DStress: Automatic Synthesis of DRAM Reliability Stress Viruses using Genetic Algorithms

Lev Mukhanov
Queen's University Belfast, UK
l.mukhanov@qub.ac.uk

Dimitrios S. Nikolopoulos
Virginia Tech, USA
dsn@vt.edu

Georgios Karakonstantis
Queen's University Belfast, UK
g.karakonstantis@qub.ac.uk

***Abstract*—Failures become inevitable in DRAM devices, which is a major obstacle for scaling down the density of cells in future DRAM technologies. These failures can be detected by specific DRAM tests that implement the data and memory access patterns having a strong impact on DRAM reliability. However, the design of such tests is very challenging, especially for testing DRAM devices in operation, due to an extremely large number of possible cell-to-cell interference effects and combinations of patterns inducing these effects.**

In this paper, we present a new framework for the synthesis of DRAM reliability stress viruses, DStress. This framework automatically searches for the data and memory access patterns that induce the worst-case DRAM error behavior regardless the internal DRAM design. The search engine of our framework is based on Genetic Algorithms (GA) and a programming tool that we use to specify the patterns examined by GA. To evaluate the effect of program viruses on DRAM reliability, we integrate DStress with an experimental server where 72 DRAM chips can operate under various operating parameters and temperatures.

We present the results of our 7-month experimental study on the search of DRAM reliability stress viruses. We show that DStress finds the worst-case data pattern virus and the worst-case memory access virus with probabilities of $1 - 4 \times 10^{-7}$ and 0.95, respectively. We demonstrate that the discovered patterns induce by at least 45% more errors than the traditional data pattern micro-benchmarks used in previous studies. We show that DStress enables us to detect the marginal DRAM operating parameters reducing the DRAM power by 17.7 % on average without compromising reliability. Overall, our framework facilitates the exploration of new data patterns and memory access scenarios increasing the probability of DRAM errors, which is essential for improving the state-of-the-art DRAM testing mechanisms.

***Index Terms*—DRAM, reliability, program synthesis, viruses**

I. Introduction

The growing size of data processed in Cloud and Edge data centers drive the need for increasing the density of DRAM chips within server-grade systems. However, increasing this density is extremely challenging due to DRAM failures that become inevitable when scaling down the size of DRAM cells [43]. Moreover, DRAM reliability may vary significantly across different chips because of the manufacturing process, especially for the nanometer technologies [48]. This made DRAM devices the primary source of hardware failures in data centers after hard drives [104].

To protect against failures, hardware vendors are forced to use the pessimistic margins for the DRAM operating parameters, such as the refresh rate, voltage and timing parameters [13], [40], [48], [61], [73], [76], [96]. The use of such pessimistic margins results in a degradation of DRAM energy and performance efficiency. It is predicted that DRAMs will become one of the main energy consumers on servers in near future for these reasons [61]. This challenge has turned attention to improving the energy efficiency of server-grade DRAM devices by scaling down essential circuit parameters, while trying to avoid possible DRAM failures [2], [12], [13], [23], [34], [44], [45], [47], [48], [51], [55], [67], [69], [74], [77], [80], [94], [96]. In particular, several studies in this area have minimized the refresh power by identifying the error-prone cells with a small retention time for memory in operation (in the field) and by setting a shorter refresh period for such cells [61], [80], [106]. In contrast, other works have suggested using a supervised data mapping to avoid the errors induced by these cells [60], [84], [102]. All these studies use micro-benchmarks to identify the retention time of cells [48], [60]. However, it was shown that DRAM error behavior varies significantly across different data and memory access patterns expressed by running applications [1], [35], [47], [48], [48], [49], [51], [77], [80], [91], [96], and there is little evidence that the applied micro-benchmarks reveal all error-prone cells. Moreover, these micro-benchmarks need to map data to adjacent cells to provide accurate retention time estimates [47], [106]. This implies that the physical design of DRAM chips and the actual memory cell array layout are known. However, vendors never disclose this information, what makes third-party DRAM testing very challenging.

Understanding the effect of data and access patterns on DRAM errors is essential for exploring the worst-case DRAM error behavior. Previous research works may have demonstrated the effect of some patterns on DRAM errors. However, to our knowledge, none of these works provided a methodology for finding the worst-case DRAM error behavior, systematically investigating the patterns that trigger such a behavior. This methodology would enable vendors and users to identify the patterns that induce errors on specific DRAM chips and thus improve the existing DRAM testing procedures. Moreover, the methodology would facilitate the synthesis of DRAM reliability stress viruses, which can be used for predicting an abnormal behavior of DRAM devices in advance. Such a prediction is extremely important in the state-of-the-art data centers [28], [28], [36], [85], where any possible DRAM malfunction should be revealed in advance to avoid service failures. Although all modern computing devices check memory before starting and use the DRAM chips with integrated

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00035

Built-In Self-Test (BIST) mechanisms [19], the applied testing procedures often use simple data patterns [27], [106]. While some errors may manifest only when sophisticated testing procedures, which stress various memory access scenarios, are applied [54]. Thus, the default DRAM testing mechanisms may be not enough for data centers to reveal the DRAM devices that could become erroneous in future [28], [85].

The goal of our study is to develop and investigate a mechanism for synthesizing the data and memory access patterns that induce the worst-case DRAM error behavior. To this end, we implement a new framework, DStress, that uses Genetic Algorithms (GA) to automatically generate the DRAM program viruses increasing the probability of DRAM errors. We integrate DStress with a real server and a special thermal testbed to stress the reliability of DRAM devices at different temperatures.

Our contribution can be summarized as follows:

- We develop a new framework, DStress, for the automatic search of DRAM reliability stress viruses using Genetic Algorithms. To enable the search, we integrate this framework with a real server where DRAM error behavior can be monitored using the ECC (Error Correction Codes) hardware under controlled temperature. To the best of our knowledge, we are the first to present a software-hardware framework for automatic synthesizing DRAM reliability stress viruses that does not require any knowledge of DRAM internals.
- We develop a novel programming tool, integrated into DStress, to specify the type of data and memory access patterns that should be examined by the GA search engine.
- We present the results of our 7-month experimental study. In particular, we demonstrate the worst-case data and access patterns discovered by DStress that maximize the probability of single-bit and multi-bit errors. We show that the discovered data patterns, which are reported for the first time, induce at least 45 % more errors than the traditional micro-benchmarks used in previous studies. We make several important observations, such as: i) the worst-case data patterns do not change with temperature; ii) *'1100'* data pattern increases the probability of single-bit errors iii) the data patterns that induce single-bit errors differ from the patterns increasing the probability of multi-bit errors; iv) memory access viruses have a stronger impact on DRAM reliability than the data pattern viruses. We demonstrate that DStress finds the worst-case data pattern and the worst-case memory access pattern with probabilities of $1 - 4 \times 10^{-7}$ and 0.95, respectively.
- We discuss possible use cases of the DRAM stress viruses discovered by DStress. In particular, we show that these viruses can be effectively used for detecting the margins of the DRAM operating parameters that enable us to reduce the memory power by 17.7 % on average without compromising DRAM reliability. We believe that DStress can be efficiently applied by vendors to investigate the effect of various memory access scenarios, including the scenarios of "rowhammer" attacks, on DRAM reliability and improve the state-of-the-art DRAM testing mechanisms.

The rest of the paper is organized as follows. Section II presents the background. Section III introduces the design of DStress, while Section IV discusses the experimental server. Section V demonstrates the results of our experimental campaign. Section VI presents use cases. Section VII discusses the related work. Finally, conclusions are drawn in Section VIII.

II. Background

Dram organization. The modern memory subsystems have several channels to access to a number of DRAM modules, or Dual In-line Memory Modules (DIMMs). Data is transferred from/to a DIMM using a memory controller, which sends commands and accessed memory addresses to the DIMM, as shown in Figure 1a. Each DIMM implements *an address decoder* that translates the accessed addresses to the physical memory layout addresses that point to the DRAM cells containing data. DRAM cells are combined into *memory array* which is stored in several DRAM chips. Note that DIMMs usually have two ranks (sides) with DRAM chips, i.e. one memory array per rank. Memory array is organized into banks that consist of two-dimensional arrays based on rows and columns. Each cell contains a capacitor and an access transistor, and all access transistors are in a row connected by a wire called *wordline*, controlling access to the capacitors. When DRAM is accessed, an entire row is read, and sense-amplifiers measure the charge on each capacitor through *bitlines*, which are connected to access transistors [12], [48], [49] (see Figure 1a). Then the data from the row is transferred to *Row buffer* for sending it to the memory controller. Importantly, there are two different types of DRAM cells: *true-cells* and *anti-cells* [46], [108]. True-cells in the charge state store a logic value of '1', while anti-cells store a logic value of '0' in the same state. As a result, true-cells manifest errors where a logic value of '1' changes to '0', while anti-cell trigger errors where '0' a logic a value of changes to '1'.

Mapping data to memory physical layout. Mapping data to memory physical layout has a strong impact on the performance and reliability of DRAMs [106]. Figure 1a illustrates how 8-KBytes chunks of data are mapped to physical memory layout for 8GB DDR3 DRAM devices on a server used in our study (see Section IV). We see that each 8-KByte data chunk is mapped to exactly one DRAM row. However, consecutive 8-KByte data chunks are mapped to rows in different banks. For example, the first 8-KByte chunk is mapped to the first row of *Bank1*, while the second row is mapped to the first row of *Bank2*. Thus, the first 8-KByte chuck of data, the 9-th data chunk and the 17-th data chunk are mapped to the first three adjacent rows of the first bank, i.e. *Row1.Bank1*, *Row2.Bank1* and *Row3.Bank1*. Note that the elements within each 8-KByte chunk are mapped to the same row but different columns. The function for mapping of addresses to physical memory layout for the DIMMs used in our study is shown in Figure 2.

