Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization

Kevin K. Chang[1] Abhijith Kashyap[1] Hasan Hassan[1,2]
Saugata Ghose[1] Kevin Hsieh[1] Donghyuk Lee[1] Tianshi Li[1,3]
Gennady Pekhimenko[1] Samira Khan[4] Onur Mutlu[5,1]
[1]Carnegie Mellon University [2]TOBB ETÜ [3]Peking University [4]University of Virginia [5]ETH Zürich

ABSTRACT

Long DRAM latency is a critical performance bottleneck in current systems. DRAM access latency is defined by three fundamental operations that take place within the DRAM cell array: *(i) activation* of a memory row, which opens the row to perform accesses; *(ii) precharge*, which prepares the cell array for the next memory access; and *(iii) restoration* of the row, which restores the values of cells in the row that were destroyed due to activation. There is significant latency variation for each of these operations across the cells of a single DRAM chip due to irregularity in the manufacturing process. As a result, some cells are *inherently* faster to access, while others are inherently slower. Unfortunately, existing systems do not exploit this variation.

The goal of this work is to *(i)* experimentally characterize and understand the latency variation across cells within a DRAM chip for these three fundamental DRAM operations, and *(ii)* develop new mechanisms that exploit our understanding of the latency variation to reliably improve performance. To this end, we comprehensively characterize 240 DRAM chips from three major vendors, and make several new observations about latency variation within DRAM. We find that *(i)* there is large latency variation across the cells for each of the three operations; *(ii)* variation characteristics exhibit significant spatial locality: slower cells are clustered in certain regions of a DRAM chip; and *(iii)* the three fundamental operations exhibit different reliability characteristics when the latency of each operation is reduced.

Based on our observations, we propose Flexible-LatencY DRAM (FLY-DRAM), a mechanism that exploits latency variation across DRAM cells within a DRAM chip to improve system performance. The key idea of FLY-DRAM is to exploit the spatial locality of slower cells within DRAM, and access the faster DRAM regions with reduced latencies for the fundamental operations. Our evaluations show that FLY-DRAM improves the performance of a wide range of applications by 13.3%, 17.6%, and 19.5%, on average, for each of the three different vendors' real DRAM chips, in a simulated 8-core system. We conclude that the experimental characterization and analysis of latency variation within modern DRAM, provided by this work, can lead to new techniques that improve DRAM and system performance.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMETRICS '16, June 14–18, 2016, Antibes Juan-Les-Pins, France.
© 2016 ACM. ISBN 978-1-4503-4266-7/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2896377.2901453

1. INTRODUCTION

Over the past few decades, the long latency of memory has been a critical bottleneck in system performance. Increasing core counts, emergence of more data-intensive and latency-critical applications, and increasingly limited bandwidth in the memory system are together leading to higher memory latency. Thus, low-latency memory operation is now even more important to improving overall system performance [9, 11, 16, 27, 36, 37, 38, 45, 49, 53, 54, 56, 66, 68, 70, 79].

The latency of a memory request is predominantly defined by the timings of two fundamental operations: *activation* and *precharge*. These operations take place on the two-dimensional arrays of memory *cells* that store data. Activation is the process of "opening" a row of cells, in order to allow data within that row to be accessed. Once a row is *activated*, the memory controller can read from or write to it one cache line at a time. Precharge is the process of "closing" the activated row, and preparing the cell array for the next memory access. Once the array is *precharged*, another row can be activated. Specifically for DRAM, we also need to consider the latency of a third operation, known as *restoration*. A DRAM cell uses a capacitor, whose charge level represents the stored data value. The activation process of a row affects the charge level in the capacitor, which can destroy the data value stored within the cell. To prevent data loss, DRAM must *restore* the charge level of each DRAM cell in the row to reflect the cell's data value before activation, which takes time.

The latencies of these three DRAM operations, as defined by vendor specifications, have *not* improved significantly in the past decade, as depicted in Figure 1. This is especially true when we compare latency improvements to the capacity ($64\times = \frac{8Gb}{128Mb}$) and bandwidth improvements ($16\times \approx \frac{2133MT/s}{133MT/s}$) [21, 23, 37, 38, 70] commodity DRAM chips experienced in the past decade. In fact, the activation and precharge latencies *increased* from 2013 to 2015, when DDR DRAM transitioned from the third generation (12.5ns for DDR3-1600J [21]) to the fourth generation (14.06ns for DDR4-2133P [23]). As the latencies specified by vendors have not reduced over time, the system performance bottleneck caused by raw main memory latency remains largely unaddressed in modern systems.

In this work, we observe that the three fundamental DRAM operations can *actually* complete with a much lower

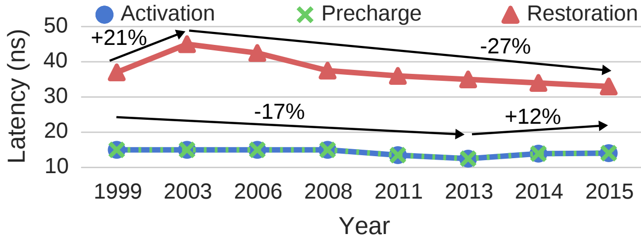


Figure 1: DRAM latency trends over time [20, 21, 23, 51].

latency for many DRAM cells than the specification, because *there is inherent latency variation present across the DRAM cells within a DRAM chip*. This is a result of manufacturing process variation, which causes the *sizes* and *strengths* of cells to be different, thus making some cells faster and other cells slower to be accessed reliably [15, 31, 41]. The speed gap between the fastest and slowest DRAM cells is getting worse [5, 57], as the technology node continues to scale down to sub-20nm feature sizes. Unfortunately, instead of optimizing the latency specifications for the common case, DRAM vendors use a single set of standard access latencies, which provide reliable operation guarantees for the *worst case* (i.e., slowest cells), to maximize manufacturing yield.

We find that the (widening) speed gap among DRAM cells presents an opportunity to reduce DRAM access latency. If we can understand and characterize the inherent variation in cell latencies, we can use the resulting understanding to reduce the access latency for those rows that contain faster cells. **The goal of this work** is to *(i)* experimentally characterize and understand the impact of latency variation in the three fundamental DRAM operations for cell access (activation, precharge, and restoration), and *(ii)* develop new mechanisms that take advantage of this variation to improve system performance.

To this end, we build an FPGA-based DRAM testing infrastructure and characterize 240 DRAM chips from three major vendors. We analyze the variations in the latency of the three fundamental DRAM operations by operating DRAM at multiple reduced latencies. Faster cells do not get affected by the reduced timings, and can be accessed reliably without changing their stored value; however, slower cells *cannot* be reliably read with reduced latencies for the three operations, leading to bit flips. In this work, we define a *timing error* as a bit flip in a cell that occurs due to a reduced-latency access, and characterize timing errors incurred by the three DRAM operations. Our experimental characterization yields six new **key observations**.

First, we find that significant variation is present in modern DRAM chips for the latencies of all three fundamental DRAM operations. For example, we observe that 68%, 100%, and 36% of cache lines can be read reliably when activation/restoration/precharge latencies are reduced by 43%/36%/43%, respectively, across all 240 of our tested DRAM chips. We conclude that exploiting latency variation in DRAM cells has the potential to greatly reduce the access latency.

Second, we find that when we reduce the latency for different DRAM operations, there is spatial locality in inherently slower cells: such cells are clustered in certain *regions* of a DRAM chip, as opposed to being randomly distributed. We conclude that such spatial locality can be exploited to develop low-cost mechanisms to reduce latency, where fast regions are accessed with lower latency, and slow regions are accessed with the standard high latency.

Third, when we reduce the three latencies, we observe that each latency exhibits a different level of impact on the inherently-slower cells. Lowering the activation latency affects *only* the cells read in the first accessed cache line. In contrast, lowering the restoration or precharge latencies affects *all* cells in the row. We explain in detail why this is the case. We also find that the number of timing errors introduced is very sensitive to reductions in activation and precharge latencies, but not that sensitive to reduction in restoration latency. We conclude that different levels of mitigation are required to address the timing errors that result from lowering each of the different DRAM operation latencies, and that reducing restoration latency does not introduce timing errors in our experiments.

Fourth, we analyze the number of timing errors that occur when DRAM access latencies are reduced, and experimentally demonstrate that most of the erroneous cache lines have a single-bit error, with only a small fraction of cache lines experiencing more than one bit flip. We conclude, therefore, that using simple error-correcting codes (ECC) can correct most of these errors, thereby enabling lower latency for many inherently slower cells.

Fifth, we find no clear correlation between temperature and variation in cell access latency. We believe that it is not essential for latency reduction techniques that exploit such variation to be aware of the operating temperature.

Sixth, we find that the stored data pattern in cells affects access latency variation. Certain patterns lead to more timing errors than others. For example, the bit value `1` can be read significantly more reliably at a reduced access latency than the bit value `0`. We conclude that it is promising to investigate asymmetric data encoding or error correction mechanisms that favor `1`s over `0`s.

Based on these major conclusions from our comprehensive analysis and characterization of 240 DRAM chips from three major DRAM manufacturers, we propose and evaluate a new mechanism, called FLY-DRAM (Flexible-LatencY DRAM). FLY-DRAM's key idea is to *(i)* categorize the DRAM cells into fast and slow regions, *(ii)* expose this information to the memory controller, and *(iii)* reduce overall DRAM latency by accessing the fast regions with a lower latency. Our simulation-based analysis shows that FLY-DRAM improves the performance of a wide range of applications in an 8-core system. Based on our experimental observations, we also discuss a page allocator design that exploits the latency variation in DRAM to improve system performance.

We hope that our extensive analysis leads to other new mechanisms to improve DRAM performance and reliability. To facilitate this, we will make our characterization results for all tested DRAM chips and the FLY-DRAM simulator publicly available [10].

To our knowledge, this is the first work to make the following major contributions:

- It experimentally demonstrates and characterizes the significant variation in latency of three fundamental DRAM operations (activation, restoration, and precharge) across different cells within a DRAM chip.
- It experimentally demonstrates that reducing the latency of each of these three fundamental DRAM operations has a different effect on slower cells. It shows that *(i)* while the

memory controller can introduce timing errors in slower cells by reducing the activation and precharge latencies, it can reduce the restoration latency without impacting these cells, thus providing greater opportunities to improve performance without causing timing errors; and *(ii)* errors due to reducing the activation latency appear only in the first cache line accessed in a row, limiting the scope of impact.

