SuDoku: Tolerating High-Rate of Transient Failures for Enabling Scalable STTRAM

Prashant J. Nair* Bahar Asgari† Moinuddin K. Qureshi†

*Department of Electrical and Computer Engineering
University of British Columbia
prashantnair@ece.ubc.ca

†Department of Electrical and Computer Engineering
Georgia Institute of Technology
{bahar.asgari,moin}@gatech.edu

***Abstract*—Conventionally, systems have relied on technology scaling to provide smaller cells, which helps in increasing the capacity of on-chip and off-chip structures. Unfortunately, scaling technology to smaller nodes causes increased susceptibility to faults. We study the problem of efficiently tolerating transient failures using scalable *Spin-Transfer Torque RAM (STTRAM)* as an example. At smaller feature sizes, the energy required to flip a STTRAM cell reduces, which makes these cells more susceptible to random failures caused by thermal noise. Such failures can be tolerated by periodic scrubbing and provisioning each line with Error Correction Code (ECC). However, to tolerate a desired bit error-rate, the cache needs ECC-6 (six bit error correction) per line, incurring impractical storage overheads. Ideally, we want to tolerate these faults without relying on multi-bit ECC.**

We propose *SuDoku*, a design that provisions each line with ECC-1 and a strong error detection code, and relies on a region-based RAID-4 to perform correction of multi-bit errors. Unfortunately, simply having such a RAID-4 based architecture is ineffective at tolerating a high-rate of transient faults and provides an MTTF in the order of only a few seconds. We describe a novel data resurrection scheme that can repair multiple faulty lines in a RAID-4 region to increase the MTTF to several hours. We propose an extension of SuDoku, which hashes a given line into two regions of RAID-4 to significantly enhance reliability and increase the MTTF to trillions of hours. Our evaluations show that SuDoku provides 874x higher reliability than ECC-6, incurs 30% less storage than ECC-6, and performs within 0.1% of an ideal fault-free baseline.

***Keywords*-STTRAM, Reliability, Transient Failure.**

I. Introduction

Technology scaling is a key enabler for high-capacity memory systems. Technology scaling has aided the advancement of several popular memory technologies such as Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM). Unfortunately, while reducing the sizes of the cells improves memory density, it also makes them vulnerable to faults. As memory systems scale, they tend to become susceptible to faults that occur intermittently over time (transient faults) and reduce their ability to retain data. For instance, at smaller nodes, SRAM cells cannot retain data operate at lower voltages and tend to fail randomly. Similarly, at sub-20 nm nodes, DRAM cells have a lower retention time and tend to show a high rate of transient faults. Transient faults tend to be an even bigger concern for incorporating new memory technologies. To study the impact of transient faults, this paper uses an emerging technology, Spin-Transfer Torque Random-Access Memory (STTRAM).

Spin-Transfer Torque Random-Access Memory (STTRAM) is an emerging technology that can enable large on-chip caches with 3X-4X as high as SRAM [1], [2]. STTRAM cells store data in the form of the orientation of a soft ferromagnetic material, which changes the state with passage of current [3]. The ability of STTRAM to retain stored data is dictated by a metric called as *Thermal Stability Factor (*Δ*)* [2], [4]. While demonstrations of STTRAM have shown years of data retention, such designs typically use a $\Delta \geq 60$ and require larger cell area, higher energy per write, and longer write latencies [5], [6]. Unfortunately, scaling STTRAM reduces its Δ and retention time, and causes transient faults [5], [7], [8].

The Δ of STTRAM cells is also subjected to variations. Recent studies from the industry depict up to a 10% σ in Δ due to process variation [1], [5], [8]. For our studies, we choose an STTRAM device with a mean Δ of 35 with a normalized standard deviation (σ) of 10% (corresponding to 22nm node [5], we analyze various values of Δ in VII-G). The mean time to failure (MTTF) for a cell with a Δ of 35 is approximately 18 days, however, after taking the variation (σ=10%) into account, on average, it takes only one hour for a cell to fail [9], [10]. We deem that this range of retention time may still be sufficient for caching applications. Unfortunately, a large cache contains millions of cells, all of which must operate reliably to avoid data loss. Therefore, even though the MTTF is an hour, our analysis shows that in a period of 20ms, we can expect 2880 bits to experience retention failures in a 64MB STTRAM cache. This translates to a bit error rate (BER) of 5.3×10^{-6} within 20ms.[1]

Retention failures are a problem in other technologies as well. For example, DRAM systems rely on periodic refreshes to maintain data integrity. Prior studies [10], [11] have advocated a DRAM-style refresh for STTRAM, whereby each line is periodically read into a buffer and written back. However, the retention failure occurs quite differently in STTRAM. While in DRAM, retention failure occurs as a result of charge leakage, in STTRAM, it occurs because of a random thermal noise that flips the direction of the magnetic cell. Therefore, unlike DRAM, in which we can maintain data integrity by restoring the charge before it leaks below a certain threshold, we cannot restore the value of a STTRAM cell by reading and rewriting it. Moreover, the probability that a bit flips because of thermal noise within a given time window is independent of the duration since the last access. Therefore, DRAM-style refresh is ineffective for STTRAM.

In essence, retention failures in STTRAM are akin to transient failures in charge-based memories caused by external

[1]We pick a period of 20ms for our analysis to ensure that the LLC can be scrubbed while incurring an overhead of not more than a few percent [5]. Analysis with other scrub interval is presented in Section VII-E.

high energy particles. Prior techniques [12]–[15], which are highly effective at handling permanent faults also become inapplicable for such faults as they are transient, and any cell is liable to incur a bit flip at a certain time interval.

Note: Prior mitigation techniques [12]–[15] that are highly effective at tolerating permanent faults become ineffective at tolerating transient faults. These techniques rely on disabling or sparing the failed bit. At our error rates, every bit in the cache is expected to encounter a failure once every few hours.

A practical solution to mitigate retention failures in STTRAM is employing periodic scrubbing and using per-line Error Correction Code (ECC) to tolerate all faults within the scrub interval [5], [6], [10]. We use a 64MB STTRAM and employ a scrub interval of 20ms for our studies. We seek a target FIT-Rate (Failure in Time period of 1 billion hours) of 1 for the STTRAM cache. For tolerating a BER of 5.3×10^{-6}, each cache line would need to provisioned with ECC-6 (correcting six errors per line). ECC-6 would have a storage overhead of 60 bits per 64-byte line (11.7%). These multi-bit encoders and decoders use complex arithmetic calculations. As these ECC calculations are non-trivial, multi-bit ECC encoders and decoders incur latencies of several tens of cycles. On the contrary, simple ECC-1, can perform encoding and decoding using a small lookup table and a single cycle latency. Ideally, we want to tolerate high rate of retention failures by paying the low cost of ECC-1.

