The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions

Minesh Patel[§‡] Jeremie S. Kim[‡§] Onur Mutlu[§‡]
[§]ETH Zürich [‡]Carnegie Mellon University

ABSTRACT

Modern DRAM-based systems suffer from significant energy and latency penalties due to conservative DRAM refresh standards. Volatile DRAM cells can retain information across a wide distribution of times ranging from milliseconds to many minutes, but each cell is currently refreshed every 64ms to account for the extreme tail end of the retention time distribution, leading to a high refresh overhead. Due to poor DRAM technology scaling, this problem is expected to get worse in future device generations. Hence, the current approach of refreshing all cells with the worst-case refresh rate must be replaced with a more intelligent design.

Many prior works propose reducing the refresh overhead by extending the default refresh interval to a higher value, which we refer to as the *target refresh interval*, across parts or all of a DRAM chip. These proposals handle the small set of failing cells that cannot retain data throughout the entire extended refresh interval via *retention failure mitigation mechanisms* (e.g., error correcting codes or bit-repair mechanisms). This set of failing cells is discovered via *retention failure profiling*, which is currently a brute-force process that writes a set of known data to DRAM, disables refresh and waits for the duration of the target refresh interval, and then checks for retention failures across the DRAM chip. We show that this brute-force approach is too slow and is detrimental to system execution, especially with frequent online profiling.

This paper presents *reach profiling*, a new methodology for retention failure profiling based on the key observation that an overwhelming majority of failing DRAM cells at a target refresh interval fail more reliably at both *longer refresh intervals* and *higher temperatures*. Using 368 state-of-the-art LPDDR4 DRAM chips from three major vendors, we conduct a thorough experimental characterization of the complex set of tradeoffs inherent in the profiling process. We identify three key metrics to guide design choices for retention failure profiling and mitigation mechanisms: *coverage*, *false positive rate*, and *runtime*. We propose reach profiling, a new retention failure profiling mechanism whose key idea is to profile failing cells at a longer refresh interval and/or higher temperature relative to the target conditions in order to maximize failure coverage while minimizing the false positive rate and profiling runtime. We thoroughly explore the tradeoffs associated with reach profiling and show that there is significant room for improvement in DRAM retention failure profiling beyond the brute-force approach. We show with experimental data that on average, by profiling at 250ms above the target refresh interval, our first implementation of reach profiling (called REAPER) can attain greater than 99% coverage of failing DRAM cells with less than a 50% false positive rate while running 2.5x faster than the brute-force approach. In addition, our end-to-end evaluations show that REAPER enables significant system performance improvement and DRAM power reduction, outperforming the brute-force approach and enabling high-performance operation at longer refresh intervals that were previously unreasonable to employ due to the high associated profiling overhead.

ISCA '17, Toronto, ON, Canada

DOI: http://dx.doi.org/10.1145/3079856.3080242

KEYWORDS

DRAM, refresh, retention failures, reliability, testing, memory

1 INTRODUCTION

DRAM stores data in volatile capacitors that constantly leak charge and therefore requires periodic charge restoration to maintain data correctness. As cell capacitor sizes decrease with process scaling and the total number of cells per chip increases each device generation [36], the total amount of time and energy required to restore all cells to their correct value, a process known as DRAM *refresh*, scales unfavorably [23, 41, 63]. The periodic refresh of DRAM cell capacitors consumes up to 50% of total DRAM power [63] and incurs large performance penalties as DRAM cells are unavailable during refresh [23, 63, 73, 75].

The DRAM refresh rate, dictated by the *refresh interval* or t_{REFI}, is a standardized constant to ensure interoperability of devices from different vendors and across product generations. DRAM chips are designed to operate reliably at this refresh rate under worst-case operating conditions. However, it is well known that DRAM cells exhibit large variations in charge retention time [42, 47, 54, 62, 63, 81]. Therefore, a fixed refresh interval causes all DRAM cells to be refreshed at the worst-case rate even though most DRAM cells can hold data for much longer. Prior works explore increasing the default refresh interval to what we call a *target refresh interval* while maintaining correctness of DRAM operation. These works assume that a *small finite set* of *failing cells* (due to the extended refresh interval) are handled with various *retention failure mitigation mechanisms* (e.g., ECC, more frequent refreshes, bit repair mechanisms) [6, 7, 26, 42, 61–63, 76, 81, 95–97].

Many of these works that extend the refresh interval assume that the set of failing cells can be quickly and efficiently identified using a *brute-force* retention failure *profiling mechanism*, which involves writing known data to DRAM, waiting for the duration of the required target refresh interval, and reading the data back out to check for errors, for every row of cells in a DRAM chip. However, these works do *not* rigorously explore the efficacy and reliability of such a mechanism. Effects such as variable retention time (VRT) [41, 42, 62, 72, 81, 82, 101] and data pattern dependence (DPD) [42–44, 49, 59, 60, 62] invalidate the assumption that a small *finite* set of failing cells exists and therefore complicate the brute-force approach, likely requiring *efficient online profiling* mechanisms to discover a *continuously-changing* set of failing cells. To this end, **our goal** is to (1) thoroughly analyze the different tradeoffs inherent to retention failure profiling via experimental analysis of a large number of real state-of-the-art LPDDR4 DRAM chips and (2) use our observations to develop a robust retention failure profiling mechanism that identifies an overwhelming majority of all possible failing cells at a given target refresh interval within a short time.

We identify three key properties that any effective retention failure profiling mechanism should have. First, for a given target refresh interval, profiling should achieve high *coverage*, i.e., the ratio of the number of failing cells discovered by the profiling mechanism to the number of all possible failing cells at the target refresh interval. This is required to minimize the cost of the mitigation mechanism for the failing cells at the target refresh interval. Second, profiling should result in only a small number of *false positives*, i.e., cells that are observed to fail during profiling but never during actual operation at the target refresh interval. This is beneficial to minimize the necessary work done by, and thus the overhead of, the retention

failure mitigation mechanism. Third, the profiling *runtime* should be as short as possible to minimize the performance impact of profiling on the system.

To understand the complex tradeoffs inherent in DRAM retention failure profiling with a focus on these three metrics, we conduct a thorough experimental analysis of 368 state-of-the-art LPDDR4 DRAM chips across three major DRAM vendors. Our rigorous experimental analysis yields three new major observations. First, the brute-force approach to profiling is inefficient and does not attain high coverage within a short time. Second, the cells that fail at a given refresh interval fail more reliably at a higher refresh interval and at a higher temperature. Third, retention failure profiling has a complex tradeoff space between three key parameters (coverage, false positive rate, and runtime).

Based on our observations from this rigorous experimental characterization of the complex tradeoff space, we propose a novel low-overhead and high-coverage DRAM retention failure profiling methodology called *reach profiling*. The key idea of reach profiling is to profile at *reach conditions*, which consist of either a larger refresh interval and/or a higher temperature relative to the target refresh interval/temperature to quickly discover an overwhelming majority of all possible failing cells at the target conditions. Intuitively, given that failing cells are more likely to fail at a higher refresh interval and at a higher temperature, we can quickly obtain all failing cells for a target condition by profiling at reach conditions, at the cost of also identifying some false positive cells as failing. The profiling parameters can be adjusted to *maximize* coverage while *minimizing* the false positive rate and runtime.

We find that on average, across 368 different DRAM chips from three vendors, reach profiling attains a speedup of 2.5x over brute-force profiling while providing over 99% coverage of failing cells with less than a 50% false positive rate. Further speedups of up to 3.5x can be obtained at the expense of significantly greater false positive rates. By manipulating the profiling conditions relative to the target conditions, we can select an appropriate tradeoff between coverage, false positive rate, and runtime that suits the desired system configuration.

Reach profiling results in a set of failing cells that can be handled by various retention failure mitigation mechanisms (e.g., error correcting codes, higher refresh rates, and remapping mechanisms for failing cells) to reliably operate a system at a target refresh rate. As an example, consider a scheme where the DRAM memory controller maps addresses with failing cells out of the system address space. For any target refresh interval, reach profiling can be used to quickly determine the set of failing cells with high coverage. The memory controller can then map out the addresses containing the failed cells from the system address space in order to maintain system reliability at a higher refresh interval. Alternatively, the system can employ any other error mitigation mechanism proposed by prior work to handle the failing cells (e.g., [6, 7, 26, 42, 61–63, 76, 81, 95, 97]). For example, when used with ArchShield [76], REAPER reliably enables an average end-to-end system performance improvement of 12.5% (maximum 23.7%) on our workloads (Section 7.3.2).

This paper makes the following major **contributions**:

- It is the first work to 1) experimentally characterize data retention behavior and retention failures for state-of-the-art LPDDR4 DRAM chips (Section 5) and 2) demonstrate the complex tradeoff space between the three key metrics associated with DRAM retention failure profiling: coverage, false positive rate, and profiling runtime (Section 6).
- It proposes *reach profiling*, a novel DRAM cell retention failure profiling technique that efficiently and reliably identifies cells with a high likelihood to fail at a target refresh interval. The key idea of reach profiling is to profile at a longer refresh interval and/or a higher temperature relative to the target refresh interval/temperature in order to quickly discover an overwhelming majority of all possible failing cells at the target conditions. Reach profiling enables a wide range of previously proposed refresh reduction techniques [61, 63, 76, 79, 95, 96] (Section 3).
- It experimentally shows, with analysis of 368 LPDDR4 DRAM chips at many different operating conditions, that reach profiling is effective. We experimentally demonstrate that by profiling at 250ms above the target refresh interval, reach profiling can provide greater than 99% coverage of failing cells with less than a 50% false positive rate, while running 2.5x faster than the state-of-the-art brute-force profiling mechanism.
- It shows that DRAM cells *cannot* easily be classified as "weak" or "strong", based on a rigorous analysis of cell retention time distribution and individual cell retention failure data. We use this analysis and data to provide a theoretical basis for the effectiveness of reach profiling (Section 5.5).
- It presents a formal analysis of tolerable raw bit error rates (RBERs) based on using ECC to better understand profiling requirements. We use this analysis to present a new model for the minimum length of time that a retention profile remains valid. For example, given a system with a desired uncorrectable bit error rate (UBER) of 10^{-15} and a target refresh interval of 1024ms operating at 45°C, reach profiling results in a profile with 99% coverage that is valid for approximately 2.3 days (Section 6.2.2).
- It provides an implementation of reach profiling, called *REAPER*, which enables significant end-to-end system performance improvement and DRAM power reduction. REAPER outperforms the brute-force approach and enables high-performance operation at longer refresh intervals that were previously unreasonable to employ due to the high associated profiling overhead (Section 7).

2 BACKGROUND

We discuss the necessary background material required for understanding our experimental investigation and new profiling mechanism. For further detail on fundamental DRAM operation, we refer the reader to [21–25, 32, 48, 52, 54–58, 62, 63, 87–90].

2.1 DRAM Organization

DRAM can be thought of as a hierarchy of 2-dimensional cell arrays. Figure 1 illustrates the basic DRAM array structure and function. Figure 1(a) depicts a DRAM array along with peripheral circuitry used for *i*) selecting a row to read (*row decoder*) and *ii*) reading out the data in the selected row (*sense amplifiers*). Each row of cells is connected with a wire called the *wordline*, and each column of cells with a wire called the *bitline*. Each cell, shown in Figure 1(b), encodes one bit of data in a capacitor using logic "high" (V_{dd}) to represent one binary value and logic "low" ($0V$) for its inverse.

During an access, the row decoder asserts a single wordline, connecting an entire *row* of cells to their respective bitlines and sense amplifiers. Each bitline is initially charged to $\frac{V_{dd}}{2}$ and connection with the cell capacitor results in a shift in bitline voltage towards either V_{dd} or $0V$. Each sense amplifier resolves the directional shift in voltage on its bitline to determine the original value stored in the cell. After resolving the shift, the data has been read out of the row of cells and can be accessed by the system and written back into the cells via the sense amplifiers.