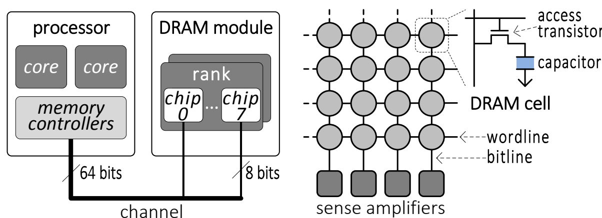
- It experimentally demonstrates that access latency variation exhibits spatial locality within DRAM, and that the error rate with reduced latencies is correlated with the stored data pattern in cells, but not with temperature.
- It proposes a new mechanism, FLY-DRAM, which exploits the lower latencies of DRAM regions with faster cells by introducing heterogeneous timing parameters into the memory controller. We find that FLY-DRAM improves performance in an 8-core system by 13.3%, 17.6%, and 19.5%, on average, for each of the three different vendors' real DRAM chips, across a wide range of applications.

2. BACKGROUND & MOTIVATION

In this section, we first provide necessary background on DRAM organization and operation to enable a better understanding of the major DRAM timing parameters we will characterize. Then, we discuss how we can exploit DRAM variation to reduce the DRAM access latency.

2.1 High-Level DRAM System Organization

A modern DRAM system consists of a hierarchy of channels, modules, ranks, and chips, as shown in Figure 2a. Each *memory channel* drives DRAM commands, addresses, and data between a memory controller in the processor and one or more DRAM modules. Each *module* contains multiple DRAM chips that are divided into one or more ranks. A *rank* refers to a group of chips that operate in lock step to provide a wide data bus (usually 64 bits), as a single DRAM chip is designed to have a narrow data bus width (usually 8 bits) to minimize chip cost. Each of the eight chips in the rank shown in Figure 2a transfers 8 bits simultaneously to supply 64 bits of data.



(a) DRAM System (b) DRAM Bank

Figure 2: DRAM system organization.

2.2 Internal DRAM Organization

Within a DRAM chip, there are multiple banks (e.g., eight in a typical DRAM chip [21]) that can process DRAM commands independently from each other to increase parallelism. A *bank* consists of a 2D-array of DRAM cells that are organized into rows and columns, as shown in Figure 2b. A row typically consists of 8K cells. The number of rows varies depending on the chip density. Each DRAM cell has *(i)* a *capacitor* that stores binary data in the form of electrical charge (i.e., fully charged and discharged states represent `1` and `0`, respectively), and *(ii)* an *access transistor* that serves as a switch to connect the capacitor to the *bitline*. Each column of cells share a bitline, which connects them to a *sense amplifier*. The sense amplifier senses the charge stored in a cell, converts the charge to digital binary data, and buffers it. Each row of cells share a wire called the *wordline*, which controls the cells' access transistors. When a row's wordline is enabled, the entire row of cells gets connected to the row of sense amplifiers through the bitlines, enabling the sense amplifiers to sense and latch that row's data. The row of sense amplifiers is also called the *row buffer*.

2.3 Accessing DRAM

Accessing (i.e., reading from or writing to) a bank consists of three steps: *(i)* **Row Activation & Sense Amplification**: opening a row to transfer its data to the row buffer, *(ii)* **Read/Write**: accessing the target column in the row buffer, and *(iii)* **Precharge**: closing the row and the row buffer. We use Figure 3 to explain these three steps in detail. The top part of the figure shows the phase of the cells within the row that is being accessed. The bottom part shows both the DRAM command and data bus timelines, and demonstrates the associated timing parameters.

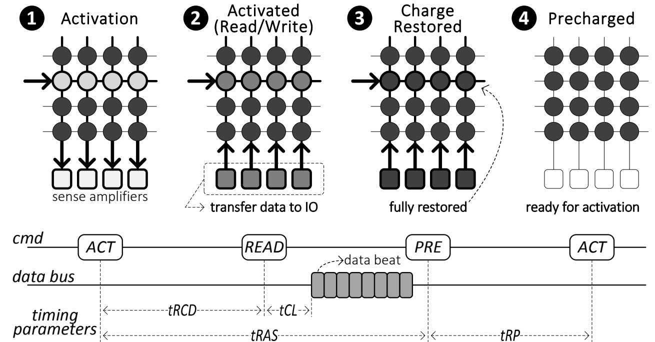


Figure 3: Internal DRAM phases, DRAM command/data timelines, and timing parameters to read a cache line.

Initial State. Initially, the bank remains in the *precharged* state (❹ in Figure 3), where all of the components are ready for activation. All cells are fully charged, represented with the dark grey color (a darker cell color indicates more charge). Second, the bitlines are charged to $V_{DD}/2$, represented as a thin line (a thin bitline indicates the initial voltage state of $V_{DD}/2$; a thick bitline means the bitline is being driven). Third, the wordline is disabled with 0V (a thin wordline indicates 0V; a thick wordline indicates V_{DD}). Fourth, the sense amplifier is *off* without any data latched in it (indicated by lighter color in the sense amplifier).

Row Activation & Sense Amplification Phases. To open a row, the memory controller sends an ACTIVATE command to raise the wordline of the corresponding row, which connects the row to the bitlines (❶). This triggers an *activation*, where charge starts to flow from the cell to the bitline (or the other way around, depending on the initial charge level in the cell) via a process called *charge sharing*. This process perturbs the voltage level on the corresponding bitline by a small amount. If the cell is initially charged (which we assume for the rest of this explanation, without loss of generality), the bitline voltage is perturbed upwards. Note that this causes the cell itself to discharge, losing its data temporarily (hence the lighter color of the accessed row),

but this charge will be restored as we will describe below. After the activation phase, the sense amplifier *senses* the voltage perturbation on the bitline, and turns *on* to further *amplify* the voltage level on the bitline by injecting more charge into the bitline and the cell (making the activated row darker in ❷). When the bitline is amplified to a certain voltage level (e.g., $0.8V_{DD}$), the sense amplifier latches in the cell's data, which transforms it into binary data (❷). At this point in time, the data can be read from the sense amplifier. The latency of these two phases (activation and sense amplification) is called the *activation latency*, and is defined as **tRCD** in the standard DDR interface [21, 23]. This activation latency specifies the latency from the time an ACTIVATE command is issued to the time the data is ready to be accessed in the sense amplifier.

Read/Write & Restoration Phases. Once the sense amplifier (row buffer) latches in the data, the memory controller can send a READ or WRITE command to access the corresponding column of data within the row buffer (called a *column access*). The column access time to read the cache line data is called **tCL** (**tCWL** for writes). These parameters define the time between the column command and the appearance of the *first beat of data* on the data bus, shown at the bottom of Figure 3. A *data beat* is a 64-bit data transfer from the DRAM to the processor. In a typical DRAM [21], a column READ command reads out 8 data beats (also called an 8-beat burst), thus reading a complete 64-byte cache line.

After the bank becomes activated and the sense amplifier latches in the binary data of a cell, it starts to *restore* the connected cell's charge back to its original fully-charged state (❸). This phase is known as *restoration*, and can happen in parallel with column accesses. The restoration latency (from issuing an ACTIVATE command to fully restoring a row of cells) is defined as **tRAS**, as shown in Figure 3.

Precharge Phase. In order to access data from a different row, the bank needs to be re-initialized back to the precharged state (❹). To achieve this, the memory controller sends a PRECHARGE command, which *(i)* disables the wordline of the corresponding row, disconnecting the row from the sense amplifiers, and *(ii)* resets the voltage level on the bitline back to the initialized state, $V_{DD}/2$, so that the sense amplifier can sense the charge from the newly opened row. The latency of a precharge operation is defined as **tRP**, which is the latency between a PRECHARGE and a subsequent ACTIVATE within the same bank.

Summary. As shown at the bottom of Figure 3, the latency of back-to-back accesses to different rows in DRAM is decided by tRAS + tRP (restoration latency + precharge latency). The latency of accessing data from a row is decided by tRCD (activation latency), and is then followed by tCL and the data transfer latency (both of which are independent of the activation/restoration/precharge operations). In this work, we focus on the three critical timing parameters: tRCD, tRP, and tRAS.

2.4 Opportunities for Reducing Latency

DRAM standards define fixed values that are used for each of the timing parameters that we have described (e.g., DDR3 DRAM [21, 24]). Unfortunately, these latencies do *not* reflect the *actual* time the DRAM operations take for each cell. This is because the true access latency varies for each cell, as every cell is different in size and strength due to manufacturing process variation effects. For simplicity, and to ensure that DRAM yield remains high, DRAM manufacturers define a single set of latencies that guarantees reliable operation, based on the *slowest* cell in *any* DRAM chip across *all* DRAM vendors. As a result, there is a significant opportunity to reduce DRAM latency if, instead of always using worst-case latencies, we employ the true latency for each cell that enables the three operations reliably.

Our goal in this work is to *(i)* understand the impact of cell variation in the three fundamental DRAM operations for cell access (activation, precharge, and restoration); *(ii)* experimentally characterize the latency variation in these operations; and *(iii)* develop new mechanisms that take advantage of this variation to reduce the latency of these three operations.

To achieve this goal, we discuss the impact of reducing activation (Section 4), precharge (Section 5), and restoration (Section 6) latencies on DRAM cells by experimentally analyzing and characterizing the latency variation in cells in 240 real DRAM chips.

3. EXPERIMENTAL METHODOLOGY

To study the effect of using different timing parameters on modern DDR3 DRAM chips, we developed a DRAM testing platform that allows us to precisely control the value of timing parameters and the tested DRAM location (i.e., banks, rows, and columns) within a module. The testing platform, shown in Figure 4, consists of Xilinx FPGA boards [80] and host PCs. We use the RIFFA [19] framework to communicate data over the PCIe bus from our customized *testing software* running on the host PC to our customized *test engine* on the FPGA. Each DRAM module is tested on an FPGA board, and is located inside a heat chamber that is connected to a temperature controller. Unless otherwise specified, we test modules at an ambient temperature of 20±1℃. We examine various temperatures in Section 4.5.