This paper proposes *SuDoku*,[2] a resilient cache architecture that efficiently handles scalability challenges without incurring overheads of strong ECC. SuDoku uses the insight that even at a BER of 5.3×10^{-6}, only six bits in every Million bits will be faulty. Therefore, 99.9999% of cache lines will either have no faulty bits, or have only one faulty bit. SuDoku handles the common case by a single bit ECC (ECC-1) with each line. To handle multi-bit faults, SuDoku appends each line with cyclic redundancy code (CRC-31) [16], a strong detection code that detects up to seven faults. SuDoku relies on a region-based RAID-4 scheme for correcting multi-bit failures, whereby each group of 512 lines is provisioned with one dedicated parity line. On detecting an uncorrectable failure in any line of a group, we use the associated parity line to reconstruct data. The likelihood of invoking RAID-4-based correction is small (on average, only four lines in a 64MB cache will have 2+ failures in a 20ms scrub interval). We refer to this base SuDoku as *SuDoku-X* (Section III). Unfortunately, the MTTF of SuDoku-X is only 3.7 seconds and we need a stronger scheme to supplement SuDoku-X.

[2]SuDoku is a numerical puzzle, where the data value in a blank cell is constructed using the available data values in row, column, and smaller grid. Our design shares a similar spirit in data recovery for faults, hence the name.

The dominant failure mode of SuDoku-X is due to groups with two faulty lines, each with exactly two bit-failures. Traditional RAID-4 would be unable to perform correction if two units of the region are deemed faulty. However, we use the insight that 2-bit failures can be corrected with ECC-1, if the position of one of the faulty bits is known, because one can fix this bit error by flipping the faulty bit. We call this scheme to as *Sequential Data Resurrection (SDR)*. To perform SDR, we first scan the region of RAID-4 (group) and correct all line with single bit faults. Then, we compute the parity of the group and compare it against the stored parity, and identify the bit positions with a mismatch. For the remaining faulty lines, we sequentially flip each identified position of faulty-bit and perform ECC-1 based correction. If after performing the ECC-1 correction, the per-line CRC indicates no error, then the line is deemed to be successfully corrected. We refer to this design with SDR as *SuDoku-Y* (Section IV). The MTTF of SuDoku-Y is 3.9 hours.

Note: Simply having a RAID-based design, as proposed in recent works [17], [18], in insufficient at tolerating a high-rate of transient faults. We propose two novel optimization that significantly improves (by 15 orders of magnitude) the robustness of RAID-based design at tolerating a high-rate of transient faults.

SuDoku-Y fails in two situations: First, when a region has two lines each with two overlapping faults, so, parity cannot detect the their positions. Second, when a group has 3+ faulty lines each with 2+ faulty bits, so, SDR cannot be applied. To overcome this, SuDoku uses the concept of skewed-hashing and significantly enhances its effectiveness. Rather than restricting a line to participate in exactly one parity group, we use two hashes and let each line in the cache to map to two separate groups. If the faulty lines for a given region are uncorrectable under the first hash, SuDoku tries to repair the faulty lines using the group formed using the second hash. We refer to this design of SuDoku with skewed hashing as *SuDoku-Z* (Section V). The MTTF of SuDoku-Z is 8250 billion hours.

SuDoku-Z is consistently stronger than ECC-6 and tolerates a higher variability in Δ. For instance, at σ = 10%, SuDoku-Z is 874x as reliable as ECC-6 (0.092 FIT). SuDoku achieves the high reliability without relying on the storage and latency overheads of ECC-6. SuDoku uses 256KB of SRAM to store parities for a 64MB cache (0.39% area SRAM overhead). The latency overheads of error correction with SuDoku are incurred rarely and do not have any measurable impact on performance ($<$0.01%). While SuDoku tolerates high rates of transient faults, it is also effective for tolerating permanent faults. We also discuss how SuDoku can be used to improve the reliability of other technologies (Section VI).

II. Motivation

When memory cells are scaled to smaller technology nodes, they become more susceptible to failures [14], [19]–[22]. Several recent studies have proposed cost-effective solutions that can tolerate permanent fault rates of as high as 100 parts per million (ppm) in DRAM [14], [15] and 1000 parts per million in SRAM caches [12], [21]–[23] while avoiding the storage overheads of using strong ECC codes. Unfortunately, unlike cells with permanent faults which tend to be deterministic, any cell can become faulty at run-time due to a transient fault. We study the problem of high-rate of transient faults using scalable STTRAM as an example. This section provides a brief background on STTRAM scaling, traditional techniques for tolerating scaling-related failures, and shortcomings of these techniques.

A. Overview of STTRAM

STTRAM is a highly dense replacement of SRAM for large on-chip caches. STTRAM stores data by polarizing the magnetic tunnel junction (MTJ) within each cell. To this end, each cell has a free layer and a fixed layer. The fixed layer is always polarized in a constant direction. By applying current in different directions, the free layer can be polarized in the parallel and anti-parallel direction with respect to the fixed layer. This difference in the orientation of polarization changes the resistance of the STTRAM cell and determines the bit stored within each cell.

B. Background on STTRAM Scaling

Once the MTJ layer is polarized, it is susceptible to temperature variations. Robustness of the MTJ-layer to temperature variations is indicated by the BER (p_{cell}), which follows a Poisson distribution, and it depends on the thermal stability factor (Δ) of the MTJ layer. Equation 1 shows the impact of Δ on p_{cell} for a given time period (t_s), where f_0 is the thermal attempt frequency (1GHz) [5]–[7], [24].

$$p_{cell}(t_s) = 1 - e^{-\lambda \cdot t_s} \qquad \left(\text{where } \lambda = \frac{f_0}{e^{\Delta}} \implies \frac{10^9}{e^{\Delta}} \right) \quad (1)$$

As cells scale, their Δ must remain unaffected to maintain a low p_{cell}. However, Δ is proportional to the volume of the free layer (V_f). Thus, as STTRAM scales, maintaining V_f becomes more challenging. While reducing the feature size, if the V_f decreases by 2x, then Δ also reduces by 2x and increases p_{cell}. Table I shows that as mean Δ reduces from 60 to 35, the BER (over 20ms) increases nearly six orders of magnitude to 5.3×10^{-6}. Such random thermal failures are one of the biggest obstacles of scaling STTRAM [19], [20].

Table I
Thermal Stability vs Error Rate (20ms period). Data recomputed from Figure 13 and Figure 17 in [5].

Mean Thermal Stability (Δ) with σ = 10%	60 (32nm node)	35 (22nm node)
Bit-Error Rate (p_{cell})	2.7×10^{-12}	5.3×10^{-6}

C. Ineffectiveness of DRAM-Style Refresh

Unfortunately, the failure model of STTRAM due to thermal noise is quite different from DRAM retention failures. Retention failure in DRAM cell occurs because of gradual loss of charge, so rewriting the data replenishes that charge. However, thermal failures cause STTRAM to lose data instantly and at unpredictable times. If a cell flips within the refresh interval, simply reading the same value and rewriting it to the cell will not tolerate failures in STTRAM. Therefore, DRAM-style refreshing is ineffective for STTRAM [5], [6].

D. Effective Solution: Scrubbing and ECC

A practical solution to mitigate thermal failures in STTRAM is to employ periodic scrubbing and use a strong per-line ECC [5], [6], [10] to tolerate all the bit errors within a scrub interval. We can draw analogies between transient faults in DRAM due to lower refresh-rate and transient faults in STTRAM due to lower scrubbing-rate. This is because, in both case, a lower rate of refresh or scrubbing causes retention failures. While DRAM simply requires refreshing charge in data cells, STTRAM requires reading data and applying ECC to correct the read data before writing it back.