2.2 DRAM Refresh

Data stored in DRAM cell capacitors is volatile as a result of a combination of various charge leakage mechanisms [62, 84]. Over time, the data stored in a DRAM cell degrades and is eventually lost. This is called a *retention failure*. In order to prevent a retention failure, cell data must be periodically restored with a process called DRAM *refresh*. The JEDEC specification [37] defines the time interval between refresh commands, t_{REFI}, to be 64ms for typical operating conditions and 32ms at above 85°C.

The refresh operation relies on using the sense amplifiers to read out a row of data and write it back into the cells in order to restore the

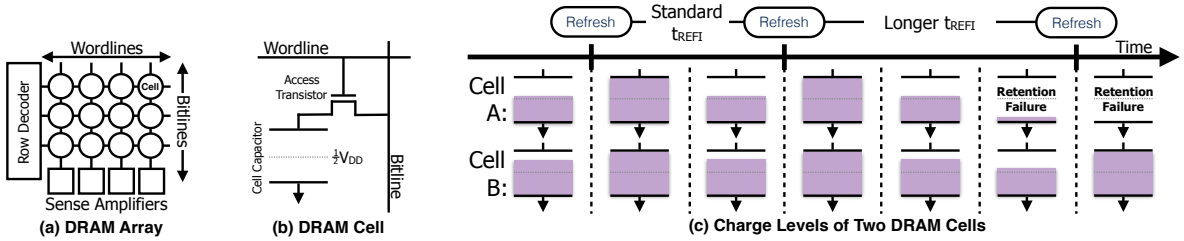

Figure 1: (a) A 2-D DRAM array composed of cells. (b) A DRAM cell consisting of a capacitor and transistor. (c) Effect of extending the refresh interval on two cells with different leakage rates (high-leakage cell A and low-leakage cell B).

original fully charged/discharged states of cells. However, refreshing a cell is conditional on a correct read, so if a cell's logical state has already changed by the time a refresh operation is performed, refresh could simply restore the incorrectly-read data. Figure 1(c) demonstrates the charge levels of two DRAM cells, A and B, with different leakage rates, when they are both refreshed at two different refresh intervals. While both cells retain data correctly with the standard refresh interval, cell A is unable to retain data with a *longer* refresh interval.

Prior works show that due to process variation and other effects discussed in Section 2.3, cells can hold data for a wide distribution of *retention times* ranging from milliseconds to minutes, and only a small percentage of cells, called *worst-case cells*, actually require the JEDEC-specified 64ms refresh interval [33, 42, 47, 54, 60, 62, 63, 81, 95]. However, determining the set of cells that require a given refresh interval is a nontrivial process, so each cell is simply refreshed at the rate required by the worst-case cells to guarantee correct operation across all of DRAM [63]. As a result, the memory controller is responsible for ensuring that each DRAM row is refreshed every t_{REFI}, which in turn results in considerable overhead in both energy consumption and performance due to extra DRAM accesses and in-progress row refresh operations that spend energy and delay program-issued DRAM requests [21, 23, 63, 74, 81, 95].

2.3 DRAM Retention Phenomena

In addition to the inherent process variation from manufacturing, two major phenomena affect the data retention behavior of a DRAM cell: 1) *variable retention time* (VRT) [41, 42, 62, 72, 81, 82, 101], and 2) *data pattern dependence* (DPD) [42–44, 49, 59, 60, 62]. These phenomena *dynamically* change the effective retention time of each cell, which complicates the identification of the set of all cells that fail under a longer refresh interval. We discuss these phenomena in detail in this section in order to demonstrate the difficulty of the retention failure identification problem.

2.3.1 Variable Retention Time (VRT). VRT is the phenomenon where a DRAM cell's retention characteristics alternate between two or more different retention time values. This phenomenon has been demonstrated to be *ubiquitous* and *unpredictable* [41, 42, 62, 72, 81, 82, 101]. A VRT cell stays in a given retention time state for an unpredictable length of time, and this time period is based on a memoryless random process that changes with supply voltage and ambient temperature [101]. This results in a dynamically-changing set of cells that experience retention failures at a given target condition (i.e., the more time passes after determining the profile of failing cells at a target condition, the more the profile coverage degrades).

2.3.2 Data Pattern Dependence (DPD). The retention time of a cell has been shown to depend on the data values stored both in the cell itself and the data stored in its neighboring cells [42–44, 49, 59, 60, 62]. DPD effects occur due to coupling between adjacent circuit elements, exacerbating charge leakage in affected cells and resulting in difficult-to-predict retention times [43, 59, 62]. This means that different data patterns (DPs) stored in DRAM induce retention failures at different refresh intervals for each cell. Operating conditions, architectural design, and process variation all change the DPD effects, further complicating the problem.

3 REFRESH OVERHEAD MITIGATION

Given the ability to perfectly predict each DRAM cell's charge retention behavior, we would ideally refresh each cell only when it is about to fail, wasting no cycles for issuing unnecessary refresh operations. However, achieving this ideal is very difficult, and many prior works propose mechanisms to approximate the ideal case. We provide a brief overview of prior efforts to eliminate unnecessary refresh operations and discuss their limitations with respect to retention failure discovery, to motivate our experimental characterization in Section 5 and new mechanism in Section 6.

3.1 Retention Failure Mitigation

Many prior works attempt to reduce unnecessary refresh operations by extending the default refresh interval while mitigating the handful of resulting retention failures. RAIDR [63] refreshes DRAM rows at different intervals according to the retention time of the worst-case cell in each row. Ohsawa et al. [79] propose storing the retention time of each row's weakest cell within DRAM structures and varying the refresh interval based on this information. ProactiveDRAM [96] extends the default refresh interval and issues additional refreshes to rows that cannot sustain the longer refresh interval. RAPID [95] prioritizes allocating data to rows with longer retention times and chooses the refresh interval based on the retention time of the allocated row with the highest leakage rate. SECRET [61] identifies the set of failing cells at a longer refresh interval and remaps such cells to known-good cells. ArchShield [76] maintains a data structure of known-failing cells at an extended refresh interval and replicates these cells using a portion of DRAM at an architectural level.

These works all demonstrate significant improvements in overall system performance and energy consumption due to reduction in unnecessary refresh operations (e.g., 35% average performance improvement [81]). Unfortunately, all of these works depend on *accurate identification* of failing cells at a longer refresh interval. As we discuss in Section 3.2, no such identification mechanism that guarantees data correctness exists. Additionally, challenges that arise from VRT effects (Sections 2.3.1 and 5.3) necessitate an *online* profiling mechanism [42, 56, 62, 81].

3.2 Retention Failure Profiling

Prior works propose approaches to profiling retention failures. We find that there is no solution that is both reliable and efficient. We identify two major approaches to profiling: 1) *ECC-scrubbing* [81], a method that relies on ECC to detect retention failures, and 2) *brute-force profiling* [62], a trial-and-error method that detects failures by testing cells many times and with different data patterns. Unfortunately, each of these proposals has shortcomings that limit their effectiveness in DRAM retention failure profiling.

ECC scrubbing approaches (e.g., AVATAR [81]) suffer from a lack of reliability due to their *passive* approach to failure detection. These techniques periodically check the ECC codes for every DRAM word and record the set of rows/cells that fail at a given refresh

interval. Unfortunately, these techniques do *not* account for data pattern changes in a given DRAM row between the periodic scrubs. As a result, they can extend the refresh interval for a row at the end of a period, but that row may be written to with a new unfavorable data pattern, which leads to uncorrectable errors in the next period. Without deliberately (i.e., *actively*) testing with the *worst-case* data patterns, ECC scrubbing *cannot* make an estimate as to what fraction of all possible failures have been detected, which limits the reliability of this approach.

In order to account for all possible failures with different data patterns, a second class of profiling mechanisms takes an *active* approach: they extensively test DRAM for many *rounds* of tests and with different data patterns. We call this approach *brute-force profiling* since it runs multiple *iterations* of a brute-force testing loop that 1) writes data patterns to DRAM, 2) waits for the target refresh interval, and 3) checks for retention failures, as shown in Algorithm 1. Prior works identify effective data patterns useful for retention failure discovery, including solid 1s and 0s, checkerboards, row/column stripes, walking 1s/0s, random data, and their inverses [42, 49, 62]. By testing sufficiently many data patterns, this approach approximates testing with the worst-case data pattern and therefore can find a significant fraction of all failing cells at the target refresh interval. In addition, multiple iterations of testing are required in order to account for the probabilistic nature of retention failures (as we show in Section 5.5). However, as we experimentally demonstrate in Sections 5.3-5.5, brute-force profiling requires a large number of iterations in order to achieve a high coverage of failures at the target refresh interval. This causes brute-force profiling to have a high performance overhead, which is only exacerbated by circumstances that require frequent profiling (Section 7.3).

Algorithm 1: Brute-Force Profiling Algorithm

```
1  PROFILE(target_tREF, num_iterations):
2      failed_cells = []
3      for it ← {1 to num_iterations}:
4          for dp ∈ data_patterns:
5              write_DRAM(dp)
6              disable_refresh()
7              wait(target_tREF)
8              enable_refresh()
9              this_iteration_failures ← get_DRAM_errors()
10             failed_cells.add(this_iteration_failures)
11     return failed_cells
```

Our goal in this paper is to develop an *effective and efficient profiling technique* that can reliably identify the set of failing DRAM cells at target conditions so that we can 1) extend the refresh interval to the target interval and 2) employ an error mitigation mechanism to handle the retention failures that result from the longer refresh interval. We would like this profiling mechanism to provide high failure coverage and a low number of false positives, while having a short runtime.

To this end, we need to understand the tradeoffs involved in profiling and its associated metrics. We first discuss our experimental methodology to develop this understanding based on analysis of real DRAM chips. We then experimentally characterize the retention behavior of modern DRAM chips with respect to the three profiling metrics: coverage, false positive rate, and runtime.

4 EXPERIMENTAL METHODOLOGY

In order to develop an effective and efficient retention failure profiling mechanism for modern DRAM chips, we need to first better understand their data retention characteristics. To this end, we developed a thermally-controlled DRAM testing infrastructure to characterize state-of-the-art LPDDR4 DRAM chips. Our infrastructure provides precise control over DRAM commands, which we verified via a logic analyzer by probing the DRAM command bus.

All tests, unless specified otherwise, were performed using 368 2y-nm LPDDR4 DRAM chips [37] from three major vendors in a thermally-controlled chamber at 45°C ambient temperature. In our infrastructure, ambient temperature is maintained using heaters and fans controlled via a microcontroller-based PID loop to within an accuracy of 0.25°C, with a reliable range of 40°C to 55°C. DRAM temperature is held at 15°C above ambient using a separate local heating source and temperature sensors to smooth out temperature variations due to self-heating.

5 NEW LPDDR4 CHARACTERIZATION

We experimentally study the retention characteristics of 368 LPDDR4 chips and compare our findings with prior works, which examine DDR3 DRAM [42, 49, 62, 81]. We present data showing failure discovery rates, data pattern dependence (DPD) effects and variable retention time (VRT) effects to demonstrate the difficulties involved in retention failure profiling. We then extensively analyze single-cell failure probabilities to motivate profiling at a longer refresh interval and/or a higher temperature to quickly discover failing cells. We present four key observations resulting from our experimentation and describe the implications of each observation on retention failure profiling.