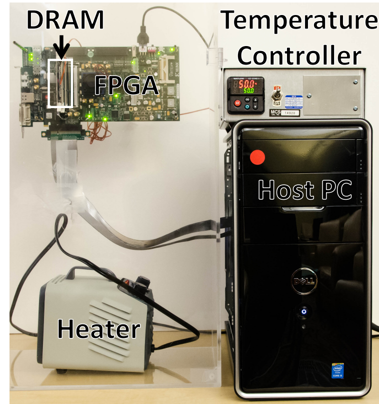


Figure 4: FPGA-based DRAM testing infrastructure.

3.1 DRAM Test

To achieve the goal of controlling timing parameters, our FPGA test engine supports a list of DRAM commands that get processed directly by the memory controller on the FPGA. Then, on the host PC, we can write a *test* that specifies a sequence of DRAM commands along with the delay between the commands (i.e., timing parameters). The test sends the commands and delays from the host PC to the FPGA test engine.

Test 1 shows the pseudocode of a test that reads a cache line from a particular bank, row, and column with timing parameters that can be specified by the user. The test first sends an ACTIVATE to the target row (line 2). After a tRCD delay that we specify (line 3), it sends a READ (line 4) to the target cache line. Our test engine enables us to specify the exact delay between two DRAM commands, thus allowing us to tune certain timing parameters. The read delay (tCL) and data transfer latency (BL) are two DRAM internal timings that cannot be changed using our infrastructure. After our test waits for the data to be fully transferred (line 5), we precharge the bank (line 6) with our specified tRP (line 7). We describe the details of the tests that we created to characterize latency variation of tRCD, tRP, and tRAS in the next few sections.

```
1 ReadOneCacheLine(my_tRCD, my_tRP, bank, row, col)
2   ACT(bank, row)
3   cmdDelay(my_tRCD)      ▷ Set activation latency (tRCD)
4   READ(bank, row, col)
5   cmdDelay(tCL +BL)      ▷ Wait for read to finish
6   PRE(bank)
7   cmdDelay(my_tRP)       ▷ Set precharge latency (tRP)
8   readData()             ▷ Send the read data from FPGA to PC
```

Test 1: Read a cache line with specified timing parameters.

3.2 Characterized DRAM Modules

We characterize latency variation on a total of 30 DDR3 DRAM modules, comprising 240 DRAM chips, from the three major DRAM vendors that hold more than 90% of the market share [2]. Table 1 lists the relevant information about the tested DRAM modules. All of these modules are *dual in-line* (i.e., 64-bit data bus) with a single rank of DRAM chips. Therefore, we use the terms *DIMM* (dual in-line memory module) and module interchangeably. In the rest of the paper, we refer to a specific DIMM using the label D_v^n, where n and v stand for the DIMM number and vendor, respectively. In the table, we group the DIMMs based on their model number, which provides certain information on the process technology and array design used in the chips.

Vendor	DIMM Name	Model	Timing (ns) (tRCD/tRP/tRAS)	Assembly Year
A (Total of 8 DIMMs)	D_A^{0-1}	M0	13.125/13.125/35	2013
	D_A^{2-3}	M1	13.125/13.125/36	2012
	D_A^{4-5}	M2	13.125/13.125/35	2013
	D_A^{6-7}	M3	13.125/13.125/35	2013
B (Total of 9 DIMMs)	D_B^{0-5}	M0	13.125/13.125/35	2011-12
	D_B^{6-8}	M1	13.125/13.125/35	2012
C (Total of 13 DIMMs)	D_C^{0-5}	M0	13.125/13.125/34	2012
	D_C^{6-12}	M1	13.125/13.125/36	2011

Table 1: Properties of tested DIMMs.

4. ACTIVATION LATENCY ANALYSIS

In this section, we present our methodology and results on varying the activation latency, which is expressed by the tRCD timing parameter. We first describe the nature of errors caused by tRCD reduction in Section 4.1. Then, we describe the FPGA test we conducted on the DRAM modules to characterize tRCD variation in Section 4.2. The remaining sections describe different major observations we make based on our results.

4.1 Behavior of Activation Errors

As we discuss in Section 2.3, tRCD is defined as the minimum amount of time between the ACTIVATE and the first column command (READ/WRITE). Essentially, tRCD represents the time it takes for a row of sense amplifiers (i.e., the row buffer) to sense and latch a row of data. By employing a lower tRCD value, a column READ command may potentially read data from sense amplifiers that are still in the *sensing and amplification* phase, during which the data has not been fully latched into the sense amplifiers. As a result, reading data with a lowered tRCD can induce timing errors (i.e., flipped bits) in the data.

To further understand the nature of activation errors, we perform experiments to answer two fundamental questions: *(i)* Does lowering tRCD incur errors on *all* cache lines read from a sequence of READ commands on an opened row? *(ii)* Do the errors propagate back to the DRAM cells, causing *permanent* errors for all future accesses?

4.1.1 Errors Localized to First Column Command

To answer the first question, we conduct Test 2 that first activates a row with a specific tRCD value, and then reads every cache line in the entire row. By conducting the test on every row in a number of DIMMs from all three vendors, we make the following observation.

```
1 ReadOneRow(my_tRCD, bank, row)
2   ACT(bank, row)
3   cmdDelay(my_tRCD)          ▷ Set activation latency
4   for c ← 1 to Col_MAX
5     READ(bank, row, c)       ▷ Read one cache line
6     findErrors()             ▷ Count errors in a cache line
7   cmdDelay(tCL + BL)
8   PRE(bank)
9   cmdDelay(tRP)
```

Test 2: Read one row with a specified tRCD value.

Observation 1: *Activation errors are isolated to the cache line from the first* READ *command, and do not appear in subsequently-read cache lines from the same row.*

There are two reasons why errors do *not* occur in the subsequent cache line reads. First, a READ accesses only its corresponding sense amplifiers, without accessing the other columns. Hence, a READ's effect is isolated to its target cache line. Second, by the time the second READ is issued, a sufficient amount of time has passed for the sense amplifiers to properly latch the data. Note that this observation is independent of DIMMs and vendors as the fundamental DRAM structure is similar across different DIMMs. We discuss the number of activation errors due to different tRCD values for each DIMM in Section 4.3.1.

4.1.2 Activation Errors Propagate into DRAM Cells

To answer our second question, we run two iterations of Test 2 (i.e., reading a row that is activated with a specified tRCD value) on the same row. The first iteration reads a row that is activated with a lower tRCD value, then closes the row. The second iteration re-opens the row using the standard tRCD value, and reads the data to confirm if the errors remain in the cells. Our experiments show that if

the first iteration observes activation errors within a cache line, the second iteration observes the same errors. This demonstrates that activation errors not only happen at the sense amplifiers but also propagate back into the cells.

We hypothesize this is because reading a cache line early causes the sense amplifiers to latch the data based on the *current* bitline voltage. If the bitline voltage has not yet fully developed into V_{DD} or 0V, the sense amplifier latches in unknown data and amplifies this data to the bitline, which is then restored back into the cell during restoration phase.

Observation 2: *Activation errors occur at the sense amplifiers and propagate back into the cells. The errors persist until the data is overwritten.*

After observing that reducing activation latency results in timing errors, we now consider two new questions. First, after how much activation latency reduction do DIMMs start observing timing errors? Second, how many cells experience activation errors at each latency reduction step?

4.2 FPGA Test for Activation Latency

To characterize activation errors across every cell in DIMMs, we need to perform an ACTIVATE and a READ on one cache line at a time since activation errors only occur in one cache line per activation. To achieve this, we use Test 3, whose pseudocode is below, for every cache line within a row.

```
1 tRCDColOrderTest(my_tRCD, data)
2   for b ← 1 to Bank_MAX
3     for c ← 1 to Col_MAX                          ▷ Column first
4       for r ← 1 to Row_MAX
5         WriteOneCacheLine(b, r, c, data)
6         ReadOneCacheLine(tRCD, tRP, b, r, c)
7         assert findErrors() == 0                  ▷ Verify data
8         ReadOneCacheLine(my_tRCD, tRP, b, r, c)
9         findErrors()          ▷ Count errors in a cache line
```

Test 3: Read each cache line with a specified tRCD value.

The test iterates through each cache line (lines 2-4) and performs the following steps to test the cache line's reliability under a reduced tRCD value. First, it opens the row that contains the target cache line, writes a specified data pattern into the cache line, and then precharges the bank (line 5). Second, the test re-opens the row to read the cache line with the standard tRCD (line 6), and verifies if the value was written properly (line 7). Then it precharges the bank again to prepare for the next ACTIVATE. Third, it re-activates the row using the reduced tRCD value (*my_tRCD* in Test 3) to read the target cache line (line 8). It records the number of timing errors (i.e., bit flips) out of the 64-byte (512-bit) cache line (line 9).

In total, we have conducted more than 7500 rounds of tests on the DIMMs shown in Table 1, accounting for at least 2500 testing hours. For each round of tests, we conducted Test 3 with a different tRCD value and data pattern. We tested five different tRCD values: 12.5ns, 10ns, 7.5ns, 5ns, and 2.5ns. Due to the slow clock frequency of the FPGA, we can only adjust timings at a 2.5ns granularity. We used a set of four different data patterns: `0x00`, `0xaa`, `0xcc`, and `0xff`. Each data pattern represents the value that was written into each byte of the entire cache line.

4.3 Activation Error Distribution

In this section, we first present the distribution of activation errors collected from all of the tests conducted on every DIMM. Then, we categorize the results by DIMM model to investigate variation across models from different vendors.

4.3.1 Total Bit Error Rates

Figure 5 shows the box plots of the *bit error rate* (BER) observed on every DIMM as tRCD varies. The BER is defined as the fraction of activation error bits in the total population of tested bits. For each box, the bottom, middle, and top lines indicate the 25th, 50th, and 75th percentile of the population. The ends of the whiskers indicate the minimum and maximum BER of all DIMMs for a given tRCD value. Note that the y-axis is in log scale to show low BER values. As a result, the bottom whisker at tRCD=7.5ns cannot be seen due to a minimum value of 0. In addition, we show *all* observation points for each specific tRCD value by overlaying them on top of their corresponding box. Each point shows a BER collected from one round of Test 3 on one DIMM with a specific data pattern and a tRCD value. Based on these results, we make several observations.

