[5] The Apache HTTP Server Project. http://httpd.apache.org/.

[6] B. Abali, H. Franke, D. Poff, R. Saccone, C. Schulz, L. Herger, and T. Smith. Memory Expansion Technology (MXT): Software support and performance. In *IBM Journal of Research and Development*, 2001.

[7] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*, 2015.

[8] J. Ahn, S. Yoo, O. Mutlu, and K. Choi. PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. In *ISCA*, 2015.

[9] J.-H. Ahn, B.-H. Jeong, S.-H. Kim, S.-H. Chu, S.-K. Cho, H.-J. Lee, M.-H. Kim, S.-I. Park, S.-W. Shin, J.-H. Lee, B.-S. Han, J.-K. Hong, P. Moran, and Y.-T. Kim. Adaptive Self Refresh Scheme for Battery Operated High-Density Mobile DRAM Applications. In *ASSCC*, 2006.

[10] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Improving System Energy Efficiency with Memory Rank Subsetting. In *ACM TACO*, 2012.

[11] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs. In *IEEE CAL*, 2009.

[12] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go? In *VLDB*, 1999.

[13] A. Alameldeen and D. Wood. Interactions Between Compression and Prefetching in Chip Multiprocessors. In *HPCA*, 2007.

[14] A. R. Alameldeen and D. A. Wood. Adaptive Cache Compression for High-Performance Processors. In *ISCA*, 2004.

[15] A. R. Alameldeen and D. A. Wood. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. In *Technical Report, UM-Madison 1599, University of Wisconsin-Madison*, 2004.

[16] C. Alvarez, J. Corbal, E. Salamí, and M. Valero. On the Potential of Tolerant Region Reuse for Multimedia Applications. In *ICS*, 2001.

[17] C. Alvarez, J. Corbal San Adrian, and M. Cortes. Dynamic Tolerance Region Computing for Multimedia. In *IEEE Trans. Comput.*, 2012.

[18] AMD. *AMD Opteron 4300 Series processors.* http://www.amd.com/en-us/products/server/4000/4300.

[19] AMD. BKDG for AMD Family 16h Models 00h-0Fh Processors, 2013.

[20] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's law through EPI throttling. In *ISCA*, 2005.

[21] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis. Eliminating Redundant Fragment Shader Executions on a Mobile GPU via Hardware Memoization. In *ISCA*, 2014.

[22] R. Ausavarungnirun, K. Chang, L. Subramanian, G. H. Loh, and O. Mutlu. Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. In *ISCA*, 2012.

[23] R. Ausavarungnirun, C. Fallin, X. Yu, K. Chang, G. Nazario, R. Das, G. Loh, and O. Mutlu. Design and Evaluation of Hierarchical Rings with Deflection Routing. In *SBAC-PAD*, 2014.

[24] R. Ausavarungnirun, C. Fallin, X. Yu, K. Chang, G. Nazario, R. Das, G. Loh, and O. Mutlu. A Case for Hierarchical Rings with Deflection Routing: An Energy-Efficient On-Chip Communication Substrate. In *Parallel Computing*, 2016.

[25] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu. Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance. In *PACT*, 2015.

[26] R. Balasubramonian, N. Muralimanohar, K. Ramani, and V. Venkatachalapathy. Microarchitectural Wire Management for Performance and Power in Partitioned Architectures. In *HPCA*, 2005.

[27] S. Borkar and A. A. Chien. The future of microprocessors. In *CACM*, 2011.

[28] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *DATE*, 2013.

[29] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN*, 2015.

[30] Y. Cai, Y. Luo, E. Haratsch, K. Mai, and O. Mutlu. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *HPCA*, 2015.

[31] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *ICCD*, 2013.

[32] Y. Cai, G. Yalcin, O. Mutlu, E. Haratsch, A. Cristal, O. Unsal, and K. Mai. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *ICCD*, 2012.

[33] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. Unsal, and K. Mai. Error Analysis and Retention-Aware Error Management for NAND Flash Memory. In *ITJ*, 2013.

[34] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai. Neighbor-cell Assisted Error Correction for MLC NAND Flash Memories. In *SIGMETRICS*, 2014.

[35] P. Cao, E. W. Felten, A. R. Karlin, and K. Li. A Study of Integrated Prefetching and Caching Strategies. In *SIGMETRICS*, 1995.

[36] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum. Scheduling and page migration for multiprocessor compute servers. In *ASPLOS*, 1994.

[37] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens. Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization. In *DATE*, 2014.

[38] K. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu. HAT: Heterogeneous Adaptive Throttling for On-Chip Networks. In *SBAC-PAD*, 2012.

[39] K. Chang, A. Kashyap, H. Hassan, S. Khan, K. Hsieh, D. Lee, S. Ghose, G. Pekhimenko, T. Li, and O. Mutlu. Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. In *SIGMETRICS*, 2016.

[40] K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. Improving DRAM performance by parallelizing refreshes with accesses. In *HPCA*, 2014.

[41] K. Chang, P. J. Nair, S. Ghose, D. Lee, M. K. Qureshi, and O. Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.

