2017 IEEE International Symposium on High Performance Computer Architecture

Defect Analysis and Cost-effective Resilience Architecture for Future DRAM Devices

Sanguhn Cha*, Seongil O*, Hyunsung Shin*, Sangjoon Hwang*, Kwangil Park*, Seong Jin Jang*, Joo Sun Choi*,
Gyo Young Jin*, Young Hoon Son*, Hyunyoon Cho*†, Jung Ho Ahn† and Nam Sung Kim‡
*Samsung Electronics, †Seoul National University, ‡University of Illinois Urbana-Champaign
gajh@snu.ac.kr, nskim@illinois.edu

***Abstract*—Technology scaling has continuously improved the density, performance, energy efficiency, and cost of DRAM-based main memory systems. Starting from sub-20nm processes, however, the industry began to pay considerably higher costs to screen and manage notably increasing defective cells. The traditional technique, which replaces the rows/columns containing faulty cells with spare rows/columns, has been able to cost-effectively repair the defective cells so far, but it will become unaffordable soon because an excessive number of spare rows/columns are required to manage the increasing number of defective cells. This necessitates a synergistic application of an alternative resilience technique such as In-DRAM ECC with the traditional one.**

Through extensive measurement and simulation, we first identify that aggressive miniaturization makes DRAM cells more sensitive to random telegraph noise or variable retention time, which is dominantly manifested as a surge in randomly scattered single-cell faults. Second, we advocate using In-DRAM ECC to overcome the DRAM scaling challenges and architect In-DRAM ECC to accomplish high area efficiency and minimal performance degradation. Moreover, we show that advancement in process technology reduces decoding/correction time to a small fraction of DRAM access time, and that the throughput penalty of a write operation due to an additional read for a parity update is mostly overcome by the multi-bank structure and long burst writes that span an entire In-DRAM ECC codeword. Lastly, we demonstrate that system reliability with modern rank-level ECC schemes such as single device data correction is further improved by hundred million times with the proposed In-DRAM ECC architecture.

***Keywords*-DRAM; resilience; In-DRAM ECC; single-cell faults; long-burst writes;**

I. Introduction

Process scaling inevitably compromises the reliability of components of integrated circuits [7], [14], [33]. For DRAM, arbitrarily distributed single-cell failure (SCF) is the primary source of device failures, especially when the fabrication process of a certain technology generation matures [39]. The sources of this SCF include spatial and temporal variation in retention time [31], [43], random telegraph noise (RTN [16]), fluctuation in write recovery time [26], and cosmic rays [17]. Traditionally, each DRAM bank is equipped with spare (redundant) rows and columns, which can fix these SCFs as well as the failure of an entire row and column. However, the frequency of SCF increases as DRAM cell size decreases as smaller transistors and capacitors are more vulnerable to process variation and manufacturing imperfection. Therefore, the area overhead of spare rows and columns should drastically increase to fix the SCFs and keep DRAM device yield high.

There have been various proposals to address the area overhead of the traditional reliability scheme: ArchShield [36], XED [37], CiDRA [47], and In-DRAM ECC (Error Correction Code) [28], [35], [40]. ArchShield [36] stores the information associated with faulty DRAM cells in a certain main memory area, enabling faulty words to be replicated. CiDRA [47] places a small SRAM cache in each DRAM device to replace accesses to the addresses associated with faulty cells into the corresponding cached data. These promising proposals, however, collect faulty cell information through runtime test on every boot-up or by leveraging external non-volatile memory, which limit their wide adoption. They also cannot cope with newly added faults, such as those due to RTN and retention time variation.

By contrast, In-DRAM ECC, a rather traditional DRAM reliability scheme that incorporates spare (parity) bit provisioning and ECC coding within a DRAM device, has relatively higher area overhead at a given SCF rate, but does not require costly faulty map storage or reconstruction processes. It also combats newly added faults after a device is shipped. The overheads of In-DRAM ECC on DRAM area and performance were unacceptably high in the past, for example 50% for DRAMs that should operate in a unit of a single byte [28]. However, advances in fabrication technologies have improved ECC coding speed. Also, increase access granularity per DRAM device due to continuous growth in bandwidth demands amortizes area overhead for In-DRAM ECC, both alleviating these issues significantly. Samsung reported an LPDDR4 DRAM prototype that employs In-DRAM ECC, where area, latency, and power overheads are limited to 6.3%, 10%, and 3%, respectively [40].

In this paper, we first reveal the detailed failure mechanism of modern DRAM devices through our extensive measurement and simulation results. Second, we analyze the impacts of In-DRAM ECC in depth on contemporary main memory systems over the aspects of reliability, area, power, and performance. We identify that the wiring limitation of a DRAM bank fabricated at a nano-scale process enforces data to be accessed at the multiple of eight bits, strongly influencing the selection of In-DRAM ECC types. With eight parity bits, single-bit error correction (SEC) is possible for

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.30

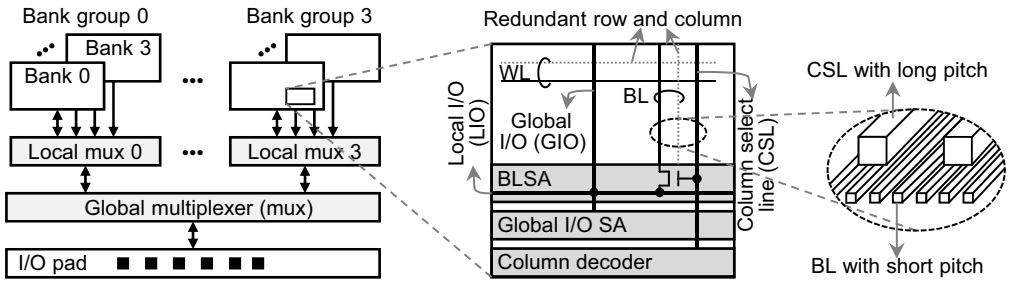


Figure 1: Modern DRAM organization. WL, BL, and SA stand for wordline, bitline, and sense amplifier. GDDR5 and DDR4 adopt the concept of bank group, which divides banks in a device into multiple groups. Two consecutive accesses within a group should be separated by tCCD_L, whereas the ones between groups should be as least by tCCD_S and $\text{tCCD}_L > \text{tCCD}_S$.

128-bit data, whereas SEC and double-bit error detection (SECDED) is possible for 64-bit data. Even if the latter is a stronger code, both may interfere with the SECDED code employed at the rank-level. Opposite to the analysis of [47], however, we show that the both codes can behave in harmony with other rank-level ECC types, such as SDDC (single-device-data-correction), DDDC (double-device-data-correction), and SSCDSD (single-symbol-error-correction and double-symbol-error-detection) that can survive from an entire DRAM device failure in a rank. Therefore, we choose SEC as the In-DRAM ECC type as it incurs significantly lower area overhead than SECDED. We also show that In-DRAM ECC significantly reduces the probability of undetected/uncorrectable errors at a given SCF rate.

The access granularity of a device with In-DRAM ECC, which is equal to the codeword size (= data + parity), is larger than the unit size of device's external data transfers, turning a DRAM write operation into a sequence of read-modify-write sub-operations internally. This leads to additional power and performance overheads on top of ECC encoding/decoding delays. Write operations are less frequently in the critical paths of applications, and thus they have less pronounced impact of this limited DRAM write bandwidth on the system performance. However, it does influence system performance when an application has bursty writes to a DRAM bank, such as block copies. We propose to augment DRAM commands tailored for a long burst write (LBWR), where the burst size equals to the codeword size of In-DRAM ECC, obviating the need of read-modify-write and hence restoring most of the performance degradation (to less than 2.5% on average of memory-intensive SPEC CPU2006 benchmarks [12]).