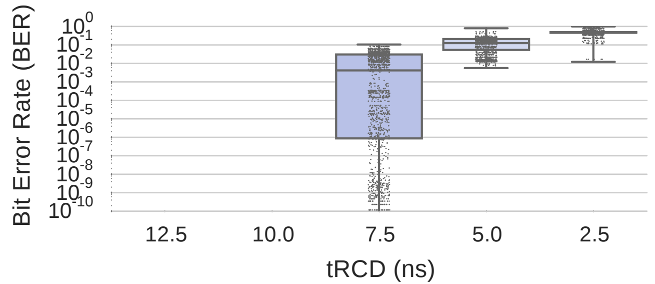


Figure 5: Bit error rate of all DIMMs with reduced tRCD.

First, we observe that BER exponentially increases as tRCD decreases. With a lower tRCD, fewer sense amplifiers are expected to have enough strength to properly sense the bitline's voltage value and latch the correct data. Second, at tRCD values of 12.5ns and 10ns, we observe no activation errors on any DIMM. This shows that the tRCD latency of the slowest cells in our tested DIMMs likely falls between 7.5 and 10ns, which are lower than the standard value (13.125ns). The manufacturers use the extra latency as a *guardband* to provide additional protection against process variation.

Third, the BER variation among DIMMs becomes smaller as tRCD value decreases. The reliability of DIMMs operating at tRCD=7.5ns varies significantly depending on the DRAM models and vendors, as we demonstrate in the Section 4.3.2. In fact, some DIMMs have no errors at tRCD=7.5ns, which cannot be seen in the plot due to the log scale. When tRCD reaches 2.5ns, most DIMMs become rife with errors, with a median BER of 0.48, similar to the probability of a coin toss.

4.3.2 Bit Error Rates by DIMM Model

Since the performance of a DIMM can vary across different models, vendors, and fabrication processes, we provide a detailed analysis by breaking down the BER results by DIMM model (listed in Table 1). Figure 6 presents the distribution of every DIMM's BER grouped by each vendor and model combination. Each box shows the quartiles and median, along with the whiskers indicating the minimum and maximum BERs. Since all of the DIMMs work reliably at 10ns and above, we show the BERs for tRCD=7.5ns and tRCD=5ns.

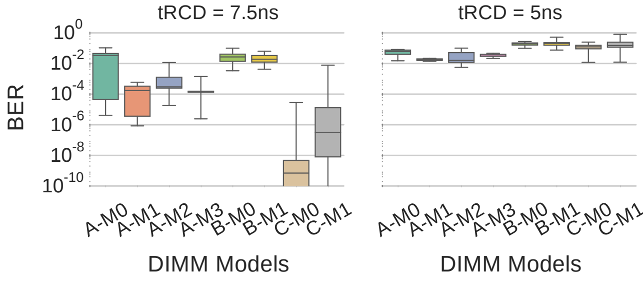


Figure 6: BERs of DIMMs grouped by model, when tested with different tRCD values.

By comparing the BERs across models and vendors, we observe that BER variation exists not only across DIMMs from different vendors, but also on DIMMs manufactured from the same vendor. For example, for DIMMs manufactured by vendor C, *Model 0* DIMMs have fewer errors than *Model 1* DIMMs. This result suggests that different DRAM models have different circuit architectures or process technologies, causing latency variation between them.

Similar to the observation we made across different DIMM models, we observe variation across DIMMs that have the same model. Due to space constraints, we omit figures to demonstrate this variation, but all of our results are available online [10]. The variation across DIMMs with the same model can be attributed to process variation due to the imperfect manufacturing process [5, 41, 55, 57].

4.4 Impact of Data Pattern

In this section, we investigate the impact of reading different data patterns under different tRCD values. Figure 7 shows the average BER of test rounds for three representative DIMMs, one from each vendor, with four data patterns. We do not show the BERs at tRCD=2.5ns, as rows cannot be reliably activated at that latency. We observe that pattern `0x00` is susceptible to more errors than pattern `0xff`, while the BERs for patterns `0xaa` and `0xcc` lie in between.[1] This can be clearly seen on D_C^0, where we observe that `0xff` incurs 4 orders of magnitude fewer errors than `0x00` on average at tRCD=7.5ns. We make a similar observation for the rest of the 12 DIMMs from vendor C.

With patterns `0xaa` and `0xcc`, we observe that bit `0` is more likely to be misread than bit `1`. In particular, we examined the flipped bits on three DIMMs that share the same model as D_C^0, and observed that *all of the flipped bits* are due to bit `0` flipping to `1`. From this observation, we can infer that there is a bias towards bit `1`, which can be more reliably read under a shorter activation latency than bit `0`.

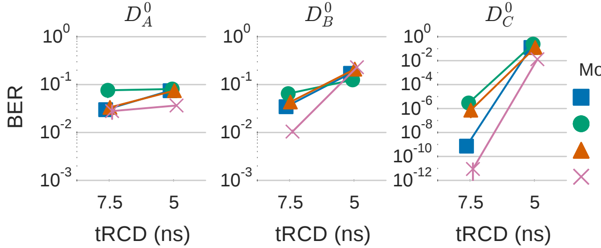


Figure 7: BERs due to four different data patterns on three different DIMMs as tRCD varies.

[1] In a cache line, we write the 8-bit pattern to every byte.

We believe this bias is due to the sense amplifier design. One major DRAM vendor presents a circuit design for a contemporary sense amplifier, and observes that it senses the V_{DD} value on the bitline faster than 0V [42]. Hence, the sense amplifier is able to sense and latch bit `1` faster than `0`. Due to this pattern dependence, we believe that it is promising to investigate asymmetric data encoding or error correction mechanisms that favor `1`s over `0`s.

Observation 3: *Errors caused by reduced activation latency are dependent on the stored data pattern. Reading bit `1` is significantly more reliable than bit `0` at reduced activation latencies.*

4.5 Effect of Temperature

Temperature is an important external factor that may affect the reliability of DIMMs [12, 29, 43, 63]. In particular, Schroeder et al. [63] and El-Sayed et al. [12] do not observe clear evidence for increasing DRAM error rates with increased temperature in data centers. Other works find that data retention time strongly depends on temperature [29, 43, 59]. However, none of these works have studied the effect of temperature on DIMMs when they are operating with a lower activation latency.

To investigate the impact of temperature on DIMMs operating with an activation latency lower than the standard value, we perform experiments that adjust the *ambient temperature* using a closed-loop temperature controller (shown in Figure 4). Figure 8 shows the average BER of three example DIMMs under three temperatures: 20°C, 50°C, and 70°C for tRCD=7.5/5ns. We include error bars, which are computed using 95% confidence intervals.

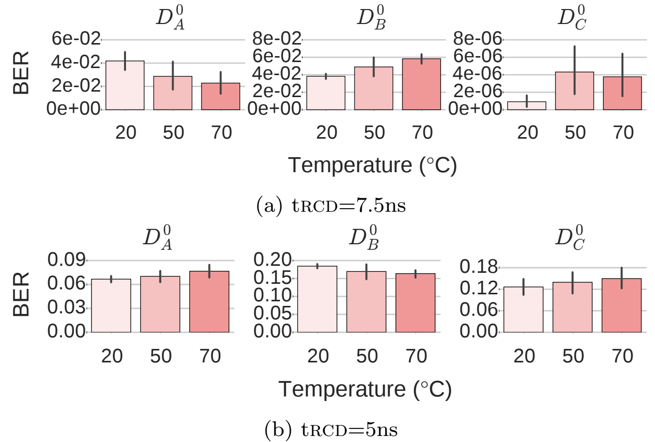


Figure 8: BERs of three example DIMMs operating under different temperatures.

We make two observations. First, at tRCD=7.5ns (Figure 8a), every DIMM shows a different BER trend as temperature increases. By calculating the *p-value* between the BERs of different temperatures, we find that the change in BERs is not statistically significant from one temperature to another for two out of the three tested DIMMs, meaning that we cannot conclude that BER increases at higher temperatures. For instance, the p-values between the BERs at 20°C and 50°C for D_A^0, D_B^0, and D_C^0 are 0.084, 0.087, and 0.006, respectively. Two of the three DIMMs have p-values greater than an α of 0.05, meaning that the BER change is statistically insignificant. Second, at lower tRCD values (5ns), the

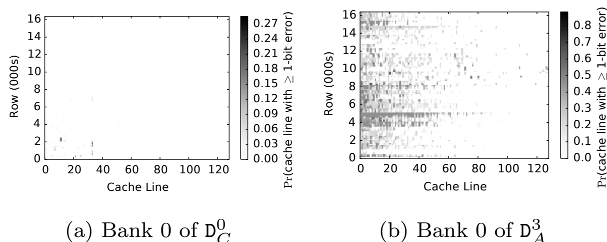
difference between the BERs due to temperature becomes even smaller.

Observation 4: *Our study does not show enough evidence to conclude that activation errors increase with higher temperatures.*

4.6 Spatial Locality of Activation Errors

To understand the locations of activation errors within a DIMM, we show the probability of experiencing at least one bit error in each cache line over a large number of experimental runs. Due to limited space, we present the results of two representative DIMMs from our experiments.

Figure 9 shows the locations of activation errors in the first bank of two DIMMs using tRCD=7.5ns. Additional results showing the error locations in every bank for some DIMMs are available online [10]. The x-axis and y-axis indicate the cache line number and row number (in thousands), respectively. In our tested DIMMs, a row size is 8KB, comprising 128 cache lines (64 bytes). Results are gathered from 40 and 52 iterations of tests for $\mathtt{D}_C^0$ and $\mathtt{D}_A^3$, respectively.



(a) Bank 0 of $\mathtt{D}_C^0$ (b) Bank 0 of $\mathtt{D}_A^3$

Figure 9: Probability of observing activation errors.

The main observation on $\mathtt{D}_C^0$ (Figure 9a) is that errors tend to cluster at certain *columns of cache lines*. For the majority of the remaining cache lines in the bank, we observe no errors throughout the experiments. We observe similar characteristics in other DIMMs from the same model. In addition, we observe clusters of errors at certain regions. For example, $\mathtt{D}_A^3$ (Figure 9b) shows that the activation errors repeatedly occur within the first half of the majority of rows.

We hypothesize that the cause of such spatial locality of errors is due to the locality of variation in the fabrication process during manufacturing: certain cache line locations can end up with less robust components, such as weaker sense amplifiers, weaker cells, or higher resistance bitlines.