[42] M.-T. Chao, H.-Y. Yang, R.-F. Huang, S.-C. Lin, and C.-Y. Chin. Fault Models for Embedded-DRAM Macros. In *DAC*, 2009.

[43] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer. Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access. In *MICRO*, 2012.

[44] P.-F. Chiu, M.-F. Chang, C.-W. Wu, C.-H. Chuang, S.-S. Sheu, Y.-S. Chen, and M.-J. Tsai. Low Store Energy, Low VDDmin, 8T2R Nonvolatile Latch and SRAM With Vertical-Stacked Resistive Memory (Memristor) Devices for Low Power Mobile Applications. In *JSSC*, 2012.

[45] J. Choi, W. Shin, J. Jang, J. Suh, Y. Kwon, Y. Moon, and L.-S. Kim. Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM. In *ISCA*, 2015.

[46] D. Citron, D. Feitelson, and L. Rudolph. Accelerating Multi-media Processing by Implementing Memoing in Multiplication and Division Units. In *ASPLOS*, 1998.

[47] J. Collins and D. M. Tullsen. Hardware Identification of Cache Conflict Misses. In *MICRO*, 1999.

[48] D. A. Connors and W.-m. W. Hwu. Compiler-directed Dynamic Computation Reuse: Rationale and Initial Results. In *MICRO*, 1999.

[49] R. Cooksey, S. Jourdan, and D. Grunwald. A Stateless, Content-directed Data Prefetching Mechanism. In *ASPLOS*, 2002.

[50] F. Dahlgren, M. Dubois, and P. Stenström. Sequential Hardware Prefetching in Shared-Memory Multiprocessors. In *IEEE TPDS*, 1995.

[51] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In *HPCA*, 2013.

[52] R. Das, O. Mutlu, T. Moscibroda, and C. Das. Application-aware prioritization mechanisms for on-chip networks. In *MICRO*, 2009.

[53] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Aergia: Exploiting Packet Latency Slack in On-Chip Networks. In *ISCA*, 2010.

[54] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *ICAC*, 2011.

[55] R. de Castro, A. Lago, and M. Silva. Adaptive compressed caching: design and implementation. In *SBAC-PAD*, 2003.

[56] Dell. Dell PowerEdge R415 Spec Sheet. http://www.dell.com/learn/us/en/04/shared-content~data-sheets~en/documents~r415-specsheet.pdf, 2009.

[57] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM: A hybrid PRAM and DRAM main memory system. In *DAC*, 2009.

[58] F. Douglis. The Compression Cache: Using On-line Compression to Extend Physical Memory. In *Winter USENIX Conference*, 1993.

[59] J. Dundas and T. Mudge. Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss. In *ICS*, 1997.

[60] J. Dusser, T. Piquet, and A. Seznec. Zero-content Augmented Caches. In *ICS*, 2009.

[61] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Fairness via Source Throttling: A Configurable and High-performance Fairness Substrate for Multi-core Memory Systems. In *ASPLOS*, 2010.

[62] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Prefetch-aware Shared Resource Management for Multi-core Systems. In *ISCA*, 2011.

[63] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu, and Y. N. Patt. Parallel application memory scheduling. In *MICRO*, 2011.

[64] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt. Coordinated Control of Multiple Prefetchers in Multi-core Systems. In *MICRO*, 2009.

[65] E. Ebrahimi, O. Mutlu, and Y. Patt. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems. In *HPCA*, 2009.

[66] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature Management in Data Centers: Why Some (Might) Like It Hot. In *SIGMETRICS*, 2012.

[67] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature Management in Data Centers: Why Some (Might) Like It Hot. In *Technical Report, CSRG-615, University of Toronto*, 2012.

[68] C. Elm, M. Klein, and D. Tavangarian. Automatic on-line memory tests in workstations. In *MTDT*, 1994.

[69] Enhanced Memory Systems. Enhanced SDRAM SM2604, 2002.

[70] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. In *IEEE Micro*, 2008.

[71] C. Fallin, C. Wilkerson, and O. Mutlu. The heterogeneous block architecture. In *ICCD*, 2014.

[72] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *HPCA*, 2015.

[73] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. Modeling within-die spatial correlation effects for process-design co-optimization. In *ISQED*, 2005.

[74] M. Gao, G. Ayers, and C. Kozyrakis. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *PACT*, 2015.

[75] P. Gillingham, R. C. Foss, V. Lines, G. Shimokura, and T. Wojcicki. High-speed, high-reliability circuit design for megabit DRAM. In *JSSC*, 1991.

[76] A. Grew. MLP yes! ILP no! In *WACI, ASPLOS*, 1998.

[77] B. Grot, J. Hestness, S. Keckler, and O. Mutlu. Express Cube Topologies for on-Chip Interconnects. In *HPCA*, 2009.

[78] B. Grot, J. Hestness, S. Keckler, and O. Mutlu. A QoS-Enabled On-Die Interconnect Fabric for Kilo-Node Chips. In *IEEE Micro*, 2012.

[79] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu. Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees. In *ISCA*, 2011.

[80] B. Grot, S. W. Keckler, and O. Mutlu. Preemptive Virtual Clock: A flexible, efficient, and cost-effective QOS scheme for networks-on-chip. In *MICRO*, 2009.