II. SCFs: A Primary Source of DRAM Failure

A. DRAM organization and fault types

A device consists of multiple DRAM banks. A bank is structured in two dimensions and typically contains around a billion DRAM cells, each holding a bit of data. A DRAM cell consists of a storage capacitor and an access transistor. To access data stored in a cell, its access transistor is first turned on by applying a high voltage to a wordline (WL) connected to the gate of the transistor. Then, charges can transfer from the corresponding storage capacitor to bitline sense amplifier (BLSA) (for reads) or vice versa through bitline (BL) [21], [31], [32] (for writes). Charges stored in storage capacitor leak over time and an insufficient amount of charges in the cell is evaluated to a wrong value. To retrieve correct data, cells have to be periodically refreshed to restore the charge in storage capacitor. JEDEC SDRAM standards specify to perform minimum 8,192 refresh operations to a DRAM device for a refresh interval of 64ms [22], [23], [32]. If the retention time of a cell is shorter than this refresh interval, the leaky cell becomes a faulty cell.

Modern DRAM devices adopt hierarchical organizations to store billions of bits in a single die as depicted in Figure 1. In order to access data in a bank, a row of the bank holding the data is first selected by an activate command and the entire values of the row are latched at BLSAs through BLs which are perpendicular to WLs. A row consists of thousands of cells (e.g. typically 8K cells in a DDR3/DDR4 die [22], [23]) and the pitch between two consecutive rows is smaller than 100nm in modern DRAM technologies. Driving this row with a single driver circuit through a narrow and hence highly resistive WL cannot meet latency requirement. Instead, signals from multiple global WLs are used to drive local WLs that constitute a row. Then, a read or write command specifies a location within the row, whose data are latched at the corresponding BLSAs. There are thousands of rows in a bank and the pitch between two columns is sub-100nm as well, whereas a read/write command transfers only up to few hundred bits through global I/Os (GIOs). Therefore, the datapath of a bank is hierarchical as well; a bank has multiple rows of BLSAs and there are muxes between BLSAs and GIOs, called local I/Os (LIOs). Besides, column-select lines (CSLs) control the multiplexers and span an entire bank vertically; CSLs are thick (whose pitch is several times higher than that of bitline) so that several BLSAs are selected as a group.

Faults are inevitable to integrated circuits including DRAM devices due to manufacturing imperfection and pro-

cess variation [7], [14], [33]. Improving yield or decreasing the rates of DRAM device failures that originate from faults [41] has been a critical task to the cost-sensitive DRAM industry. There have been various DRAM fault types such as cell, row, column, bank, pin (e.g., command, address, and data pins), and device faults. Among them, randomly distributed faulty cells have been recognized as a primary source of faults, followed by row and column faults [36], [39], [47]. For example, Sridharan et al. [48] reported that cell, row, and column faults contribute to 50%, 13%, and 11% of entire DRAM faults, respectively. Recent studies that collect DRAM failure statistics from massively parallel systems also support this distribution [45], [49] of DRAM failure types, even if these results are from deployed DRAM devices (after *t0*, the time of device shipment) whereas [39] showed the results before *t0*.

B. Root causes and characteristics of SCFs

These randomly distributed faulty cells are mainly caused by two fault types – random faults and parametric faults [26], [39]. Random faults are mainly caused by RTN or variable retention time (VRT), leading to the failure rate of 10^{-4} on contemporary 8GB main memory DRAM modules. Parametric faults are observed in write recovery (tWR) and retention (tREFW) of data, which are currently screened with a large margin while testing DRAMs (before *t0*). Because random and parametric faults are arbitrarily distributed single-cell faults (SCFs), they cannot be properly treated by sparing each with a whole column or row [36], [47].

In 28nm or finer-pitch DRAM processes, we observe from our device samples that the percentage of SCFs in total faults are increasing (surpassing 80%), which in contrast to a recent speculation that clustered DRAM cell defects may become significant due to aggressive scaling of device feature size. As the fabrication technology advances, the size of a DRAM cell gets smaller, but the size of peripheral circuits, such as row/column decoders, sense amplifiers, and I/O buffers, and the pitch of lengthy (millimeter-scale) wires has not decreased at the same pace [13]. Therefore, other types of DRAM faults originating from these peripherals are relatively easier to be managed by both manufacturing and circuit/design techniques compared with SCFs.

C. Forecasting SCFs for future DRAM devices

Through experiments with fabricated chips and simulation based on our projected technology model, we expect that the aforementioned trends will continue in the future as well. That is, SCFs due to DRAM cell retention time and write recovery time violations will dominate DRAM failures.

Spatial variation of retention time: As DRAM process scales down, it is expected that leakage current increases for given DRAM cell capacitance [33] and the capacitance of a storage capacitor decreases. Furthermore, it becomes more difficult to build a storage capacitor structure of a high aspect ratio [26], [36], which denotes the ratio of the height to bottom diameter of a storage capacitor. Consequently, the retention time of a DRAM cell is likely to become shorter with smaller DRAM technology nodes.

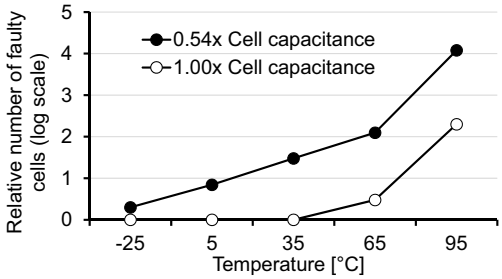


Figure 2: The number of error bits due to short retention time in DRAM devices over temperature ranges. Two (average) cell capacitance values are used: 1) ordinary 25nm cells (1×) and 2) smaller capacitors reflecting difficulty of cell manufacturing in future processes (0.54×). All measured values are normalized to those of ordinary cells at -25°C.

To understand the retention times of future DRAM cells more precisely, we fabricated and measured 4,500 DRAM devices with cells that have 1× and 0.54× cell capacitance values manufactured with Samsung's 25nm DRAM process technology.[1] We examined faults of the fabricated DRAM devices, which enables us to analyze the intrinsic characteristics of raw DRAM cell faults utilizing the test patterns and test environments actually used in DRAM manufacturing before the faults are remedied. Figure 2 shows that the DRAM cell fault rate increases as capacitance decreases at each temperature point. The DRAM cell fault rate with 0.54× relative cell capacitance was increased by up to 100 times compared to the case of 1× capacitance. The experiment shows that 1) the cell retention time is very sensitive to temperature and 2) DRAM cell faults due to short retention time are randomly distributed, not clustered.

Write recovery time (tWR): To properly write data into DRAM cells, the associated page including the cells should stay open before being closed for at least a certain amount of time, called tWR. Otherwise an adequate amount of charge cannot be transferred to the cell, leading to a wrong data being evaluated during a subsequent read. The value of tWR has been specified as a fixed value of 15ns, not a multiple of DRAM clock cycles (tCK) in JEDEC standards [22], [23]. As DRAM feature sizes decrease below 20nm, the resistance of BL contacts increases and on-current of access transistor decreases [26]. Consequently, the number of DRAM cells that cannot be successfully written under the tWR constraint

[1]We targeted to cut the cell capacitance into half but the average cell capacitance of the fabricated chips was 0.54× of that of the reference chip. We did not change the access transistor size as it is determined by the fabrication technology used. We used test equipment group (TEG) circuity located at scribe lines between DRAM dies for measurement.

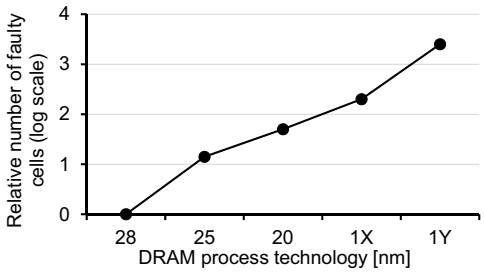


Figure 3: The number of faults caused by violation of tWR constraint (write recovery time) obtained by simulation using sub-30nm DRAM process technology nodes.