We use a 64MB STTRAM for our studies and employ a scrub interval of 20ms. We seek a target FIT rate of atmost one, translating to at most one uncorrectable errors in one billion hours of operation.[3] To reach the target FIT rate, we need to equip each line with ECC-6, as shown in Table II. Unfortunately, provisioning each cache line with ECC-6 incurs significant overheads in terms of latency and storage (60 bits per line). Ideally, we would like to overcome the latency, area, and complexity challenges using simple ECC-1 (10 bits per line) in the common case and still be able to tolerate six or more bits of transient faults.

E. Insight: Optimize for the Common Case

As shown in Table II, the likelihood of multi-bit error is very uncommon. For example, even if each line was provisioned with ECC-1, only four lines in a 64MB cache would experience multi-bit failures within the 20ms scrub interval. Unfortunately, we do not know which line would encounter the multi-bit failures. Moreover, the lines with multi-bit failure will change between scrub intervals. As we don't know which lines will encounter failures, the prior work on tolerating STTRAM failures [5], [6] naively allocated uniform amount of error correction entries with each line, thus incurring significant ECC overheads. Our insight for reducing the overhead of tolerating high error rates is to give lines enough ECC entries to tolerate the common case (ECC-1). We equip each line with strong detection code (CRC-31) to detect multi-bit failures and rely on an alternate low-cost mechanism to perform multi-bit error correction.

[3]Typically, a Chipkill protected DRAM memory has a FIT rate slightly exceeding 1 FIT, so our target FIT ensures that the reliability of the overall system is not dominated by the cache.

Table II
FIT RATE OF 64MB CACHE FOR VARIOUS ECC, BER OF 5.3×10^{-6} IN SCRUB INTERVAL OF 20MS.

ECC per line	ECC-1	ECC-2	ECC-3	ECC-4	ECC-5	ECC-6
Probability of line-failure in 20ms	3.9×10^{-6}	3.8×10^{-9}	2.9×10^{-12}	1.9×10^{-15}	10^{-18}	4.9×10^{-22}
Probability of cache-failure in 20ms	9.8×10^{-1}	4×10^{-3}	3.1×10^{-6}	2×10^{-9}	1.1×10^{-12}	5.1×10^{-16}
Cache FIT-Rate	$>10^{14}$	7.2×10^{11}	5.5×10^{8}	3.5×10^{5}	191	0.092

III. SUDOKU-X: BASE DESIGN

We propose *SuDoku*, a resilient architecture, which tolerates a high rate of transient faults at low cost. We explain the basic design, *SuDoku-X*, before discussing our enhancements of SuDoku. We use the insight that even at a BER of 5.3×10^{-6}, only five in every Million bits will be faulty. Therefore, 99.9999% of cache lines (64Byte size) will either have zero or one faulty bit. SuDoku handles this common case by provisioning each line with ECC-1 and provides an alternate means for multi-bit error correction. We use a scrub interval of 20ms with a BER of 5.3×10^{-6} (Δ,σ = 35,10%) within the scrub interval (sensitivity to these parameters is provided in Section VII).

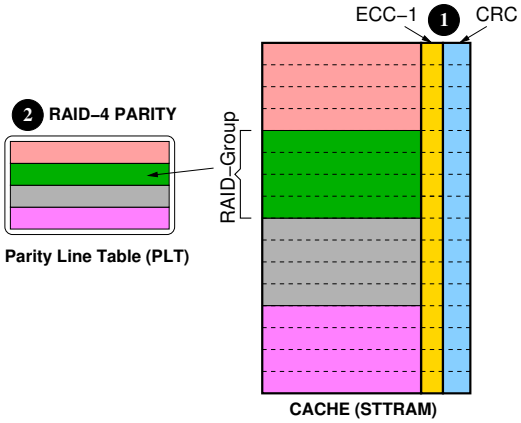

Figure 1. The Organization of SuDoku-X. Each line uses ECC-1 and CRC-21. RAID-4 corrects multi-bit errors. The PLT stores the parities.

A. The Organization

Even at a high BER (5.3×10^{-6}), only a few lines would encounter multi-bit errors. For example, within a 64MB cache, only four lines are expected to have multi-bit errors. SuDoku-X provides two levels of protection – one to handle single-bit error (common case) and another to handle multi-bit errors (low cost). Figure 1 shows the organization of a cache with SuDoku-X. Each line is equipped with an ECC-1 to handle the case of one bit error locally and quickly. Each line is also provisioned with a strong error detection code, CRC-31, which detects up to seven errors in a line (all even number of errors beyond seven bits are detected with extremely high probability). The CRC-31 requires a storage overhead of 31 bits per line. When CRC detects a line with multi-bit error, SuDoku-X uses an alternate mechanism for correction. To achieve multi-bit error correction at low cost, we use a scheme based on the concept of RAID-4 [25], [26].

Unlike strong ECC, RAID-4 requires simple encoders and decoders to correct multi-bit errors. However, RAID-4 corrects only one faulty line within the protected region. Unfortunately, we expect several lines (four on average) with multi-bit errors within a scrub interval. If we have a RAID-4 across all the lines, then we will be unable to correct them. Therefore, we partition the cache into equal-sized regions, called *RAID-Group*, each provisioned by a parity line. This line maintains the parity information for all the lines in the RAID-Group. For example, in Figure 1 the cache contains 16 lines, which are split into four RAID-Groups of four lines each. The parity for each Raid-Group is stored in a separate structure called the *Parity Line Table (PLT)*. We use a RAID-Group of 512 lines, so the PLT is only 0.2% of the cache size. As each write to the cache must also update the PLT, one can provide sufficient bandwidth to the PLT, either by making it heavily banked or in SRAM or both. A line with multi-bit error can be repaired using the respective parity line stored in the PLT for the RAID-Group.

B. Error-Free Operation

This section explains read and write operations for SuDoku-X in the common case. A read operation fetches the ECC-1 and CRC-31 along with the data line. The cache controller first verifies if the line is faulty by checking its CRC syndrome. This can be performed within one cycle. If the syndrome is "0", the line is deemed to be non-faulty. Note that PLT is not accessed for a read operation. For a write operation, the cache controller must update data in the stored cache line, as well as the parity information in the PLT. These updates can be performed as two sequential read-modify-write operations. STTRAM usually employs a read-modify-write scheme to reduce the number of bit-flips and reducing write power and latency [27], [28]. The first read-modify-write is to the data line in the cache. As part of this operation, the controller identifies the position of the bits get modified by the write. The second read-modify-write is to the respective parity line in the PLT, and flips the bits corresponding to the locations of the modified data bits.

C. Performing Error Correction

This section examines how SuDoku-X performs correction. When a line is accessed, the CRC associated with the line will detect possible errors. The repair depends on whether the line encountered a single-bit or multi-bit error.

1) Repairing Single-Bit Errors: As the most common case of errors is a single-bit error, once we find a CRC error, we first try the ECC-1-based repair for the line. If the ECC-1 performed a correction, we recompute the CRC using the corrected data value. If the CRC matches with the stored CRC, we deem the line to be corrected successfully.