[81] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti. 3D-Stacked Memory-Side Acceleration: Accelerator and System Design. In *WoNDP*, 2014.

[82] T. Hamamoto, S. Sugiura, and S. Sawada. On the Retention Time Distribution of Dynamic Random Access Memory (DRAM). In *IEEE TED*, 1998.

[83] C. A. Hart. CDRAM in a Unified Memory Architecture. In *COMPCON*, 1994.

[84] M. Hashemi, Khubaib, E. Ebrahimi, O. Mutlu, and Y. N. Patt. Accelerating Dependent Cache Misses with an Enhanced Memory Controller. In *ISCA*, 2016.

[85] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu. ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality. In *HPCA*, 2016.

[86] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima. The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory. In *IEEE Micro*, 1990.

[87] HP. Configuring and using DDR3 memory with HP ProLiant Gen8 Servers, 2012. http://h20000.www2.hp.com/bc/docs/support/ SupportManual/c03293145/c03293145.pdf.

[88] HPC Challenge. GUPS. http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/.

[89] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Conner, N. Vijaykumar, O. Mutlu, and S. W. Keckler. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *ISCA*, 2016.

[90] W.-C. Hsu and J. E. Smith. Performance of Cached DRAM Organizations in Vector Supercomputers. In *ISCA*, 1993.

[91] J. Huang and D. Lilja. Exploiting Basic Block Value Locality with Block Reuse. In *HPCA*, 1999.

[92] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic Rays Don'T Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *ASPLOS*, 2012.

[93] Hybrid Memory Cube Consortium. HMC Specification 1.1, 2013.

[94] Hybrid Memory Cube Consortium. Hybrid Memory Cube, 2013.

[95] Hybrid Memory Cube Consortium. HMC Specification 2.0, 2014.

[96] M. Inoue, T. Yamada, H. Kotani, H. Yamauchi, A. Fujiwara, J. Matsushima, H. Akamatsu, M. Fukumoto, M. Kubota, I. Nakao, N. Aoi, G. Fuse, S. Ogawa, S. Odanaka, A. Ueno, and H. Yamamoto. A 16-Mbit DRAM with a relaxed sense-amplifier-pitch open-bit-line architecture. In *JSSC*, 1988.

[97] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA*, 2008.

[98] ITRS. International Technology Roadmap for Semiconductors: Process Integration, Devices, and Structures. http://www.itrs.net/Links/2007ITRS/Home2007.htm, 2007.

[99] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer. High Performance Cache Replacement using Re-Reference Interval Prediction. In *ISCA*, 2010.

[100] JEDEC. Application Specific Memory for a Server Centric World. http://www.jedec.org/sites/default/files/EugeneKim_dcrp.pdf, 2012.

[101] JEDEC. DDR3 SDRAM, JESD79-3F, 2012.

[102] JEDEC. DDR4 SDRAM Standard, 2012.

[103] JEDEC. Low Power Double Data Rate 3 (LPDDR3), 2012.

[104] JEDEC. *Standard No. 21C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules*, Aug. 2012.

[105] JEDEC. *Standard No. 79-3F. DDR3 SDRAM Specification*, July 2012.

[106] JEDEC. High Bandwidth Memory (HBM) DRAM. Standard No. JESD235, 2013.

[107] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Bottleneck identification and scheduling in multithreaded applications. In *ASPLOS*, 2012.

[108] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Utility-based Acceleration of Multi-threaded Applications on Asymmetric CMPs. In *ISCA*, 2013.

[109] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das. Exploiting Core-Criticality for Enhanced GPU Performance. In *SIGMETRICS*, 2016.

[110] T. Johnson, D. Connors, M. Merten, and W.-M. Hwu. Run-time cache bypassing. In *IEEE TC*, 1999.

[111] T. S. Jung. Memory technology and solutions roadmap. http://www.sec.co.kr/images/corp/ir/irevent/techforum_01.pdf, 2005.

[112] D. Kang, S. Baek, J. Choi, D. Lee, S. H. Noh, and O. Mutlu. Amnesic Cache Management for Non-Volatile Memory. In *MSST*, 2015.

[113] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. Choi. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *The Memory Forum*, 2014.

[114] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das. Managing GPU Concurrency in Heterogeneous Architectures. In *MICRO*, 2014.

[115] B. Keeth and R. J. Baker. *DRAM Circuit Design: A Tutorial*. Wiley-IEEE Press, 2000.

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

[117] D. Keitel-Schulz and N. Wehn. Embedded DRAM development: Technology, physical design, and application issues. In *DTC*, 2001.

[118] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutlu, and D. A. Jimenez. Improving Cache Performance by Exploiting Read-Write Disparity. In *HPCA*, 2014.

[119] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.

[120] S. Khan, D. Lee, and O. Mutlu. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016.

[121] K. Kim and J. Lee. A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs. In *EDL*, 2009.

[122] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014.

[123] Y. Kim et al. Ramulator source code. http://www.ece.cmu.edu/~safari/tools.html.

[124] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*, 2010.

[125] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *MICRO*, 2010.