Observation 5: *Activation errors do not occur uniformly within DRAM. They instead exhibit strong spatial concentration at certain regions.*

4.7 Density of Activation Errors

In this section, we investigate how errors are distributed within the erroneous cache lines. We present the distribution of error bits at the granularity of *data beats*, as conventional error-correcting codes (ECC) work at the same granularity. We discuss the effectiveness of employing ECC in Section 4.8. Recall from Section 2.3 that a cache line transfer consists of eight 64-bit data beats.

Figure 10 shows the distribution of error bits observed in each data beat of all erroneous cache lines when using tRCD=7.5ns. We show experiments from 9 DIMMs, categorized into three DIMM models (one per vendor). We select the model that observes the lowest average BER from each vendor, and show the frequency of observing 1, 2, 3, and $\geq$4 error bits in each data beat. The results are aggregated from all DIMMs of the selected models. We make two observations.

First, most data beats experience only fewer than 3 error bits at tRCD=7.5ns. We observe that more than 84%, 53%, and 91% of all the recorded activation errors are just 1-bit errors for DIMMs in A-M1, B-M1, and C-M0, respectively. Across all of the cache lines that contain at least one error bit, 82%, 41%, and 85% of the data beats that make up each cache line have no errors for A-M1, B-M1, and C-M0, respectively. Second, when tRCD is reduced to 5ns, the number of errors increases. The distribution of activation errors in data beats when using tRCD=5ns is available online [10], and it shows that 68% and 49% of data beats in A-M1 and C-M0 still have no more than one error bit.

Observation 6: *For cache lines that experience activation errors, the majority of their constituent data beats contain either no errors or just a 1-bit error.*

4.8 Effect of Error Correction Codes

As shown in the previous section, a majority of data beats in erroneous cache lines contain only a few error bits. In contemporary DRAM, ECC is used to detect and correct errors at the granularity of data beats. Therefore, this creates an opportunity for applying error correction codes (ECC) to correct activation errors. To study of the effect of ECC, we perform an analysis that uses various strengths of ECC to correct activation errors.

Figure 11 shows the percentage of cache lines that do *not* observe any activation errors when using tRCD=7.5ns at various ECC strengths, ranging from single to triple error bit correction. These results are gathered from the same 9 DIMMs used in Section 4.7. The first bar of each group is the percentage of cache lines that do not exhibit any activation errors in our experiments. The following data bars show the fraction of error-free cache lines after applying single, double, and triple error correction codes.

We make two observations. First, without any ECC support, a large fraction of cache lines can be read reliably without any errors in many of the DIMMs we study. Overall, 92% and 99% of cache lines can be read without any activation errors from A-M1 and C-M0 DIMMs, respectively. On the other hand, B-M1 DIMMs are more susceptible to reduced activation latency: only 12% of their cache lines can be read without any activation errors.

Observation 7: *A majority of cache lines can be read without any activation errors in most of our tested DIMMs. However, some DIMMs are very susceptible to activation errors, resulting in a small fraction of error-free cache lines.*

Second, ECC is very effective in correcting the activation errors. For example, with a single error correction code (1EC), which is widely deployed in many server systems, the fraction of reliable cache lines improves from 92% to 99% for A-M1 DIMMs. Even for B-M1 DIMMs, which exhibit activation errors in a large fraction of cache lines, the triple error correcting code is able to improve the percentage of error-free cache lines from 12% to 62%.

Observation 8: *ECC is an effective mechanism to correct activation errors, even in modules with a large fraction of erroneous cache lines.*

5. PRECHARGE LATENCY ANALYSIS

In this section, we present the methodology and results on varying the precharge latency, represented by the tRP

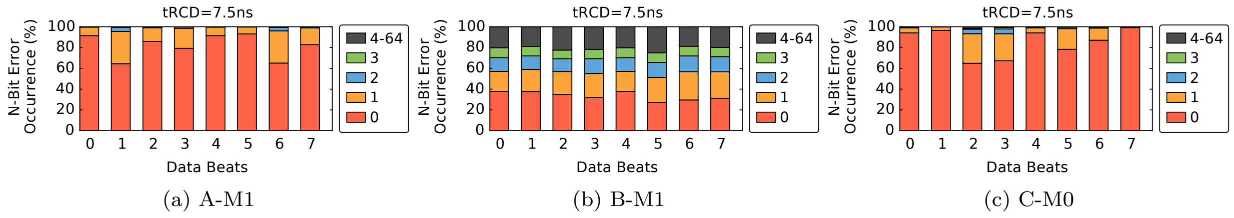


Figure 10: Breakdown of the number of error bits observed in each data beat of erroneous cache lines at tRCD=7.5ns.

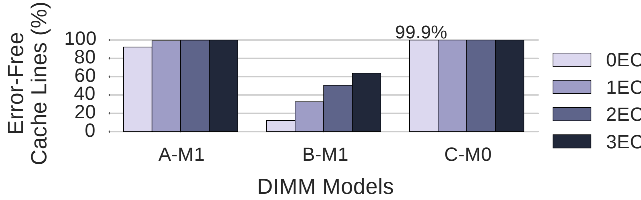


Figure 11: Percentage of error-free cache lines with various strengths of error correction (EC), with tRCD=7.5ns.

timing parameter. We first describe the nature of timing errors caused by reducing the precharge latency in Section 5.1. Then, we describe the FPGA test we conducted to characterize tRP variation in Section 5.2. In the remaining sections, we describe four major observations from our result analysis.

5.1 Behavior of Precharge Errors

In order to access a new DRAM row, a memory controller issues a PRECHARGE command, which performs the following two functions in sequence: *(i)* it closes the currently-activated row in the array (i.e., it disables the activated row's wordline); and *(ii)* it reinitializes the voltage value of every bitline inside the array back to $V_{DD}/2$, to prepare for a new activation.

Reducing the precharge latency by a small amount affects only the reinitialization process of the bitlines without interrupting the process of closing the row. The latency of this process is determined by the *precharge unit* that is placed by each bitline, next to the sense amplifier. By using a tRP value lower than the standard specification, the precharge unit may not have sufficient time to reset the bitline voltage from either V_{DD} (`bit 1`) or 0V (`bit 0`) to $V_{DD}/2$, thereby causing the bitline to float at some other intermediate voltage value. As a result, in the *subsequent* activation, the sense amplifier can incorrectly sense the wrong value from the DRAM cell due to the extra charge left on the bitline. We define precharge errors to be timing errors due to reduced precharge latency.

To further understand the nature of precharge errors, we use a test similar to the one for reduced activation latency in Section 4.1. The test reduces only the precharge latency, while keeping the activation latency at the standard value, to isolate the effects that occur due to a reduced precharge latency. We attempt to answer two fundamental questions: *(i)* Does lowering the precharge latency incur errors on multiple cache lines in the row activated *after* the precharge? *(ii)* Do these errors propagate back to the DRAM cells, causing permanent errors for all future accesses?

5.1.1 Precharge Errors Are Spread Across a Row

Throughout repeated test runs on DIMMs from all three vendors, we observe that reducing the precharge latency induces errors that are spread across multiple cache lines in the row activated after the precharge. This is because reducing the tRP value affects the latency between two *row-level* DRAM commands, PRECHARGE and ACTIVATE. As a result, having an insufficient amount of precharge time for the array's bitlines affects the entire row.

Observation 9: *Timing errors occur in multiple cache lines in the row activated after a precharge with reduced latency.*

Furthermore, these precharge errors are due to the sense amplifiers sensing the wrong voltage on the bitlines, causing them to latch incorrect data. Therefore, as the restoration operation reuses the data latched in the sense amplifiers, the wrong data is written back into the cells.

5.2 FPGA Test for Precharge Latency

In contrast to activation errors, precharge errors are spread across an *entire* row. As a result, we use a test that varies tRP at the row level. The pseudocode of the test, Test 4, is shown below.

```
1  tRPRowOrderTest(my_tRP, data)
2    for b ← 1 to Bank_MAX
3      for r ← 1 to Row_MAX                          ▷ Row order
4        WriteOneRow(b, r, data)
5        ReadOneRow(tRCD, tRP, b, r)
6        WriteOneRow(b, r + 1, data_bar)        ▷ Inverted data
7        ReadOneRow(tRCD, tRP, b, r + 1)
8        assert findErrors() == 0   ▷ Verify data, data_bar
9        ReadOneRow(tRCD, my_tRP, b, r)
10       findErrors()                      ▷ Count errors in row r
```

Test 4: Read each row with a specified tRP value.

In total, we have conducted more than 4000 rounds of tests on the DIMMs shown in Table 1, which accounts for at least 1300 testing hours. We use three groups of different data patterns: (`0x00`, `0xff`), (`0xaa`, `0x33`), and (`0xcc`, `0x55`). Each group specifies two different data patterns, which are the inverse of each other, placed in consecutive rows in the same array. This ensures that as we iterate through the rows in order, the partially-precharged state of the bitlines will not favor the data pattern in the adjacent row to be activated.

5.3 Precharge Error Distribution

In this section, we first show the distribution of precharge errors collected from all of the tests conducted on every DIMM. Then, we categorize the results by DIMM model to investigate variation across models from different vendors.

5.3.1 Total Bit Error Rates

Figure 12 shows the box plots of the BER observed for every DIMM as tRP is varied from 12.5ns down to 2.5ns. Based on these results, we make several observations. First, similar to the observation made for activation latency, we do not observe errors when the precharge latency is reduced to 12.5

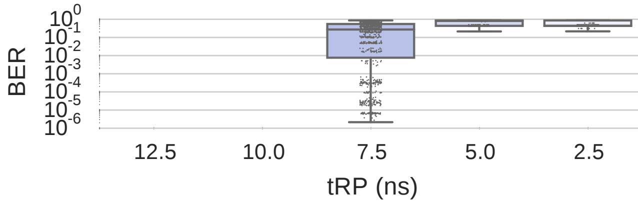


Figure 12: Bit error rate of all DIMMs with reduced tRP.

and 10ns, as the reduced latencies are still within the guardband provided. Second, the precharge BER is significantly higher than the activation BER when errors start appearing at 7.5ns – the median of the precharge BER is 587x higher than that of the activation BER (shown in Figure 5). This is partially due to the fact that reducing the precharge latency causes the errors to span across multiple cache lines in an *entire row*, whereas reducing the activation latency affects *only the first cache line* read from the row. Third, once tRP is set to 5ns, the BER exceeds the tolerable range, resulting in a median BER of 0.43. In contrast, the activation BER does not reach this high an error rate until the activation latency is lowered down to 2.5ns.