In some rows, the order of the cells can be changed due to scrambling or row remapping [49], [106]. For example, in Figure 1a, the physical memory layout address space is scrambled for *Rown.Bank1*. In this row, the right neighbor of

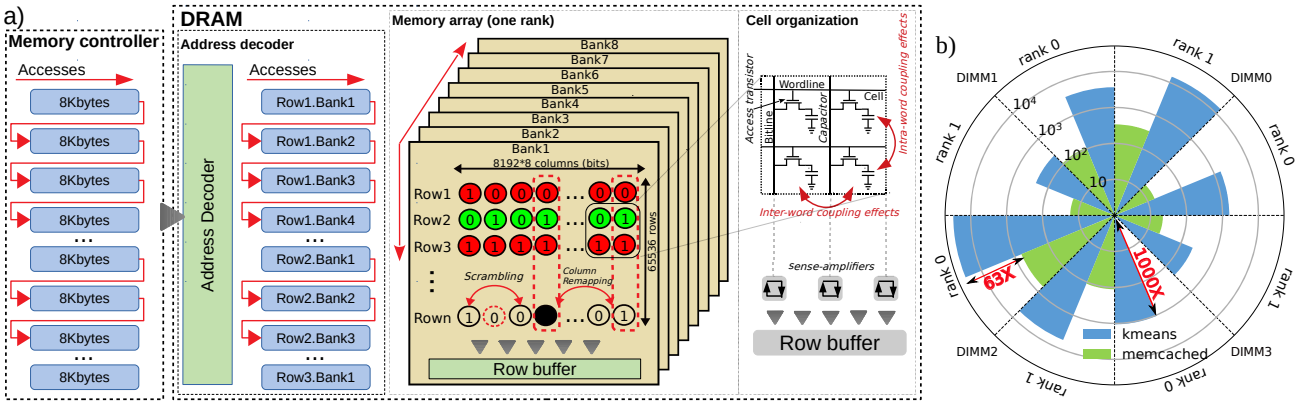

Fig. 1. a) DRAM organization (DDR3 8Gb,1 Gig);b) The number of 1-bit errors detected for *kmeans* and *memcached* when DRAM operates under 2.283 s T_{REFP} and lowered V_{DD} (1.428 V) at 50°C.

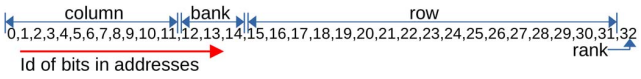

Fig. 2. The mapping function of 64-bit aligned addresses to physical memory layout (8 GB DDR3 DIMM).

the first cell should be located in the second column. However, due to the data scrambling, this neighbor is a cell from the third column. Moreover, vendors may remap DRAM columns with faulty cells to the redundant columns which do not have such cells. For example, in Figure 1a, the 4-th column of *Bank1* which has a faulty cell is remapped to the last column of this bank. Thus, two consecutive addresses do not always point to the data from neighbour cells.

DRAM operating parameters. The main drawback of the DRAM technology is the limited *retention time* [60] of a cell's charge. To avoid any errors induced by the charge leakage over time, DRAMs employ an *Auto-Refresh* mechanism that recharges the cells in the array periodically. Conventionally, all DDR technologies adopt a *refresh period*, T_{REFP}, of 64 ms (or 32 ms) for refreshing each cell of DIMMs; although, many cells may have a much higher retention time than this T_{REFP} [40]. Another critical parameter for the operation of DRAMs is supply voltage. The nominal supply *voltage*, V_{DD}, of DRAM chips is chosen conservatively by vendors to ensure that each chip operates correctly under a wide range of conditions [13], [18]. Apart from the above circuit parameters, one of the main environmental conditions affecting the reliability of DRAM is temperature [33], [60].

Workload-dependent DRAM behavior. Previous studies have shown that the data and memory access pattern of a running program may significantly affect DRAM error behavior [48], [60]. The effect of data patterns is explained by the fact that the retention time of a cell depends on the value stored in this cell. While specific memory access patterns may induce the DRAM cell-to-cell interference [1], [51], [96], which increases the probability of some cells from neighbouring rows to leak charge. This effect has been widely exploited for "rowhammer" attacks [51], [71], [99]. Specifically, the data in *Row2.Bank1* from Figure 1a may be compromised when *Row1.Bank1* and *Row3.Bank1* are accessed too often. Thus, inherent program features that change the data and memory access pattern of a running workload may have a significant effect on DRAM reliability. Figure 1b illustrates how DRAM error behavior may vary when running different workloads. This figure shows the spatial and density distribution of the detected single-bit errors across 4 DIMMs (8 ranks) when running *kmeans* and *memcached* benchmarks for DRAM operating under relaxed T_{REFP} and V_{DD} at 50 ^oC (2-hour runs). We present the distribution as a polar plot, where $\theta - axis$ specifies DIMM slots and ranks, while $\rho - axis$ reflects the number of single-bit errors detected by hardware (ECC). We see that the number of errors manifested by the benchmarks in DIMM3/rank0 differs by 1000x.

DIMM-to-DIMM variation. It was shown that DRAM reliability may vary across chips [49], [60], [60]. This variation is due the manufacturing process [50] and the internal design of DRAM modules [46], [47], [49], [98]. For example, Figure 1b shows that the number of errors manifested by *memcached* in DIMM2/rank0 is 633x higher than the number of errors triggered by the same benchmark in DIMM3/rank0.

DRAM errors. DRAM errors may occur in the address decoder, read/write circuity and memory cell array [106]. To detect the errors vendors run specific test suites, such as MARCH tests and MATS tests [27], [106], which can reveal the vast majority of DRAM errors. However, these tests were originally designed to stress DRAM reliability at the inter-word level [54], [106] and identify coupling faults triggered by bits within a word (see Figure 1a). Thus, such tests may be not effective for detecting intra-word coupling faults [27], [106]. Moreover, the MARCH and MATS tests imply that the physical design of the actual memory cell array layout is known, since it is important to map data and access to the particular cells in a specific order for testing DRAM reliability. However, the physical layout is not known and it is not provided by vendors. Furthermore, in some particular rows the order of the cells can be changed due to scrambling or row remapping [46], [106] (see *DRAM Organization*). Thus, any testing of DRAM reliability by storing a specific data pattern in adjacent cells may be incorrect when the address mapping scheme for the scrambling process is not known. Moreover, vendors may apply some advanced mapping techniques which

```
->parameters
 $$$_ARRAY1_VEC_$$$ [N1][DB1,UP1]
 $$$_ARRAY2_VEC_$$$ [N2][0,N1]
 $$$_VAR1_$$$ [DB3,UP3]
<-parameters

->global_data
 volatile unsigned long long var1[]=$$$_ARRAY1_VEC_$$$;
 volatile unsigned long long var2[]=$$$_ARRAY2_VEC_$$$;
<-global_data

->local_data
 unsigned long long var3=$$$_VAR1_$$$;
 volatile unsigned long long* temp_array;
 int i,j;
<-local_data

->body
 templ_array = (unsigned long long*)(malloc(N1*sizeof(unsigned long long)));

 /* data pattern */
 for(i=0;i<N1;i++)
 {
   templ_array[i]=var[i];
 }

 /* access pattern*/
 j=0;
 while(j<var3)
 {
    for(i=0;i<N2;i++)
    {
       templ_array[var2[i]];
    }
    j++;
 }
<-body
```

Fig. 3. Example of a code generation template.

are not exposed to the user. As a result, the MARCH and MATS tests may be not effective for the characterization of DRAM reliability when running these tests without knowing the physical memory cell layout.

The goal of our work is to develop a mechanism that enables us to generate viruses to stress DRAM reliability and reveal the worst-case DRAM error behavior without any knowledge of the DRAM internals.

III. The Proposed Design

A. Pattern programming

To generate DRAM reliability stress viruses, we develop a new framework, DStress. This framework enables the user to define data and memory access patterns through specific variables using program templates and find those variable values that maximize the number of memory errors. To define these variables, we introduce several annotations in *C Programming Language*. Figure 3 shows an instance of a program template, which contains four sections: *parameters*, *global_data*, *local_data* and *body*. *Parameters* define the type of variables which will be changed by the search engine (see Section III-E). For example, *$$$_VAR1_$$$ [DB3,UB3]* defines the type of a scalar variable, *VAR1*, which values vary from *DB3* up to *UB3*. For instance, the bound range *[0,100]* defines *VAR1* which values vary from 0 up to 100. *$$$_ARRAY1_VEC_$$$ [N1][DB1][UB1]* defines the type of a vector variable, *ARRAY1*, consisting of $N1$ elements that vary from $DB1$ up to $UB1$.

In *global data*, two arrays are declared, i.e. *var1* and *var2*, which are initialized by the search engine following the types defined in *parameters*. These arrays are allocated in the heap (*global data*). In *local data*, we declare four variables of *standard C types* (*templ1_array*, *templ2_array*, *i* and *j*) and one scalar variable (*var3*) which are initialized by the search engine, as defined in *parameters*. Section *body* contains a C program that will be generated at the synthesis phase (see Section III-E). In this particular example, we allocate memory for *templ_array*. Then we initialize *templ_array* using the data stored in *var1*. In this program, *var1* defines the pattern of the data which we store in *templ_array*. In the last loop, we access to the elements of *templ_array* in the order provided in *var2*, which defines the access pattern.

B. Data and memory access patterns

Data patterns: In our study, we first search for a 64-bit word (data pattern) that maximizes the number of DRAM errors when we fill the memory with this word. To this end, we implement a template where we define the type of such a pattern as a binary array with a size of 64 elements. Thus, the size of the search space is 2^{64}, which motivates the use of GA since it is impossible to check every possible combination within a reasonable period of time.