2) Repairing Multi-Bit Errors: If a line encounters a multi-bit error, then even after correction using ECC-1, the CRC will continue to deem the line to be faulty. For correcting multi-bit errors, SuDoku relies on a RAID-4 scheme, in which each group of 512 lines is provisioned with a dedicated parity line. For correction, we first read all the lines in the RAID-group and fix all single-bit errors. Then, the data for the faulty line is generated by computing effective parity over the parity line and the lines in the RAID-Group, except for the line being repaired. The likelihood of invoking a RAID-4 based correction is small (on average, only four lines have 2+ errors in a scrub interval).

For instance, Figure 2 shows a cache with 16 lines and each RAID-Group contains four lines. Lines A-D form a RAID-Group and the parity line for this group is the top-most line in the PLT. Line B encounters a six-bit error, which is detected by the CRC. Even after undergoing an ECC-1-based correction, the CRC still indicates error. As a result, we reconstruct the data for B by computing the parity of lines A, C, D, and the parity line. If a line encounters any single-bit error, then such an error is corrected before participating in the RAID based correction. Thus, SuDoku-X can repair the multi-bit errors in Line B, without requiring any storage or circuitry for multi-bit ECC.

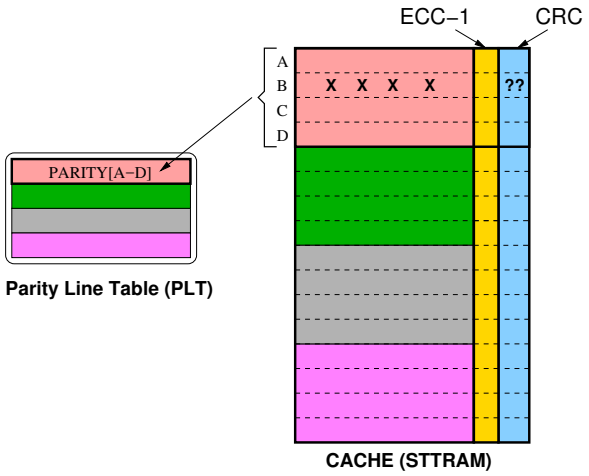

Figure 2. Correction of multi-bit errors with SuDoku-X. Line B encounters a multi-bit error, which is detected by CRC. The data for B is repaired by performing an XOR of the data for lines A,C,D and the respective parity line from the PLT.

D. Considerations on Size of RAID-Group

We use a default size of 512 lines per RAID-Group. The size of the RAID-Group determines the storage overhead for the parity lines, the latency for performing error correction, and the overall reliability. With a RAID-Group of 512 lines, the RAID-4 based correction would incur a storage overhead of 128KB for a cache of 64MB (2K RAID-Groups of 512 lines each). This storage overhead is sufficiently small to be stored in SRAM. Furthermore, the latency overhead of repairing using RAID-4 (512 line reads) is incurred infrequently – on average four lines in a scrub interval of 20ms. This repair latency (of approximately 4μs per repair) is 16μs per 20ms. Even if we encounter all of the repairs on demand read, the overall latency impact is less than 0.08%.

E. Organizing Data, CRC, and ECC within a line

To enable high fault detection and correction rates, we compute the CRC over the data and then compute the ECC over CRC and data. This enables the ECC to correct single-bit faults in both the CRC and the data. Furthermore, it also enables use to detect miscorrections from ECC in lines with 1+ faults. This is because, in such lines, the CRC recomputed after a miscorrection from ECC will not match.

F. SDC Rate of SuDoku-X

The Silent Data Corruption (SDC) of SuDoku-X is dictated by the error detection capability CRC-31, which detects up to seven bit errors [29]. For 8+ bit errors, CRC-31 has a small misdetection probability of 2^{-31}. Unfortunately, with SuDoku-X, a line with 7-bit error can get miscorrected an 8-bit error by the ECC-1, and subsequently the CRC-31 can let this event go undetected. In fact, this is the dominant source of SDC in SuDoku-X. Table III lists the SDC Rate for cases when the line has either 7 or 8+ errors. Note that, if the line has six or fewer errors, SuDoku-X will not result in SDC, as CRC-31 is guaranteed to detect them. Fortunately, the total SDC FIT-Rate of SuDoku-X is 8.9×10^{-9} which is much lower than our target of 1 FIT. However, there is an uncorrectable line every 3.71 seconds, which dictates the overall MTTF of SuDoku-X to be 3.71 seconds.

Table III
SDC RATES OF CACHE WITH SUDOKU-X

Vulnerability	7 Faults/Line	8+ Faults/Line
Event (per Billion Hours)	191	0.09
CRC-31: Misdetection Probability	2^{-31}	2^{-31}
SDC Rate (per Billion Hours)	**8.9×10^{-9}**	**4.2×10^{-11}**

IV. SUDOKU-Y: DATA RESURRECTION

The dominant failure mode of SuDoku-X is when two lines in the same RAID-Group have two errors each. Out of all failures from SuDoku-X, the case of exactly two faulty lines (with multi-bit errors) accounts for 99.98% of the cases. Furthermore, even in the case of multi-bit failures, the case of two bit failures dominates (99.98%). We use the insight that ECC-1 corrects two-bit failures (without extra storage), if we can identify their position and flip the faulty-bits. This technique is called *Sequential Data Resurrection (SDR)*, and SuDoku-X equipped with SDR is called as *SuDoku-Y*.

A. Overview of SDR

In general, if two disks fail, then RAID-4 cannot recover faulty data. We use the insight that unlike disk failures, our design is handling bit faults, and faulty lines tend to have an large number of fault-free bits (for example, in the typical case of two-bit fault, 510 bits are still fault-free). For a line with two-bit faults, if we can identify one faulty bit position,

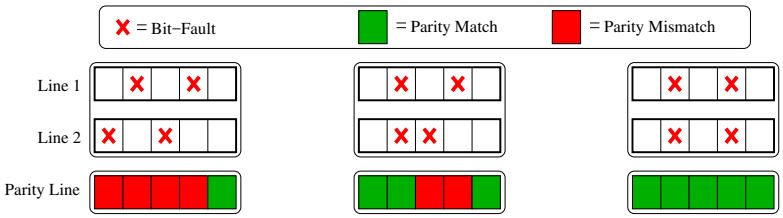

Figure 3. Three scenarios for Selective Data Resurrection using SuDoku. In general, ECC-1 cannot repair lines with two errors. However, if we know the position of one of the errors (from the Parity Line) we can correct the other error using ECC-1 by flipping one of the faulty bit (the CRC of line can validate if the correction was indeed successful).

then we can perform two-bit error correction by flipping that faulty bit and using per-line ECC-1 to fix the other fault. This correction can be checked and validated with the per-line CRC-31. In case of SuDoku-X, when multiple lines with multi-bit failures exist, the parity of the RAID-Group is used to identify the location of faulty bits as they can lead to parity mismatches (in the common case). We then perform error correction by flipping each of the bits in the mismatched positions one by one, applying the per-line ECC-1 to fix the other faulty bit, and then rechecking with CRC. If the CRC does not match, we try with the next mismatched bit position until all the positions are exhausted. Note that if we correct even N-1 faulty lines out of the N faulty lines of a RAID-Group using SDR, we correct the final uncorrectable line using the RAID-4 based correction. We analyze the effectiveness of SDR for the case of two faulty lines in the group, with two faults each.