5.1 Temperature Dependence

We observe an exponential dependence of failure rate on temperature for refresh intervals below 4096ms, as consistent with prior work [31, 34, 62]. We find and make use of the following temperature relationships throughout the rest of this paper:

$$R_A \propto e^{0.22\Delta T} \qquad R_B \propto e^{0.20\Delta T} \qquad R_C \propto e^{0.26\Delta T} \quad (1)$$

where R_X represents the proportion of failures for Vendor X and ΔT is the change in ambient temperature. These relationships approximately translate to scaling the retention failure rate by a factor of 10 for every 10°C increase in temperature.

5.2 Aggregate Retention Time Distributions

Figure 2 shows the effect of increasing the refresh interval on the bit error rates (BER) averaged across 368 total chips from three different vendors. For each refresh interval, we compare the population of failing cells to the population of failing cells at all *lower* refresh intervals. Failures that are 1) *not* observed at lower refresh intervals are shown in purple (called *unique*), 2) observed at lower refresh intervals but *not* at the given interval are shown in orange (called *non-repeat*), and 3) *also* observed at lower refresh intervals are shown in green (called *repeat*). We make one key observation.

Observation 1: A large proportion of cells that are observed to fail at a given refresh interval are likely to *fail again at a higher refresh interval*.

Corollary 1: Determining the set of failing cells at a given refresh interval provides a large proportion of the failures found at *lower* refresh intervals.

5.3 Variable Retention Time Effects

We observe variable retention time (VRT) effects that prevent reliable detection of failing cells. Figure 3 shows the number of failing cells discovered over six days of data collection using brute-force profiling at a refresh interval of 2048ms at 45°C ambient temperature. The data is shown for a single representative chip from Vendor B, but we find a similar trend across all chips from all vendors. Profiling follows the procedure described in Algorithm 1, using six data patterns and their inverses (Section 3.2) per iteration over 800 iterations throughout the length of the test. The cumulative set of discovered failures is shown in orange and the set of failures discovered each iteration is broken up into unique (i.e., newly-discovered) failures and repeat failures, shown in purple and green, respectively.

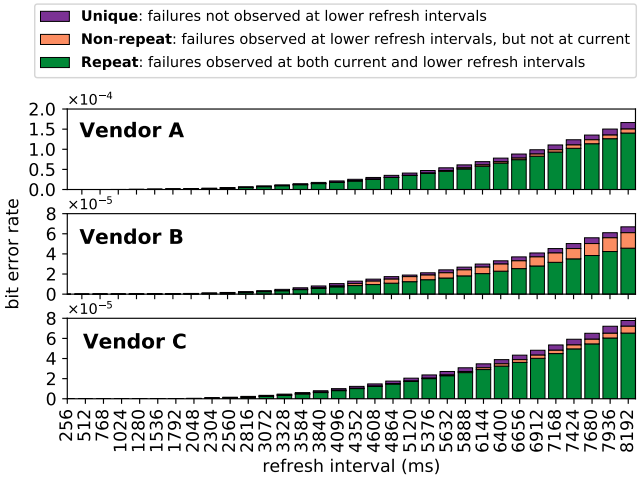

Figure 2: Retention failure rates for different refresh intervals. Cells are categorized as unique, repeat, or non-repeat based on whether or not they are observed at the given interval *and* at lower refresh intervals.

We find that after about 10 hours of testing, brute-force profiling enters the *steady-state accumulation* phase, in which new failures continue to accumulate with a rate of approximately one cell every 20 seconds. In other words, it takes about 10 hours to find the base set of failures for a given refresh interval using the brute-force approach. We attribute the continual discovery of new failures to the VRT phenomenon, which can cause a cell to shift from a retention time greater than 2048ms to one that is lower. However, we also observe that the total set of failures (unique + repeat) found in each iteration is nearly constant in size, implying that the rate of failure accumulation is very close to the rate of failures leaving the failing set. This finding is consistent with prior work on DDR3 chips [62, 81], showing that newer DRAM generations face similar profiling difficulties to older generations.

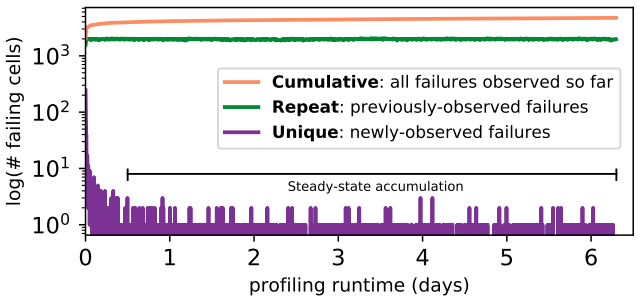

Figure 3: Number of failing cells discovered using brute-force profiling at a refresh interval of 2048ms at 45°C ambient temperature using a single representative chip (from Vendor B).

We conduct this same analysis for *different refresh intervals* on 368 chips from across the three DRAM vendors at 45°C. Figure 4 shows the *steady-state new failure accumulation* rates for different refresh intervals aggregated across all chips of each vendor. The y-axis represents the steady-state new failure accumulation rate, and the x-axis shows the refresh interval. Each data point is drawn at the average value across all chips from the vendor with error bars representing the respective standard deviation. We find that the steady-state failure accumulation rate grows at a *polynomial* rate with respect to the refresh interval. Figure 4 overlays the data with well-fitting polynomial regressions of the form $y = a * x^b$, and the exact fit equations are provided in the figure itself.

Observation 2: No matter how comprehensive the set of failures discovered is, the population of failing cells continues to change due to VRT effects.

Corollary 2: The retention failure profile inevitably needs to be re-generated after some period of time. In other words, *online* profiling is required.

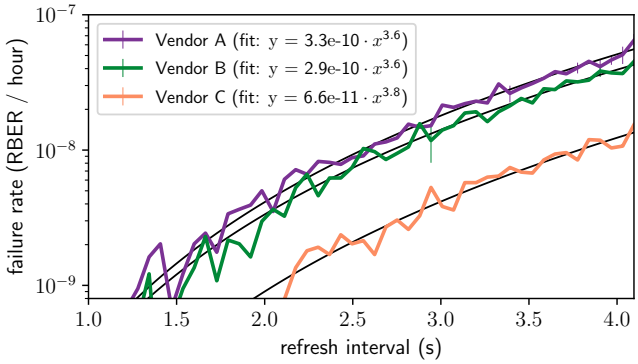

Figure 4: Steady-state failure accumulation rates vs. refresh interval for 368 chips across the three DRAM vendors at 45°C. The best-fit curve for each vendor is shown in black.

5.4 Data Pattern Dependence Effects

Figure 5 shows the cumulative number of retention failures discovered over 800 iterations spanning 6 days of continuous brute-force profiling with different data patterns across the three vendors at 45°C. Profiling uses Algorithm 1, in which each iteration consists of writing a data pattern to DRAM, pausing refresh for 2048ms, and then checking for retention failures. New failures found by each data pattern each iteration are added to a running set representing the total number of failures discovered so far. The ratio of failures discovered by each data pattern to the *total* number of failures discovered by *all data patterns together* is plotted against time. We see that failures continue to accumulate over time, as expected from Section 5.3, and different data patterns uncover different fractions of failures. Of the six data patterns tested, the random pattern (solid dark purple) discovers the most failing cells across all three vendors.

Observation 3: Unlike DDR3 DRAM [62], the random data pattern most closely approaches full coverage after 800 iterations spanning 6 days, but it still *cannot* detect *every* failure on its own.

Corollary 3: A robust profiling mechanism should use *multiple data patterns* to attain a high coverage of the set of failing cells.

5.5 Individual Cell Failure Probability

We experimentally study the retention characteristics of individual DRAM cells. We find that each cell's probability of retention failure follows a normal distribution with respect to the refresh interval. Figure 6(a) shows the effect of changing the refresh interval (shown normalized from between 64ms and 4096ms to $x = 0.00$s) on the probability of retention failure for almost all of the failing cells from a representative DRAM chip of Vendor B.[1] Each curve represents the failure cumulative distribution function (CDF) of a single cell, with failure probabilities of 0.0 and 1.0 representing 0% and 100% incorrect reads out of 16 total test iterations, respectively. All distributions' means are normalized to $x = 0.00$s in order to highlight the similarity in failure pattern between different cells. We see that due to the normally-distributed failure probability of each cell, *every cell becomes more likely to fail at longer refresh intervals*.

Figure 6(b) shows a histogram of the standard deviations of each cell's unique failure distribution. We find that the standard deviations

[1]Cells exhibiting VRT behavior (~2% of all cells for these conditions) are excluded from this plot, for ease of explanation.

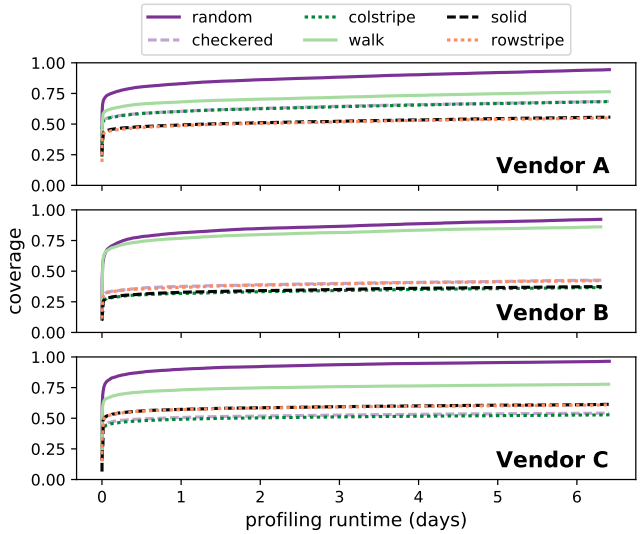

Figure 5: Coverage of unique retention failures discovered by different data patterns using brute-force profiling across 800 iterations spanning 6 days.

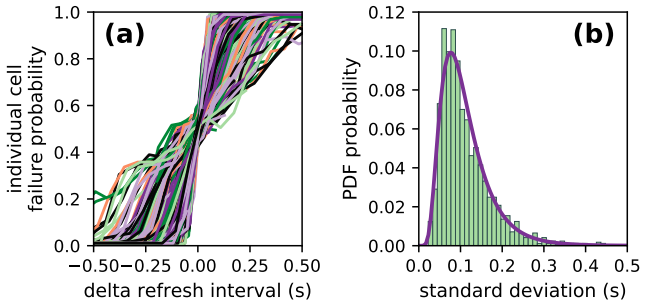

Figure 6: (a) Individual cells fail with a normally-distributed cumulative distribution function with respect to refresh interval (CDF means are normalized to x = 0.00s) and (b) their standard deviations follow a lognormal distribution (right). This data is taken from a single representative chip of Vendor B at 40°C and base refresh intervals ranging from 64ms to 4096ms.

follow a tight lognormal distribution, with the majority of cells having a standard deviation of less than 200ms at these conditions.[2] This means that *slightly extending the refresh interval results in significantly increasing the failure probability of almost all cells, which makes cells with low failure probabilities easier to detect*.

We can explain the shape of these distributions using circuit-level DRAM characteristics. As described in Section 2.2, the DRAM cell read process requires sense amplifiers to resolve the direction of a shift in bitline voltage. This shift is known as the *sense amplifier voltage offset*, and due to various manufacturing variations, the magnitude of this shift across different cells is found to be normally distributed [60, 100]. While using an extended refresh interval, DRAM cell leakage causes some cells' voltage offsets to be too small to resolve correctly, causing the sense amplifier to probabilistically read an incorrect value. Therefore, our observation of normally-distributed failure probabilities for all cells (Figure 7(a)) is a direct consequence of the normal distribution of the sense amplifier voltage offsets. Furthermore, Li et al. [60] find that leakage components in DRAM cells follow lognormal distributions, which we hypothesize is responsible for our observation of lognormally-distributed standard deviations (Figure 7(b)). Given that different cells leak at lognormally-distributed rates, we would expect the length of time that a cell's charge is within the noise margin of the sense amplifier to also be lognormally distributed.