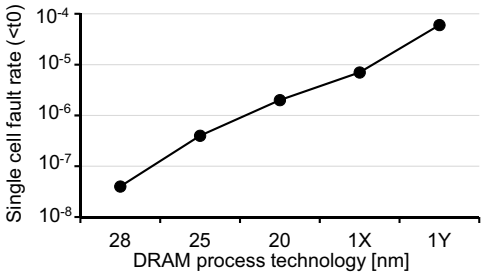


Figure 4: Expected SCF rates before *t0* below 30nm DRAM fabrication technology nodes.

is expected to grow.

In order to analyze the characteristics of DRAM cell faults occurred by violating the tWR constraint in future DRAM technologies, we simulated DRAM cells with our in-house failure estimation tool using 28nm, 25nm, 20nm, and sub-20nm (1Xnm and 1Ynm) DRAM technologies.[2] As plotted in Figure 3, more cells violate the tWR constraint and become faulty as DRAM fabrication technology scales down. Besides, we measured the tWR values of the DRAM devices fabricated with our 28nm DRAM fabrication technology and confirmed that the faults due to violation of the tWR constraint are randomly scattered. This randomly distributed SCF characteristic is expected to be maintained in future DRAM technology nodes due to the characteristics of its fault causes, such as irregularity in resistance of BL contact and in on-current of DRAM cells' access transistors.

Temporal variation of retention time (VRT): VRT is a phenomenon where the retention time of a DRAM cell changes over time [43], [54]. VRT is widely present in modern DRAM devices [31] and expected to be more eminent in future DRAMs. DRAM manufacturers try to detect faulty cells due to VRT in diverse ways, but there is no precise way to screen them out during the test phase for now. Therefore, an error handling scheme (e.g, In-DRAM ECC) is required to handle errors resulted from VRT in future process technologies besides conventional DRAM rank-level error correction techniques called rank-level ECC.

Implications of DRAM cell faults: Based on our analyses of the characteristics of write recovery time and temporal/spatial variation of retention time, we can estimate the SCF rate before *t0* for various DRAM process technology nodes below 30nm. As DRAM process technology scales down further, SCFs are expected to increase rapidly and to become the major source of errors as shown in Figure 4. Therefore, *we will focus on SCFs, the dominant DRAM fault types in the present and future*, hereafter in this paper.

III. Combat SCF with Resilience Architecture

A. Techniques to combat SCFs

The traditional redundant row and column sparing method is an essential fault handling technique that has been employed in commodity DRAM devices [14]. Each DRAM bank is equipped with spare rows and columns that can replace faulty rows, columns, and cells. These redundant rows/columns are set up and utilized as follows. During the test phase of fabricated devices, the addresses of faulty cells are identified and gathered by an automatic test equipment. A repair algorithm calculates and assigns target redundant rows and columns for the faulty cells, columns, and rows to efficiently leverage these spares. The target addresses are stored in one-time programmable memory, such as electrical fuses within the device (Figure 1). When a device is turned on, the target addresses stored in the fuses are loaded to target address registers in each physical bank. For each device operation, a comparator compares an input address with target ones before the input address is decoded by row/column decoders. When the input address matches a target address in the registers, the data is read from or written to the redundant row/column. During the test phase, the time to gather and analyze the addresses of faulty cells and to store target addresses into fuses is proportional to the number of faulty cells, rows, and columns. Accordingly, new test methods are anticipated to reduce the test time for mass production with increasing SCF rates over generations.

If the number of redundant rows and columns in a DRAM device is not enough to replace all faulty cells, rows, and columns, the device is discarded. All DRAM devices at *t0* appear as fault-free owing to the redundant row and column sparing method. However, this sparing method is inefficient for randomly distributed SCFs because redundant rows and columns consist of thousands of cells and they are replaced as a whole [29]. This method is reasonable when the SCF rates are low, but the area overhead rapidly grows as DRAM fabrication technology scales down and hence the SCF rates surge accordingly.

Recent proposals [36], [47] try to architecturally address SCFs with small area overhead. ArchShield [36] stores the location information of faulty cells in a certain main memory area (i.e., fault maps), enabling faulty words to be

[2]Notations such as 1X and 1Y are conventional expressions widely used in DRAM vendors; X, Y, and Z ($X > Y > Z$) are non-negative integers smaller than ten.

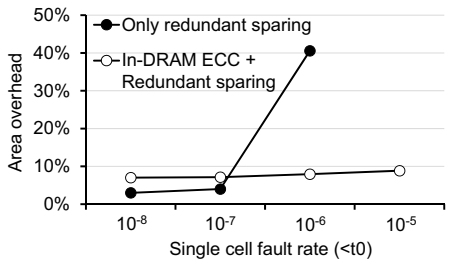


Figure 5: Area overhead of using only row/column sparing method vs. the sparing method and In-DRAM ECC collectively over the SCF rates before *t0*.

replicated in a memory area without any fault. ArchShield requires changes to both memory system hardware and OS. CiDRA [47] complements a small SRAM cache in each DRAM device to replace the access to the addresses associated with faulty cells into corresponding cached data. Even though promising, both proposals collect the information of faulty cells through a runtime test phase on every boot-up or by leveraging external non-volatile memory, both of which are a stiff hurdle for wide acceptance. Moreover, neither can manage newly added faults during normal operations.

DRAM manufacturers have also tried to incorporate ECC within DRAM devices, which is called In-DRAM ECC [28], [35]. In-DRAM ECC typically adopts a block code [9], [11], [42]; data in a device is divided into codewords, each with a fixed size of k symbols and encoded into n symbols denoted by (n, k) where the codeword has ($n - k$) parity symbols. Even if the default alphabet of a symbol is a bit, a group of bits corresponding to the width of a DRAM device (e.g. $\times 4$ and $\times 8$) can be a symbol. In a block code, the Hamming distance stands for the minimum number of symbols that should be changed for a valid codeword to be another. The DRAM proposals that adopted ECC in the past were very difficult to be mass-produced because of huge area overhead associated with extra parity cells, such as 50% for devices that operate in a unit of a single byte (8b) [28]. However, the increased size of a unit data transfer in DRAM devices (device width $\times$ burst length[3]) to satisfy high bandwidth demands and the advances in fabrication technologies have substantially reduced the area and latency overhead of In-DRAM ECC parity bits and coding logic, respectively. These help In-DRAM ECC becoming an attractive solution in handling SCFs cost-effectively. For example, Samsung presented the first commodity LPDDR4 DRAM [24] employing Single-bit Error Correction (SEC) In-DRAM ECC [40] with a codeword size of 136 bits, parity cell area overhead of 6.25%, and decoding overhead of less than 2ns.

[3]Modern devices have multiple data pins denoted by $\times$N (N is the number of data pins). The data transfer rate of these pins is several times higher than the operating frequency of DRAM banks; so multiple bits from BLSAs are transferred through a pin, determining the burst length.

	Data	Parity	Area overhead	Encoding	Decoding
SEC	128b	8b	6.9%	1.5ns	1.6ns
SECDED	64b	8b	11.8%	1.1ns	1.2ns
DEC	128b	16b	14.2%	2.0ns	5.1ns
S8EC	128b	16b	13.2%	2.0ns	3.5ns

Table I: Encoding/decoding time and estimated area overhead including parity area and ECC engines.

Combining the aforementioned redundant row/column sparing method and In-DRAM ECC can be attractive as the SCF rates increase. Figure 5 compares estimated area overheads of using only the redundant sparing method with that of a combined use of the sparing method and In-DRAM ECC as the SCF rates before *t0* are varied. In this estimation, (136, 128) SEC code is assumed for In-DRAM ECC. When the SCF rates are larger than 10^{-6}, the combined use becomes more area efficient than only using the redundant sparing method. Therefore, the combined use of redundant sparing method and In-DRAM ECC is compelling in area overhead perspective, whose details are elaborated in Section IV-A.