B. Operation of SDR for Two Faulty Lines

If a region has two faulty lines, each with two faulty bits, only a maximum of four locations will show a mismatch in the parity line. The parity is computed by correcting all lines with 1-bit error in the group, and by using the original (uncorrected data values) for both the faulty lines with two-bit errors. Figure 3 illustrates a scenario in which two lines (Line 1 and Line 2) in the same RAID-Group encounter two faults. For simplicity, lets assume none of the other lines in the group encounter any faults. We explain the operation of SDR for three possible scenarios.

Case 1: No overlapping faults (99.22% probability) Figure 3(a) shows a scenario with no overlapping faults and the parity line indicates four mismatch locations that can be faulty in each line. SDR then fetches Line-1 and sequentially flips only the bits in the mismatched locations in the parity line and invoke ECC-1. If the flipped bit was indeed faulty, then ECC-1 corrects the remaining faulty bit and the CRC will indicate that the cache line is non-faulty. Thereafter, Line-2 can be corrected by using RAID-4.

Case 2: One overlapping fault (0.78% probability) Figure 3(b) shows the scenario in which one overlapping fault occurs and the parity line will have only two mismatches. SDR fetches Line-1 and sequentially tries to flip only the bits in the mismatched locations in the parity line and invoke ECC-1. If the flipped bit was indeed faulty, then ECC-1 can correct the remaining faulty-bit and CRC will indicate that the cache line is non-faulty. Even if the location of one faulty-bit was unknown, we could still correct both the faulty bits. Thereafter, Line-2 can be corrected using RAID-4.

Case 3: Both faults overlap (0.0004% probability) Figure 3(c) shows the scenario in which both the faults overlap. Thus, the parity line will not have any mismatches and SDR cannot be applied. The likelihood that two faulty bits of one line (512 bits) will overlap exactly with the two faulty bits of another line is quite low ($\frac{2}{512} \cdot \frac{1}{511} = 0.0004\%$).

The latency of SDR-based correction is only a few cycles of trial and error on the mismatched positions (4-6), so it is in the regime of few tens of nanosecond. However, this latency is incurred once every 3.71 seconds (the MTTF of SuDoku-X), so the overall latency impact remains negligible.

C. Effectiveness of SDR in Other Cases

SDR is also effective in the case of two faulty lines, each with two faulty bits, as it can repair them 99.9996% of the time. While SDR may seem ineffective if one of the lines have 3 or more faulty bits, it can fix these cases too. This is because, in the common case when there are two faulty lines in a group and one line has a 3 bit fault, the other line tends to have a 2 bit fault, as shown in Figure 4 (if two faulty bits overlap then SDR cannot repair). If we can repair Line 1 using SDR, then we can repair Line 2 using RAID-4.

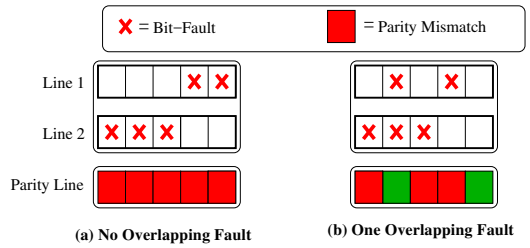

Figure 4. SDR can repair a line with 3-bit fault if it does not have ¿1 bit of overlap with a line with 2-bit fault.

We have only analyzed cases in which the RAID-Group has only two faulty lines. But SDR is useful in other cases too. For example, if there are three faulty lines with two-bit failures each (the most common case of three lines with multi-bit failure), we can repair each of the faulty lines using SDR. Just that in this scenario, we would have six possible position of mismatch and each line will sequentially undergo repair through these six possible locations. The effectiveness of SDR even in this case is 99.9%. We do not perform SDR if there are more than six mismatches.

D. SDC Rate of SuDoku-Y

SuDoku-Y based correction is invoked only when SuDoku-X encounters several lines with multi-bit failures. Once invoked, it is extremely unlikely for SuDoku-Y to encounter SDC, as it would require that the miscorrected lines go undetected by the CRC-31 and also go undetected in the Parity Line of the RAID-Group (about 10^{-28} probability). The dominant scenario for SuDoku-Y to cause SDC is identical to that of SuDoku-X (one line in the group has 7+ errors and the CRC-31 is unable to detect). As per Table III, the total SDC rate of SuDoku-Y is also 8.9×10^{-9}, which is about eight orders of magnitude lower than our target goal of one FIT. Thus, SDC rate of SuDoku-Y is not a concern.

E. Limitations of SuDoku-Y: DUE Rate

SuDoku-Y faces Detectable Uncorrectable Error (DUE) when an SDR fails to correct in two scenarios. First, when there are multiple faulty lines and at least two of them have three or more errors. Second, when two faulty bits overlap. As SuDoku-Y fixes most multi-line failures, it has an MTTF of 3.49 hours (3387X higher than SuDoku-X) and provides a DUE FIT of 286 Million. The next section describes a scheme that reduces the FIT-Rate to 0.0001.

V. SuDoku-Z: Skewed-Hash For RAID

SuDoku-Y fails when a RAID-Group contains multiple faulty lines each with 3+ bit error. As these lines have three or more errors, identifying one faulty bit position will not enable SDR to repair the line using ECC-1. We use the concepts of skewed-hashing [30] and multi-hash Bloom Filters [31] to enhance the effectiveness of SuDoku. Rather than restricting a line to exactly one RAID-Group, we use two hashes (Hash-1 and Hash-2) and let each line participate in two different RAID-Groups. If the faulty lines for a given RAID-Group is deemed uncorrectable under Hash-1 (i.e. SuDoku-Y fails), then this design tries to repair each of the uncorrectable lines using the RAID-Groups formed under Hash-2. This design of SuDoku with skewed hashing is called *SuDoku-Z*.[4]

A. Organization

Figure 5 shows the organization of SuDoku-Z. SuDoku-Z contains two Parity Line Tables (PLT-Hash1 and PLT-Hash2). The lines are mapped to the two PLT using two Hashes, Hash-1 and Hash-2. PLT-Hash1 stores the parity lines of the RAID-Groups formed under Hash-1 (identical to SuDoku-Y). Similarly, PLT-Hash2 stores the parity lines of the RAID-Groups formed under Hash-2 (newly added for SuDoku-Z). The hashes are selected such that the lines are guaranteed to be mapped to different RAID-Groups. This avoids the same set of lines from making a RAID-Group fail under both Hash-1 and Hash-2. For example, if the size of the RAID-Group is 512 lines, we can form Hash-1 by masking out the 9 least significant bits (CacheLineAddr[8:0]) of the cache line address, and Hash-2 by masking out the next nine least significant bits (CacheLineAddr[17:9]). Each write into the STTRAM cache must also write to both PLT-1 and PLT-2 to maintain updated parity.