It was previously shown that when accessing a row often, some cells from the adjacent rows may leak charge quickly due to the cell-to-cell interference [51], [65], [82]. We assume that a specific data pattern in a row may induce such an interference in the neighbouring rows. For example, in Figure 1, the data in *Row1.Bank1* and *Row3.Bank1* may induce errors in *Row2.Bank1*. To investigate this, we designed a specific template that searches for the data pattern for all rows such as *Row1.Bank1*, *Row2.Bank1* and *Row3.Bank1* where *Row2.Bank1* is erroneous (see Figure 1). In our experiments, we use DDR3 8GB DIMMs (1 Gig) where the size of each row is 8 KByte. Thus, the size of the template is 24 KByte. Note that in this template we fill only the erroneous rows and their neighbouring rows (see Section V). Similar to the 64-bit data pattern, we define the type of the 24-KByte data pattern as a binary array with a size of 24576 elements.

To investigate if the data in the adjacent rows from different banks affect the probability of error manifestation in erroneous rows, we designed the template where the size of the data pattern is 512 KByte. We fill with this data pattern the rows from all the banks next to the erroneous rows. For example, in Figure 1 such rows are: *Row1.Bank1*, *Row1.Bank2*, ..., *Row1.Bank8*, *Row2.Bank1*, *Row2.Bank2*, ..., *Row2.Bank8*, *Row3.Bank1* (17 rows in total). The 512-KByte data pattern is defined as a binary array with a size of 524288 elements.

Access patterns: We first investigate which rows that are next to the erroneous rows should be repeatedly accessed to increase the probability of DRAM errors. By accessing to these rows, we induce the cell-to-cell interference between rows and thus increase the probability of DRAM errors. Based on these grounds, we design a specific template where 32 rows before and 32 rows after the erroneous rows, 64 rows in total, are repeatedly accessed. In this template, we define the access pattern also as a binary array with a size of 64 elements, i.e.

if the i-th row is accessed, then the i-the element in the array is '1' ('0' otherwise).

Finally, we investigate if accesses to specific elements of the rows next to the erroneous rows increase the risk of DRAM errors. However, in this experiment, to reduce the search space, we consider only 16 neighbouring rows. We calculate indexes of the accessed elements as:

$$a_i \times x + b_i \quad (1)$$

where x is a variable being changed in a loop from 0 to 65536 with increments of 1; i is the index of accessed rows ($i \in [1, 16]$). In this experiment, we search for the a_i and b_i for each row (i) which increase the probability of DRAM errors. To find these coefficients, we define the memory access pattern as an array of integer elements with the size of 32 elements. In this array, the first 16 elements correspond to a_i, while the second 16 elements correspond to b_i. Note that a_i and b_i may vary from 0 to 65536, however, to accelerate the search process, we limit these coefficients: $a_i, b_i \in$[0,20].

C. The search criteria

Our goal is to find the memory access and data patterns that maximize the number of manifested DRAM errors. There are two types of DRAM errors that may occur: single-bit errors (when only one bit is corrupted per a 64-bit word) and multi-bit errors (when several bits are corrupted per a 64-bit word). Since the majority of DRAM errors are single-bit [46], [73], most server-grade processors implement ECC SECDED (Single Error Correction Double Error Detection) which automatically corrects single-bit errors (**CE**, Correctable Errors) and detect multi-bit errors (**UE**, Unocrrectable Errors). Note that ECC SECDED detects 100 % of 2-bit errors, while errors where more than 2 bit are corrupted may be not detected by ECC SECDED. Such errors manifest so called Silence Data Corruption (**SDC**s).

In this study, we investigate and search for the memory and access patterns that maximize the number of Correctable errors(**CE**s) and Uncorrectable Errors (**UE**s).

x_i \ y_i	1	0
1	a	b
0	c	d

TABLE I
OTUS EXPRESSION OF BINARY FEATURES

D. Processing phase

The overall workflow of DStress is presented in Figure 4. Our framework processes all the templates that are created by the user at the processing phase. In this phase, we make lexical, syntax and semantic analyses to extract variables which will be used by DStress for searching the worst-case memory and access patterns. The results of the analyses are passed to the next phase (*the synthesis phase*).

E. Synthesis phase

To find the memory access and data patterns that manifest the highest number of DRAM errors, we need to check all possible combinations of memory and access patterns. However, the extremely large number of such combinations renders the evaluation the effect of every possible memory and

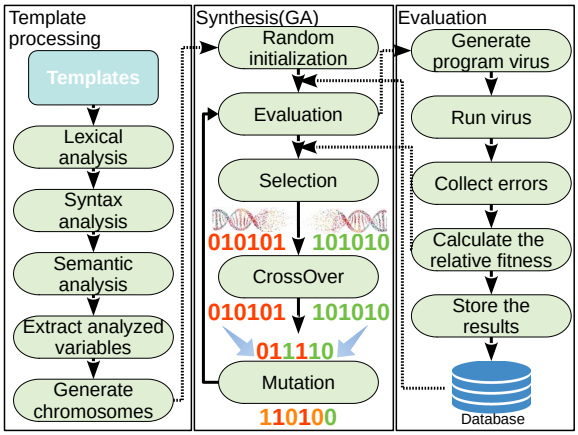

Fig. 4. Overview of DStress.

access pattern on DRAM reliability practically impossible. To this end, our framework formulates a Genetic Algorithm (GA), which implements a stochastic search to generate programs, or viruses, that maximize the number of manifested errors. GAs are stochastic search algorithms inspired by evolutionary biology to search for a solution to optimization problems. Similar to the natural selection process, the GA algorithm defines the following parameters.

Chromosomes. Chromosomes are used to encode information about parameters of the search. Particularly, in our framework, each chromosome defines a specific memory access and data pattern.

Selection. The algorithm selects chromosomes only when they fit the target evaluation function. Specifically, DStress searches for the chromosomes that maximize the number of CEs or UEs. GA measures this number for a specific chromosome by running the corresponding virus on a server and recording all the manifested errors (see Section III-F).

Mutation. GA varies chromosomes through mutation to expand the search space. In particular, the mutation operation stochastically changes parts of a chromosome, in our case, the data which defines data and memory access patterns.

Crossover. GA also generates new chromosomes through the crossover operation. The crossover operation stochastically combines different parts of parent chromosomes to generate new chromosomes (offsprings).

Generations. The number of offspring generations (or algorithm iterations) generated by GA is in principle unlimited. However, DStress interrupts GA when the latest offspring contains chromosomes which do not vary a lot, i.e. when the chromosomes are similar. This indicates that GA found a virus that maximizes the relative target function (the number of DRAM errors).

Similarity and Convergence Criteria: We measure the similarity between chromosomes in offsprings using the *Sokal & Michener* similarity function [87]. To formally define this metric, let us assume that there are two chromosomes $\vec{X} = (x_1, x_2, ...x_n)$ and $\vec{Y} = (y_1, y_2, ...y_n)$ where x_i and y_i are binary features of $\vec{X}$ and $\vec{Y}$, respectively. To measure the similarity between $\vec{X}$ and $\vec{Y}$, we first need to introduce *Op-*

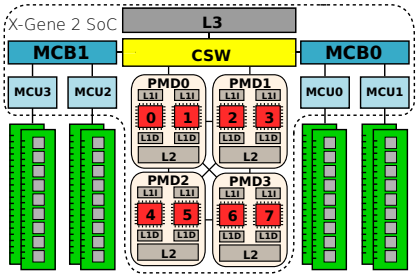

Fig. 5. X-Gene 2 SoC server block diagram.

erational Taxonomic Units (OTUs) [87]. OTUs are expressed as a 2 x 2 contingency table where a is the number of x_i and y_i ($i \in [1, n]$) features which are both '1'; b is the number of x_i and y_i which are '0' and '1', respectively; c is the number of x_i and y_i which are '1' and '0', respectively; and d is the number of x_i and y_i which both equal to '0'.

The *Sokal & Michener* similarity function is defined as:

$$SMF = \frac{a+d}{a+b+c+d} \quad (2)$$

This function (SMF) shows the ratio of the matching binary features on two chromosomes. We estimate SMF for each possible pair of chromosomes in an offspring and average it. We stop running GA when the average SFM estimated for the last offspring is greater than 0.85. It should be noted that the chromosomes from the first offspring are generated randomly. Finally, we also limit the time of each GA search run by two weeks if the search does not converge.

Note that we use SMF only to measure the similarity between chromosomes encoded by binary vectors, while for vectors with real numbers, which are used in memory access patterns, we apply the weighted Jaccard similarity [15]:

$$J_W(\vec{X}, \vec{Y}) = \frac{\sum_i min(x_i, y_i)}{\sum_i max(x_i, y_i)} \quad (3)$$

where x_i and y_i are real features of $\vec{X}$ and $\vec{Y}$, respectively. We estimate J_W for the last offspring and use the same threshold (0.85) for this function to stop the search process.

F. Evaluation phase

To collect the number of DRAM errors (CEs or UEs) manifested by the virus generated at the synthesis phase for a specific chromosome, we compile and run the virus on a real server and measure the number of manifested DRAM errors using ECC. We discuss the hardware platform used in our study in the next section. Finally, we record each virus, i.e. the chromosomes that encode the data and memory access patterns, and the number of manifested DRAM errors for the virus in a database. This enables us to start a new search process using the discovered worst-case viruses if the previous search process has been interrupted.

IV. Experimental Setup

To enable our study, we use a commodity server featuring the *AppliedMicro/Ampere X-Gene 2* [89] ARMv8 processor (CPU). All memory operations in X-Gene 2 are handled by 4 MCUs, which are divided in 2 groups of *Memory Controller Bridges* (MCBs), as shown in Figure 5. Each MCU can be populated with up to 2 DDR3 DIMMs running at 1866 MT/s. In our setup, we use 4 DDR3 8GB DIMMs from a major vendor , one for each MCU, consisting of 72 DRAM chips. For confidentiality purposes, we anonymize the DRAM vendor information.