We extend this analysis to different temperatures, and we aggregate the normal distribution fit parameters (μ, σ) for each failing cell into the distributions shown in Figure 7 (a) and (b), respectively. We see that at higher temperatures, the distributions for both the means and standard deviations shift left. This means that on average, cell retention times shift to lower values and their failure distribution becomes narrower around that retention time. We can take advantage of this observation by profiling not only at a longer refresh interval, but also at a *higher temperature*, thus ensuring that a large majority of the cells observed to *inconsistently* fail at the target refresh interval and temperature are very likely to fail at the (longer) profiling refresh interval and the (higher) profiling temperature.

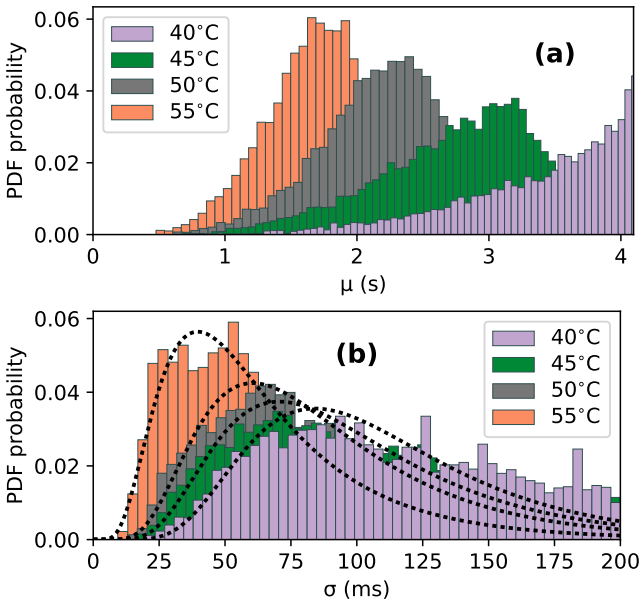

Figure 7: Distributions of individual cells' normal distribution parameters (μ, σ) over different temperatures taken from a representative chip from Vendor B. We see that both distributions shift left with increasing temperature, which means that an increase in temperature causes individual cells to both fail at a lower refresh interval and also exhibit a narrower failure probability distribution.

Observation 4: Cells that inconsistently fail at the target conditions may be missed with brute-force profiling, but are more likely to be found when profiling at a longer refresh interval and/or a higher temperature.

Corollary 4: In order to find failures quickly and consistently, we should profile at *a longer refresh interval* or *a higher temperature*.

By combining the normal distributions of individual cell failures from a representative chip of Vendor B, we obtain the data in Figure 8, which shows the failure probability for the combined distribution over different temperatures and refresh intervals. The dashed regions represent the combined standard deviation for each tested temperature, and the solid black curve in-between represents the combined mean. From this data, we draw two major conclusions: 1) at a higher temperature or a longer refresh interval, the typical cell

[2]Note that this is *not* the same result as the well-known lognormal distribution of the number of retention failures with respect to refresh interval [31, 60]. Our observation is that the spread of *each individual cell failure distribution* is lognormally-distributed as opposed to the *overall number of failing cells* at different refresh intervals.

is more likely to fail, and 2) raising the temperature and extending the refresh interval have similar effects on cell failure probability (e.g., at 45°C, a 1s change in refresh interval has a similar effect to a 10°C change in temperature).

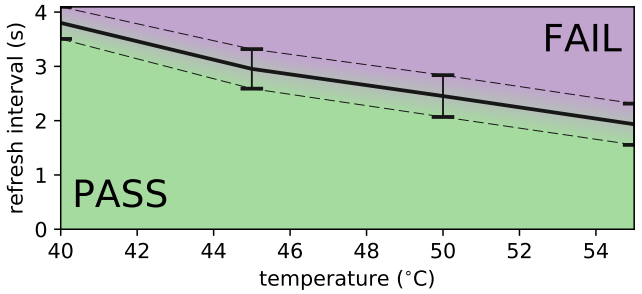

Figure 8: Effect of manipulating temperature/refresh interval on the combined normal distribution of failing cells from a representative chip from Vendor B. Dashed regions represent the combined standard deviation and the central line represents the combined mean for each tested temperature.

6 REACH PROFILING

Reach profiling accelerates the process of finding the set of DRAM cells that experience retention failures by profiling DRAM under conditions that increase the likelihood of failure, called *reach conditions*. This enables the profiler to discover the failures much more quickly, leading to a significant reduction in overall profiling runtime. Given a set of desired *target conditions* consisting of a target refresh interval and a target operating temperature, the *key idea* of reach profiling is to profile at *reach conditions*, a combination of a *longer refresh interval* and a *higher temperature* relative to the target conditions. As experimentally demonstrated in Section 5.5, each DRAM cell is *more likely* to fail at such reach conditions than at the target conditions. While this enables reach profiling to attain high failure *coverage* (i.e., the ratio of correctly identified failing cells over the total number of failing cells at the target conditions), it also causes reach profiling to have *false positives* (i.e., failing cells at the reach conditions that do *not* fail at the target conditions), leading to more work for a retention failure mitigation mechanism that is used to correct the retention failures (including false positives). More work done by the mitigation mechanism leads to higher overhead in terms of performance, energy, or area. This results in a *complex tradeoff space* between the choice of reach conditions and the resulting profiling coverage, false positive rate, and runtime.

Given this complex tradeoff space, we need to address three **key questions** to develop an effective implementation of reach profiling:

(1) What are the desirable *reach conditions*?
(2) Given the continuous accumulation of new failures (e.g., due to VRT), how long does a retention failure profile remain useful (i.e., how often must we reprofile)?
(3) What information does the profiler need in order to determine desirable reach conditions for a given system?

We explore the answers to these questions in the rest of this section. Then, in Section 7, we use our exploration to develop REAPER, a robust implementation of reach profiling.

6.1 Desirable Reach Conditions

The three key metrics of profiling, *coverage*, *false positive rate*, and *runtime* (Section 1), lead to contradictory optimization goals. While an ideal configuration would achieve high coverage, low false positive rate, and low runtime, we find that there is a large tradeoff space involving these three goals. Using experimental data, we demonstrate the large scale of the tradeoff space inherent in reach profiling and show the effects of changing the refresh interval and the temperature on the three key profiling metrics. We present data for interesting choices of reach conditions and analyze the results to show how the profiling system can make a reasonable choice.

6.1.1 Manipulating Refresh Interval and Temperature. Figure 9 demonstrates the effect of choosing different reach profiling conditions on failure coverage and false positive rate for a representative chip. In order to perform this analysis, brute-force profiling is conducted at regularly spaced points throughout the graphs in Figure 9 using 16 iterations of 6 different data patterns and their inverses as per Algorithm 1. Each point in the graph space is then treated as a target condition with all other points as its reach conditions. This results in a distribution of coverage/false positive rates for *every* delta temperature/refresh interval combination.

We find that these distributions are highly similar for each delta temperature/refresh interval combination, with standard deviations of less than 10% of the data range for both coverage and false positive rate. This allows us to use the means of these distributions to reasonably demonstrate the overall effect of manipulating the temperature and/or refresh interval *independently* of a particular target refresh interval. We show these means in Figure 9, with $(x, y) = (0.00, 0)$ effectively representing *any* target refresh interval and all other points in the figure representing its reach conditions. The contours represent the coverage (top graph) and false positive rate (bottom graph).[3] As these two graphs show, by increasing the refresh interval and/or temperature, we can obtain a higher failure coverage (Figure 9, top), as expected given the individual cell failure probability analysis in Section 5.5. However, this also results in an increase in the false positive rate (Figure 9, bottom). Thus, there is a direct tradeoff between coverage and false positive rate.

Profiling runtime is more difficult to evaluate since it depends on 1) the profiling refresh interval, 2) number of profiling iterations (see Algorithm 1), and 3) overheads of reading and writing the data patterns to all of DRAM. We experimentally find that the amount of time taken to read/write data to all DRAM channels and check for errors is slightly less than 250ms for our evaluated chips. So, we assume a fixed overhead of 250ms per profiling iteration per data pattern tested. Figure 10 shows the results of such an analysis for a representative chip where $(x, y) = (0.00, 0)$ represents profiling at the target conditions (i.e., brute-force profiling) and all other points show the results of reach profiling with the same analysis as in Figure 9. Here, each contour curve shows the profiling runtime at different reach conditions, all normalized to the runtime at the target refresh interval (i.e., brute-force profiling runtime). Profiling runtime is determined by the number of profiling iterations required to achieve over 90% coverage.

We see that we can obtain drastic profiling runtime speedups by aggressively increasing the reach conditions (e.g., +0.5s, +5°C), but from Figure 9, we know that this will result in a very high number of false positives (e.g., over 90% false positive rate). Although the runtime numbers in Figure 10 assume a fixed coverage, we observe the same trends across different coverage requirements. Therefore, we conclude that high coverage and low runtime are generally *competing goals* to low false positive rates, and choosing a good set of reach conditions at which to profile depends on the user's and the system's requirements and failure mitigation capabilities.

We repeat this analysis for all 368 of our DRAM chips and find that each chip demonstrates the same trends, showing that the same tradeoff analysis we have done for Figures 9 and 10 also applies to every other chip we tested. In the following section, we analyze the data presented here and determine how to select a desirable set of reach conditions.

[3]The contours are not perfectly smooth due to variations in the data resulting from small shifts (<0.25°C) in temperature throughout testing and the probabilistic nature of retention failures, including VRT effects.

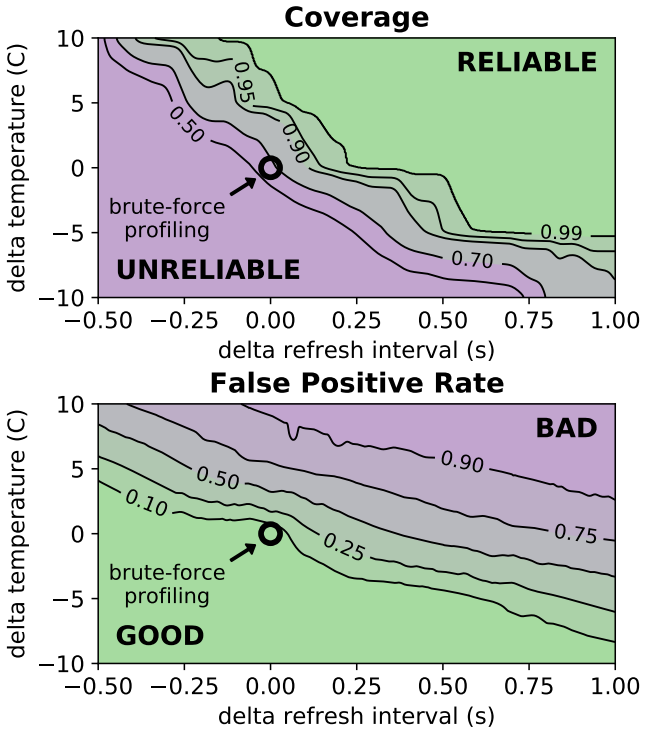

Figure 9: Failure coverage (top) and false positive rates (bottom) as a result of different choices of reach profiling conditions for a representative chip from Vendor B. Contours represent coverage (top) and false positive rate (bottom) obtained relative to brute-force profiling at $(\mathbf{x}, \mathbf{y}) = (\mathbf{0.00}, \mathbf{0})$.