B. Determining right In-DRAM ECC types

Even though there are numerous block codes that can be used for In-DRAM ECC, implementable ECC types within DRAM devices are greatly restricted when we consider their impacts on system performance and energy efficiency, as well as DRAM area and testing overheads. To determine practical ECC types, we simulated numerous ECC codes at the 20nm DRAM technology, through which we identify that SEC [11], SECDED (SEC with Double-bit Error Detection) [15], Double-bit Error Correction (DEC) [11], [38], and Single-symbol (8b) Error Correction (S8EC) codes [8] are feasible for In-DRAM ECC due to their low coding latency and area overheads. Table I tabulates encoding time, decoding time, and estimated area overhead of parity bit area and ECC engines in an 8Gb device. We limit the numbers of data bits not to exceed 128 bits because bringing too many data bits per read or write results in a large size prefetching, and hence high energy overhead. Furthermore, we consider the number of parity bits is in multiples of eight because the width of DRAM internal data buses, control path, and cell arrangement layout are fixed based on multiples of eight-bit width in modern DDR3/DDR4 as explained in Section II-A. Making the number of parity bits non-multiples of eight incurs additional area overhead for data buses across DRAM banks, I/O sense amplifiers (IOSAs), write drivers per bank, and others. For example, the actual area overhead of (137, 128) SECDED code in an 8Gb DRAM device is 12.1%, which is larger than that of (72, 64) SECDED code considering these design issues. From the simulation results, DEC and S8EC codes are excluded because they require longer than 3ns decoding time, which

is added to the time from a read command to the output of the first corresponding data called tCL, results in large system performance degradation, as main memory reads are often in the critical paths of applications. Therefore, implementable code candidates become (136, 128) SEC and (72, 64) SECDED codes.

The SEC and SECDED codes are typically constructed using binary linear block codes [9], [11]. A binary (n, k) systematic linear block code is represented as an $r \times n$ H-matrix. The H-matrix of the SECDED code is constructed under the three constraints: 1) there is no zero-weight column; 2) each column is different from any other column; and 3) each exclusive-or-sum of any two columns is different from any column in the H-matrix. SEC code can be constructed if both the first and second constraints are satisfied. The third constraint is related to the DED function. Therefore, we construct compatible (136, 128) SEC and (72, 64) SECDED codes for DRAM device architecture considering decoding time and area overhead of an ECC engine under the above constraints.

C. Architecting In-DRAM ECC

Supporting In-DRAM ECC entails the addition of ECC engines within DRAM devices, the changes to internal DRAM datapath such as GIOs, the introduction of a new DRAM timing parameter, and the increase in values of existing timing constraints. To implement (136, 128) SEC code in $\times$4 or $\times$8 DRAM devices with burst length of 8, additional data bits should be internally prefetched during read and write operations because the number of bits to be transferred per read/write, (8 $\times$ 4) = 32 bits and (8 $\times$ 8) = 64 bits, is smaller than the data size (128b) of a 136b codeword. All the bits of a codeword in a device is stored at the same row of a DRAM bank. The (72, 64) SECDED code also requires this additional internal data prefetching for $\times$4 DRAM devices with burst length of 8. During a write operation, only the written data is partially overwritten in a codeword and parity bits are newly regenerated for the entire 136b codeword. Consequently, this additional data prefetching requires an internal read-modify-write (RMW) operation for each write [26].[4] During a write operation, an internal read command is first generated to get a corrected data in the target address using an ECC engine. The data associated with the write command is modified by combining the corrected data and input data. Then the parity bits are calculated based on the modified data using the ECC engine. Finally, a complete codeword is written to the target address. This internal RMW operation is similar to the data-masked write operation in existing LPDDR4 products [40].

It is possible to locate an In-DRAM ECC engine either right next to a bank or I/O pins (see Figure 6). Between the two options, the latter has one engine per device, leading to lower area overhead than the former option that implements a dedicated In-DRAM ECC engine per bank. To prevent data bus contention between accesses to different banks of a device, which can occur while conducting an internal RMW operation for a write operation on the latter option (one engine per device), we place an ECC engine at each physical bank. An In-DRAM ECC engine consists of an ECC encoder, a syndrome calculator, and a corrector in each physical bank. Figure 6(b) shows a flow of internal sequence of read, modify, and write sub-operations for a write. As an internal read operation is conducted only in a target bank, other writes can be executed in other banks simultaneously. Consequently, the timing constraints caused by an internal RMW operations are confined to a single bank.

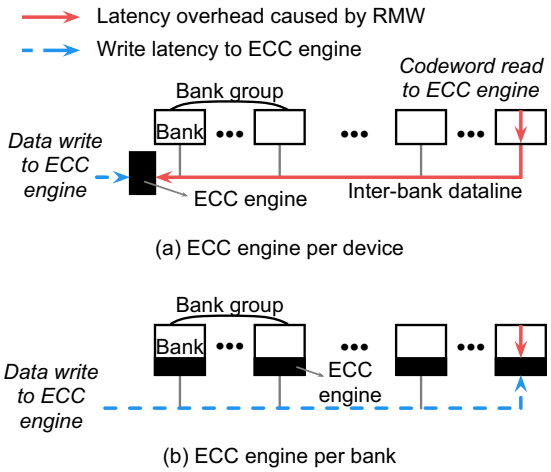


Figure 6: In-DRAM ECC engine can be (a) next to I/O pins (one per device) and (b) right next to a bank (one per bank).

Because a write operation is processed by a RMW operation internally for using In-DRAM ECC, the interval between two consecutive writes to the same bank should be increased. One way to reflect this is to increase the column-to-column delay (tCCD), the minimum column command (i.e., read and write commands) timing [21]. As specified in Section V-A, this internal RMW operation requires an interval of 20ns for near future DRAM process technology nodes, which would significantly degrade the effective bandwidth of a device. Starting from DDR4 [23], there are two types of tCCD: tCCD_L, which defines the minimum delay between two column commands to the same bank group and tCCD_S, the one to different bank groups, where banks in a DRAM device are divided into multiple bank groups (e.g., a DDR4 device has four bank groups, each with four banks). We can leverage this and increase only tCCD_L as each bank has a dedicated ECC engine. It is possible to further reduce the impact of this internal RMW operation by adding a new timing constraint called tCCD_{WR}, which is

[4] A read does not update data back to the BLSAs even if the ECC engine corrects the corresponding codeword in our In-DRAM ECC scheme.

applied only to the consecutive writes to the same bank. This internal RMW operation does not deteriorate the penalty in latency between read and write switches because the read and modify sub-operations of a write are performed before the write sub-operation is finished, making the subsequent read operations unaffected. As for a read operation ahead of a write operation, because the trailing write data can be injected to the internal data bus of a device after the read data are completely transferred out of the device through the data bus, there is enough time for the internal read sub-operation of the trailing write is conducted.

IV. Impacts of In-DRAM ECC on Testing, Reliability, and Performance

A. Synergy between row/column sparing and In-DRAM ECC

The combined use of the redundant row/column sparing method and In-DRAM ECC is effective in handling randomly scattered SCFs for the next generation DRAM devices. The combined use solves the so-called birthday problem [14], [50], which is the probability that, in a given sample of people, two or more have the same birthday. When a (72, 64) SECDED or (136, 128) SEC In-DRAM ECC code is used, the probability associated with the birthday problem is the probability of two or more SCFs occurring within a single codeword. To realize the combined use considering the birthday problem, we conduct testing at ECC codeword granularity during the DRAM test phase.