[126] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*, 2012.

[127] Y. Kim, W. Yang, and O. Mutlu. Ramulator source code. https://github.com/CMU-SAFARI/ramulator.

[128] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. In *IEEE CAL*, 2015.

[129] T. Kimuta, K. Takeda, Y. Aimoto, N. Nakamura, T. Iwasaki, Y. Nakazawa, H. Toyoshima, M. Hamada, M. Togo, H. Nobusawa, and T. Tanigawa. 64Mb 6.8ns Random Row Access DRAM Macro for ASICs. In *ISSCC*, 1999.

[130] P. M. Kogge. EXECUBE-A New Architecture for Scaleable MPPs. In *ICPP*, 1994.

[131] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. Evaluating STT-RAM as an energy-efficient main memory alternative. In *ISPASS*, 2013.

[132] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *MICRO*, 2003.

[133] M. Kund, G. Beitel, C.-U. Pinnow, T. Rohr, J. Schumann, R. Symanczyk, K.-D. Ufert, and G. Muller. Conductive bridging RAM (CBRAM): an emerging non-volatile memory technology scalable to sub 20nm. In *IEDM*, 2005.

[134] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-Change Technology and the Future of Main Memory. In *IEEE Micro*, 2010.

[135] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory As a Scalable DRAM Alternative. In *ISCA*, 2009.

[136] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Phase Change Memory Architecture and the Quest for Scalability. In *CACM*, 2010.

[137] C. J. Lee, E. Ebrahimi, V. Narasiman, O. Mutlu, and Y. N. Patt. DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems. In *UT Tech Report TR-HPS-2010-002*, 2010.

[138] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt. Prefetch-Aware DRAM Controllers. In *MICRO*, 2008.

[139] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt. Improving Memory Bank-level Parallelism in the Presence of Prefetching. In *MICRO*, 2009.

[140] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. In *ACM TACO*, 2016.

[141] D. Lee, F. Hormozdiari, H. Xin, F. Hach, O. Mutlu, and C. Alkan. Fast and accurate mapping of complete genomics reads. In *Methods*, 2014.

[142] D. Lee, S. Khan, L. Subramanian, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu. Full data sets for AVA-DRAM: Reducing DRAM Latency by Exploiting Architectural Variation. http://www.ece.cmu.edu/~safari/tools/avadram-fulldata.html.

[143] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *HPCA*, 2015.

[144] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. K. Chang, and O. Mutlu. Full data sets for Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case. http://www.ece.cmu.edu/~safari/tools/aldram-hpca2015-fulldata.html.

[145] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*, 2013.

[146] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu. Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM. In *PACT*, 2015.

[147] J. Lee, K. Kim, Y. Shin, K. Lee, J. Kim, D. Kim, J. Park, and J. Lee. Simultaneously Formed Storage Node Contact and Metal Contact Cell (SSMC) for 1Gb DRAM and Beyond. In *IEDM*, 1996.

[148] X. Li, K. Shen, M. C. Huang, and L. Chu. A Memory Soft Error Measurement on Production Systems. In *ATC*, 2007.

[149] Y. Li, J. Choi, J. Sun, S. Ghose, H. Wang, J. Meza, J. Ren, and O. Mutlu. Managing Hybrid Main Memories with a Page-Utility Driven Performance Model. In *CoRR abs/1507.03303*, 2015.

[150] Y. Li, H. Schneider, F. Schnabel, R. Thewes, and D. Schmitt-Landsiedel. DRAM Yield Analysis and Optimization by a Statistical Design Approach. In *IEEE TCSI*, 2011.

[151] K. Lim, S. Kang, J. Choi, J. Joo, Y. Lee, J. Lee, S. Cho, and B. Ryu. Bit line coupling scheme and electrical fuse circuit for reliable operation of high density DRAM. In *VLSIC*, 2001.

[152] K.-N. Lim, W.-J. Jang, H.-S. Won, K.-Y. Lee, H. Kim, D.-W. Kim, M.-H. Cho, S.-L. Kim, J.-H. Kang, K.-W. Park, and B.-T. Jeong. A 1.2V 23nm 6F2 4Gb DDR3 SDRAM with local-bitline sense amplifier, hybrid LIO sense amplifier and dummy-less array architecture. In *ISSCC*, 2012.

[153] J. Lin, H. Zheng, Z. Zhu, E. Gorbatov, H. David, and Z. Zhang. Software Thermal Management of DRAM Memory for Multicore Systems. In *SIGMETRICS*, 2008.

[154] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value Locality and Load Value Prediction. In *ASPLOS*, 1996.

[155] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*, 2013.

[156] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.

[157] S. Liu, B. Leung, A. Neckar, S. Memik, G. Memik, and N. Hardavellas. Hardware/software techniques for DRAM thermal management. In *HPCA*, 2011.

[158] G. H. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *ISCA*, 2008.

[159] S.-L. Lu, Y.-C. Lin, and C.-L. Yang. Improving DRAM Latency with Dynamic Asymmetric Subarray. In *MICRO*, 2015.

[160] Y. Lu, J. Shu, J. Guo, S. Li, and O. Mutlu. High-Performance and Lightweight Transaction Support in Flash-Based SSDs. In *IEEE TC*, 2015.