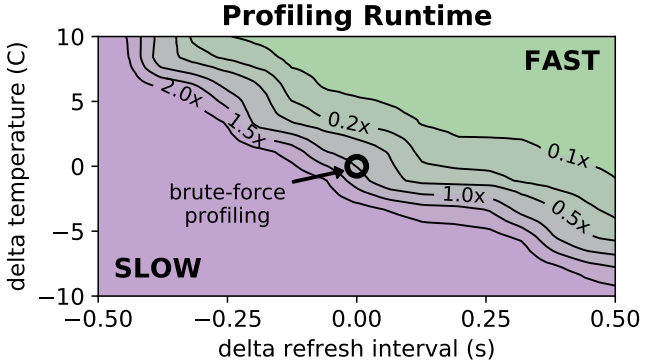

Figure 10: Profiling runtime as a result of different choices of reach conditions for a representative chip from Vendor B. Contours represent profiling runtime relative to brute-force profiling at $(\mathbf{x}, \mathbf{y}) = (\mathbf{0.00}, \mathbf{0})$.

6.1.2 Choice of Desirable Parameters. The exact choice of reach conditions depends on the overall system design and its particular optimization goals. For example, in a highly latency-sensitive system, the primary goal may be to minimize profiling runtime, in which case giving up a low false positive rate to obtain a low profiling runtime may be necessary. On the other hand, if the retention failure mitigation mechanism in use is intolerant to high false positive rates (e.g., discarding all DRAM rows containing failing cells), a low false positive rate may be the primary design goal. In general, the system designer should take these factors into account and make the best tradeoff for the target system.

However, although the exact choice of reach conditions depends on the system goals, we observe from Figures 9 and 10 that higher coverage and higher runtime are directly and positively correlated, and both come at the expense of false positives. This means that when selecting suitable reach profiling conditions, the system designer can feasibly select as high a refresh interval/temperature as possible that keeps the resulting amount of false positives tractable. This approach primarily relies on identifying the cost of false positives for the particular system in question, possibly sacrificing the ability to handle a large number of false positives in favor of low profiling runtime.

As a concrete example, from studying data similar to those presented in Figures 9 and 10 averaged across all 368 chips, we find that for 99% coverage at a modest 50% false positive rate, we attain a profiling runtime speedup of 2.5x over the brute-force mechanism when we extend the refresh interval by 250ms. We find that we can push the speedup to over 3.5x while maintaining the same level of coverage by either increasing the temperature or lengthening the refresh interval even further, but such aggressive reach profiling conditions result in greater than 75% false positive rates.

6.2 Online Profiling Frequency

In order to enable a longer refresh interval, we not only have to provide a way to profile for retention failures, but also have to guarantee that retention failure profiling provides sufficient coverage to prevent erroneous system operation. Given failures that are missed by profiling due to less-than-100% coverage and non-zero new failure accumulation rates (Section 5.3), we need a method by which to estimate *profile longevity, i.e.,* the amount of time before a profile is no longer correct (i.e., when reprofiling becomes necessary). In this section, we analyze the types of errors that can be missed by profiling and present a theoretical model for allowable error rates and profile longevity.

6.2.1 Failures Missed by Profiling. Retention failure profiling can only capture cells that fail at the profiling conditions. Cells missed during profiling due to DPD effects (Section 5.4) and newly-failing cells that did not fail during profiling due to environmental factors (e.g., temperature shifts, soft errors) or due to VRT effects (Section 5.3) cannot be observed by profiling. Prior works acknowledge that there will inevitably be failures that are missed by profiling and argue that *some form of ECC is necessary to allow safe operation with a longer refresh interval* [42, 62, 81]. Our observation of continuously-accumulating new failures in LPDDR4 DRAM chips (Section 5) leads us to agree with this argument for the use of ECC. Thus, in order to enable online profiling, we need to determine 1) what error rates can be tolerated by an ECC-based system and 2) at what point the accumulated failures will exceed the correction capability of ECC and require reprofiling.

6.2.2 Allowable Errors and Tolerable Error Rates. In order to estimate the probability of system failure given DRAM retention errors in the presence of various types of error correction codes (ECC), we estimate the error rate as observed by the system, called the uncorrectable bit error rate (UBER), as a function of the raw bit error rate (RBER), defined as the ratio of failing DRAM cells. We define *system failure* as exceeding an UBER of 1) 10^{-15} for consumer applications [71] and 2) 10^{-17} for enterprise applications [4].

Given a general system using k-bit ECC (with $k = 0$ defined as no ECC, $k = 1$ as SECDED ECC [67], etc.) and an ECC word size of w, we define the UBER as the probability of observing an error in a single DRAM ECC word, normalized to the number of bits per ECC word:

$$\text{UBER} = \frac{1}{w} \text{P}[\text{uncorrectable error in a } w\text{-bit ECC word}] \quad (2)$$

We obtain an uncorrectable error in an ECC word when we have an n-bit error $\forall\ n > k$. This means that we can expand Equation 2 in terms of n:

$$\text{UBER} = \frac{1}{w}\sum_{n=k+1}^{w} \text{P}[n\text{-bit failure in a } w\text{-bit ECC word}] \quad (3)$$

representing the sum of the probabilities of all possible k-bit ECC uncorrectable errors. In particular, for $k = 0$ and $k = 1$, we have 0 and 8 additional bits per 64-bit data word, respectively. For these cases, Equation 3 takes the form:

$$\begin{aligned}\text{UBER}(k=0) &= \frac{1}{64}\sum_{n=1}^{64} \text{P}[n\text{-bit failure in a 64-bit ECC word}] \\ \text{UBER}(k=1) &= \frac{1}{72}\sum_{n=2}^{72} \text{P}[n\text{-bit failure in a 72-bit ECC word}]\end{aligned} \quad (4)$$

The probability of an n-bit error in a k-bit word can be modeled by a binomial distribution, assuming that DRAM retention failures are independent and randomly distributed, as has been shown in previous work [6, 95]. Given that these assumptions hold, we can use the RBER R as the probability of a single-bit failure within DRAM, and expand the probability of an n-bit failure in a k-bit ECC word using a binomial distribution in terms of R:

$$\text{P}[n\text{-bit failure in a w-bit ECC word}] = \binom{w}{n} R^n (1-R)^{w-n} \quad (5)$$

Putting the pieces together, we arrive at:

$$\text{UBER} = \frac{1}{w}\sum_{n=k+1}^{w} \binom{w}{n} R^n (1-R)^{w-n} \quad (6)$$

Table 1 summarizes the maximum tolerable *RBER* for a target UBER of 10^{-15}, for various choices of ECC strength. In order to provide comparison points, the table also translates the tolerable RBER into actual *number of tolerable bit errors* for various DRAM sizes.

		ECC Strength		
		No ECC	**SECDED**	**ECC-2**
Tolerable RBER		1.0e−15	3.8e−9	6.9e−7
# Tolerable Bit Errors	512MB	4.3e−6	16.3	3.0e3
	1GB	8.6e−6	32.6	5.9e+3
	2GB	1.7e−5	65.3	1.2e+4
	4GB	3.4e−5	130.6	2.4e+4
	8GB	6.9e−5	261.1	4.7e+4

Table 1: Tolerable RBER and tolerable number of bit errors for UBER = 10^{-15} across different ECC strengths for selected DRAM sizes

Thus, for any target UBER, we can compute the *tolerable RBER*, and hence the *maximum* number of cells that can be allowed to *escape* (i.e., not be detected by) our retention failure profiling mechanism while still maintaining correct DRAM operation. By applying the maximum tolerable RBER to the RBER at any desired target refresh interval, we can directly compute the minimum coverage required from a profiling mechanism in order for the retention failure mitigation mechanism to guarantee correct system operation.

6.2.3 Profile Longevity.

Given the maximum tolerable number of retention failures N provided by Table 1, the number of failures C missed by profiling due to imperfect profiling coverage, and the accumulation rate A of new failures as measured in Section 5.3, we can estimate T, the length of time before we need to reprofile, as:

$$T = \frac{N - C}{A} \quad (7)$$

For example, with a 2GB DRAM and SECDED ECC as the failure mitigation mechanism, we can afford up to $N = 65$ failures while still maintaining correct DRAM operation. Assuming an aggressive target refresh interval of 1024ms at 45°C, we empirically observe 2464 retention failures (Figure 2) and a new failure accumulation rate of $A = 0.73$ cells / hour (Figure 4). With 99% coverage, we compute $C = 24.6 \approx 25$ cells, and applying Equation 7, we obtain $T = 2.3$ days. We can use a similar analysis to determine the profile longevity for any desired configuration and choose an appropriate tradeoff point between the different reach profiling parameters.

6.3 Enabling Reliable Relaxed-Refresh Operation

In order to determine good reach conditions at which to profile for a real system running at a longer refresh interval than the default, we need to know two key pieces of information: 1) the particular retention failure mitigation mechanism in use, so that we can constrain the reach profiling tradeoff space, and 2) detailed chip characterization data, in order to make reliable estimates of reach profiling benefits and profile longevity, allowing us to determine which point in the tradeoff space will provide the best overall system performance/energy improvement.

The choice of *retention failure mitigation mechanism* determines the hardware/energy/performance overhead of managing failing cells, which determines how aggressively we can push the target/reach conditions before the system can no longer cope with the number of failures and false positives. The mitigation mechanism therefore constrains both the target conditions (i.e., the resultant RBER *without* mitigation) and the maximum number of false positives, which in turn restricts the range of the reach conditions.

Detailed chip characterization data (as in Figures 9 and 10) is necessary to produce accurate estimates of all of the parameters of the actual system, including the expected RBER, the profile longevity, the required coverage, and even the reach conditions themselves. While these parameters can be estimated from general trends across many chips, the variations among different chips mean that truly reliable relaxed refresh operation requires estimating profiling parameters based on data from the actual chip.

Currently, DRAM vendors do not provide this data, but it would be reasonable for vendors to provide this data in the on-DIMM serial presence detect (SPD) as done in [52]. Otherwise, the user would have to characterize his/her own device. Even though a detailed characterization may take prohibitive amounts of time, a few sample points around the tradeoff space could provide enough information in conjunction with the general trends across many devices to develop accurate estimations. However, the general problem of efficiently obtaining per-chip characterization data is itself an open research direction and is beyond the scope of this work.

Once these two critical pieces of information are available, the system designer can decide what the best tradeoff is between profile longevity, coverage, false positive rate, and runtime according to his/her own system configuration. Despite the overall trend similarities we observe between the DRAM chips we evaluate in Section 5, the *optimal* choice of reach conditions depends on the particular system and its tradeoffs between cost, performance, power, and complexity, and it is up to the user to determine the most suitable configuration for his/her own needs.

7 END-TO-END IMPLEMENTATION

Reach profiling provides a general-purpose retention failure profiling methodology whose exact use and design parameters (Section 6.3) depend on the error mitigation mechanism used for reduced-refresh-rate operation. A system designer should utilize reach profiling uniquely depending on desired performance/energy targets and ease/overhead of implementation. For example, the designer is free to choose where to implement the core profiling algorithm (e.g., at the memory controller, the operating system, etc.), how to save and restore the state of DRAM before and after a profiling *round* (i.e.,

each individual instance of online profiling consisting of all iterations and data patterns), how to efficiently profile large portions of DRAM without significant performance loss, etc. However, a thorough exploration of the design space regarding the implementation of reach profiling is beyond the scope of this paper.

In order to effectively present the insight behind the tradeoffs involved in reach profiling, we provide and evaluate REAPER, a naïve (yet robust) example implementation of reach profiling that assumes a full-system pause each time retention failures are profiled. REAPER enables a variety of retention failure mitigation mechanisms [61, 63, 76, 79, 95, 96] for reliable operation at longer refresh intervals. We experimentally validate that, even with these worst-case assumptions, REAPER preserves a significant amount of the benefit of an ideal profiling mechanism that has zero overhead while outperforming brute-force profiling in terms of both performance and power consumption.