During the test phase (before device shipment at *t0*), we leverage In-DRAM ECC to manage intermittent SCFs (e.g., parametric faults due to in-spec but marginal time in retention and write recovery of data). A codeword is classified to be either correctable or uncorrectable by In-DRAM ECC depending on the number of faulty cells in the codeword. The codeword having a SCF alone is regarded as correctable; the codeword with two or more SCFs, column faults, and row faults are categorized as an uncorrectable one that is expected to be resolved by the redundant row and column sparing method. Even in single-bit error correctable cases, permanent SCFs such as stuck-at faults and the ones due to out-of-spec retention and write recovery time are preferred to be handled by the redundant sparing method whenever possible to prevent frequent error correction after shipment. During in-field operations (after *t0*), In-DRAM ECC also corrects newly added faults such as RTN and VRT, which is another strength of In-DRAM ECC.

Based on the implication of fault rates in Section II-C, the number of correctable codewords becomes approximately 1k to 100k in an 8Gb DRAM device before *t0*, even though DRAM process scales down. In other words, an 8Gb (= 2^{33}) DRAM has 64M (= 2^{26}) codewords with data bits of 128b (= 2^7) assuming the use of (136, 128) SEC code. Only 100k codewords of In-DRAM ECC are used before *t0* and single-bit errors in remaining 63.9M (= 64M - 100k) codewords are still correctable after *t0*. This can extremely improve the reliability and availability of DRAM devices and main memory systems as quantified in Section V-A.

B. Interaction between In-DRAM and rank-level ECC

In server or desktop systems, multiple DRAM devices form a rank and all the devices in a rank operate in tandem. Each rank often has an error detection and correction feature by using additional devices to manage various faults after *t0*, such as DIMM faults, I/O (pin) faults, and SCFs caused by RTN, VRT, and cosmic rays. This rank-level ECC, which typically adopts a block code as well, consequently interacts with In-DRAM ECC because the codewords of both codes overlap, as pointed out by [47].

Terminology: A main memory system is comprised of memory controllers, channels, and dual-inline-memory-modules (DIMMs). One or more memory controllers are typically implemented in a CPU and each memory controller is connected to one or more channels. Each channel contains an independent set of address, command, and data buses, and is connected to one or more DIMMs. Each DIMM has DRAM devices that are organized logically in one or more ranks. A rank is composed of several DRAM devices that receive the same address/command and operate in tandem. The aggregate width of a data bus in a rank is 64, typically from 16 4-bit wide ($\times$4 DRAM) devices or eight 8-bit wide ($\times$8 DRAM) devices. On DDR3/4, mismatch between internal DRAM operation frequency (tCCD) and external interface frequency (tCK) causes the minimal access granularity as the eight times of the 64-bit bus (burst length of 8), which often matches the size of a CPU cache line. Besides, an ECC DIMM has few DRAM devices for storing ECC parity bits per rank.

Rank-level SECDED: We denote SECDED code used at a rank level in ECC DIMMs by rank-level SECDED to distinguish it from SECDED code used within a DRAM device. Rank-level SECDED typically employs the (72, 64) SECDED code and deals with soft and hard errors that appear after *t0*. An ECC engine in a memory controller generates eight parity bits when 64 data bits are written. Then, 64 data bits and eight parity bits are separately stored into the DRAM devices of a rank. During a read, a single-bit error is corrected and a double-bit error is detected. Previous studies demonstrate that Chipkill reduces the uncorrectable error rate by 10 to 42 times compared to rank-level SECDED [25], [48]. Also, a system outage due to DRAM device failures has been decreased by Chipkill [10]. Hence, to improve reliability of main memory systems, rank-level SECDED is used for low-end servers whereas Chipkill is widely used for high-end servers and datacenters.

Chipkill: Chipkill commonly tolerates failures of an entire DRAM device per rank (or per logical rank consisting of few physical ranks grouped together). There are various correction and detection methods to construct Chipkill depending on server vendors. A widely used commercial Chipkill

method is to use the single-symbol-correction and double-symbol-detection (SSCDSD) codes based on the Galois Field (GF) arithmetic [2]. The symbol size is set to the output width of a DRAM device, thereby tolerating an entire device failure. To keep the ECC parity storage overhead low, a codeword consists of data from multiple memory channels (e.g. two channels are combined together to transfer 8 bursts of 144b codewords per DRAM read/write) [25], [55]. Unless using lengthy CPU cache lines (e.g. 128B), a burst from multiple channels that operate in tandem corresponds to multiple cache lines. Then, the accessed cache lines are split across DIMMs and superfluous cache lines are over-fetched to conduct Chipkill. Consequently, a Chipkill-enabled memory system based on an SSCDSD code incurs wasted power consumption and memory bandwidth in main memory systems [20], [52].

Single-device-data-correction (SDDC) from Intel remedies a single DRAM device failure using a combination of cyclic-redundancy-check (CRC) and parity check codes [6], [20]. SDDC can map out a single failed one and correct a single-bit error even after the failed device is mapped out [6], [19]. SDDC needs one device for CRC and another for parity check for $\times 4$ DRAM devices, thereby supporting Chipkill with a single ECC DIMM without cacheline overfetching.[5] However, two memory channels are required to conduct SDDC with cacheline overfetching for $\times 8$ devices. In IBM POWER7, a single DRAM device failure is recovered out of 18 $\times 8$ devices with cacheline overfetching [18]. Also, if a device is identified to be permanently failed and marked as a bad one, a single clustered four-bit error in an I/O pin can be additionally corrected.

To provide higher reliability and availability, Intel's Chipkill method that remedies two DRAM device failures, double-device-data-correction (DDDC), uses two devices for CRC, one for parity check, and one for spare device [19]. DDDC can recover two sequential DRAM device failures but not simultaneously. If one DRAM device fails in a logical rank, the failed one is recovered by SDDC and is substituted with the spare device. After substitution, another device failure can be remedied by SDDC. DDDC is available only for $\times 4$ DRAM devices with cacheline overfetching.

Interference between rank-level SECDED and In-DRAM ECC: A codeword of In-DRAM ECC and that of rank-level ECC overlap, but neither includes the other. Therefore, these two ECC codes can interact [47]. When (72, 64) SECDED code is used for rank-level ECC, an In-DRAM ECC codeword and a rank-level ECC codeword share 8 bits within a $\times 8$ DRAM device of a rank (see Figure 7(a)). Consider a case that there is a faulty cell that is recognized

[5]Cacheline overfetching refers to the case that the size of an ECC codeword is larger than the datapath width of a memory channel (72 bits for systems supporting ECC DIMMs) and hence two channels are ganged together to serve a memory request. Then, the access granularity to main memory, 128B data, becomes larger than the typical cacheline size, 64B.

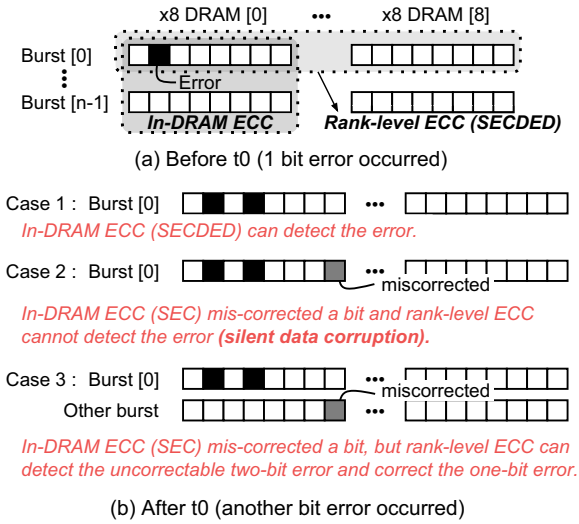


Figure 7: Interference between In-DRAM ECC and rank-level SECDED.

before *t0*, which is supposed to be fixed by In-DRAM ECC. If another bit error occurs next to the already faulty cell in the shared eight bit region, as shown in Figure 7(b), In-DRAM ECC and rank-level SECDED will cope with this by one of the following ways. First, if In-DRAM ECC uses a (72, 64) SECDED code, both In-DRAM ECC and rank-level SECDED codes can detect the error but neither can correct the error. Second, if In-DRAM ECC uses a (136, 128) SEC code and it miscorrects a third bit in the same shared 8 bit region, rank-level SECDED cannot even detect this error, causing a silent data corruption (SDC). Third, if In-DRAM ECC uses a (136, 128) SEC code and it miscorrects a bit within the In-DRAM ECC codeword but outside of the shared 8 bit region, rank-level SECDED can detect the uncorrectable two-bit error and correct the one-bit error. It is possible to construct (136, 128) SEC code in a way that the second scenario mentioned above can never occur (enforcing the miscorrection by a two-bit error within a shared eight ($\times 8$ device) or four ($\times 4$ device) bit region to always occur in another shared bit region), overcoming the SDC problem. In summary, it is unlikely, but still possible for a single-bit failure after *t0* to cause an uncorrectable error when rank-level SECDED is used with In-DRAM ECC.