[161] Y. Lu, J. Shu, L. Sun, and O. Mutlu. Loose-Ordering Consistency for Persistent Memory. In *ICCD*, 2014.

[162] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.

[163] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke. Composite Cores: Pushing Heterogeneity Into a Core. In *MICRO*, 2012.

[164] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In *MSST*, 2015.

[165] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu. Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory. In *DSN*, 2014.

[166] M. Meterelliyoz, F. Al-amoody, U. Arslan, F. Hamzaoglu, L. Hood, M. Lal, J. Miller, A. Rama-sundar, D. Soltman, W. Ifar, Y. Wang, and K. Zhang. 2nd Generation Embedded DRAM with 4X Lower Self Refresh Power in 22nm Tri-Gate CMOS Technology. In *VLSI*, 2014.

[167] J. Meza et al. Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management. In *IEEE CAL*, 2012.

[168] J. Meza, J. Li, and O. Mutlu. A Case for Small Row Buffers in Non-Volatile Main Memories. In *ICCD, Poster Session*, 2012.

[169] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu. A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory. In *WEED*, 2013.

[170] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. A large-scale study of flash memory failures in the field. In *SIGMETRICS*, 2015.

[171] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *DSN*, 2015.

[172] Micron. 4Gb DDR3 SDRAM. http://www.micron.com/get-document/?documentId=5605&file=4Gb_DDR3_SDRAM.pdf.

[173] Micron. RLDRAM 2 and 3 Specifications. http://www.micron.com/products/dram/rldram-memory.

[174] Micron. Mobile DRAM Power-Saving Features and Power Calculations. http://www.micron.com/~/media/documents/products/technical-note/dram/tn4612.pdf, 2005.

[175] Micron. DDR3 SDRAM System-Power Calculator, 2010.

[176] Micron. 4Gb DDR3 SDRAM (MT41J512M8), 2012.

[177] K.-S. Min, J.-T. Park, S.-P. Lee, Y.-H. Kim, T.-H. Yang, J.-D. Joo, K.-M. Lee, J.-K. Wee, and J.-Y. Chung. A post-package bit-repair scheme using static latches with bipolar-voltage programmable antifuse circuit for high-density DRAMs. In *VLSI*, 2001.

[178] A. K. Mishra, O. Mutlu, and C. R. Das. A Heterogeneous Multiple Network-on-Chip Design: An Application-Aware Approach. In *DAC*, 2013.

[179] Y. Moon, Y.-H. Cho, H.-B. Lee, B.-H. Jeong, S.-H. Hyun, B.-C. Kim, I.-C. Jeong, S.-Y. Seo, J.-H. Shin, S.-W. Choi, H.-S. Song, J.-H. Choi, K.-H. Kyung, Y.-H. Jun, and K. Kim. 1.2V 1.6Gb/s 56nm $6F^2$ 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture. In *ISSCC*, 2009.

[180] Y. Mori, K. Ohyu, K. Okonogi, and R.-I. Yamada. The origin of variable retention time in DRAM. In *IEDM*, 2005.

[181] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-core Systems. In *USENIX Security*, 2007.

[182] T. Moscibroda and O. Mutlu. Distributed Order Scheduling and Its Application to Multi-core Dram Controllers. In *PODC*, 2008.

[183] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *MICRO*, 2011.

[184] N. Muralimanohar and R. Balasubramonian. Interconnect Design Considerations for Large NUCA Caches. In *ISCA*, 2007.

[185] O. Mutlu. Asymmetry everywhere (with automatic resource management). In *NSF ACAR*, 2010.

[186] O. Mutlu. Memory Scaling: A Systems Architecture Perspective. In *MemCon*, 2013.

[187] O. Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IMW*, 2013.

[188] O. Mutlu, H. Kim, and Y. Patt. Address-value delta (AVD) prediction: increasing the effectiveness of runahead execution by exploiting regular memory allocation patterns. In *MICRO*, 2005.

[189] O. Mutlu, H. Kim, and Y. Patt. Techniques for efficient processing in runahead execution engines. In *ISCA*, 2005.

[190] O. Mutlu, H. Kim, and Y. N. Patt. Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses. In *IEEE TC*, 2006.

[191] O. Mutlu, H. Kim, and Y. N. Patt. Efficient Runahead Execution: Power-efficient Memory Latency Tolerance. In *IEEE Micro*, 2006.

[192] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*, 2007.

[193] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems. In *ISCA*, 2008.

[194] O. Mutlu, J. Stark, C. Wilkerson, and Y. Patt. Runahead execution: an alternative to very large instruction windows for out-of-order processors. In *HPCA*, 2003.

[195] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt. Runahead execution: An effective alternative to large instruction windows. In *IEEE Micro*, 2003.

[196] O. Mutlu and L. Subramanian. Research Problems and Opportunities in Memory Systems. In *SUPERFRI*, 2015.