The X-Gene 2 server features a dedicated processor to enable power management, monitoring of sensors and configuration of system parameters, such as T_{REFP} and V_{DD}.

DRAM operating parameters. The maximum allowed T_{REFP} in the X-Gene 2 platform is 2.283 s (35× more than the nominal 64 ms) and each MCU can independently be configured. V_{DD} can be controlled at the MCB level, i.e. per 2 MCUs. In our experiments, we reduce V_{DD} from the default 1.5 V down to to 1.425 V (5% reduction). Note that this is the minimum voltage specified by the DRAM vendor. Our experiments indicated that any further reduction of V_{DD} results in instantaneous crashing of the server.

Fig. 6. The X-Gene2 server.

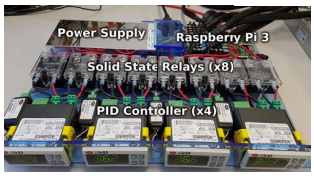

Fig. 7. Temperature controller board.

Temperature. To perform the experiments under controlled temperatures, we implement a unique temperature-controlled server testbed using heating elements [39]. Figure 6 shows the X-Gene 2 board with four DIMMs fitted with our custom adapters. The temperature of each element is regulated by a controller board, as shown in Figure 7, which contains a Raspberry Pi 3 [21], four closed-loop PID controllers [10], and eight solid-state relays controlling the resistive elements of each DIMM and rank independently.

Memory Configuration. To improve the memory performance, modern servers implement memory interleaving [26], [110]. Interleaving enables the uniform distribution of consecutive accesses across different MCUs, maximizing the memory bandwidth. Hardware interleaving poses a challenge to our study, since all data is distributed across MCUs. As a result, we cannot allocate data in a specific DIMM when it is enabled. To this end, we changed the firmware and disabled hardware interleaving in such a way that all the kernel data is allocated in MCU0, while the application data can be allocated in other MCUs. In our study, we operate MCU0 and MCU1 (MCB0) under the nominal DRAM operating parameters, while parameters of the second memory domain (i.e. MCU1 and MCU2) are relaxed.

V. Experimental Results

Parameters of the GA search. To find the optimal GA parameters that provide a quick search of the worst-case data and access patterns, we simulated the GA search for the fitness function that counts the number of bits in a 64-bit chromosome equal to '1'. We found that GA finds the 64-bit

chromosome where all bits set to '1' for the minimum number of generations, which is about 80, when: i) the mutation probability is 0.5; ii) the crossover probability is 0.9 and iii) the size of population is 40.

DRAM parameters and Temperature. We run our experiments for DIMMs allocated at MCU2 (DIMM2) and MCU3 (DIMM3) operating under the maximum refresh period, which is 2.283 s T_{REFP}, and the minimum supply voltage, i.e. 1.428 V V_{DD}. We test the reliability of DRAM chips operating at 55^oC, 60^oC, 65^oC and 70^oC. We use this temperature range to follow previous studies [48], [60] and the DIMM specification , in which the vendor reports the maximum operating temperature of $70^{\circ}C$. Our experiments with various micro-benchmarks used for DRAM characterization, such as *MSCAN* [27], [106], revealed that only correctable errors (CEs) are manifested for temperatures below 70^oC, while uncorrectable errors (UEs) appear at 70^oC.

A. Data pattern search

1) 64-bit patterns: In our first experimental search, we run GA to find the 64-bit word that maximizes the number of CEs, as discussed in Section III-B. In this particular experiment, we use the DIMM which is connected to MCU2,i.e. DIMM2. We run the search at 55^oC. *Note that we run each virus ten times and average the number of obtained CEs since the number of errors may vary from run-to-run due to the phenomenon called Variable Retention Time (VRT) [83].*

Figure 8a shows the 40 discovered worst-case 64-bit data patterns which trigger the highest number of CEs. GA stopped the search process when the similarity function (SFM, see Section III-E) for the 40 worst-case 64-bit patterns exceeded 0.85. This search took about one week and 80 generations were produced. In this figure, the y-axis shows the number of CEs detected for 40 data patterns, while the x-axis depicts each bit of the discovered data patterns. We clearly see that the discovered worst-case data patterns have almost the same sequence of '0's and '1's (the similarity function SFM is 0.89). In other words, the GA search converged to a data pattern with a specific set of '0's and '1's.

Interestingly, we can identify in all discovered data patterns a simple sub-pattern - *'1100'*. Thus, filling the memory with the data that contains this sub-pattern increases the probability to obtain DRAM errors. This effect can be explained by the physical design of the DIMMs. Particularly, such a sub-pattern will increase the probability of DRAM failures in the designs where cells are organized in the following order: *true-cell, true-cell, anti-cell, anti-cell* [51]. Based on these grounds, we assume the DRAM devices used in our study have a similar design. **Observation:** *The repeating sequence '1100' in the data pattern increases the probability of CEs.*

To investigate how the worst-case pattern varies with the DRAM temperature, we run the same search for DIMM2 operating at 60^oC. Figure 8b shows the 40 worst-case data patterns discovered by GA for DIMM2. We see that the GA search converges to the same data pattern that has been discovered at 55^oC. The average similarity SMF between two sets of the worst-case data patterns obtained at 55^oC and 60^oC is 0.90. **Observation:** *The worst-case data pattern does not change with the DRAM temperature.*

To find out if there is a data pattern that reduces the probability of CEs, we change the fitness function to the function that minimizes the number of CEs. Figure 8c shows the 40 best-case 64-bit data patterns discovered for DIMM2 operating at 55^oC. Similar to the worst-case data patterns, the GA search of the 64-bit patterns that minimize the number of CEs converged to the data patterns with a specific set of '0's and '1's (SMF is about 0.92). We see that these patterns significantly differ from the worst-case data patterns; the average similarity SMF between the worst-case patterns and the best-case patterns is about 0.62. Notably, the number of CEs induced by the worst-case data pattern is almost 8x greater than the number of CEs triggered by the best-case data pattern. As follows, the maximum difference in the number of CEs manifested by the same application for which the input data varies is up to 8x (for temperatures below 55^oC) if the memory access pattern does not change from run-to-run.

Further, we compare the number of CEs triggered by popular data pattern micro-benchmarks used for DRAM characterization. In particular, we use: *MSCAN* in which memory is filled with the data where all bits are *'1'* (*all1s*) or *'0'* (*all0s*); *checkerboard*; *walking0s*; *walking1s*; and the micro-benchmark with the random pattern [27], [48], [60], [106]. Figure 8e compares the number of CEs induced by these micro-benchmarks and the revealed 64-bit worst-case and best-case data patterns for two ranks of DIMM2 and DIMM3. Our first observation is that the worst-case data pattern discovered by DStress induces the highest number of CEs for all DIMMs and ranks. For example, the 64-bit worst-case data pattern induces 45% more CEs on average than *walking0s*, which triggers the maximum number of errors among all micro-benchmarks. Thus, the discovered data pattern is more effective for detecting DRAM errors than the micro-benchmarks used in the previous studies. Importantly, to our knowledge, we are the first to report that such a data pattern with repeating sequences of *'1100'* increases the probability of DRAM errors. Our second observation is that this data pattern manifests the highest number of CEs across all DIMMs and ranks. This implies that the pattern maximize the number of CEs not only for one DIMM, but also for other DIMMs used in our study. Finally, our third observation is that the 64-bit best-case data pattern discovered by DStress manifests the lowest number of CEs among all micro-benchmarks and across all DIMMs and ranks. These results confirm that DStress indeed found the best-case 64-bit data pattern which is valid for different DRAM chips. **Implication:** *The data pattern viruses are valid for different DRAM chips used in our study.*

Note that in this experiment we run each virus and micro-benchmark for 2 hours and average the number of detected CEs over the number of runs. Thus, we minimize the bias of the measurements introduced by VRT (Variation Retention Time) effects [83].