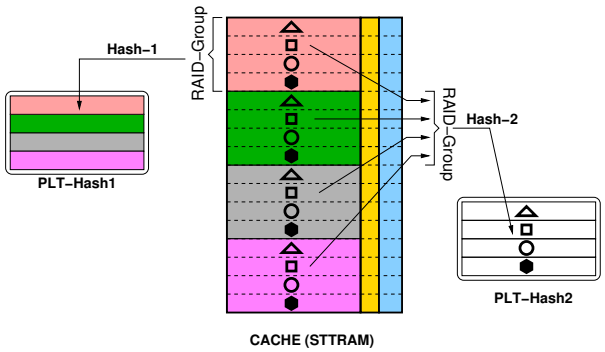

Figure 5. Organization of SuDoku-Z using two Hash functions: Hash-1 and Hash-2. A newly added structure (PLT-Hash2) stores parity lines of RAID-Groups formed under Hash-2. SuDoku-Z performs correction with Hash-2 only if correction fails under Hash-1.

Figure 5 shows an example of a cache with 16 lines implementing SuDoku-Z. The size of the RAID-Group is four lines. Under Hash-1, the consecutive four lines (same color) form a RAID-Group, and their parity is stored in PLT-Hash1. Under Hash-2, every fourth line (same symbol) form a RAID-Group, and their parity is stored in PLT-Hash2. Note that if lines shares a RAID-Group under Hash-1, then they do not share a RAID-Group under Hash-2. This helps with correction – if faulty lines are unrepairable under Hash-1, then they are guaranteed to map to different RAID-Groups under Hash-2, and undergo correction by using SuDoku-Y on those RAID-Groups. Skewed-hashing is highly effective because the likelihood of a faulty line mapping into two different uncorrectable RAID-Groups is extremely small.

B. Repairing Faulty Lines

The correction of SuDoku-Z is invoked only if the SuDoku-Y-based correction fails using Hash-1. Note that this is a relatively infrequent event, and occurs once every 3.9 hours on average. When this occurs, we first identify the set of lines that are unrepairable under Hash-1. For each such line, we try to repair using the RAID-Group under Hash-2. For a line to be deemed uncorrectable under SuDoku-Z, it will have to be unrepairable under both Hash-1 and Hash-2. In fact, if there are N unrepairable lines in a Hash-1 RAID-Group, and we are able to repair N-1 lines under Hash-2, then the remaining uncorrectable line can be repaired using Hash-1 and the corrected values of the remaining faulty lines. As the RAID-Group will have only one faulty line, the RAID-4 based correction can correct the line. Thus, for SuDoku-Z to fail, we must have two lines that are uncorrectable under both Hash-1 and Hash-2 – an extremely unlikely scenario.

[4]Although we implement SuDoku-Z with SuDoku-Y, we can implement SuDoku-Z alone too. Such a design will not be effective because of the high DUE rate, causing a FIT rate of 4 Million.

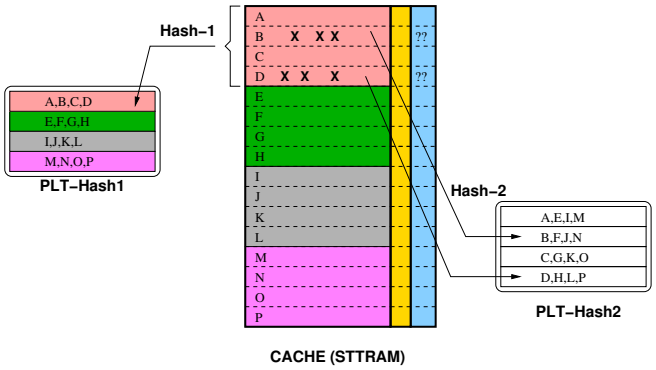

Figure 6. Correction with SuDoku-Z. Lines B and D have and uncorrectable error under Hash-1. Under Hash-2, they map to different RAID-Group and can be corrected.

For example, Figure 6 shows a cache with 16 lines (A-P) which uses two hash functions, Hash-1 and Hash-2. Two faulty lines (B and D) with three faulty bits reside in the same RAID-Group under Hash-1 and correction under Hash-1 fails. By design, lines B and D map to different RAID-Groups under Hash-2. Line B can perform correction under the Hash-2 RAID-Group that is formed with lines B,F,J and N. If SuDoku-Y can correct this RAID-Group, then line B is repaired. Similarly, line D can perform correction under the Hash-2 RAID-Group that is formed with lines D,H,L and P. If SuDoku-Y can correct this RAID-Group, then line D is repaired. In fact, even if one of the lines is repaired (say only Line D is repaired), then we can use the corrected value of that line to repair the other line (using corrected Line D under Hash-1 to repair Line B). SuDoku-Z fails only if both lines are uncorrectable under both Hash-1 and Hash-2.

The analysis can be extended to the cases when there are more than two uncorrectable lines in a RAID-Group. For example, consider there are N faulty lines in a RAID-Group formed under Hash-1. Then, we will try correction for all N lines under Hash-2. As long as at least (N-1) lines can be corrected using Hash-2, we will be able to repair all N lines.

C. SDC Rate and Effectiveness of SuDoku-Z

SuDoku-Z is invoked only when SuDoku-Y encounters an uncorrectable error. The likelihood that this correction will yield an undetected error is negligibly small (miscorrected lines go undetected by CRC-31 and Parity Lines match under both Hash-1 and Hash-2). The dominant cause of SDC for SuDoku-Z is identical to that of SuDoku-X (one line has 7+ errors and the CRC-31 is unable to detect). From Table III, the SDC Fit-Rate of SuDoku-Z is also 8.9×10^{-9}, nearly eight orders of magnitude lower than our target of one FIT.

SuDoku-Z encounters a DUE when the faulty line cannot be corrected using both Hash-1 and Hash-2. Given the likelihood of a group failing is quite small (nearly 6.9×10^{-10}), the likelihood that a line fails under both Hash-1 and Hash-2 is extremely small, and for the system to fail, we will need two of such lines. The DUE FIT-Rate of SuDoku-Z is 10^{-4}.

As the SDC FIT-Rate of SuDoku-Z is 11200x lower than the DUE FIT-Rate, the total FIT-Rate of SuDoku-Z is determined by its DUE Rate. Thus, the total FIT-Rate of SuDoku-Z is 0.0001. As shown in Figure 7, the MTTF of SuDoku-Z is 874x as high as that of ECC-6. Note that SuDoku-Z provides this level of resilience without requiring the storage and latency overheads of ECC-6.

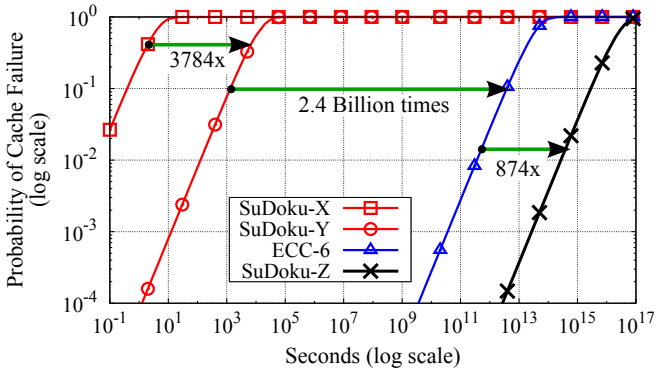

Figure 7. Cache failure probability (DUE+SDC) with SuDoku-X, SuDoku-Y, SuDoku-Z, and ECC-6. SuDoku-Z has 874x lower MTTF than ECC-6.