Chipkill vs. In-DRAM ECC: Chipkill is different from rank-level SECDED in that it corrects any number of error bits from a single DRAM device. The additional errors that cannot be corrected by the rank-level ECC can be corrected by Chipkill, such as SSCDSD, SDDC, and DDDC, because these additional bit errors are confined to a single DRAM device. Besides, there is no difference between (72, 64) SECDED and (136, 128) SEC In-DRAM ECC codes in the reliability perspective, because both can correct one bit error per codeword and additional error bits in the codeword are

handled by the rank-level Chipkill anyway. Hereafter, we take (136, 128) SEC as the In-DRAM ECC code.

C. Overcoming the performance overhead of In-DRAM ECC with long burst writes

In-DRAM ECC turns a write operation into a sequence of read, modify, and write sub-operations internally due to mismatch in the unit of data transfer per read/write and the In-DRAM ECC codeword size (Section III-C). Therefore, the minimal interval between two consecutive writes to the same bank ($tCCD_{WR}$) is several times higher than that between two consecutive reads to the same bank ($tCCD_L$). This significantly deteriorates the performance of bursty writes to a single bank, which can occur during a block copy, such as duplicating/migrating an OS page or a large object, as the least significant bits of logical/physical addresses are typically mapped to few banks in modern computer systems to exploit the spatial locality of applications [56].

To alleviate this performance penalty, we propose a new DRAM command called a long-burst write (LBWR), a variant of a write (WR) command. The difference between a normal write and this LBWR is that the latter accompanies data whose length is equal to the number of data bits of a codeword (i.e., 128 bits of a (136, 128) SEC In-DRAM ECC code) through a burst length longer than the normal one (8 for DDR3/DDR4). Therefore, the read sub-operation can be eschewed in a LBWR. The burst length of this LBWR is 32 for $\times 4$ devices and 16 for $\times 8$ devices.

Modern memory controllers that can service requests different from the receiving order [3], [34], [44] can exploit LBWR by detecting a group of write requests that span an entire codeword and generating a single LBWR command for the group instead of a sequence of WR commands. Read and write requests are clustered within a controller due to timing overhead in read-write switches and the fact that read requests, which are often in a critical path of cores' execution, have higher priority than write requests. Therefore, the memory controller often has several write requests when it is about to service one and has a high chance of generating LBWR commands from the requests with high spatial locality. Bigger write queues [51] can further increase the chance.

V. Evaluation

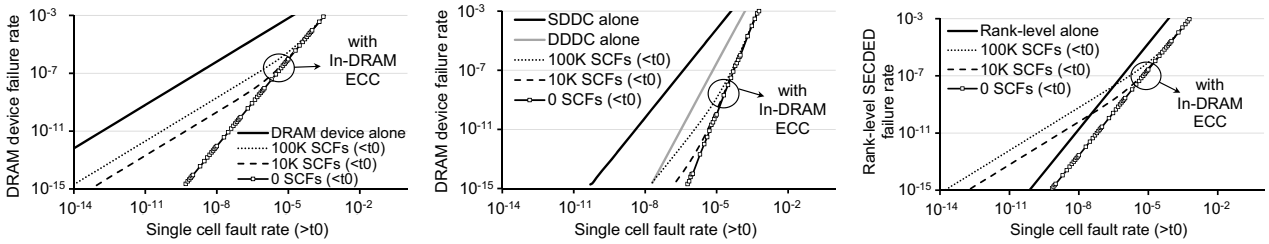
A. Impact of In-DRAM ECC on reliability

To evaluate the effectiveness of In-DRAM ECC on reliability, failure rates are calculated for usage cases of 1) DRAM devices alone and 2) DRAM devices with Chipkill and Rank-level SECDED, based on SCF rates after the device shipment time *t0*. After *t0*, SCFs can occur due to VRT, cosmic neutrons, alpha particles, and radiations. SCFs after *t0* can be either transient, permanent, or intermittent faults [4], [48], [49]. We assume that they are permanent faults to conservatively evaluate reliability and availability. SCFs after *t0* are assumed to be randomly distributed. Column, row, and connection faults after *t0* are omitted because they occur rarely [49], and are easily detected and handled by fault handling schemes, such as rank isolation and DIMM changes. The (136, 128) SEC code is employed for In-DRAM ECC.

DRAM device: Figure 8(a) shows single DRAM device failure rates versus various SCF rates after *t0*. 8Gb $\times 8$ devices with burst length of 8 are used. A DRAM device failure occurs when one or more bit errors exist within any 64b block of a $\times 8$ device after In-DRAM ECC correction. We also calculate DRAM device failure rates when 10k and 100k SCFs are managed not by redundant sparing method but by In-DRAM ECC before *t0*. Even though there are 100k SCFs on average before *t0* in an 8Gb device, DRAM device failure rate with In-DRAM ECC is reduced by about 1,000 times compared to devices without it.

Interaction with Chipkill: Figure 8(b) illustrates SDDC failure rates before and after In-DRAM ECC is employed for various SCF rates after *t0* on $\times 8$ DRAM devices. The failure rate of Chipkill using SSCDSD code is expected to be the same as that of SDDC. To conduct SDDC for $\times 8$ DRAM, 18 $\times 8$ DRAM devices with burst length of 8 are required for cacheline overfetching. An SDDC failure occurs when errors are uncorrected by In-DRAM ECC and SDDC within the overfetched cacheline of 18 $\times 8$ DRAM devices. When there are 100k SCFs before *t0* in each 8Gb device, the SDDC failure rate with In-DRAM ECC is reduced by about 10^6 times at a 10^{-7} SCF rate after *t0* compared with devices without In-DRAM ECC. The synergistic use of In-DRAM ECC and SDDC reduces the SDDC failure rates extremely because SDDC corrects burst errors from a single device. Moreover, the SDDC failure rate with In-DRAM ECC is much smaller than even the DDDC failure rate without it. Consequently, future scaled devices employing In-DRAM ECC can provide much higher reliability than modern devices to main memory systems using Chipkill.

Interaction with rank-level SECDED: Figure 8(c) shows rank-level SECDED failure rates before and after In-DRAM ECC is employed for various SCF rates after *t0*. The rank-level SECDED is assumed to consist of nine $\times 8$ devices employing In-DRAM ECC to calculate the rank-level SECDED failure rate. A rank-level SECDED failure occurs when one or more bit errors exist within any cacheline of nine $\times 8$ DRAM devices with burst length of 8 after corrections by In-DRAM ECC and rank-level SECDED. At the same SCF rate before *t0*, combining In-DRAM ECC with SDDC gives a lower rank-level failure rate than combining In-DRAM ECC with rank-level SECDED, which demonstrates the strength of Chipkill and its variants that fix errors from an entire device. In the cases of fixing 10k and 100k SCFs with In-DRAM ECC before *t0*, the failure rates of rank-level SECDED with In-DRAM ECC are larger than those without In-DRAM ECC when SCF rates after *t0* are smaller



(a) Device failure rates (b) SDDC failure rates (c) Rank-level SECDED failure rates

Figure 8: SCF rates after *t0* versus (a) the DRAM device, (b) the SDDC, and (c) the rank-level SECDED failure rates with or without (136, 128) SEC In-DRAM ECC where 0, 10k, and 100k SCFs before *t0* are corrected not by the redundant row/column sparing method but by In-DRAM ECC.