DStress was not able to detect the data patterns that trigger

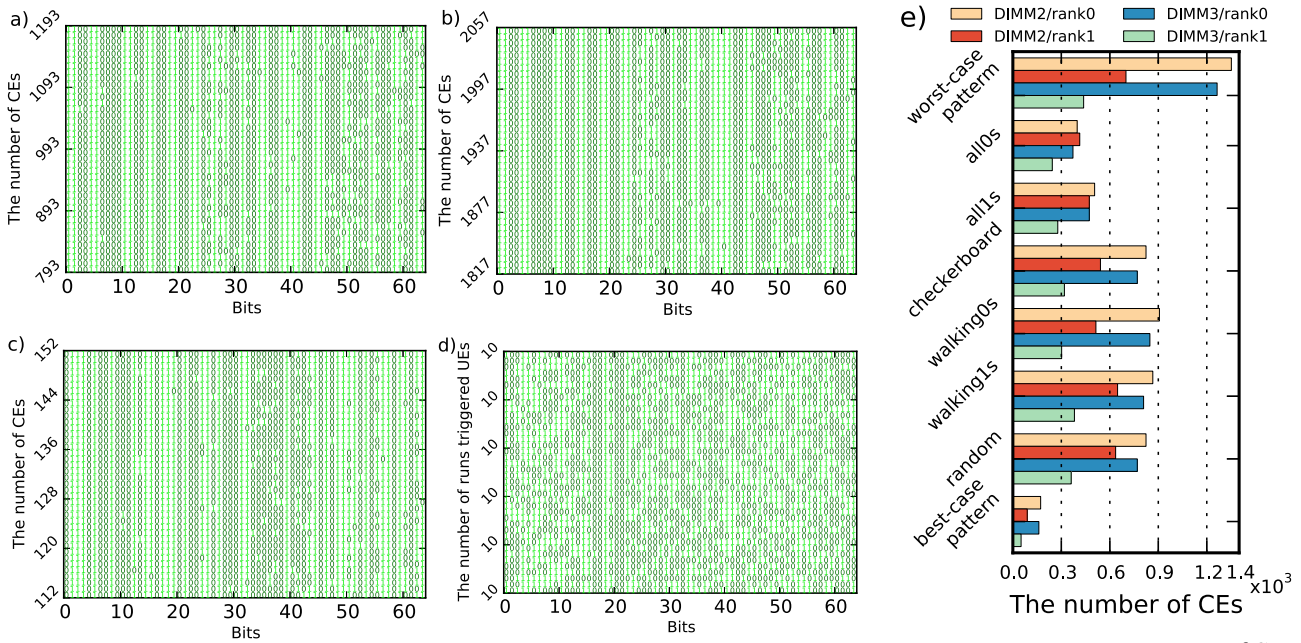

Fig. 8. 40 worst-case 64-bit data patterns that maximize(a)/minimize(c) the number of CEs for DIMM2 operating under relaxed parameters at 55^oC; b) 40 worst-case data patterns that maximize the number of CEs for DIMM2 at 60^oC; d) 40 64-bit data patterns that maximize the probability of an UE for DIMM2 at 62^oC. e) The number of CEs triggered by different micro-benchmarks at 60^oC.

UEs at the temperatures below 62^oC. However, the search engine discovered the patterns that trigger UEs when DRAM operates at 62^oC. Figure 8d depicts the 64-bit data patterns that trigger UEs. The y-axis of Figure 8d shows the number of experimental runs when UEs have been obtained, while the x-axis depicts each bit of the discovered data patterns. Note that in our framework, a virus is automatically stopped by OS when an UE is detected by ECC. As we see, the 40 discovered 64-bit data patterns trigger UEs in 100 % of experimental runs. Unlike our previous experiments, the GA search took 2 weeks and did not converge, since the similarity function SMF for the 40 data patterns that trigger the highest number of UEs is 0.58. This may indicate that there are several data patterns with completely different sequences of '1's and '0's that trigger UEs. However, we clearly see that in all the patterns 17-th, 18-th, 21-st and 22-nd bits are always '0'. Nonetheless, these bits cannot be an indicator of the data patterns that trigger UEs, since the same bits are '0' in the 64-bit data patterns maximizing the number of CEs which do not manifest UEs at 62^oC. To validate this, we run each virus with data patterns depicted in Figure 8a for 2 hours, however we have not discovered any UE when DRAM operates at 62^oC. **Observation:***The worst-case 64-bit patterns that manifest CEs and UEs differ.*

2) 24-Kbyte data patterns: To search for the worst-case 24-KByte data pattern that stresses the cell-to-cell interference in the adjacent rows, we designed a template discussed in Section III-B. In this template, we search for a 24-KByte data pattern that maximizes the number of CEs or UEs when accessing to two rows next to the erroneous rows. To find erroneous rows, we collected all the errors that we have obtained in our experiments for DRAM operating under relaxed parameters at 55^oC, 60^oC and 62^oC. We identified the row addresses where errors were detected using the mapping function discussed in Section II.

Figure 9 shows the 40 worst-case 24-Kbyte data patterns (see Section III-B) that trigger the highest number of CEs. In this figure, the y-axis shows the number of CEs detected for each data pattern, and the x-axis depicts each bit (light green lines are '1's, dark green lines are '0's) of the discovered data patterns. *BankX/Row2* indicates the 8-KByte data pattern discovered for the error-prone rows, while *BankX/Row1* is the 8-KByte data pattern of the rows preceding the error-prone rows and *BankX/Row3* is the data pattern of the rows following the error-prone rows in the same bank (*BankX*). Similar to our previous experiments with CEs, the GA search converged for 24-Kbyte patterns since the similarity function SMF for the 40 worst-case patterns is 0.89. We clearly see the straight lines in this figure for *BankX/Row1*, *BankX/Row2* and *BankX/Row3*. This also indicates that the GA search converged, since all the discovered patterns have almost the same order of '0's and '1's. Interestingly, the discovered 24-KByte data patterns trigger 16 % more errors than the worst-case 64-bit data pattern on average. Note that we obtained the same difference when running the worst case 64-bit and 24-Kbyte data patterns for 2 hours. **Observation**: *The worst-case 24-KByte data pattern differs from the worst-case 64-bit data pattern and manifests 16% more DRAM errors.*

We assume that this difference is due to the cell-to-cell interference induced by the data stored in the adjacent rows.

3) 512-KByte data patterns: Figure 10 shows the 40 worst-case 512-KByte data patterns discovered by GA for DIMM2 operating at 60^oC. Similar to the 24-KByte data patterns, the GA search, which took about 2 weeks, converged since the

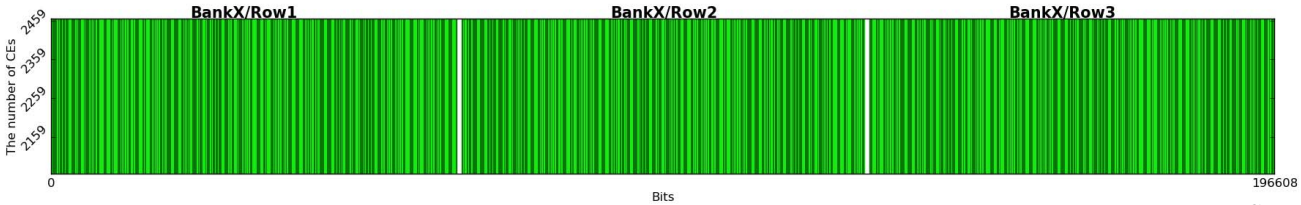

Fig. 9. 40 worst-case 24-Kbyte data patterns that maximize the number of CEs for DIMM2 (MCU2) operating under relaxed parameters at 60^oC.

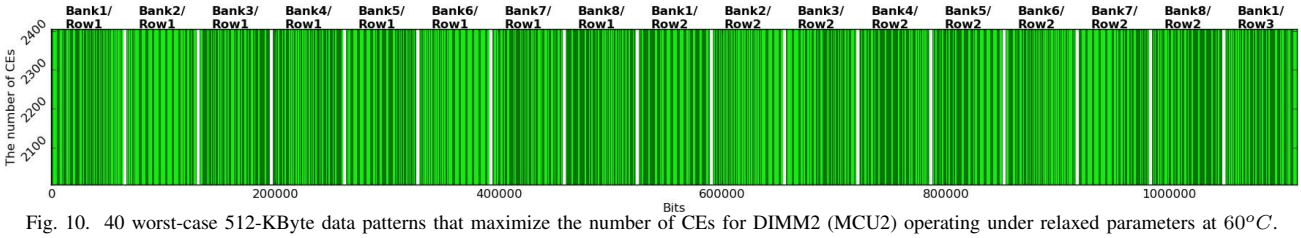

Fig. 10. 40 worst-case 512-KByte data patterns that maximize the number of CEs for DIMM2 (MCU2) operating under relaxed parameters at 60^oC.

similarity coefficient SMF is about 0.88. However, we see that the maximum number of detected CEs does not exceed the highest number of CEs discovered by the 24-KByte data pattern. This indicates that the probability of CEs in the erroneous rows is affected only by the data in neighbouring rows from the same bank. Thus, there is no any cell-to-cell interference across rows from different banks. It is logical, and it confirms that the address mapping function provided in Section II is correct.

Finally, we also run the GA search to find the worst-case 24-KByte and 512-KByte data patterns for UEs, however the search did not converge within 2 weeks for both patterns. Nonetheless, we found that the worst-case patterns differ from the patterns that maximize the number of CEs.

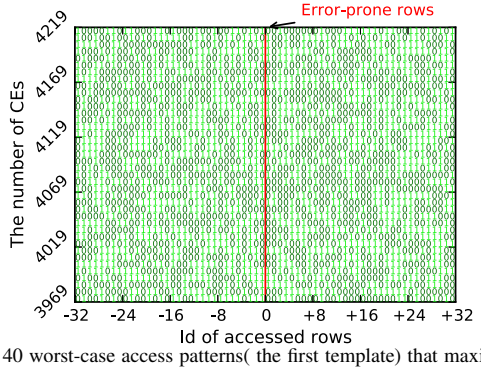

Fig. 11. 40 worst-case access patterns(the first template) that maximizes the number of CEs for DIMM2 operating under relaxed parameters at 60^oC.

4) Access patterns: To investigate how memory access patterns affect DRAM error behavior, we first implement a template where only adjacent rows of the error-prone rows are accessed, as discussed in Section III-B. We investigate accesses to which 32 predecessor rows and 32 successor rows of the error-prone rows increase the probability to obtain a CE in these rows. Note that in this scheme, we form the list of predecessors and successor rows considering DRAM banks, e.g. the predecessors of *Row2.Bank2* are *Row1.Bank8*, *Row1.Bank7*, *Row1.Bank6*,...,*Row1.Bank1*. Importantly, to avoid a parallel search of the worst-case data and memory access patterns, we fill the memory in this experiment with the worst-case 64-bit data pattern discovered previously.