[197] S. Narasimha, P. Chang, C. Ortolland, D. Fried, E. Engbrecht, K. Nummy, P. Parries, T. Ando, M. Aquilino, N. Arnold, R. Bolam, J. Cai, M. Chudzik, B. Cipriany, G. Costrini, M. Dai, J. Dechene, C. Dewan, B. Engel, M. Gribelyuk, D. Guo, G. Han, N. Habib, J. Holt, D. Ioannou, B. Jagannathan, D. Jaeger, J. Johnson, W. Kong, J. Koshy, R. Krishnan, A. Kumar, M. Kumar, J. Lee, X. Li, C. Lin, B. Linder, S. Lucarini, N. Lustig, P. McLaughlin, K. Onishi, V. Ontalus, R. Robison, C. Sheraw, M. Stoker, A. Thomas, G. Wang, R. Wise, L. Zhuang, G. Freeman, J. Gill, E. Maciejewski, R. Malik, J. Norum, and P. Agnello. 22nm High-Performance SOI Technology Featuring Dual-Embedded Stressors, Epi-Plate High-K Deep-Trench Embedded DRAM and Self-Aligned Via 15LM BEOL. In *IEDM*, 2012.

[198] S. Narasimha, K. Onishi, H. Nayfeh, A. Waite, M. Weybright, J. Johnson, C. Fonseca, D. Corliss, C. Robinson, M. Crouse, D. Yang, C.-H. Wu, A. Gabor, T. Adam, I. Ahsan, M. Belyansky, L. Black, S. Butt, J. Cheng, A. Chou, G. Costrini, C. Dimitrakopoulos, A. Domenicucci, P. Fisher, A. Frye, S. Gates, S. Greco, S. Grunow, M. Hargrove, J. Holt, S.-J. Jeng, M. Kelling, B. Kim, W. Landers, G. LaRosa, D. Lea, M. Lee, X. Liu, N. Lustig, A. McKnight, L. Nicholson, D. Nielsen, K. Nummy, V. Ontalus, C. Ouyang, X. Ouyang, C. Prindle, R. Pal, W. Rausch, D. Restaino, C. Sheraw, J. Sim, A. Simon, T. Standaert, C. Sung, K. Tabakman, C. Tian, R. Van Den Nieuwenhuizen, H. van Meer, A. Vayshenker, D. Wehella-Gamage, J. Werking,

R. Wong, J. Yu, S. Wu, R. Augur, D. Brown, X. Chen, D. Edelstein, A. Grill, M. Khare, Y. Li, S. Luning, J. Norum, S. Sankaran, D. Schepis, R. Wachnik, R. Wise, C. Warm, T. Ivers, and P. Agnello. High Performance 45-nm SOI Technology with Enhanced Strain, Porous Low-k BEOL, and Immersion Lithography. In *IEDM*, 2006.

[199] NEC. Virtual Channel SDRAM uPD4565421, 1999.

[200] K. Nesbit, A. Dhodapkar, and J. Smith. AC/DC: an adaptive data cache prefetcher. In *PACT*, 2004.

[201] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *MICRO*, 2006.

[202] D. Niu, Y. Xiao, and Y. Xie. Low power memristor-based ReRAM design with Error Correcting Code. In *ASP-DAC*, 2012.

[203] G. Nychis, C. Fallin, T. Moscibroda, and O. Mutlu. Next Generation On-chip Networks: What Kind of Congestion Control Do We Need? In *Hotnets*, 2010.

[204] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects. In *SIGCOMM*, 2012.

[205] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *MICRO*, 2004.

[206] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. In *IEEE Micro*, 1997.

[207] D. A. Patterson. Latency lags bandwith. *Commun. ACM*, Oct. 2004.

[208] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *SOSP*, 1995.

[209] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler. Toggle-Aware Bandwidth Compression for GPUs. In *HPCA*, 2016.

[210] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. P. Gibbons, M. A. Kozuch, and T. C. Mowry. Exploiting Compressed Block Size as an Indicator of Future Reuse. In *HPCA*, 2015.

[211] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Linearly Compressed Pages: A Low-complexity, Low-latency Main Memory Compression Framework. In *MICRO*, 2013.

[212] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches. In *PACT*, 2012.

[213] S. Pelley, P. M. Chen, and T. F. Wenisch. Memory Persistency. In *ISCA*, 2014.

[214] S. Phadke and S. Narayanasamy. MLP Aware Heterogeneous Memory System. In *DATE*, 2011.

[215] T. Piquet, O. Rochecouste, and A. Seznec. Exploiting Single-Usage for Effective Memory Management. In *ACSAC*, 2007.

147

[216] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO*, 2009.

[217] M. Qureshi, D.-H. Kim, S. Khan, P. Nair, and O. Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.

[218] M. K. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer. Adaptive Insertion Policies for High Performance Caching. In *ISCA*, 2007.

[219] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A Case for MLP-Aware Cache Replacement. In *ISCA*, 2006.

[220] M. K. Qureshi and Y. N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO*, 2006.

[221] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *ISCA*, 2009.

[222] M. Rahman, B. Childers, and S. Cho. COMeT: Continuous Online Memory Test. In *PRDC*, 2011.

[223] Rambus. DRAM Power Model. http://www.rambus.com/energy, 2010.

[224] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page placement in hybrid memory systems. In *ICS*, 2011.