Observation 10: *With the same amount of latency reduction, the number of precharge errors is significantly higher than the number of activation errors.*

5.3.2 Bit Error Rates by DIMM Model

To examine the precharge error trend for individual DIMM models, we show the BER distribution of every DIMM categorized by DRAM model in Figure 13. Similar to the observation we made for activation errors in Section 4.1, variation exists across different DIMM models. These results provide further support for the existence and prevalence of latency variation in modern DRAM chips.

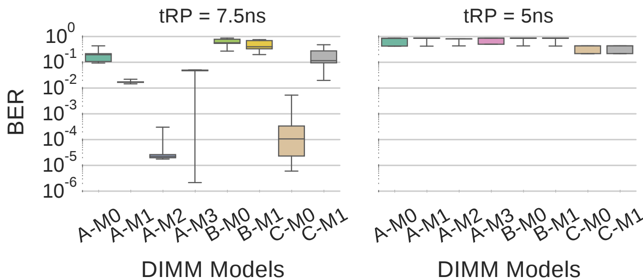
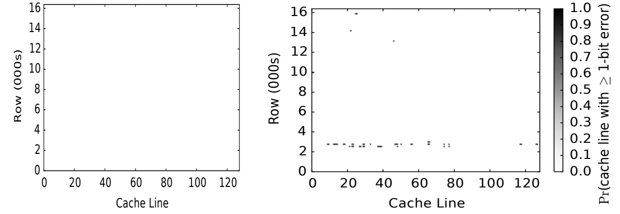


Figure 13: BERs of DIMMs grouped by model, when tested with different tRP values.

5.4 Spatial Locality of Precharge Errors

In this section, we investigate the location and distribution of precharge errors. Due to the large amount of available data, we show representative results from a single DIMM, D_C^0 (model C-M0). All of our results for all DIMMs will be made available publicly [10]. Figure 14 shows the probability of each cache line seeing at least a one-bit precharge error in Bank 0 and Bank 7 of D_C^0 when we set tRP to 7.5ns. The x-axis indicates the cache line number, and the y-axis indicates the row number (in thousands). The results are gathered from 12 iterations of tests. We make several observations based on our results.

First, some banks do not have any precharge errors throughout the experiments, such as Bank 0 (Figure 14a, hence the plot is all white). Similar to the activation er-



(a) Bank 0 of D_C^0 (b) Bank 7 of D_C^0

Figure 14: Probability of observing precharge errors.

rors, precharge errors are *not* distributed uniformly across locations within DIMMs. Second, Figure 14b shows that the errors concentrate on a certain region of rows, while the other regions experience much fewer or no errors. This demonstrates that certain sense amplifiers, or cells at certain locations are more robust than others, allowing them to work reliably under a reduced precharge latency.

Observation 11: *Precharge errors do not occur uniformly within DIMMs, but exhibit strong spatial concentration at certain regions.*

Overall, we observe that 71.1%, 13.6%, and 84.7% of cache lines contain no precharge errors when they are read from A-M1, B-M1, and C-M0 model DIMMs, respectively, with tRP=7.5ns. Similar to the trend discussed in Section 4.8, C-M0 DIMMs have the highest fraction of reliable cache lines among the DIMMs tested, while B-M1 DIMMs experience the largest amount of errors. Even though the number of error-free cache lines at tRP=7.5ns is lower than that at tRCD=7.5ns, the portion is still significant enough to show the prevalence of precharge latency variation in modern DIMMs.

Observation 12: *When precharge latency is reduced, a majority of cache lines can be read without any timing errors in some of our tested DIMMs. However, other DIMMs are largely susceptible to precharge errors, resulting in a small fraction of error-free cache lines.*

6. RESTORATION LATENCY ANALYSIS

In this section, we present a methodology and findings on varying the restoration latency, defined by the tRAS timing parameter. First, we elaborate on the impact of reducing tRAS on performance and reliability in Section 6.1. Then, we explain our FPGA test conducted to characterize tRAS variation, and present our observations.

6.1 Impact of Reduced tRAS

As mentioned in Section 2.3, tRAS specifies the minimum amount of time between issuing an ACTIVATE and a PRECHARGE command to a bank. By reducing tRAS, we can complete an access to one row faster, and quickly switch to access the next row. From the perspective of *reliability*, reducing the restoration latency may potentially induce errors in the cells due to having insufficient time to restore the lost charge back to the cells. When a row of cells is activated, the cells temporarily lose their charge to the bitlines, so that the sense amplifiers can sense the charge. During the restoration phase, the sense amplifiers restore charge back into the cells, bringing them back to the fully-charged state. By reducing the restoration latency, the amount of restored charge reduces, and the cells may not reach the fully-charged state. As a result, a subsequent access to the same row may not able to sense the correct value, thereby leading to errors.

6.2 Test Methodology and Results

To characterize the variation in restoration latency (tRAS), we consider another important factor that affects the amount of charge stored in DRAM cells, which is *leakage*. DRAM cells lose charge over time, thus requiring a periodic *refresh* operation to restore the charge. Reducing the restored charge in the cells can cause them to lose too much charge before the next refresh, generating an error.

To perform a conservative characterization, we integrate this leakage factor into our test methodology. We access each row by issuing a pair of commands, ACTIVATE and PRECHARGE, with a specific tRAS value between these two commands. Then, we wait for a full refresh period (defined as 64ms in the DRAM standard [21, 23]) before we access the row again to verify the correctness of its data. We test this sequence on a representative set of DIMMs from all three DRAM vendors and we use four data patterns: `0x00`, `0xff`, `0xaa`, and `0xcc`.

In our previously described tests on activation and precharge variation, we test every time step from the default timing value to a minimum value of 2.5ns, with a reduction of 2.5ns per step. Instead of reducing tRAS all the way down to 2.5ns from its standard value of 35ns, we lower it until $t\text{RAS}_{min} = t\text{RCD}+t\text{CL}+\text{BL}$, which is the latency of activating a row and reading a cache line from it. In a typical situation where the memory controller reads or writes a piece of data after opening a row, lowering tRAS below $t\text{RAS}_{min}$ means that the memory controller can issue a PRECHARGE while the data is still being read or written. Doing so risks terminating READ or WRITE operations prematurely, causing unknown behavior.

In order to test tRAS with a reasonable range of values, we iterate tRAS from 35ns to $t\text{RAS}_{min}$. Our $t\text{RAS}_{min}$ is calculated by using the standard tCL=13.125ns and BL=5ns along with a fast tRCD=5ns. $t\text{RAS}_{min}$ is rounded up to the nearest multiple of 2.5ns, which is *22.5ns*.

We do not observe errors across the range of tRAS values we tested in any of our experiments. This implies that charge restoration in modern DRAMs completes within the duration of an activation and a read. Therefore, tRAS can be reduced aggressively without affecting data integrity.

Observation 13: *Modern DIMMs have sufficient timing margin to complete charge restoration within the period of an* ACTIVATE *and a* READ*. Hence,* tRAS *can be reduced without introducing any errors.*

7. EXPLOITING LATENCY VARIATION

Based on our extensive experimental characterization, we propose two new mechanisms to reduce DRAM latency for better system performance. Our mechanisms exploit the key observation that different DIMMs have different amounts of tolerance for lower DRAM latency, and there is a strong correlation between the location of the cells and the lowest latency that the cells can tolerate. The first mechanism (Section 7.1) is a pure hardware approach to reducing DRAM latency. The second mechanism (Section 7.2) leverages OS support to maximize the benefits of the first mechanism.

7.1 Flexible-Latency DRAM

As we discussed in Sections 4.6 and 5.4, the timing errors caused by reducing the latency of the activation/precharge operations are concentrated on certain DRAM regions, which implies that the latency heterogeneity among DRAM cells exhibits strong locality. Based on this observation, we propose *Flexible-LatencY DRAM (FLY-DRAM)*, a software-transparent design that exploits this heterogeneity in cells to reduce the overall DRAM latency. The key idea of FLY-DRAM is to determine the shortest reliable access latency of each DRAM region, and to use the memory controller to apply that latency to the corresponding DRAM region at runtime. There are two key design challenges of FLY-DRAM, as we discuss below.

The first challenge is determining the shortest access latency. This can be done using a *latency profiling* procedure, which *(i)* runs Test 3 (Section 4.2) with different timing values and data patterns, and *(ii)* records the smallest latency that enables reliable access to each region. This procedure can be performed at one of two times. First, the system can run the procedure the very first time the DRAM is initialized, and store the profiling results to non-volatile memory (e.g., disk or flash memory) for future reference. Second, DRAM vendors can run the procedure at manufacturing time, and embed the results in the Serial Presence Detect (SPD) circuitry (a ROM present in each DIMM) [22]. The memory controller can read the profiling results from the SPD circuitry during DRAM initialization, and apply the correct latency for each DRAM region. While the second approach involves a slight modification to the DIMM, it can provide better latency information, as DRAM vendors have detailed knowledge on DRAM cell variation, and can use this information to run more thorough tests to determine a lower bound on the latency of each DRAM region.

The second design challenge is limiting the storage overhead of the latency profiling results. Recording the shortest latency for each cache line can incur a large storage overhead. For example, supporting four different tRCD and tRP timings requires 4 bits per 512-bit cache line, which is almost 0.8% of the entire DRAM storage. Fortunately, the storage overhead can be reduced based on a new observation of ours. As shown in Figures 9a and 9b, timing errors typically concentrate on certain DRAM *columns*. Therefore, FLY-DRAM records the shortest latency *at the granularity of DRAM columns*. Assuming we still need 4 bits per DRAM cache line, we need only 512 bits per DRAM bank, or an insignificant 0.00019% storage overhead for the DIMMs we evaluated. One can imagine using more sophisticated structures, such as Bloom Filters [4], to provide finer-grained latency information within a reasonable storage overhead, as shown in prior work on variable DRAM refresh time [44, 59]. We leave this for future work.

The FLY-DRAM memory controller *(i)* loads the latency profiling results into on-chip SRAMs at system boot time, *(ii)* looks up the profiled latency for each memory request based on its memory address, and *(iii)* applies the corresponding latency to the request. By reducing the latency values of tRCD, tRAS, and tRP for some memory requests, FLY-DRAM improves overall system performance, which we quantitatively demonstrate in the next two sections.