VI. SuDoku For Other Technologies

The goal of our paper was to develop an effective scheme to tolerate a high rate of transient failures, without incurring the substantial storage overhead and latency of strong ECC. While our evaluations were done for STTRAM, there is nothing in SuDoku that is designed specifically for STTRAM. SuDoku is a general technique that can tolerate high-rate of transient faults in any technology.

For instance, it is desirable to operate SRAM arrays at low voltage to reduce system power. Some SRAM cells do not operate reliably below a certain operating voltage V_{min} and fail in a persistent manner. Mitigating SRAM failures at low voltage has been a subject of active research [12], [21], [22]. Unfortunately, prior studies either require strong ECC per line (ECC-8 for operating below 500mV) or require that explicit testing is done at runtime or boot-time to identify lines with multibit failures [32]. Such runtime testing is complex and increases the downtime of the system. If SuDoku is used, we can get the cost-effectiveness of ECC-1 per line and avoid the runtime testing altogether. Table IV shows how SuDoku (ECC-1 + CRC-31) can enable reliable operation at lower V_{min} compared to uniform ECC-based scheme, and while avoiding the runtime testing of prior schemes [12], [21], [22].

Table IV
Probability of SRAM Cache Failure

ECC Scheme	Prob. of Cache Failure (BER = 10^{-3}) V_{min} < 500mV [21]
ECC-7	0.11
ECC-8	0.0066
ECC-9	3.5×10^{-4}
SuDoku	3.8×10^{-10}

In similar spirit, error rates in the ultra-low power memory SRAM devices (such as near threshold devices) are orders of magnitude higher compared to standard SRAM [33]. Even in technologies like DRAM, Variable Retention Time (VRT)

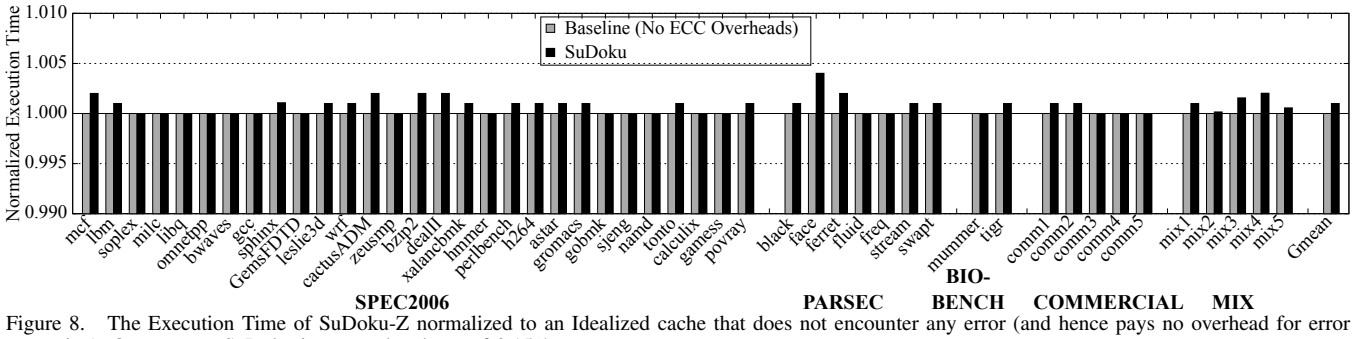

Figure 8. The Execution Time of SuDoku-Z normalized to an Idealized cache that does not encounter any error (and hence pays no overhead for error correction). On average, SuDoku incurs a slowdown of 0.15%.

can cause DRAM cells to cause retention failures in an unpredictable fashion. DRAM, Flash, and emerging Non-Volatile Memories are susceptible to disturb errors, as shown in Table V. SuDoku can tolerate all these faults, regardless of whether they are permanent or transient in nature.

Table V
EXTENDING SUDOKU TO OTHER TECHNOLOGIES

Concern	Technology
Operational Failures	SRAM, DRAM [34]
Low-Voltage Failures	SRAM [12], [21], [22], [32], [35], DRAM [36]
Retention Failures	SRAM [37], DRAM [38]–[40]
Disturb Failures	PCM [41], Flash [42]

VII. RESULTS AND ANALYSIS

We describe performance, power, and sensitivity analysis.

A. Methodology

Performance Evaluation: The performance impact of SuDoku comes from two aspects. First, the increased delay incurred due to CRC decoding (one cycle). Second, the latency incurred in performing error correction for multi-bit errors. As corrections are performed infrequently, the impact on performance is negligibly small. To assess the performance impact of SuDoku, we integrate an STTRAM cache module (a clone of CMP$im [43]) into USIMM [44]. Table VI lists the key parameters for the Baseline System.

Table VI
BASELINE SYSTEM CONFIGURATION

Number of cores (OoO)	8 cores, 3.2GHz/core
Processor ROB size	160
Fetch and retire width	4
STTRAM Last Level Cache	64MB (Shared)
No. of ways and linesize	8-Way, 64B lines
STTRAM Latencies	Read: 9ns, Write: 18ns
DDR3 Memory (800Mhz)	2 Channels, 8GB Each

We choose all benchmarks in the SPEC2006 suite [45] and PARSEC [46] , BioBench (BIO) [47] and commercial (COMM) benchmarks from the MSC suite [48]. For SPEC2006, we generate a representative slice of one billion instructions using Pinpoints(i.e., using Pintools [49] and Simpoints). In the MSC Suite, we directly use the traced workloads present in the suite. We also form four MIXED workloads by randomly selecting benchmarks. We perform timing simulation until all the benchmarks in the workload finish execution, and measure the average execution time.

Reliability and Energy Evaluations: We use analytical models to perform reliability evaluations. To this end, we derive the BER (p_{cell}) from Equation 1. We then derive the failure probabilities for a cache over a billion hours (FIT-Rate) by using basic binomial probability distribution. Alternately, one could also use monte-carlo techniques [50]–[52].

Table VII
CHARACTERISTICS OF STTRAM AND SRAM [53]

Characteristic	STTRAM	SRAM
Write energy per access (nJ)	0.35	0.11
Read energy per access (nJ)	0.13	0.05
Static power per cell (nW)	0.07	4.02

For our energy evaluations, we use the parameters shown in Table VII. As per [54], nearly 40 pJ is spent to encode and decode a line using ECC-6 and we conservatively assume CRC-31 + ECC-1 will also consume same energy.

B. Analysis of Correction Latency

Lines with 1 error can be corrected with the per-line ECC-1 at low latency. However, for lines with multi-bit error, a RAID based correction is invoked. For our fault rate, the system encounters a line with multi-bit error, on average, four times every 20ms. Such lines would invoke SuDoku-X and require reading all the 512 lines in the RAID-Group, incurring a latency of at-most 16μs (9ns per line). Fortunately, incurring 16μs overhead for correction once every 20ms would cause a degradation of less than 0.01%. The correction latency of SuDoku-Y (20μs) and SuDoku-Z (80μs) is longer, however, these are incurred every 3.71 seconds and 3.9 hours, respectively, so the performance impact from such corrections remains negligible (<0.001%).