[225] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. In *IBM Journal of Research and Development*, 2008.

[226] B. Razavi. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill, 2000.

[227] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu. ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems. In *MICRO*, 2015.

[228] P. J. Restle, J. W. Park, and B. F. Lloyd. DRAM variable retention time. In *IEDM*, 1992.

[229] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory Access Scheduling. In *ISCA*, 2000.

[230] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In *ISCA*, 2009.

[231] M. Saito, J. Ogawa, K. Gotoh, S. Kawashima, and H. Tamura. Technique for controlling effective Vth in multi-Gbit DRAM sense amplifier. In *VLSIC*, 1996.

[232] Samsung. DRAM Data Sheet. http://www.samsung.com/global/business/semiconductor.

[233] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[234] R. H. Sartore, K. J. Mobley, D. G. Carrigan, and O. F. Jones. Enhanced DRAM with embedded registers. U.S. patent, 1999. Patent number 5887272.

[235] Y. Sato, T. Suzuki, T. Aikawa, S. Fujioka, W. Fujieda, H. Kobayashi, H. Ikeda, T. Nagasawa, A. Funyu, Y. Fuji, K. Kawasaki, M. Yamazaki, and M. Taguchi. Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM. In *Symposium on VLSI Circuits*, 1998.

[236] Y. Sazeides and J. E. Smith. The Predictability of Data Values. In *MICRO*, 1997.

[237] B. Schroeder and G. Gibson. A Large-Scale Study of Failures in High-Performance Computing Systems. In *TDSC*, 2010.

[238] V. Seshadri, A. Bhowmick, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry. The Dirty-Block Index. In *ISCA*, 2014.

[239] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. Kozuch, O. Mutlu, P. Gibbons, and T. Mowry. Fast Bulk Bitwise AND and OR in DRAM. In *IEEE CAL*, 2015.

[240] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. RowClone: Fast and Energy-efficient in-DRAM Bulk Data Copy and Initialization. In *MICRO*, 2013.

[241] V. Seshadri, T. Mullins, A. Boroumand, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses. In *MICRO*, 2015.

[242] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry. The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing. In *PACT*, 2012.

[243] V. Seshadri, G. Pekhimenko, O. Ruwase, O. Mutlu, P. B. Gibbons, M. A. Kozuch, T. C. Mowry, and T. Chilimbi. Page overlays: An enhanced virtual memory framework to enable fine-grained memory management. In *ISCA*, 2015.

[244] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. Mitigating Prefetcher-Caused Pollution Using Informed Caching Policies for Prefetched Blocks. In *ACM TACO*, 2015.

[245] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis. MemZip: Exploring Unconventional Benefits from Memory Compression. In *HPCA*, 2014.

[246] S. M. Sharroush, Y. S. Abdalla, A. A. Dessouki, and E.-S. A. El-Badawy. Dynamic random-access memories without sense amplifiers. In *Elektrotechnik & Informationstechnik*, 2012.

[247] W. Shin, J. Yang, J. Choi, and L.-S. Kim. NUAT: A Non-Uniform Access Time Memory Controller. In *HPCA*, 2014.

[248] A. Singh, D. Bose, and S. Darisala. Software based in-system memory test for highly available systems. In *MTDT*, 2005.

[249] B. J. Smith. A pipelined, shared resource MIMD computer. In *ICPP*, 1978.

[250] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS*, 2000.

[251] A. Sodani and G. S. Sohi. Dynamic Instruction Reuse. In *ISCA*, 1997.

[252] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn. Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations. In *ISCA*, 2013.

[253] V. Sridharan and D. Liberty. A Study of DRAM Failures in the Field. In *SC*, 2012.

[254] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults. In *SC*, 2013.

[255] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt. Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers. In *HPCA*, 2007.

[256] Standard Performance Evaluation Corporation. SPEC CPU2006.

[257] H. S. Stone. A Logic-in-Memory Computer. In *IEEE TC*, 1970.

[258] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. The Blacklisting Memory Scheduler: Achieving high performance and fairness at low cost. In *ICCD*, 2014.

[259] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. The Blacklisting Memory Scheduler: Balancing Performance, Fairness and Complexity. In *TPDS*, 2016.

[260] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu. The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory. In *MICRO*, 2015.

[261] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu. MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems. In *HPCA*, 2013.

[262] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micropages: Increasing DRAM efficiency with locality-aware data placement. In *ASPLOS*, 2010.

[263] M. A. Suleman, O. Mutlu, J. A. Joao, Khubaib, and Y. N. Patt. Data Marshaling for Multi-core Architectures. In *ISCA*, 2010.

[264] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS*, 2009.

[265] A. Tanabe, T. Takeshima, H. Koike, Y. Aimoto, M. Takada, T. Ishijima, N. Kasai, H. Hada, K. Shibahara, T. Kunio, T. Tanigawa, T. Saeki, M. Sakao, H. Miyamoto, H. Nozue, S. Ohya, T. Murotani, K. Koyama, and T. Okuda. A 30-ns 64-Mb DRAM with built-in self-test and self-repair function. In *JSSC*, 1992.