7.1.1 Evaluation Methodology

We evaluate the performance of FLY-DRAM on an eight-core system using Ramulator [32, 33], an open-source cycle-level DRAM simulator, driven by CPU traces generated from Pin [46]. We will make our source code publicly available [10]. Table 2 summarizes the configuration of our eval-

uated system. We use the standard DDR3-1333H timing parameters [21] as our baseline.

Processor	8 cores, 3.3 GHz, OoO 128-entry window
LLC	8 MB shared, 8-way set associative
DRAM	DDR3-1333H [21], open-row policy [61], 2 channels, 1 rank per channel, 8 banks per rank, Baseline: tRCD/tCL/tRP = 13.125ns, tRAS = 36ns

Table 2: Evaluated system configuration.

FLY-DRAM Configuration. To conservatively evaluate FLY-DRAM, we use a randomizing page allocator that maps each virtual page to a randomly-located physical page in memory. This allocator essentially distributes memory accesses from an application to different latency regions at random, and is thus unaware of FLY-DRAM regions.

Because each DIMM has a different fraction of fast cache lines, we evaluate FLY-DRAM on three different yet representative real DIMMs that we characterized. We select one DIMM from each vendor. Table 3 lists the distribution of cache lines that can be read reliably under different tRCD and tRP values, based on our characterization. For each DIMM, we use its distribution as listed in the table to model the percentage of cache lines with different tRCD and tRP values. For example, for D_A^2, we set 93% of its cache lines to use a tRCD of 7.5ns, and the remaining 7% of cache lines to use a tRCD of 10ns. Although these DIMMs have a small fraction of cache lines (<10%) that can be read using tRCD=5ns, we conservatively set tRCD=7.5ns for them to ensure high reliability. FLY-DRAM dynamically sets tRCD and tRP to either 7.5ns or 10ns for each memory request, based on which cache line the request is to. For the tRAS timing parameter, FLY-DRAM uses 27ns ($\lceil$tRCD+tCL$\rceil$) for all cache lines in these three tested DIMMs, as we observe no errors in any of the tested DIMMs due to lowering tRAS (see Section 6.2).

DIMM Name	Vendor	Model	tRCD Dist. (%)		tRP Dist. (%)	
			7.5ns	10ns	7.5ns	10ns
D_A^2	A	M1	93	7	74	26
D_B^7	B	M1	12	88	13	87
D_C^2	C	M0	99	1	99	1

Table 3: Distribution of cache lines under various tRCD and tRP values for three characterized DIMMs.

FLY-DRAM Upper-Bound Evaluation. We also evaluate the upper-bound performance of FLY-DRAM by assuming that *every* DRAM cell is fast (i.e., 100% of cache lines can be accessed using tRCD/tRP=7.5ns).

Applications and Workloads. To demonstrate the benefits of FLY-DRAM in an 8-core system, we generate 40 8-core multi-programmed workloads by assigning one application to each core. For each 8-core workload, we randomly select 8 applications from the following benchmark suites: SPEC CPU2006 [73], TPC-C/H [75], and STREAM [48]. We use PinPoints [58] to obtain the representative phases of each application. Our simulation executes at least 200 million instructions on each core [9, 16, 35, 38].

Performance Metric. We measure system performance with the *weighted speedup (WS)* metric [69], which is a measure of job throughput on a multi-core system [13]. Specifically, $WS = \sum_{i=1}^{N} \frac{IPC_i^{shared}}{IPC_i^{alone}}$. N is the number of cores in the system. IPC_i^{shared} is the IPC of an application that runs on $core_i$ while other applications are running on the other cores. IPC_i^{alone} is the IPC of an application when it runs alone in the system without any other applications. Essentially, WS is the sum of every application's slowdown compared to when it runs alone on the same system.

7.1.2 Multi-Core System Results

Figure 15 illustrates the system performance improvement of FLY-DRAM over the baseline for 40 workloads. The x-axis indicates the evaluated DRAM configurations, as shown in Table 3. The percentage value on top of each box is the average performance improvement over the baseline.

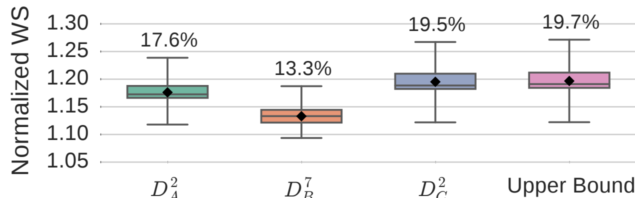


Figure 15: System performance improvement of FLY-DRAM for various DIMMs (listed in Table 3).

We make the following observations. First, FLY-DRAM improves system performance significantly, by 17.6%, 13.3%, and 19.5% on average across all 40 workloads for the three real DIMMs that we characterize. This is because FLY-DRAM reduces the latency of tRCD, tRP, and tRAS by 42.8%, 42.8%, and 25%, respectively, for many cache lines. In particular, DIMM D_C^2, whose great majority of cells are reliable at low tRCD and tRP, performs within 1% of the upper-bound performance (19.7% on average). Second, although DIMM D_B^7 has only a small fraction of cells that can operate at 7.5ns, FLY-DRAM still attains significant system performance benefits by using low tRCD and tRP latencies (10ns), which are 23.8% lower than the baseline, for the majority of cache lines. We conclude that FLY-DRAM is an effective mechanism to improve system performance by exploiting the widespread latency variation present across DRAM cells.

7.2 Discussion: DRAM-Aware Page Allocator

While FLY-DRAM significantly improves system performance in a software-transparent manner, we can take better advantage of it if we expose the different latency regions of FLY-DRAM to the software stack. We propose the idea of a DRAM-aware page allocator in the OS, whose goal is to better take advantage of FLY-DRAM by intelligently mapping application pages to different-latency DRAM regions in order to improve performance.

Within an application, there is heterogeneity in the access frequency of different pages, where some pages are accessed much more frequently than other pages [3, 60, 70, 74, 77, 83]. Our DRAM-aware page allocator places more frequently-accessed pages into lower-latency regions in DRAM. This *access frequency aware placement* allows a greater number of DRAM accesses to experience a reduced latency than a page allocator that is oblivious to DRAM latency variation, thereby likely increasing system performance.

For our page allocator to work effectively, it must know which pages are expected to be accessed frequently. In or-

der to do this, we extend the OS system calls for memory allocation to take in a Boolean value, which states whether the memory being allocated is expected to be accessed frequently. This information either can be annotated by the programmer, or can be estimated by various dynamic profiling techniques [1, 6, 26, 47, 60, 74, 77, 83]. The page allocator uses this information to find a free physical page in DRAM that suits the expected access frequency of the application page that is being allocated.

We expect that by using our proposed page allocator, FLY-DRAM can perform close to the upper-bound performance reported in Section 7.1.2, even for DIMMs that have a smaller fraction of fast regions.

8. RELATED WORK

To our knowledge, this is the first work to *(i)* provide a detailed experimental characterization and analysis of latency variation for three major DRAM operations (tRCD, tRP, and tRAS) *across different cells within a DRAM chip*, *(ii)* demonstrate that a reduction in latency for each of these fundamental operations has a different impact on slower cells, *(iii)* show that access latency variation exhibits spatial locality, *(iv)* demonstrate that the error rate due to reduced latencies is correlated with the stored data pattern but not conclusively correlated with temperature, and *(v)* propose mechanisms that take advantage of variation *within a DRAM chip* to improve system performance.

DRAM Latency Variation. Adaptive-Latency DRAM (AL-DRAM) also characterizes and exploits DRAM latency variation, but does so at a much coarser granularity [37]. This work experimentally characterizes latency variation across different DRAM chips under different operating temperatures. AL-DRAM sets a uniform operation latency for the *entire* DIMM. In contrast, our work characterizes latency variation *within each chip*, at the granularity of individual DRAM cells. Our mechanism, FLY-DRAM, can be combined with AL-DRAM to further improve performance.

Chandrasekar et al. study the potential of reducing some DRAM timing parameters [7]. Similar to AL-DRAM, this work observes and characterizes latency variation *across* DIMMs, whereas our work studies variation across cells *within a DRAM chip*.

DRAM Error Studies. There are several studies that characterize various errors in DRAM. Many of these works observe how specific factors affect DRAM errors, analyzing the impact of temperature [12, 37] and hard errors [18]. Other works have conducted studies of DRAM error rates in the field, studying failures across a large sample size [40, 50, 63, 71, 72]. There are also works that have studied errors through controlled experiments, investigating errors due to retention [28, 29, 43, 59], disturbance from neighboring DRAM cells [34], and latency variation *across DRAM chips* [7, 37]. None of these works study errors due to latency variation across the cells within a DRAM chip, which we extensively characterize in our work.

DRAM Latency Reduction. Several types of commodity DRAM (Micron's RLDRAM [52] and Fujitsu's FCRAM [62]) provide low latency at the cost of high area overhead [35, 38]. Many prior works (e.g., [8, 9, 17, 35, 38, 45, 56, 65, 66, 70, 84]) propose various architectural changes *within* DRAM chips to reduce latency. In contrast, FLY-DRAM does not require any changes to a DRAM chip. Other works [16, 36, 64, 67, 68] reduce DRAM latency by changing the memory controller, and FLY-DRAM is complementary to them.

ECC DRAM. Many memory systems incorporate ECC DIMMs, which store information used to correct data during a read operation. Prior work (e.g., [14, 25, 29, 30, 39, 76, 78, 81, 82]) proposes more flexible or more powerful ECC schemes for DRAM. While these ECC mechanisms are designed to protect against faults using standard DRAM timings, we show that they also have the potential to correct timing errors that occur due to reduced DRAM latencies.

9. CONCLUSION

This paper provides the first experimental study that comprehensively characterizes and analyzes the latency variation within modern DRAM chips for three fundamental DRAM operations (activation, precharge, and restoration). We find that significant latency variation is present across DRAM cells in all 240 of our tested DRAM chips, and that a large fraction of cache lines can be read reliably even if the activation/restoration/precharge latencies are reduced significantly. Consequently, exploiting the latency variation in DRAM cells can greatly reduce the DRAM access latency. Based on the findings from our experimental characterization, we propose and evaluate a new mechanism, FLY-DRAM (Flexible-LatencY DRAM), which reduces DRAM latency by exploiting the inherent latency variation in DRAM cells. FLY-DRAM reduces DRAM latency by categorizing the DRAM cells into fast and slow regions, and accessing the fast regions with a reduced latency. We demonstrate that FLY-DRAM can greatly reduce DRAM latency, leading to significant system performance improvements on a variety of workloads.