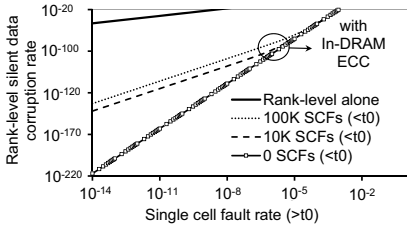


Figure 9: SCF rates after *t0* vs. rank-level SECDED silent data corruption rates with/without (136, 128) SEC In-DRAM ECC where 0, 10k, and 100k SCFs before *t0* are corrected not by the redundant sparing method but by In-DRAM ECC.

Resource	Value
Number of (cores, MCs)	(16, 4)
Coherence policy	MOESI
Per core:	
(Freq, issue/commit width)	(3.6GHz, 4 / 4)
Issue policy	Out-of-Order
L1 I/D $ size/associativity	16KB / 4
L2 $ size/associativity	1MB / 16
L1, L2 $ line size	64B
Per memory controller (MC):	
(# of channels, Req Q size)	(1, 32)
(Capacity per rank, BW)	(16GB, 19.2GB/s)
(# of ranks, sched. policy)	(4, PAR-BS [34])
DRAM page policy	Minimalist open [27]

Table II: Default parameters of the simulated system.

than 10^{-7}, which is different from the case of interacting with Chipkill. This is due to the interference between two ECC codes as analyzed in Section IV-B. As the SCF rates increase, combining rank-level ECC and In-DRAM ECC leads to lower rank-level failure rates compared with the use of rank-level SECDED ECC alone. It is always better to combine rank-level ECC and In-DRAM ECC when In-DRAM ECC does not correct any SCF before *t0*. Furthermore, as depicted in Figure 9, rank-level SECDED SDC (silent data corruption: undetected errors) rates are extremely reduced due to In-DRAM ECC, suggesting that In-DRAM ECC can be beneficial with higher SCF rates.

B. Impact of In-DRAM ECC on performance

Experimental setup: The significant improvement in DRAM device and rank failure rates by In-DRAM ECC accompanies increases in DRAM access latency (tCL) and the minimal interval between two consecutive writes to the same bank ($tCCD_{WR}$). We ran single-threaded, multi-programmed, and multi-threaded workloads in a simulated multicore system to evaluate the system-level impact of In-DRAM ECC. We modified McSimA+ [1] to support $tCCD_{WR}$ and adjusted the tCL values accordingly. The simulator also models the bank-group related timing constraint of DDR4, where a DRAM device consists of four bank groups, each having four banks. A read or write operation to the same group should be separated at least by $tCCD_L$, which is larger than the device peak bandwidth specified by $tCCD_S$. Table II summarizes the default parameters of the simulated system.

We used SPEC CPU2006 [12], SPLASH-2X [53], PARSEC [5], and MICA [30] for evaluation. We sorted the SPEC applications by the main memory accesses per kilo-instructions (MAPKI) and categorized the following nine applications with the highest MAPKI values as spec-high: mcf, milc, soplex, GemsFDTD, libquantum, lbm, omnetpp, leslie3d, and sphinx3. Simpoint [46] was used to identify the representative phases, where two slices with the highest weight values were chosen per application. Each slice consists of 100M instructions. Only one memory channel was used while simulating single-threaded applications in order to stress the main memory system. Among the multi-programmed workloads, mix-high assigns spec-high applications to cores; the simulation points from all the SPEC applications are blended to mix-blend. We simulated the regions of interest of the following multi-threaded workloads, radix and fft from SPLASH-2X, fluidanimate and streamcluster from PARSEC, and a key-value store MICA.

Results: Figure 10 shows the relative IPC (instructions per cycle) of the following five DRAM configurations shown in Table III: 1) the baseline without In-DRAM ECC, which has

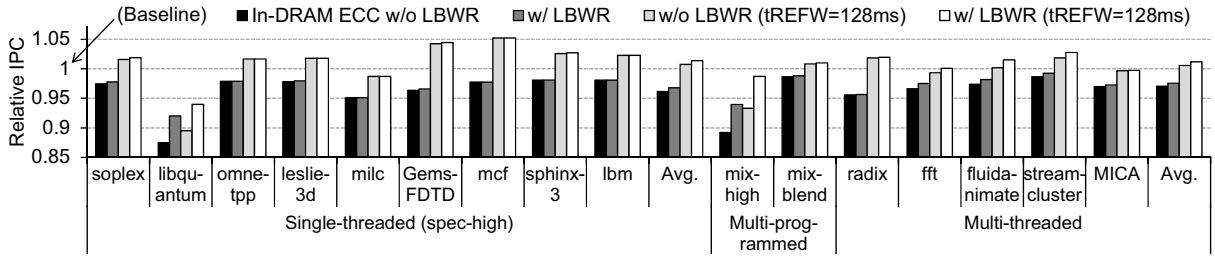


Figure 10: Performance impact of In-DRAM ECC. Five DRAM configurations were simulated using single-threaded, multi-programmed, and multi-threaded workloads; 1) baseline without In-DRAM ECC, 2) with In-DRAM ECC, 3) 2) + LBWR (long-burst write) support, 4) 2) + longer cell retention time, and 5) 4) + LBWR.

Parameter	Baseline	In-DRAM ECC	
		w/o LBWR	w/ LBWR
tRCD (ns)	13.3	13.3	13.3
tCL (ns)	13.3	14.9	14.9
tRC (ns)	45.3	45.3	45.3
tWR (ns)	15.0	15.0	15.0
tRP (ns)	13.3	13.3	13.3
tREFW (ms)	64	64 / 128	64 / 128
$tCCD_S$ (ns)	3.3	3.3	3.3
$tCCD_L$ (ns)	5.0	5.0	5.0
$tCCD_{WR}$ (ns)	N/A	20.0	20.0

Table III: Default DRAM timing values. LBWR stands for a long-burst write command.

no performance penalty but incurs high cost premium due to high device failure rates; 2) the one with In-DRAM ECC, and hence has higher tCL and $tCCD_{WR}$ values; 3) the one with In-DRAM ECC and LBWR (long-burst write); 4) the one with In-DRAM ECC and 2× extended data retention time (tREFW = 128ms) assuming that In-DRAM ECC fixes SCFs induced by cells that have the actual retention time less than 128ms; and 5) the one with In-DRAM ECC, LBWR, and 2× extended data retention time. The IPC values are normalized to that of the baseline for each tested workload.

We made the following key observations. First, In-DRAM ECC reduced the performance of memory intensive applications, but the degree of performance reduction is smaller than 5% on most applications as out-of-order cores partially hide memory access latency from their critical execution paths. *libquantum* and *mix-high* are two workloads that experienced more than 10% of performance degradation by adopting In-DRAM ECC because they frequently generate write requests with high spatial locality and hence suffer from the high $tCCD_{WR}$ value. Second, the configuration supporting the LBWR commands alleviates the performance drop due to this large interval between write commands by combining subsequent write commands that constitute the data bits of an entire In-DRAM ECC codeword into the corresponding LBWR commands. For example, adopting LBWR increased the performance of write-intensive *libquantum* by 5.0% compared to the configuration with In-DRAM ECC. Third, doubling the DRAM retention time by utilizing In-DRAM ECC gave positive impacts on performance, concurring with the results in [57]. For example, longer retention time gives 5.1% and 4.8% performance improvement over the baseline on *mcf* and *GemsFDTD*. With increased retention time, In-DRAM ECC outperforms the baseline with few exceptions. The effectiveness of LBWR is orthogonal to longer tREFW.