7.1 REAPER Implementation

Reach profiling requires the ability to manipulate the DRAM refresh interval and/or the DRAM temperature, both of which can be achieved with commodity off-the-shelf DRAM. REAPER implements reach profiling in firmware running directly in the memory controller. Each time the set of retention failures must be updated, profiling is initiated by gaining exclusive access to DRAM. REAPER has the ability to manipulate the refresh interval directly, but for simplicity, we assume that *temperature* is not adjustable (as shown in Section 5.5, manipulating either the temperature or the refresh interval achieves the same effect). REAPER conducts reach profiling and stores the addresses of the failing cells it finds at locations dictated by the retention failure mitigation mechanism of choice.[4] At the completion of profiling, REAPER releases exclusive DRAM access and the system resumes normal operation. In the remainder of this section, we describe how this implementation of reach profiling can be combined with two example retention failure mitigation mechanisms from prior work to reduce refresh operations.

7.1.1 REAPER Supporting ArchShield. As a demonstrative example, we combine REAPER with ArchShield [76] to enable refresh rate reduction. ArchShield requires a small amount of additional logic in the memory controller for detecting accesses to known-faulty addresses, which are stored in 4% of DRAM in a reserved segment known as the FaultMap. The REAPER firmware provides these faulty addresses via periodically profiling DRAM for retention failures following the methodology in Section 3.2. All detected failures are stored into the FaultMap, which ArchShield accesses to determine which failing addresses must be remapped.

7.1.2 REAPER Supporting Multi-Rate Refresh. In order to show REAPER's flexibility for integration with other retention failure mitigation mechanisms, we describe REAPER working with RAIDR [63]. RAIDR groups DRAM rows into different bins, according to the rows' retention times, and applies different refresh rates to each bin. REAPER enables RAIDR by periodically updating the bins using the set of failures discovered each time profiling is conducted. This enables the system to reduce the refresh interval for most of the rows within DRAM, which results in overall system performance increase and energy reduction.

7.2 Evaluation Methodology

Given that profiling rounds require on the order of seconds or minutes (Section 7.3.1) and the time between online profiling rounds can be on the order of hours (Section 6.2.3), we cannot feasibly simulate workloads in order to demonstrate latency effects due to online profiling. Instead, we simulate our workloads without the profiling overhead and report performance results using throughput in terms of instructions-per-cycle (IPC). We then use the following model to compute overall system performance accounting for online profiling overhead:

$$\text{IPC\_real} = \text{IPC\_ideal} \times (1 - \text{profiling\_overhead}) \quad (8)$$

where *IPC_ideal* is the measured throughput of simulated workloads, and *profiling_overhead* is the proportion of real system time spent in profiling. Our profiling overhead model assumes worst-case configurations, pessimistically assuming that applications make zero forward progress while profiling. Our full system performance model roughly estimates the worst-case performance degradation expected from an implementation of reach profiling.

We use Ramulator [2, 53] to evaluate performance and DRAMPower [1] to evaluate DRAM power consumption. We simulate 20 multiprogrammed heterogeneous workload mixes, each of which is constructed by randomly selecting 4 benchmarks from the SPEC CPU2006 benchmark suite [3]. Multi-core system performance is measured in terms of the weighted speedup metric [28, 91]. We provide our evaluated system configuration in Table 2.

Processor	4 cores, 4GHz clock frequency, 3-wide issue, 8 MSHRs/core, 128-entry instruction window
Last-level Cache	64B cache line, 16-way, 8MB cache size
Memory Controller	64-entry read/write request queues, FR-FCFS scheduling policy [83, 102], open/closed row policy [50, 51] for single/multi-core
DRAM	LPDDR4-3200 [37], 4 channels, 1 rank, 8 banks/rank, 32K-256k rows/bank, 2KB row buffer

Table 2: Evaluated system configuration

7.3 Performance and Energy Evaluation

We develop a detailed model for profiling overhead, taking into account extra DRAM accesses required for profiling, latencies involved in reading/writing data patterns to DRAM, and time consumed waiting for the extended refresh intervals. We use this model in conjunction with Equation 8 to reliably estimate worst-case system performance impact and additional DRAM power consumption. In this section, we present our 1) model for profiling performance and power consumption overhead for brute-force profiling and reach profiling, and 2) results for end-to-end system performance and DRAM power consumption using brute-force profiling, reach profiling, and the ideal profiling mechanism that does not impact system performance or power consumption. We find that while profiling power consumption overheads are very low across a wide range of DRAM sizes and refresh intervals, profiling performance overheads become significant for large DRAM chips and high online profiling frequencies. In such cases where profiling performance significantly impacts overall system performance, reach profiling maintains a large fraction of the benefits of using a longer refresh interval (e.g., providing 14.8% higher performance than brute-force profiling on average for a 64Gb chip running at a 1280ms refresh interval at 45°C), as we show in Section 7.3.2.

7.3.1 Profiling Overhead. We model the overhead of online profiling by accumulating the latencies of individual operations required to run a single *round* of profiling. We assume that profiling requires a full system pause, with no useful work being done while the profiler is running. One round of profiling consists of N_{it} iterations, each of which consists of reading/writing DRAM with N_{dp} different data patterns. Our end-to-end runtime model is as follows:

$$T_{\text{profile}} = (T_{REFI} + T_{\text{wr DRAM}} + T_{\text{rd DRAM}}) * N_{\text{dp}} * N_{\text{it}} \quad (9)$$

where T_{REFI} is the profiling refresh interval that DRAM must wait with refresh disabled in order to accumulate errors; $T_{wr\ DRAM}$ is the

[4] Profiling involves modifying the contents of DRAM. Data must be saved and restored before and after profiling is run, respectively. While a naïve implementation may flush all DRAM data to secondary storage (e.g., hard disk drive, SSD, NVM) or other parts of DRAM, a more efficient implementation could hide most or all of this latency. Efficiently implementing DRAM data save and restore is an orthogonal problem to this work. Therefore, we do not take DRAM data save and restore overheads into account in our performance and energy evaluations.

time to write a data pattern into DRAM; and $T_{rd\ DRAM}$ is the time to read DRAM and compare against the original data pattern. For 32 8Gb DRAM chips with T_{REFI} = 1024ms, $T_{\text{rd/wr DRAM}}$ = 0.125s[†], N_{dp} = 6, and N_{it} = 6, we find that $T_{\text{profile}} \approx$ 3.01 minutes, and for 32 64Gb chips, $T_{\text{profile}} \approx$ 19.8 minutes.

In order to demonstrate the overall system performance degradation from profiling, Figure 11 plots the proportion of overall system time spent profiling for a variety of different profiling intervals and DRAM sizes assuming 16 iterations of brute-force profiling at 1024ms using 6 data patterns and their inverses. The x-axis represents the online profiling frequency in hours, and the y-axis shows the proportion of total system-time spent profiling. To compare the brute-force method against REAPER, we plot the performance overheads with hashed (brute-force) and solid (REAPER) bars. We show results for the 2.5x profiling runtime speedup of reach profiling over brute-force profiling, found experimentally in Section 6.1.2. The differently-colored bars represent DRAM modules consisting of 32 individual DRAM chips, with each chip ranging from 8Gb to 64Gb in size. We observe that at shorter profiling intervals, the performance degradation is prohibitively high and is exacerbated by larger DRAM chip sizes. For example, we find that for a profiling interval of 4 hours and a 64Gb chip size, 22.7% of total system time is spent profiling with brute-force profiling while 9.1% of time is spent profiling with REAPER.

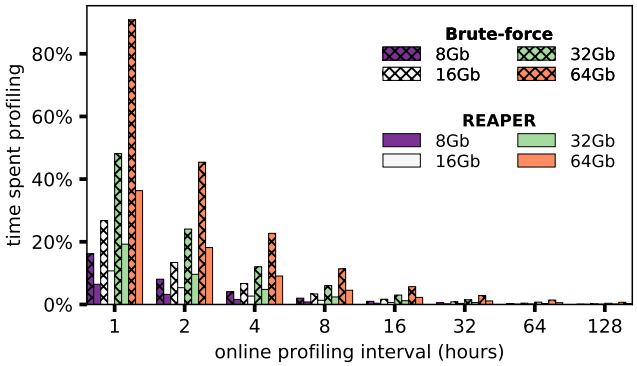

Figure 11: Total system time spent profiling with REAPER and brute-force profiling for different online profiling intervals using 32 DRAM chips, for different chip sizes.

In order to demonstrate the overall DRAM power overhead associated with profiling, Figure 12 shows the total DRAM power required for profiling across the same sweep of configurations as Figure 11. Power consumption overhead is calculated using the average energy-per-bit costs for different DDR commands obtained from our LPDDR4 DRAMPower model [1]. We estimate the number of extra commands required for profiling and show the total DRAM energy consumed for one round of profiling divided by the online profiling interval. The x-axis represents the online profiling frequency in hours, and the y-axis shows the DRAM power consumption in nanowatts. We see that power consumption has a strong dependence on DRAM size and shows a similar scaling trend as profiling performance overhead with respect to the profiling interval. However, the absolute power consumption is very low (on the order of nanowatts), because the majority of profiling runtime is spent waiting for retention failures to occur (i.e., the profiling refresh interval) rather than actively accessing DRAM. We show in Section 7.3.2 how profiling itself results in negligible additional power in the DRAM system, even in the extreme case of very frequent profiling and large DRAM sizes.

[†]This value is based on empirical measurements from our infrastructure using 2GB of LPDDR4 DRAM. We scale this number according to DRAM size throughout our evaluation to account for the larger number of accesses.

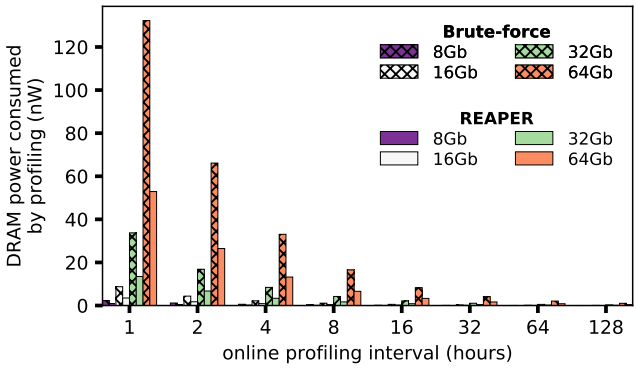

Figure 12: DRAM power consumption of REAPER and brute-force profiling for different online profiling intervals using 32 DRAM chips, for different chip sizes

.

7.3.2 End-to-end System Evaluation Results. We evaluate the overall effects of REAPER on system performance and DRAM power consumption by applying our profiling overhead model to Equation 8. We compare REAPER with both online brute-force profiling and the ideal profiling mechanism that incurs no performance or energy overhead. The ideal mechanism mirrors the approach taken by prior works on refresh rate reduction [61, 63, 76, 79, 95, 96], which assume that *offline* profiling is sufficient. Considerable evidence against the sufficiency of offline profiling is provided by both prior work on DRAM characterization [42, 62] and our own observations (Sections 5.3 and 6.2.1). We exclude ECC-scrubbing based mechanisms from our evaluations due to their passive approach to failure profiling, which cannot guarantee failure coverage in the same way as an active profiling mechanism, as discussed in Section 3.2.