Figure 11 shows the results of a two week GA search at 60^oC for the discussed template. The search did not converge, since the similarity function SMF is 0.5. As a results, it is hard to conclude accesses to which particular rows increases the probability to obtain a CE. In the previous research studies, it was shown that frequent accesses to the adjacent rows of an error-prone row belonging to the same bank increases the probability of an error [71], [99]. Such rows correspond to rows with indexes *-8* and *+8* in Figure 11. However, we do not see any correlation between accesses to these rows and the number of CEs [71], [99], [108]. We explain this by the fact that in the previous works the frequency of memory accesses was much higher than in our study, since these works use the *clflush* instruction to access to DRAM [71], [99]. This instruction enables the user to flush the lines of caches into DRAM and thus increase the memory access rate. While in our study we use only explicit memory accesses, which, as we assume, are partially handled by caches. As a result, we access to DRAMs only when a row is not cached and thus we obtain a much lower DRAM access intensity in our experiments. Nonetheless, in general, we see that accesses to the neighbouring rows increase the number of CEs manifested in the error-prone rows by 71 % compared to the worst-case 24-KByte data pattern. Thus, to stress DRAM reliability, it is essential to use both data and memory access patterns.

Observation: *The memory access-based viruses induce 71% more CEs compared to the worst-case data pattern viruses.*

Based on these results, we may conclude that the techniques proposed in previous studies, which characterize DRAMs using only data pattern micro-benchmarks, can be ineffective.

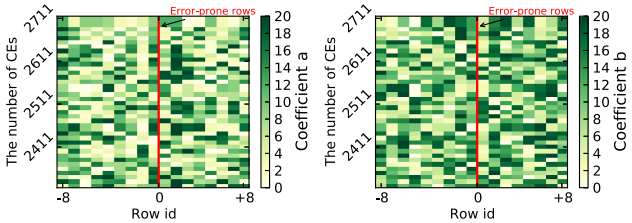

Fig. 12. 40 worst-case access patterns(the second template) that maximize the number of CEs for DIMM2 operating under relaxed parameters at 60^oC.

In our next experiment with memory access patterns, we

investigate how accesses to particular elements of rows that are adjacent to the error-prone rows affect DRAM error behavior, as discussed in Section III-B. Figure 12 shows the coefficients a and b discovered for a two week GA search at 60^oC. Similar to our previous experiment, it is hard to identify any specific access pattern, i.e. elements in rows, that increases the probability to obtain a CE (the average similarity coefficient J_W is 0.45, see Section III-E). Importantly, we use the same 64-bit data pattern to initialize the memory as in our experiments with the first access template.

We see that the maximum number of manifested CEs is lower by 56% than the number of CEs discovered in our previous experiment with access patterns. We explain this by the fact that in the first access template we access to 64 neighbouring rows, while in the this template only 16 adjacent rows are accessed. As a result, there is a higher probability to induce the cell-to-cell interference in the first template. Nonetheless, the number of errors manifested by the second access template is almost 10 % higher than the maximum number of CEs discovered using the worst-case 24-KByte data pattern. Thus, we may conclude that DRAM accesses to the adjacent rows of error-prone rows significantly increases the probability to obtain DRAM errors. However, our study did not reveal accesses to what particular rows and elements in the neighbouring rows increase the probability of CEs.

Finally, our GA search engine did not find particular access patterns that increase the probability of UEs, since the search did not converge to specific patterns. Moreover, similar to our experiments with data patterns, the worst-case access patterns manifested UEs only at 62^oC. We attribute the lack of the GA convergence for our experiments with the memory access patterns to the fact that there can be many access patterns maximizing the probability of CEs and UEs. The lack of the convergence can be also explained by a high level of the cell-to-cell interference induced by accesses, which increases the change of DRAM error behavior from run-to-run. This implies that DStress needs more time for the search in case of UEs and memory accesses compared to the data pattern search of CEs. In our future research, we will increase the time of the GA search for UEs and memory accesses and investigate possible reasons for slow convergence.

5) Efficiency of the GA search: To examine the efficiency of the GA search, we designed a special framework that generates micro-benchmarks with randomized data patterns and collect the distribution of the number of CEs. We use this distribution to estimate the probability that there are data patterns which induce more errors (CEs) than the data patterns discovered by GA. Figure 13a shows the probability density function (PDF) for the number of CEs manifested by randomized data patterns when DIMM2 operates under relaxed parameters at 60^oC. The D'Agostino-Pearsontest test confirmed that the obtained density function (the Gaussian function) follows the normal distribution [17]. Thus, we can use the obtained Gaussian to estimate the probability that there are data patterns inducing more CEs than the patterns discovered by GA [17]. According to our estimates (see Figure 13a), the probability that there exist data patterns that trigger more errors than the discovered worst-case 64-bit and 24-KByte data patterns are 0.97 and $1 - 4 \times 10^{-7}$, respectively.

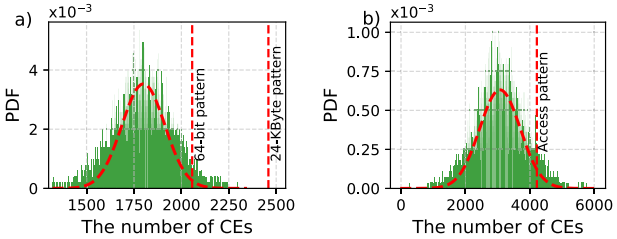

Fig. 13. The distribution of the number of CEs(DIMM2, 60^oC) triggered by a benchmark with random data patterns (a) and random access patterns (b).

We also run the micro-benchmark with random access patterns to evaluate the efficiency of the GA search for access patterns. We found that the probability that DStress discovered the worst-case access pattern for the first template, which induces the highest number of CEs, is about 0.95 (see Figure 13b). These results demonstrate that DStress is more effective for detecting error-prone data patterns rather than access patterns. Nonetheless, the probabilities obtained for both pattern types are high, which indicates that DStress can be effectively applied for generating DRAM viruses.

VI. Use Cases

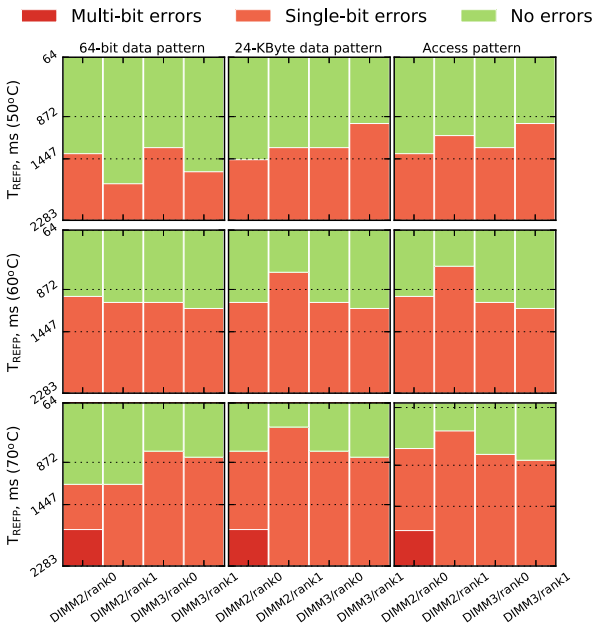

Fig. 14. The safe T_{REFP} discovered by the DRAM viruses under relaxed V_{DD} at different temperatures.

Scaling of DRAM parameters: We use the discovered viruses to find the maximum T_{REFP} (or the marginal T_{REFP}) under relaxed V_{DD} that do not trigger DRAM errors, i.e. CEs and UEs, when running any possible workload. By setting such a T_{REFP} under relaxed V_{DD}, we can reduce the DRAM power without compromising reliability. Figure 14 (*No errors*) shows the discovered marginal T_{REFPs} under relaxed V_{DD} for DIMM2 and DIMM3. In this figure, we present the discovered T_{REFP} for 3 different viruses: i) the worst-case

64-bit data pattern, ii) the worst-case 24-KByte data pattern and iii) the worst-case access pattern discussed in Section V. We use the viruses to find the marginal $T_{REFP}s$ under relaxed V_{DD} at 50^oC, 60^oC and 70^oC. We see that the viruses that use the worst-case 24-KByte data and access patterns discovered lower marginal $T_{REFP}s$ compared to the 64-bit data pattern virus. We also found that the most pessimistic margins for T_{REFP} were discovered by the access virus. These results confirm our previous observation on the efficiency of the memory access viruses.

To detect the marginal $T_{REFP}s$ for which only CEs can be observed but not UEs, we run the viruses discovered by DStress for UEs. Our first observation is that the marginal $T_{REFP}s$ discovered for UEs are higher than the marginal $T_{REFP}s$ obtained for CEs and , therefore, provide higher power savings (see Figure 1, *Single-bit errors*). However, the use of such T_{REFP} margins may be not desirable in real data centers [85], [90]. Our second observation is that the three viruses are equally effective at detecting UEs. Thus, the probability of UEs is mainly depends on temperature.

To validate the T_{REFP} margins detected by the access virus for CEs, we run different memory-intensive benchmarks from Rodinia, Parsec and Ligra suites [8], [14], [88], [92] for DRAM operating at various temperatures for three weeks, and we have not discovered any CEs or UEs. Finally, by using on-board sensors we measured the DRAM power when running different workloads. Our experimental results show that by setting the marginal T_{REFP} obtained for CEs under relaxed V_{DD} for DIMM2 and DIMM3, we can achieve 17.7 % DRAM energy savings and 8.6 % total system energy savings on average without compromising DRAM reliability.

DRAM reliability testing: We believe that DStress will be useful for vendors in exploring the effects of various data and access patterns on DRAM errors and improving the state-of-the-art DRAM testing mechanisms. The generated viruses can be also exploited for *hardware predictive maintenance* which enables the user to identify erroneous hardware components and maintenance cycles in data centers [28], [28], [36], [85].

Security: "Rowhammer" attacks become a serious security threat in computers [9], [29], [86], [93], cloud servers [16], [78], [81], [109] and mobile devices [100], [101]. We intend to use DStress for discovering new "rowhammer" attack scenarios. We believe that our framework ideally fits for this purpose, since it enables us to find the combination of data and access patterns maximizing the probability of errors without knowledge of the internal DRAM design.