[266] J. E. Thornton. Parallel operation in the Control Data 6600. In *Fall Joint Computer Conference*, 1964.

[267] M. Thottethodi, A. Lebeck, and S. Mukherjee. Self-tuned congestion control for multiprocessor networks. In *HPCA*, 2001.

[268] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies. In *ISCA*, 2008.

[269] B. Thwaites, G. Pekhimenko, H. Esmaeilzadeh, A. Yazdanbakhsh, O. Mutlu, J. Park, G. Mururu, and T. Mowry. Rollback-free Value Prediction with Approximate Loads. In *PACT*, 2014.

[270] C. Toal, D. Burns, K. McLaughlin, S. Sezer, and S. OKane. An RLDRAM II Implementation of a 10Gbps Shared Packet Buffer for Network Processing. In *AHS*, 2007.

[271] Transaction Processing Performance Council. TPC Benchmark. http://www.tpc.org/.

[272] H. Usui, L. Subramanian, K. Chang, and O. Mutlu. DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators. In *ACM TACO*, 2016.

[273] A. J. van de Goor and I. Schanstra. Address and Data Scrambling: Causes and Impact on Memory Tests. In *DELTA*, 2002.

[274] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating system support for improving data locality on CC-NUMA compute servers. In *ASPLOS*, 1996.

[275] T. Vogelsang. Understanding the Energy Consumption of Dynamic Random Access Memories. In *MICRO*, 2010.

[276] M.-J. Wang, R.-L. Jiang, J.-W. Hsia, C.-H. Wang, and J.-E. Chen. Guardband determination for the detection of off-state and junction leakages in DRAM testing. In *Asian Test Symposium*, 2001.

[277] F. Ware and C. Hampel. Improving Power and Data Efficiency with Threaded Memory Modules. In *ICCD*, 2006.

[278] J.-K. Wee, W. Yang, E.-K. Ryou, J.-S. Choi, S.-H. Ahn, J.-Y. Chung, and S.-C. Kim. An antifuse EPROM circuitry scheme for field-programmable repair in DRAM. In *JSSC*, 2000.

[279] M. V. Wilkes. The memory gap and the future of high performance memories. In *SIGARCH Comput. Archit. News*, 2001.

[280] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis. The Case for Compressed Caching in Virtual Memory Systems. In *ATEC*, 1999.

[281] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. Chen, and M.-J. Tsai. Metal Oxide RRAM. In *Proceedings of the IEEE*, 2012.

[282] H.-S. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase Change Memory. In *Proceedings of the IEEE*, 2010.

[283] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. In *SIGARCH Comput. Archit. News*, 1995.

[284] Xilinx. *Virtex-6 FPGA Integrated Block for PCI Express*, 2011. http://www.xilinx.com/support/documentation/ip_documentation/mig/v3_92/ug406.pdf.

[285] Xilinx. *ML605 Hardware User Guide*, 2012. http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf.

[286] Xilinx. *Virtex-6 FPGA Memory Interface Solutions*, 2013. http://www.xilinx.com/support/documentation/ip_documentation/mig/v3_92/ug406.pdf.

[287] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan. Accelerating read mapping with fasthash. In *BMC Genomics*, 2013.

[288] D. Yaney, C. Y. Lu, R. Kohler, M. J. Kelly, and J. Nelson. A meta-stable leakage phenomenon in DRAM charge storage - Variable hold time. In *IEDM*, 1987.

[289] A. Yazdanbakhsh, G. Pekhimenko, B. Thwaites, H. Esmaeilzadeh, O. Mutlu, and T. C. Mowry. RFVP: Rollback-Free Value Prediction with Safe-to-Approximate Loads. In *ACM TACO*, 2016.

[290] A. Yazdanbakhsh, B. Thwaites, H. Esmaeilzadeh, G. Pekhimenko, O. Mutlu, and T. Mowry. Mitigating the memory bottleneck with approximate load value prediction. In *IEEE Design and Test*, 2016.

[291] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu. Row Buffer Locality Aware Caching Policies for Hybrid Memories. In *ICCD*, 2012.

[292] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu. Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories. In *ACM TACO*, 2014.

[293] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: Throughput-Oriented Programmable Processing in Memory. In *HPDC*, 2014.

[294] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie. Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. In *ISCA*, 2014.

[295] Y. Zhang, J. Yang, and R. Gupta. Frequent value locality and value-centric data cache design. In *ASPLOS*, 2000.

[296] Z. Zhang, Z. Zhu, and X. Zhang. Cached DRAM for ILP Processor Memory Access Latency Reduction. In *IEEE Micro*, 2001.

[297] J. Zhao, O. Mutlu, and Y. Xie. FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems. In *MICRO*, 2014.

[298] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *MICRO*, 2008.

[299] H. Zhou and T. M. Conte. Enhancing Memory Level Parallelism via Recovery-free Value Prediction. In *ICS*, 2003.

[300] Q. Zhu, X. Li, and Y. Wu. Thermal management of high power memory module for server platforms. In *ITHERM*, 2008.

[301] W. K. Zuravleff and T. Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. U.S. patent, 1997. Patent number 5630096.