Figure 13 shows the results of this analysis, demonstrating the overall system performance improvement (top) and DRAM power consumption reduction (bottom) due to operating with a longer refresh interval across all of our simulated workloads for a variety of DRAM sizes each consisting of 32 individual DRAM chips of the specified capacity (8-64Gb). Each triplet of boxes presents results for online brute-force profiling, REAPER, and the ideal profiling mechanism, in order from left to right. Results are shown for various lengthened refresh intervals from 128ms to 1536ms and for the case of no refresh, which is shown as a single box. For each refresh interval, the profiling parameters used to compute overhead are obtained using the experimental data for failure rates at 45°C as shown in Section 5. Profile longevity is estimated in the best case for each configuration, assuming that the profilers achieve *full coverage* each time they are run. This assumption is reasonable for profilers achieving high (e.g., 99%) coverage and allows us to decouple the results from failure rates of specific DRAM chips. Results are all normalized to the base performance or power consumption at the default refresh interval of 64ms without profiling. The boxes represent the distribution of benchmarks from the 25th to the 75th percentiles, the whiskers show the data range, and the orange and black lines represent medians and means, respectively. We make four major observations based on these results.

First, REAPER enables a high performance improvement and a high DRAM power reduction by reliably increasing the refresh interval. REAPER enables 512ms as the *best overall operating point* across all evaluated DRAM chip sizes, providing an average of 16.3% (maximum 27.0%) performance improvement and an average of 36.4% (maximum 47.4%) power reduction for 64Gb chips. This is very close to the average performance gain of 18.8% (31.2% maximum) and average power reduction of 41.3% (54.1% maximum) that comes with eliminating all refreshes (rightmost "no ref" bars).

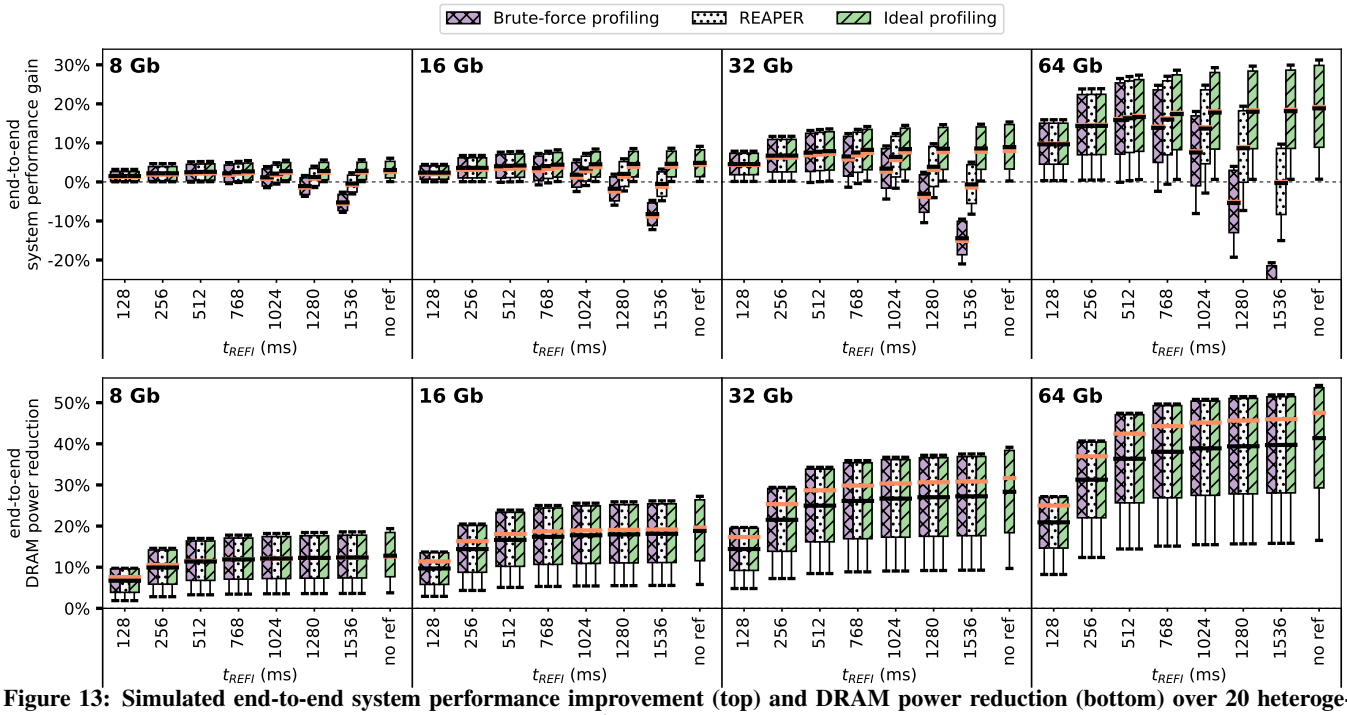

Figure 13: Simulated end-to-end system performance improvement (top) and DRAM power reduction (bottom) over 20 heterogeneous 4-core workloads for different refresh intervals at 45°C, taking into account online profiling frequency and profiling overhead.

Second, REAPER significantly outperforms the brute-force mechanism, especially at high refresh intervals (>= 1024ms). While the ideal profiling mechanism enables increasing gains with longer refresh intervals, both REAPER and brute-force profiling overheads become more significant for refresh intervals beyond 1024ms. This is due to the high VRT failure accumulation rate (Section 5.3) at long refresh intervals, which requires a high online profiling frequency. However, at such high refresh intervals, REAPER preserves much more of the ideal profiling benefit than does brute-force profiling. For example, with 64Gb DRAM chips, at a refresh interval of 1024ms, using REAPER provides 13.5% average (24.7%) maximum performance improvement while using brute-force profiling provides only 7.5% average (18% maximum) performance improvement. REAPER's performance benefit over brute-force profiling increases with longer refresh intervals because REAPER can sustain higher online profiling frequencies better, showing that REAPER is superior to brute-force profiling.

For refresh intervals below 512ms, both REAPER and brute-force profiling provide benefits that are very close to that of ideal profiling since the performance overhead of both mechanisms is low. This is due to the low VRT failure accumulation rate observed in Section 5.3 for short refresh intervals, resulting in a high profile longevity. However, short refresh intervals do *not* provide the full benefits of employing longer refresh intervals, so we would like to enable as high a refresh interval as possible with as low overhead as possible.

Third, REAPER enables high performance operation at very long refresh intervals that were previously unreasonable to operate the system at. For refresh intervals longer than 512ms, the high online profiling frequency means that profiling overhead becomes significant (as supported by Figure 11), and this overhead eventually results in overall performance degradation. At a refresh interval of 1280ms, the performance impact of profiling becomes clearly visible: using brute-force profiling with 64Gb DRAM chips leads to overall system performance *degradation* (-5.4% on average). However, REAPER still maintains significant performance benefit (8.6% on average) at 1280ms, demonstrating that REAPER enables longer refresh intervals that were previously unreasonable to operate the system at.

Fourth, both REAPER and brute-force profiling have negligible impact on overall DRAM power consumption across all refresh intervals and DRAM chip sizes. While Figure 12 shows that REAPER consumes significantly less power than brute-force profiling at the same configuration, profiling power consumption is a very small fraction of total DRAM power consumption to begin with. This means that although the profiling process has a large effect on overall system performance, it does *not* contribute significantly to DRAM power consumption. Thus, both REAPER and brute-force profiling are effective at greatly reducing DRAM power consumption by enabling a longer refresh interval.

We can use the results of Figure 13 to estimate the benefits obtained by using *any* retention failure mitigation mechanism. For example, ArchShield [76] extends the default refresh interval up to 1024ms at the cost of approximately 1% overall system performance (Section 5.1 of [76]). The overall system performance improvement can be estimated by subtracting ArchShield's reported performance cost (in [76]) from the ideal profiling performance gain (in Figure 13), resulting in an overall performance improvement of 15.7% on average (28.2% maximum) using 64Gb DRAM chips at a refresh interval of 1024ms. However, as we show the need for an online profiling mechanism (Section 6.2), the actual overall performance benefit must be adjusted to account for profiling overhead. We observe performance improvements of 6.5% on average (17% maximum) when ArchShield is combined with brute-force profiling, and 12.5% on average (23.7% maximum) when it is combined with REAPER. Thus, we find that using REAPER leads to an average performance improvement of 5.6% over using brute-force profiling. A similar analysis can be conducted using other state-of-the-art retention failure mitigation mechanisms to estimate overall system speedup and DRAM power consumption.

We conclude that REAPER is an effective and low-overhead profiling mechanism that enables high-performance operation at very

long refresh intervals (beyond 1024ms) that were previously not reasonable to employ due to the high associated profiling overhead. As a caveat to the data presented in this section, it is important to note that Figures 11, 12, and 13 are all based on specific assumptions (e.g., 45°C temperature, 2.5x performance improvement of reach profiling vs. online brute-force profiling, 32 chips per DRAM module, 100% coverage by profiling, 20 randomly-formed heterogeneous workload mixes) and we are not covering the entire design space with these results. This means that there likely exist other conditions under which REAPER may perform even better, including both different temperatures, at which we would expect failure rates and therefore profile longevity to change, and varying system requirements (e.g., choice of retention failure mitigation mechanism, target profiling coverage and false positive rate, target profile longevity, target UBER, etc.). These conditions may result in different choices of reach profiling parameters, which can lead to a higher-than-2.5x performance improvement for reach profiling over brute-force profiling. This increase in reach profiling performance translates directly to reduction in profiling overhead, which in turn translates to even greater end-to-end system performance improvement and greater DRAM power reduction than presented here.

8 RELATED WORK

To our knowledge, this is the first work that 1) analyzes and presents experimental data on the data retention characteristics of real LPDDR4 DRAM chips, 2) proposes a viable mechanism for profiling DRAM retention failures quickly and efficiently, 3) presents a rigorous analysis framework for understanding and evaluating merits of retention time profiling mechanisms, and 4) experimentally characterizes the complex tradeoff space of retention time profiling mechanisms in terms of the three major metrics we introduce for profiling: coverage, false positive rate, and profiling runtime.

As part of our analysis framework, we provide, for the first time, retention time data from 368 LPDDR4 DRAM devices. Many recent works conduct detailed experimental studies of real DRAM chips in terms of their reliability, data retention, and latency behavior [20–22, 25, 33, 38–40, 42–44, 48, 49, 54–56, 62, 70, 86, 92, 93, 98, 99]. Similarly, many recent works conduct detailed experimental studies of the reliability and data retention behavior of real NAND flash memory chips [9–19, 29, 65, 66, 69, 78, 85]. None of these works analyze and present data from real LPDDR4 DRAM chips, which we do in this paper.

We have already discussed and analyzed various retention failure mitigation mechanisms in Section 3.1. This proliferation of works [61, 63, 76, 79, 95, 96] reflects the importance of the refresh problem, which is also a major scaling challenge for modern DRAM [41, 68]. AVATAR [81] proposes a profiling technique to enable these refresh reduction mechanisms, but as discussed in Section 3.2, it does not guarantee finding the full set of failing cells in the presence of data pattern dependence. REAPER solves this by testing with numerous data patterns at longer refresh intervals and/or higher temperatures to quickly identify the set of failing cells.

A scheme that would only refresh each cell right before it is about to experience a retention failure is the ideal refresh scheme for DRAM, but it is very difficult to employ in the field because of numerous complications in DRAM cells. Many works show how retention failures also depend on many different variables [40–44, 49, 60, 62, 81, 82, 98, 101] and state that it is a very difficult problem to find a comprehensive set of failing cells [60, 62, 77]. Several prior works analyze retention failures in real DRAM devices [6, 25, 33, 38, 42–44, 49, 56, 62, 81, 86, 93, 95] and propose methods to approach benefits of the ideal refresh scheme, but they do *not* comprehensively develop efficient methods for or explore the tradeoff space of *retention failure profiling*. To our knowledge, we are the first explore the complex tradeoff space of profiling the failing cells and the first to propose an efficient method for finding such failing cells, thereby enabling many proposed mechanisms that depend on accurate and efficient identification of failing cells.