VII. Related work

Vendors test reliability of DRAM chips using MARCH and MATS tests [27], [106]. These tests are also used for scrambling and row remapping to minimize the silicon area and critical path delays [22]. Nonetheless, these tests are not effective for revealing some types of DRAM errors, such as neighbourhood pattern-sensitive faults induced by the data in adjacent cells [11], [22], [27], [106]. Moreover, to run these test, it is essential to know the physical memory layout, which is not provided by vendors and may vary across DRAM chips due to the scrambling process. While DStress enable us to identify the worst-case data and access patterns that significantly increase the probability of DRAM errors without the prior knowledge of DRAM internals.

Many studies tried to identify cells with a small retention time by stressing DRAM reliability using micro-benchmarks [13], [48], [59], [60], [60], [61], [73], [77], [80], [102], [103], [105]. All these studies imply that the used micro-benchmarks discover the vast majority of error-prone memory cells. However, as we show in this paper, these micro-benchmarks do not trigger the worst-case DRAM error behavior and thus may not detect some error-prone cells. Other works proposed to optimize DRAM parameters without identifying error-prone cells by taking advantage of frequent memory accesses to reduce refreshes, a supervised data mapping, and exploiting the inherent error resilience of applications [1], [3], [6], [7], [20], [30], [37], [41], [42], [56]–[58], [62]–[64], [66], [68], [70], [72], [75], [91], [95], [97], [107]. The efficiency of the proposed techniques can be improved by using the viruses synthesized by DStress.

Previous studies used GA-based synthesis tools to find the viruses that trigger voltage droops [4], [31], [52], [53], increase processor power [5], [24], [25], [38], [79] or temperature [32], [38]. However, the vast majority of these studies implemented models that search for the worst-case instruction sequences only, ignoring data patterns. In contrast, DStress detects the combinations of data patterns and instruction sequences that match the search criteria. This makes our framework applicable in different areas.

VIII. Conclusion

In this paper, we present DStress, a framework for the automatic synthesis of DRAM reliability stress viruses using Genetic Algorithms. This framework enables us, without knowing DRAM internals, to identify the worst-case data and memory access patterns that increase significantly the probability of DRAM errors. We present the results of our 7-month experimental study and demonstrate the 64-bit, 24-KByte and 512-KByte data patterns, as well as access patterns, discovered by our framework. We show that these patterns detect by at least 45% more errors than the micro-benchmarks used in previous studies. We demonstrate that the discovered data and memory access patterns induce the worst-case DRAM error behavior with a probability greater than 0.95. Finally, we provide several use cases of DStress which explain how our framework can be used for investigating workload-aware DRAM error behavior, including possible scenarios for "rowhammer" attacks, and revealing DRAM operating guardbands.

ACKNOWLEDGMENTS

This work was funded by the H2020 Framework Program of the European Union through the UniServer Project (Grant Agreement 688540, http://www.uniserver2020.eu) and OpreComp project (Grant Agreement 732631, http://oprecomp.eu).

REFERENCES

[1] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, "Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies," in *19th IEEE International Symposium on HPCA '13*, 2013, pp. 400–411.

[2] A. Bacchini, M. Rovatti, G. Furano, and M. Ottavi, "Characterization of data retention faults in dram devices," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2014, pp. 9–14.

[3] S. Baek, S. Cho, and R. Melhem, "Refresh now and then," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 3114–3126, Dec 2014.

[4] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 368–380.

[5] R. Bertran, A. Buyuktosunoglu, M. S. Gupta, M. Gonzalez, and P. Bose, "Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 199–211.

[6] I. Bhati, Z. Chishti, and B. Jacob, "Coordinated refresh: Energy efficient techniques for dram refresh scheduling," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 205–210.

[7] I. Bhati, Z. Chishti, S.-L. Lu, and B. Jacob, "Flexible auto-refresh: Enabling scalable and energy-efficient dram refresh reductions," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 235–246. [Online]. Available: https://doi.org/10.1145/2749469.2750408

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[9] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup est machina: Memory deduplication as an advanced exploitation vector," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 987–1004.

[10] Carel, "ir33 universale electronic control," 2012. [Online]. Available: www.carel.com/product/ir33-universale

[11] P. CaÅcaval and S. Bennett, "Efficient march test for 3-coupling faults in random access memories," *Microprocessors and Microsystems*, vol. 24, pp. 501–509, 03 2001.

[12] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization," *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 323–336, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2964791.2901453

[13] K. K. Chang, A. G. Yauglikcci, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, pp. 10:1–10:42, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3084447

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–54. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2009.5306797

[15] F. Chierichetti, R. Kumar, S. Pandey, and S. Vassilvitskii, "Finding the jaccard median," 01 2010, pp. 293–311.

[16] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 55–71, 2019.

[17] R. D'AGOSTINO and E. S. PEARSON, "Tests for departure from normality. Empirical results for the distributions of b2 and b1," *Biometrika*, vol. 60, no. 3, pp. 613–622, 12 1973. [Online]. Available: https://doi.org/10.1093/biomet/60.3.613

[18] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/1998582.1998590

[19] J. Dreibelbis, J. Barth, H. Kalter, and R. Kho, "Processor-based built-in self-test for embedded dram," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1731–1740, Nov 1998.

[20] P. G. Emma, W. R. Reohr, and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in dram for cache applications," *IEEE Micro*, vol. 28, no. 6, pp. 47–56, 2008.

[21] R. P. Foundation, "Raspberry Pi 3 Model B," 2016. [Online]. Available: https://www.raspberrypi.org/

[22] M. Franklin and K. K. Saluja, "Testing reconfigured ram's and scrambled address ram's for pattern sensitive faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1081–1087, 1996.

[23] S. Ganapathy, A. Teman, R. Giterman, A. Burg, and G. Karakonstantis, "Approximate computing with unreliable dynamic memories," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, 2015, pp. 1–4.

[24] K. Ganesan, J. Jo, W. L. Bircher, D. Kaseridis, Z. Yu, and L. K. John, "System-level max power (sympo) - a systematic approach for escalating system-level power consumption using synthetic benchmarks," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010, pp. 19–28.

[25] K. Ganesan and L. K. John, "Maximum multicore power (mampo) — an automatic multithreaded synthetic power virus generation framework for multicore systems," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[26] M. Ghasempour, A. Jaleel, J. D. Garside, and M. Luján, "Dream: Dynamic re-arrangement of address mapping to improve the performance of drams," in *Proceedings of the Second International Symposium on Memory Systems*. ACM, 2016, pp. 362–373.

[27] P. Girard, N. Nicolici, and X. Wen, *Power-Aware Testing and Test Strategies for Low Power Devices*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[28] I. Giurgiu, J. Szabo, D. Wiesmann, and J. Bird, "Predicting dram reliability in the field with machine learning," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, ser. Middleware '17. New York, NY, USA: ACM, 2017, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/3154448.3154451

[29] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," 07 2016, pp. 300–321.

[30] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, and R. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 583–595.

[31] Z. Hadjilambrou, S. Das, M. A. Antoniades, and Y. Sazeides, "Leveraging cpu electromagnetic emanations for voltage noise characterization," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 573–585.

[32] Z. Hadjilambrou, S. Das, P. N. Whatmough, D. Bull, and Y. Sazeides, "Gest: An automatic framework for generating cpu stress-tests," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 1–10.

[33] T. Hamamoto, S. Sugiura, and S. Sawada, "On the retention time distribution of dynamic random access memory (dram)," *IEEE Transactions on Electron Devices*, vol. 45, no. 6, pp. 1300–1309, June 1998.

[34] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "Chargecache: Reducing dram latency by exploiting row access locality," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 581–593.

[35] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu, "Crow: A low-cost substrate for improving dram performance, energy efficiency, and reliability," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 129–142. [Online]. Available: https://doi.org/10.1145/3307650.3322231

[36] Intel, "Intel© Memory Failure Prediction," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/software/mfp-improves-reliability.html

[37] C. Isen and L. John, "Eskimo: Energy savings using semantic knowledge of inconsequential memory occupancy for dram subsystem," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 337–346. [Online]. Available: http://doi.acm.org/10.1145/1669112.1669156

[38] A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen, "Automated microprocessor stressmark generation," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 229–239.

[39] M. Jung, D. M. Mathew, C. C. Rheinländer, C. Weis, and N. Wehn, "A platform to analyze ddr3 dramâs power and retention time," *IEEE Design & Test*, vol. 34, no. 4, pp. 52–59, 2017.

[40] M. Jung, D. M. Mathew, C. C. Rheinlander, C. Weis, and N. Wehn, "A platform to analyze DDR3 dram's power and retention time," *IEEE Design & Test*, vol. 34, no. 4, pp. 52–59, 2017. [Online]. Available: https://doi.org/10.1109/MDAT.2017.2705144

[41] M. Jung, E. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn, "Omitting refresh: A case study for commodity and wide i/o drams," in *Proceedings of the 2015 International Symposium on Memory Systems*, ser. MEMSYS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 85–91. [Online]. Available: https://doi.org/10.1145/2818950.2818964

[42] C. Kalogirou, C. D. Antonopoulos, N. Bellas, S. Lalis, L. Mukhanov, and G. Karakonstantis, "Increasing the profit of cloud providers through dram operation at reduced margins," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 549–558.

[43] U. Kang, H. soo Yu, C. Park, H. Zheng, J. B. Halbert, K. S. J. Bains, S.-J. Jang, and J. S. Choi, "Co-architecting controllers and dram to enhance dram process scaling," in *The memory forum, ISCA*, 2014.

[44] G. Karakonstantis and K. Roy, "Voltage over-scaling: A cross-layer design perspective for energy efficient systems," in *2011 20th European Conference on Circuit Theory and Design (ECCTD)*, 2011, pp. 548–551.