VI. Conclusion

In this paper, we have holistically assessed the impacts of In-DRAM ECC in contemporary and future main memory systems from reliability, area, power, and performance perspectives. More specifically, we first identified that DRAM cells are more vulnerable to random telegraph noise, retention time variation, and fluctuation in write recovery time, which lead to a surge in randomly scattered single-cell faults (SCFs). Second, we demonstrated that In-DRAM ECC could substantially reduce the DRAM device failure rates when combined with the traditional redundant row/column sparing method. Third, we showed that 136b codeword with SEC (single-bit error correction) is more desirable than 72b codeword with SECDED (SEC and double-bit error correction) by analyzing the overhead and interaction of In-DRAM ECC with the DRAM rank-level ECC such as SECDED and Chipkill. Next, to minimize the performance degradation associated with using 136b codeword, we proposed the long-burst write command and demonstrated that the performance degradation of adopting In-DRAM ECC could be limited to few percents on average, even without including a potential performance gain due to increased retention time. We expect that In-DRAM ECC enables continuous scaling down of DRAM process technologies and highly reliable main memory systems to coexist.

Acknowledgment

We thank Yuhwan Ro and Eojin Lee for their contributions on performance simulations.

References

[1] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *ISPASS*, 2013.

[2] AMD, *BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors*, 2013.

[3] R. Ausavarungnirun, K. Chang, L. Subramanian, G. Loh, and O. Mutlu, "Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems," in *ISCA*, Jun 2012.

[4] A. Avizenis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of the Secure Computing," *Transaction on Dependable and Secure Computing*, pp. 11–33, 2004.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008.

[6] R. G. Blankenship, D. W. Brzezinski, and E. F. M. Valverde, "Memory Error Detection and/or Correction," 2012, US Patent 8,250,435.

[7] S. Borkar, "Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, 2005.

[8] D. Bossen, "b-Adjacent Error Correction," *IBM Journal of Research and Development*, pp. 402–408, 1970.

[9] S. Cha and H. Yoon, "Single-Error-Correction and Double-Adjacent-Error-Correction Code for Simultaneous Testing of Data Bit and Check Bit Arrays in Memories," *Transaction on Device and Materials Reliability*, pp. 529–534, 2014.

[10] T. J. Dell, "White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," *IBM Microelectronics*, pp. 1–23, 1997.

[11] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*. John Wiley & Sons, 2006.

[12] J. L. Henning, "SPEC CPU2006 Memory Footprint," *Computer Architecture News*, pp. 84–89, 2007.

[13] R. Ho, K. W. Mai, and M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[14] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*. Springer Science & Business Media, 2011.

[15] M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM Journal of Research and Development*, pp. 395–401, 1970.

[16] K. K. Hung, H. Chenming, and Y. C. Cheng, "Random Telegraph Noise of Deep-submicrometer MOSFETs," *IEEE Electron Device Letters*, vol. 11, no. 2, pp. 90–92, 1990.

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

[18] IBM, *POWER7 System RAS Key Aspects of Power Systems Reliability, Availability, and Serviceability*.

[19] Intel, *Xeon Processor E7 Family: Reliability, Availability, and Serviceability-Advanced Data Integrity and Resiliency Support for Mission-Critical Deployments*.

[20] Intel, *Server Board S2600GZ/GL-Technical Product Specification*, June 2014.

[21] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.

[22] JEDEC, *DDR3 SDRAM Specification*, 2010.

[23] JEDEC, *DDR4 SDRAM Specification*, 2012.

[24] JEDEC, *LPDDR4 SDRAM Specification*, 2014.

[25] X. Jian, J. Sartori, H. Duwe, and R. Kumar, "High Performance, Energy Efficient Chipkill Correct Memory with Multidimensional Parity," *Computer Architecture Letters*, pp. 39–42, 2013.

[26] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *Memory Forum collocated at ISCA*, 2014.

[27] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *MICRO*, 2011.

[28] S. H. Kim *et al.*, "A Low Power and Highly Reliable 400Mbps Mobile DDR SDRAM with On-Chip Distributed ECC," in *ASSCC*, 2007.

[29] J. G. Lee, Y. H. Jun, K. H. Kyung, C. Yoo, Y. H. Cho, and S. I. Cho, "A New Column Redundancy Scheme for Yield Improvement of High Speed DRAMs with Multiple Bit Pre-fetch Structure," in *VLSIC*, 2001.

[30] S. Li *et al.*, "Architecting to Achieve a Billion Requests per Second Throughput on a Single Key-value Store Server Platform," in *ISCA*, 2015.

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

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

[33] J. A. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," *IBM Journal of Research and Development*, pp. 187–212, 2002.

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

[35] T. Nagai *et al.*, "A 65nm Low-Power Embedded DRAM with Extended Data-Retention Sleep Mode," in *ISSCC*, 2006.

[36] P. J. Nair, D. H. Kim, and M. K. Qureshi, "ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates," in *ISCA*, 2013.

[37] P. J. Nair, V. Sridharan, and M. K. Qureshi, "XED: Exposing On-Die Error Detection Information for Strong Memory Reliability," in *ISCA*, 2016.

[38] R. Naseer and J. Draper, "DEC ECC Design to Improve Memory Reliability in Sub-100nm Technologies," in *ICECS*, 2008.

[39] S. C. Oh *et al.*, "Automatic Failure Analysis System for High Density DRAM," in *ITC*, 1994.

[40] T. Y. Oh *et al.*, "A 3.2Gb/s/pin 8Gb 1.0V LPDDR4 SDRAM with Integrated ECC engine for Sub-1V DRAM Core Operation," in *ISSCC*, 2014.

[41] B. Parhami, "Defect, Fault, Error, ..., or Failure?" *Reliability, IEEE Transactions on*, vol. 46, no. 4, pp. 450–451, 1997.

[42] W. W. Peterson and J. E. J. Weldon, *Error Correcting Code*, 2nd ed. MIT Press, 1972.

[43] P. Restle, J. Park, and B. Lloyd, "DRAM Variable Retention Time," in *IEDM*, 1992.

[44] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA*, 2000.

[45] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009.

[46] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, 2002.

[47] Y. H. Son, S. Lee, S. O, S. Kwon, N. S. Kim, and J. Ahn, "CiDRA: A Cache-inspired DRAM Resilience Architecture," in *HPCA*, 2015.

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

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

[50] C. H. Stapper and H. S. Lee, "Synergistic Fault-Tolerance for Memory Chips," *Transaction on Computers*, pp. 1078–1087, 1992.

[51] Stuecheli, Jeffrey and Kaseridis, Dimitris and Daly, David and Hunter, Hillery C and John, Lizy K, "The Virtual Write Queue: Coordinating DRAM and Last-Level Cache Policies," in *ISCA*, 2010.

[52] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems," in *ISCA*, 2012.

[53] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *ISCA*, 1995.

[54] D. Yaney, C. Lu, R. Kohler, M. Kelly, and J. Nelson, "A Meta-Stable Leakage Phenomenon in DRAM Charge Storage-Variable Hold Time," in *IEDM*, 1987.

[55] D. H. Yoon, J. Chang, N. Muralimanohar, and P. Ranganathan, "BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs," in *ISCA*, 2012.

[56] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms," in *RTAS*, 2014.

[57] T. Zhang, M. Poremba, C. Xu, G. Sun, and Y. Xie, "CREAM: a Concurrent-Refresh-Aware DRAM Memory Architecture," in *HPCA*, 2014.