Other works proposed reducing refresh overheads *without* requiring knowledge of cells that fail at larger refresh intervals [5, 6, 8, 26, 27, 30, 35, 40, 45, 46, 64, 67, 73, 74, 80, 94], e.g., by better scheduling, taking advantage of software and hardware access patterns to reduce refreshes, and reducing refreshes for non-critical data. These works can be used together with the more aggressive refresh reduction techniques that REAPER enables.

9 CONCLUSION

This paper rigorously explores the complex tradeoff space associated with DRAM retention profiling mechanisms using new experimental data from 368 modern LPDDR4 DRAM chips. In an effort to develop a rigorous understanding of how retention failure profiling can be made viable for increasing the refresh interval, we experimentally characterize DRAM chips at various temperatures and refresh intervals and analyze the probability of failure of different cells in DRAM as well as tradeoffs of retention time profiling. Following rigorous analysis of the collected data, we propose *reach profiling*, a technique whose key idea is to profile DRAM at a longer refresh interval and/or a higher temperature relative to the target refresh interval/temperature in order to quickly discover an overwhelming majority of all possible failing cells at the target conditions. We show that reach profiling enables significant system performance improvement and DRAM power reduction, outperforming the brute-force approach and enabling high-performance operation at longer refresh intervals that were previously unreasonable to employ due to the high associated profiling overhead. We conclude that reach profiling can be an enabler for many past and future DRAM refresh reduction mechanisms. We also hope that the new experimental characterization and the analysis of the data retention characteristics of modern LPDDR4 DRAM devices presented in this work will serve as an enabler for others to develop new techniques to tackle the difficult yet critical problem of DRAM refresh.

ACKNOWLEDGMENTS

We thank our shepherd Alper Büyüktosunoğlu, anonymous reviewers, and SAFARI group members for feedback.

REFERENCES

[1] "DRAMPower Source Code," https://github.com/ravenrd/DRAMPower.
[2] "Ramulator Source Code," https://github.com/CMU-SAFARI/ramulator.
[3] "Standard Performance Evaluation Corporation," http://www.spec.org/cpu2006.
[4] "SAS SSDs for Enterprise: High Performance with Extreme Reliability," 2012.
[5] A. Agrawal *et al.*, "Refrint: Intelligent Refresh to Minimize Power in On-chip Multiprocessor Cache Hierarchies," in *HPCA*, 2013.
[6] S. Baek *et al.*, "Refresh Now and Then," in *TC*, 2014.
[7] I. Bhati *et al.*, "DRAM Refresh Mechanisms, Penalties, and Trade-Offs," in *TC*, 2016.
[8] I. Bhati *et al.*, "Coordinated Refresh: Energy Efficient Techniques for DRAM Refresh Scheduling," in *ISPLED*, 2013.
[9] Y. Cai *et al.*, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.
[10] Y. Cai *et al.*, "FPGA-Based Solid-State Drive Prototyping Platform," in *FCCM*, 2011.
[11] Y. Cai *et al.*, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.
[12] Y. Cai *et al.*, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.
[13] Y. Cai *et al.*, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.
[14] Y. Cai *et al.*, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.
[15] Y. Cai *et al.*, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.
[16] Y. Cai *et al.*, "Flash Memory SSD Errors, Mitigation, and Recovery," in *Proceedings of the IEEE*, to appear in 2017.
[17] Y. Cai *et al.*, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.
[18] Y. Cai *et al.*, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," in *ITJ*, 2013.
[19] Y. Cai *et al.*, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," in *SIGMETRICS*, 2014.

[20] K. Chandrasekar *et al.*, "Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization," in *DATE*, 2014.
[21] K. K. Chang, "Understanding and Improving Latency of DRAM-Based Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2017.
[22] K. K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.
[23] K. K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[24] K. K. Chang *et al.*, "Low-cost Inter-linked Subarrays (LISA): Enabling Fast Inter-subarray Data Movement in DRAM," in *HPCA*, 2016.
[25] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.
[26] Z. Cui *et al.*, "DTail: A Flexible Approach to DRAM Refresh Management," in *SC*, 2014.
[27] P. G. Emma *et al.*, "Rethinking Refresh: Increasing Availability and Reducing Power in DRAM for Cache Applications," in *MICRO*, 2008.
[28] S. Eyerman and L. Eeckhout, "System-level Performance Metrics for Multiprogram Workloads," in *IEEE Micro*, 2008.
[29] A. Fukami *et al.*, "Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices," in *Digital Investigation*, 2017.
[30] M. Ghosh and H.-H. S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-stacked DRAMs," in *MICRO*, 2007.
[31] T. Hamamoto *et al.*, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," in *TED*, 1998.
[32] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[33] H. Hassan *et al.*, "SoftMC: A Flexible and Practical Open-source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.
[34] C.-S. Hou *et al.*, "An FPGA-based Test Platform for Analyzing Data Retention Time Distribution of DRAMs," in *VLSI-DAT*, 2013.
[35] C. Isen and L. John, "ESKIMO: Energy Savings Using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM Subsystem," in *MICRO*, 2009.
[36] ITRS, "International Technology Roadmap for Semiconductors Executive Summary," 2013, http://www.itrs2.net/2013-itrs.html.
[37] JEDEC, "Low Power Double Data Rate 4 (LPDDR4) SDRAM Specification," *JEDEC Solid State Technology Association*, 2014.
[38] M. Jung *et al.*, "Reverse Engineering of DRAMs: Row Hammer with Crosshair," in *MEMSYS*, 2016.
[39] M. Jung *et al.*, "Optimized Active and Power-down Mode Refresh Control in 3D-DRAMs," in *VLSI-SoC*, 2014.
[40] M. Jung *et al.*, "Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs," in *MEMSYS*, 2015.
[41] U. Kang *et al.*, "Co-architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[42] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[43] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[44] S. Khan *et al.*, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," in *CAL*, 2016.
[45] J. Kim and M. C. Papaefthymiou, "Dynamic Memory Design for Low Data-retention Power," in *PATMOS*, 2000.
[46] J. Kim and M. C. Papaefthymiou, "Block-based Multiperiod Dynamic Memory Design for Low Data-retention Power," in *TVLSI*, 2003.
[47] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," in *EDL*, 2009.
[48] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon University, 2015.
[49] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[50] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.
[51] Y. Kim *et al.*, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.
[52] Y. Kim *et al.*, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[53] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," in *CAL*, 2015.
[54] D. Lee *et al.*, "Adaptive-latency DRAM: Optimizing DRAM Timing for the Common-case," in *HPCA*, 2015.
[55] D. Lee, "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity," Ph.D. dissertation, Carnegie Mellon University, 2016.
[56] D. Lee *et al.*, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," in *SIGMETRICS*, 2017.
[57] D. Lee *et al.*, "Tiered-latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[58] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[59] M. J. Lee and K. W. Park, "A Mechanism for Dependence of Refresh Time on Data Pattern in DRAM," in *EDL*, 2010.
[60] Y. Li *et al.*, "DRAM Yield Analysis and Optimization by a Statistical Design Approach," in *CSI*, 2011.
[61] C.-H. Lin *et al.*, "SECRET: Selective Error Correction for Refresh Energy Reduction in DRAMs," in *ICCD*, 2012.
[62] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.
[63] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[64] S. Liu *et al.*, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *ASPLOS*, 2012.
[65] Y. Luo *et al.*, "WARM: Improving NAND Flash Memory Lifetime with Write-Hotness Aware Retention Management," in *MSST*, 2015.
[66] Y. Luo *et al.*, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," in *JSAC*, 2016.
[67] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-reliability Memory," in *DSN*, 2014.
[68] J. A. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-access Memory (DRAM)," in *IBM JRD*, 2002.
[69] J. Meza *et al.*, "A Large-scale Study of Flash Memory Errors in the Field," in *SIGMETRICS*, 2015.
[70] 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.
[71] R. Micheloni *et al.*, "Apparatus and Method Based on LDPC Codes for Adjusting a Correctable Raw Bit Error Rate Limit in a Memory System," US Patent 9,092,353. 2015.
[72] Y. Mori *et al.*, "The Origin of Variable Retention Time in DRAM," in *IEDM*, 2005.
[73] J. Mukundan *et al.*, "Understanding and Mitigating Refresh Overheads in High-density DDR4 DRAM Systems," in *ISCA*, 2013.
[74] P. Nair *et al.*, "A Case for Refresh Pausing in DRAM Memory Systems," in *HPCA*, 2013.
[75] P. J. Nair *et al.*, "Refresh Pausing in DRAM Memory Systems," in *TACO*, 2014.
[76] P. J. Nair *et al.*, "ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates," in *ISCA*, 2013.
[77] Y. Nakagome *et al.*, "The Impact of Data-line Interference Noise on DRAM Scaling," in *JSSC*, 1988.
[78] I. Narayanan *et al.*, "SSD Failures in Datacenters: What, When and Why?" in *SIGMETRICS*, 2016.
[79] T. Ohsawa *et al.*, "Optimizing the DRAM Refresh Count for Merged DRAM/logic LSIs," in *ISLPED*, 1998.
[80] K. Patel *et al.*, "Energy-Efficient Value-based Selective Refresh for Embedded DRAMs," in *PATMOS*, 2005.
[81] M. K. Qureshi *et al.*, "AVATAR: A Variable-retention-time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[82] P. J. Restle *et al.*, "DRAM Variable Retention Time," in *IEDM*, 1992.
[83] S. Rixner *et al.*, "Memory Access Scheduling," in *ISCA*, 2000.
[84] K. Saino *et al.*, "Impact of Gate-induced Drain Leakage Current on the Tail Distribution of DRAM Data Retention Time," in *IEDM*, 2000.
[85] B. Schroeder *et al.*, "Flash Reliability in Production: The Expected and the Unexpected." in *FAST*, 2016.
[86] B. Schroeder *et al.*, "DRAM Errors in the Wild: a Large-scale Field Study," in *SIGMETRICS*, 2009.
[87] V. Seshadri, "Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems," Ph.D. dissertation, Carnegie Mellon University, 2016.
[88] V. Seshadri *et al.*, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.
[89] V. Seshadri *et al.*, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," in *arXiv*, 2016.
[90] V. Seshadri *et al.*, "Gather-scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," in *MICRO*, 2015.
[91] A. Snavely and D. M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Mutlithreading Processor," in *ASPLOS*, 2000.
[92] V. Sridharan *et al.*, "Memory Errors in Modern Systems: The Good, the Bad, and the Ugly," in *ASPLOS*, 2015.
[93] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *SC*, 2012.
[94] J. Stuecheli *et al.*, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *MICRO*, 2010.
[95] R. K. Venkatesan *et al.*, "Retention-aware Placement in DRAM (RAPID): Software Methods for Quasi-non-volatile DRAM," in *HPCA*, 2006.
[96] J. Wang *et al.*, "ProactiveDRAM: A DRAM-initiated Retention Management Scheme," in *ICCD*, 2014.
[97] Y. Wang *et al.*, "RADAR: A Case for Retention-aware DRAM Assembly and Repair in Future FGR DRAM Memory," in *DAC*, 2015.
[98] C. Weis *et al.*, "Retention Time Measurements and Modelling of Bit Error Rates of Wide I/O DRAM in MPSoCs," in *DATE*, 2015.
[99] C. Weis *et al.*, "Thermal Aspects and High-Level Explorations of 3D Stacked DRAMs," in *ISVLSI*, 2015.
[100] B. Wicht *et al.*, "Yield and Speed Optimization of a Latch-type Voltage Sense Amplifier," in *JSSC*, 2004.
[101] D. S. Yaney *et al.*, "A Meta-stable Leakage Phenomenon in DRAM Charge Storage-Variable Hold Time," in *IEDM*, 1987.
[102] 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," US Patent 5,630,096. 1997.