[45] G. Karakonstantis, K. Tovletoglou, L. Mukhanov, H. Vandierendonck, D. S. Nikolopoulos, P. Lawthers, P. Koutsovasilis, M. Maroudas, C. D. Antonopoulos, C. Kalogirou, N. Bellas, S. Lalis, S. Venugopal, A. Prat-Pérez, A. Lampropulos, M. Kleanthous, A. Diavastos, Z. Hadjilambrou, P. Nikolaou, Y. Sazeides, P. Trancoso, G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, and S. Das, "An energy-efficient and error-resilient server ecosystem exceeding conservative scaling limits," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1099–1104.

[46] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*, 2nd ed. Wiley-IEEE Press, 2007.

[47] S. Khan, D. Lee, and O. Mutlu, "Parbor: An efficient system-level technique to detect data-dependent failures in dram," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016, pp. 239–250.

[48] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for dram retention failures: A comparative experimental study," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 519–532, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2637364.2592000

[49] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent dram failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 27–40. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123945

[50] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled drams," *IEEE Electron Device Letters*, vol. 30, no. 8, pp. 846–848, Aug 2009.

[51] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 361–372.

[52] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, "Audit: Stress testing the automatic way," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 212–223.

[53] Y. Kim and L. K. John, "Automated di/dt stressmark generation for microprocessor power delivery networks," in *Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design*, ser. ISLPED '11. IEEE Press, 2011, p. 253–258.

[54] Kuo-Liang Cheng, Ming-Fu Tsai, and Cheng-Wen Wu, "Neighborhood pattern-sensitive fault testing and diagnostics for random-access memories," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1328–1336, Nov 2002.

[55] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency dram: Optimizing dram timing for the common-case," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 489–501.

[56] H.-H. S. Lee and M. Ghosh, "Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams," *2007 40th IEEE/ACM International Symposium on Microarchitecture*, vol. 00, pp. 134–145, 2007.

[57] X. Li, M. C. Huang, K. Shen, and L. Chu, "A realistic evaluation of memory hardware errors and software system susceptibility," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855846

[58] X. Li and D. Yeung, "Application-level correctness and its impact on fault tolerance," in *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, ser. HPCA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 181–192. [Online]. Available: https://doi.org/10.1109/HPCA.2007.346196

[59] C.-H. Lin, D.-Y. Shen, Y.-J. Chen, C.-L. Yang, and C.-Y. M. Wang, "Secret: A selective error correction framework for refresh energy reduction in drams," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 2, pp. 19:19:1–19:19:24, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2747876

[60] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 60–71. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485928

[61] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337159.2337161

[62] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *SIGPLAN Not.*, vol. 46, no. 3, pp. 213–224, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1961296.1950391

[63] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 467–478.

[64] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. D. Mannaru, A. Riska, and D. Milojicic, "Susceptibility of commodity systems and software to memory soft errors," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1557–1568, Dec 2004.

[65] D. S. M. D.-S. Min, D. I. S. D.-I. Seo, J. Y. J. You, S. C. S. Cho, D. C. D. Chin, and Y. E. Park, "Wordline coupling noise reduction techniques for scaled drams," in *Digest of Technical Papers., 1990 Symposium on VLSI Circuits*, June 1990, pp. 81–82.

[66] F. Moradi, G. Panagopoulos, G. Karakonstantis, D. Wisland, H. Mahmoodi, J. K. Madsen, and K. Roy, "Multi-level wordline driver for low power srams in nano-scale cmos technology," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011, pp. 326–331.

[67] L. Mukhanov, K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis, "Dram characterization under relaxed refresh period considering system level effects within a commodity server," in *2018 IEEE*

24th International Symposium on On-Line Testing And Robust System Design (IOLTS), 2018, pp. 236–239.

[68] L. Mukhanov, K. Tovletoglou, H. Vandierendonck, D. S. Nikolopoulos, and G. Karakonstantis, "Workload-aware dram error prediction using machine learning," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 106–118.

[69] L. Mukhanov, K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis, "Characterization of hpc workloads on an armv8 based server under relaxed dram refresh and thermal stress," ser. SAMOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 230–235. [Online]. Available: https://doi.org/10.1145/3229631.3236091

[70] J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. F. Martínez, "Understanding and mitigating refresh overheads in high-density ddr4 dram systems," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, p. 48–59, Jun. 2013. [Online]. Available: https://doi.org/10.1145/2508148.2485927

[71] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '17. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2017, pp. 1116–1121. [Online]. Available: http://dl.acm.org/citation.cfm?id=3130379.3130643

[72] P. Nair, C. Chou, and M. K. Qureshi, "A case for refresh pausing in dram memory systems," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 627–638.

[73] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 72–83. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485929

[74] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 72–83, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485929

[75] K. Nguyen, K. Lyu, X. Meng, V. Sridharan, and X. Jian, "Nonblocking memory refresh," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 588–599. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00055

[76] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and modeling on-die error correction in modern dram: An experimental study using real devices," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 13–25.

[77] M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (reaper): Enabling the mitigation of dram retention failures via profiling at aggressive conditions," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 255–268, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3140659.3080242

[78] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking deterministic signature schemes using fault attacks," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, pp. 338–352.

[79] S. Polfliet, F. Ryckbosch, and L. Eeckhout, "Automated full-system power characterization," *IEEE Micro*, vol. 31, no. 3, pp. 46–59, 2011.

[80] M. K. Qureshi, D. H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 427–437.

[81] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, p. 1–18.

[82] M. Redeker, B. F. Cockburn, and D. G. Elliott, "An investigation into crosstalk noise in dram structures," in *Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002)*, 2002, pp. 123–129.

[83] P. J. Restle, J. W. Park, and B. F. Lloyd, "Dram variable retention time," in *1992 International Technical Digest on Electron Devices Meeting*, Dec 1992, pp. 807–810.

[84] C. Roth, C. Benkeser, C. Studer, G. Karakonstantis, and A. Burg, "Data mapping for unreliable memories," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 679–685.

[85] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: A large-scale field study," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, pp. 193–204. [Online]. Available: http://doi.acm.org/10.1145/1555349.1555372

[86] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *in Black Hat USA*, 2015.

[87] S. seok Choi and S. hyuk Cha, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, pp. 43–48, 2010.

[88] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *SIGPLAN Not.*, vol. 48, no. 8, pp. 135–146, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2517327.2442530

[89] G. Singh *et al.*, "AppliedMicro X-Gene2," in *Hot Chips, 2014*.

[90] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015, pp. 297–310. [Online]. Available: http://doi.acm.org/10.1145/2694344.2694348

[91] J. Stuecheli, D. Kaseridis, H. C.Hunter, and L. K. John, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 375–384. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2010.22

[92] J. Sun, H. Vandierendonck, and D. S. Nikolopoulos, "Graphgrind: Addressing load imbalance of graph partitioning," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '17. New York, NY, USA: ACM, 2017, pp. 16:1–16:10. [Online]. Available: http://doi.acm.org/10.1145/3079079.3079097

[93] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '18. USA: USENIX Association, 2018, p. 213–225.

[94] K. Tovletoglou, L. Mukhanov, G. Karakonstantis, A. Chatzidimitriou, G. Papadimitriou, M. Kaliorakis, D. Gizopoulos, Z. Hadjilambrou, Y. Sazeides, A. Lampropulos, S. Das, and P. Vo, "Measuring and exploiting guardbands of server-grade armv8 cpu cores and drams," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 6–9.

[95] K. Tovletoglou, L. Mukhanov, D. S. Nikolopoulos, and G. Karakonstantis, "Shimmer: Implementing a heterogeneous-reliability dram framework on a commodity server," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 26–29, 2019.

[96] K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis, "Relaxing dram refresh rate through access pattern scheduling: A case study on stencil-based algorithms," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017, pp. 45–50.

[97] K. Tovletoglou, L. Mukhanov, D. S. Nikolopoulos, and G. Karakonstantis, "Harmony: Heterogeneous-reliability memory and qos-aware energy management on virtualized servers," ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 575–590. [Online]. Available: https://doi.org/10.1145/3373376.3378489

[98] A. J. van de Goor and I. Schanstra, "Address and data scrambling: causes and impact on memory tests," in *Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on*, 2002, pp. 128–136.

[99] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1675–1689. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978406

[100] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida,

"Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1675–1689. [Online]. Available: https://doi.org/10.1145/2976749.2978406

[101] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. Padmanabha Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "Guardion: Practical mitigation of dma-based rowhammer attacks on arm," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Giuffrida, S. Bardin, and G. Blanc, Eds. Cham: Springer International Publishing, 2018, pp. 92–113.

[102] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in dram (rapid): software methods for quasi-non-volatile dram," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, Feb 2006, pp. 155–165.

[103] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in dram (rapid): software methods for quasi-non-volatile dram," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, 2006, pp. 155–165.

[104] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2017, pp. 25–36.

[105] J. Wang, X. Dong, and Y. Xie, "Proactivedram: A dram-initiated retention management scheme," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 22–27.

[106] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[107] S. Wang, M. N. Bojnordi, X. Guo, and E. Ipek, "Content Aware Refresh: Exploiting the Asymmetry of DRAM Retention Errors to Reduce the Refresh Frequency of Less Vulnerable Data," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 362–374, March 2019.

[108] X.-C. Wu, T. Sherwood, F. T. Chong, and Y. Li, "Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 645–657. [Online]. Available: https://doi.org/10.1145/3297858.3304039

[109] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, p. 19–35.

[110] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd annual ACM/IEEE International Symposium on Microarchitecture*. ACM, 2000, pp. 32–41.