We conclude that it is promising to understand and exploit the inherent latency variation within modern DRAM chips. We hope that the experimental characterization, analysis, and optimization techniques presented in this paper will enable the development of other new mechanisms that exploit the latency variation within DRAM to improve system performance and perhaps reliability.

ACKNOWLEDGMENTS

We thank our shepherd Christopher Stewart, anonymous reviewers, and SAFARI group members for feedback. We acknowledge the support of Google, Intel, Nvidia, and Samsung. This research was supported in part by the ISTC-CC, SRC, and NSF (grants 1212962 and 1320531). Kevin Chang is supported in part by the SRCEA/Intel Fellowship.

References

[1] N. Agarwal *et al.*, "Page Placement Strategies for GPUs Within Heterogeneous Memory Systems," in *ASPLOS*, 2015.
[2] H. Bauer *et al.*, "Memory: Are Challenges ahead?" March 2016. Available: http://www.mckinsey.com/industries/semiconductors/our-insights/memory-are-challenges-ahead
[3] A. Bhattacharjee and M. Martonosi, "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors," in *ISCA*, 2009.
[4] B. H. Bloom, "Space/Time Tradeoffs in Hash Coding with Allowable Errors," *CACM*, July 1970.
[5] K. Chakraborty and P. Mazumder, *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories.* Prentice Hall, 2002.
[6] R. Chandra *et al.*, "Scheduling and Page Migration for Multiprocessor Compute Servers," in *ASPLOS*, 1994.
[7] K. Chandrasekar *et al.*, "Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization," in *DATE*, 2014.

[8] K. K.-W. Chang *et al.*, “Improving DRAM Performance by Parallelizing Refreshes with Accesses,” in *HPCA*, 2014.
[9] K. K. Chang *et al.*, “Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM,” in *HPCA*, 2016.
[10] CMU SAFARI Research Group Source Code Repository, https://github.com/CMU-SAFARI.
[11] J. Dean and L. A. Barroso, “The Tail at Scale,” *CACM*, 2013.
[12] N. El-Sayed *et al.*, “Temperature Management in Data Centers: Why Some (Might) Like It Hot,” in *SIGMETRICS*, 2012.
[13] S. Eyerman and L. Eeckhout, “System-Level Performance Metrics for Multiprogram Workloads,” *IEEE Micro*, 2008.
[14] S.-L. Gong *et al.*, “CLEAN-ECC: High Reliability ECC for Adaptive Granularity Memory System,” in *MICRO*, 2015.
[15] T. Hamamoto *et al.*, “On the Retention Time Distribution of Dynamic Random Access Memory (DRAM),” in *IEEE TED*, 1998.
[16] H. Hassan *et al.*, “ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality,” in *HPCA*, 2016.
[17] H. Hidaka *et al.*, “The Cache DRAM Architecture,” *IEEE Micro*, 1990.
[18] A. A. Hwang *et al.*, “Cosmic Rays Don’t Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design,” in *ASPLOS*, 2012.
[19] M. Jacobsen *et al.*, “RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators,” *RTS*, 2015.
[20] JEDEC, “DDR2 SDRAM Standard,” 2009.
[21] JEDEC, “DDR3 SDRAM Standard,” 2010.
[22] JEDEC, “Standard No. 21-C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules,” 2011.
[23] JEDEC, “DDR4 SDRAM Standard,” 2012.
[24] JEDEC, “Low Power Double Data Rate 3 (LPDDR3),” 2012.
[25] X. Jian *et al.*, “Low-Power, Low-Storage-Overhead Chipkill Correct via Multi-Line Error Correction,” in *SC*, 2013.
[26] X. Jiang *et al.*, “CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms,” in *HPCA*, 2010.
[27] S. Kanev *et al.*, “Profiling a Warehouse-Scale Computer,” in *ISCA*, 2015.
[28] S. Khan *et al.*, “PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM,” in *DSN*, 2016.
[29] S. Khan *et al.*, “The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study,” in *SIGMETRICS*, 2014.
[30] J. Kim *et al.*, “Bamboo ECC: Strong, Safe, and Flexible Codes for Reliable Computer Memory,” in *HPCA*, 2015.
[31] K. Kim and J. Lee, “A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs,” in *EDL*, 2009.
[32] Y. Kim *et al.*, “Ramulator,” https://github.com/CMU-SAFARI/ramulator.
[33] Y. Kim *et al.*, “Ramulator: A Fast and Extensible DRAM Simulator,” *IEEE CAL*, 2015.
[34] Y. Kim *et al.*, “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in *ISCA*, 2014.
[35] Y. Kim *et al.*, “A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM,” in *ISCA*, 2012.
[36] C. J. Lee *et al.*, “DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems,” in *HPS Technical Report*, 2010.
[37] D. Lee *et al.*, “Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case,” in *HPCA*, 2015.
[38] D. Lee *et al.*, “Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture,” in *HPCA*, 2013.
[39] S. Li *et al.*, “MAGE: Adaptive Granularity and ECC for Resilient and Power Efficient Memory Systems,” in *SC*, 2012.
[40] X. Li *et al.*, “A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility,” in *USENIX ATC*, 2010.
[41] Y. Li *et al.*, “DRAM Yield Analysis and Optimization by a Statistical Design Approach,” in *IEEE TCSI*, 2011.
[42] K.-N. Lim *et al.*, “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.
[43] J. Liu *et al.*, “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms,” in *ISCA*, 2013.
[44] J. Liu *et al.*, “RAIDR: Retention-Aware Intelligent DRAM Refresh,” in *ISCA*, 2012.
[45] S.-L. Lu *et al.*, “Improving DRAM Latency with Dynamic Asymmetric Subarray,” in *MICRO*, 2015.
[46] C.-K. Luk *et al.*, “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation,” in *PLDI*, 2005.
[47] J. Marathe and F. Mueller, “Hardware Profile-Guided Automatic Page Placement for ccNUMA Systems,” in *PPoPP*, 2006.
[48] J. D. McCalpin, “STREAM Benchmark.”
[49] S. A. McKee, “Reflections on the Memory Wall,” in *CF*, 2004.
[50] J. Meza *et al.*, “Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field,” in *DSN*, 2015.
[51] Micron Technology, Inc., “128Mb: x4, x8, x16 Automotive SDRAM,” 1999.
[52] Micron Technology, Inc., “576Mb: x18, x36 RLDRAM3,” 2011.
[53] O. Mutlu, “Memory Scaling: A Systems Architecture Perspective,” *IMW*, 2013.
[54] O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” *SUPERFRI*, 2015.
[55] S. Nassif, “Delay Variability: Sources, Impacts and Trends,” in *ISSCC*, 2000.
[56] S. O *et al.*, “Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture,” in *ISCA*, 2014.
[57] M. Onabajo and J. Silva-Martinez, *Analog Circuit Design for Process Variation-Resilient Systems-on-a-Chip*. Springer, 2012.
[58] H. Patil *et al.*, “Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation,” in *MICRO*, 2004.
[59] M. K. Qureshi *et al.*, “AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems,” in *DSN*, 2015.
[60] L. E. Ramos *et al.*, “Page Placement in Hybrid Memory Systems,” in *ICS*, 2011.
[61] S. Rixner *et al.*, “Memory Access Scheduling,” in *ISCA*, 2000.
[62] Y. Sato *et al.*, “Fast Cycle RAM (FCRAM): A 20-ns Random Row Access, Pipe-Lined Operating DRAM,” in *VLSIC*, 1998.
[63] B. Schroeder *et al.*, “DRAM Errors in the Wild: A Large-Scale Field Study,” in *SIGMETRICS*, 2009.
[64] V. Seshadri *et al.*, “The Dirty-Block Index,” in *ISCA*, 2014.
[65] V. Seshadri *et al.*, “Fast Bulk Bitwise AND and OR in DRAM,” *IEEE CAL*, 2015.
[66] V. Seshadri *et al.*, “RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization,” in *MICRO*, 2013.
[67] V. Seshadri *et al.*, “Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses,” in *MICRO*, 2015.
[68] W. Shin *et al.*, “NUAT: A Non-Uniform Access Time Memory Controller,” in *HPCA*, 2014.
[69] A. Snavely and D. Tullsen, “Symbiotic Jobscheduling for a Simultaneous Multithreading Processor,” in *ASPLOS*, 2000.
[70] Y. H. Son *et al.*, “Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations,” in *ISCA*, 2013.
[71] V. Sridharan *et al.*, “Memory Errors in Modern Systems: The Good, the Bad, and the Ugly,” in *ASPLOS*, 2015.
[72] V. Sridharan and D. Liberty, “A Study of DRAM Failures in the Field,” in *SC*, 2012.
[73] Standard Performance Evaluation Corp., “SPEC CPU2006,” http://www.spec.org/cpu2006.
[74] K. Sudan *et al.*, “Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement,” in *ASPLOS*, 2010.
[75] Transaction Performance Processing Council, “TPC Benchmarks,” http://www.tpc.org/.
[76] A. N. Udipi *et al.*, “LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems,” in *ISCA*, 2012.
[77] B. Verghese *et al.*, “Operating System Support for Improving Data Locality on CC-NUMA Compute Servers,” in *ASPLOS*, 1996.
[78] C. Wilkerson *et al.*, “Reducing Cache Power with Low-cost, Multi-bit Error-correcting Codes,” in *ISCA*, 2010.
[79] M. V. Wilkes, “The Memory Gap and the Future of High Performance Memories,” *SIGARCH CAN*, 2001.
[80] Xilinx, “ML605 Hardware User Guide,” Oct. 2012.
[81] D. H. Yoon *et al.*, “BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs,” in *ISCA*, 2012.
[82] D. H. Yoon and M. Erez, “Virtualized ECC: Flexible Reliability in Main Memory,” in *ASPLOS*, 2010.
[83] H. Yoon *et al.*, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” in *ICCD*, 2012.
[84] T. Zhang *et al.*, “Half-DRAM: A High-Bandwidth and Low-Power DRAM Architecture from the Rethinking of Fine-Grained Activation,” in *ISCA*, 2014.