C. Impact on Performance

Figure 8 shows the execution time for SuDoku-Z as compared to an the idealized cache that does not encounter any error. Since SuDoku-X requires a single cycle to check the ECC-1 and CRC-31 syndrome for every request, additional latency overhead is small. The overall performance impact of this syndrome check latency is negligibly small, 0.1% on an average. Furthermore, as the common-case fault is a single-bit fault and the high-latency of RAID-based error correction is incurred infrequently (16μs overhead once every 20ms). Therefore, the overall performance overhead due to SuDoku tends to be small.

D. Impact on Energy and Power

SuDoku-Z consumes additional energy as due to the parity updates in the PLT on each write to the cache. We compute the overall system energy and the Energy Delay Product (EDP) of SuDoku-Z with an idealized baseline that does not encounter any error. Figure 9 shows the System-EDP for SuDoku-Z normalized to the idealized baseline without any error correction. On average, the updates of SuDoku-Z cause an overall System-EDP to increase by at most 0.4%.

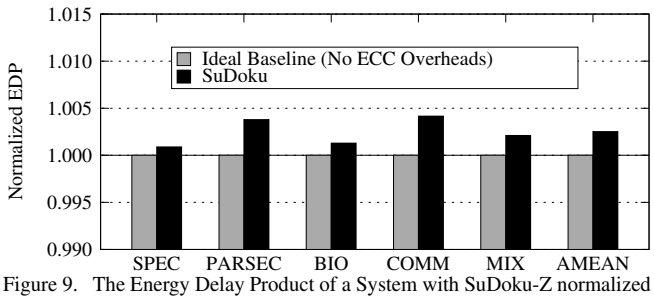

Figure 9. The Energy Delay Product of a System with SuDoku-Z normalized to an error-free baseline. SuDoku requires negligible energy to update PLTs.

E. Impact of Scrub Interval

We used a scrub interval of 20ms, which is in line with the scrub period for a 64MB STTRAM cache to keep the cache bandwidth overheads to within a few percent [5], [8]. Reducing the scrub interval reduces the BER (almost linearly), however it increases the time the cache is busy doing scrub operations. Table VIII shows the impact of varying the scrub interval from 10ms to 40ms on the FIT-Rate of ECC-5, ECC-6 and SuDoku-Z. Note that ECC-5 is insufficient at providing FIT of one even at 10ms scrub interval, whereas SuDoku-Z can provide one FIT even at 40ms.

Table VIII
FIT-RATE VS. SCRUB INTERVALS (DEFAULT: 20MS)

Scrub Interval	BER per scrub	FIT-Rate ECC-5	FIT-Rate ECC-6	FIT-Rate SuDoku-Z
10ms	2.7×10^{-6}	6.74	1.66×10^{-3}	5.49×10^{-7}
20ms	5.3×10^{-6}	215	0.092	1.05×10^{-4}
40ms	1.09×10^{-5}	6870	6.76	0.04

F. Sensitivity to Cache Sizes

We also evaluate the sensitivity of SuDoku for different cache sizes. Table IX shows the FIT-Rate for SuDoku with a 32MB cache, 64MB cache, and a 128MB cache. As a 32MB cache has 0.5x the number of lines as a 64MB cache, with all other factors remaining the same, its FIT-Rate is 0.5x to that of a 64MB cache. Similarly, the FIT-Rate of a 128MB cache is 2x to that of a 64MB cache.

Table IX
SENSITIVITY TO CACHE SIZE

Cache Size	FIT-Rate
32MB	0.52×10^{-4}
64MB	1.05×10^{-4}
128MB	2.1×10^{-4}

G. Impact of Thermal Stability (Δ)

SuDoku is also robust at lower Δ values. Lowering Δ increases the BER of STTRAM. Fortunately, SuDoku consistently offers atleast two-orders of magnitude higher reliability as compared to ECC-6. Table X highlights the strength of SuDoku when compared to ECC-6. SuDoku can be enhanced even further by replacing ECC-1 with ECC-2.

Table X
IMPACT OF Δ: ECC-6 VS SUDOKU

Thermal Stability Δ	ECC-6 FIT-Rate	SuDoku FIT-Rate	Strength of SuDoku
35	0.092	1.05×10^{-4}	**874x**
34	4.63	1.15×10^{-2}	**402x**
33	1240	8	**155x**

H. Storage Overheads of SuDoku-Z

SuDoku-Z requires 10 bits of ECC-1 and 31 bits of CRC-31 for every 512-bit cachelines. Furthermore, it uses two PLTs, each 128KB for the 64MB cache. Therefore, the amortized cost of these two PLTs is 2 bits per cache line. Therefore, SuDoku requires a total storage overheard of 43 bits per cache line, which is much less than the 60 bits per line incurred for ECC-6. Furthermore, the PLT structures are sufficiently small that they can be kept in a small 256KB SRAM structure beside the 64MB STTRAM cache.

I. Analyzing Write Traffic To the PLT

The PLT is 512x smaller than the STTRAM cache and incurs the same write intensity as the STTRAM. Therefore, to reduce bandwidth overheads to the PLT, Sudoku assumes a PLT design that uses the same number of banks as the STTRAM cache. Each bank in the PLT and the STTRAM uses a dedicated request queue. As the latency of SRAM-based PLT tends to be much smaller than the STTRAM cache, the PLT does not cause any bandwidth bottlenecks or performance overheads.

VIII. RELATED WORK

We discuss some prior work on memory reliability.

A. Parity-Based Schemes for Cache Failures

Correctable Parity Protected Cache CPPC) [17] uses a parity-based detection of a single-bit error on a per-line (or per-word basis), and tracks a global parity separately. When the line parity detects an error, the global parity is used to restore the data of the faulty line. However, CPPC was designed for a fairly low BER (a per-cell MTTF of 1 million hours), whereas we target a much higher BER (a per-cell MTTF of a few hours). CPPC fails when the line (or word) encounters multi-bit errors or when there are multiple faulty lines in the cache. One can also use RAID-6-based correction to fix two faulty lines in a group by using two additional parities. The two parities are diagonal parity and row-wise parity. However, SuDoku-Z is much stronger than RAID-6 as RAID-6 does not employ SDR.

Two-Dimensional Error Coding (2DP) [18] is parity-based scheme that keeps both horizontal and vertical parities to correct single bit errors. This scheme is low-cost and is highly effective at low-error rates. However, as the horizontal and vertical parities are formed using the same set of lines, two lines with overlapping 2+ bit errors can cause uncorrectable errors. Even if we use an optimized 2DP which uses a per-line ECC-1, it still cannot tolerate two lines with 3+ bit failures in which any bit-failure overlaps. Therefore, for the error rates we target, 2DP has a low MTTF of a few hours.

To compare CPPC, RAID-6, and 2DP with SuDoku, we provision these schemes with the same resources as SuDoku. We also assume each line is provisioned with CRC-31 detection code. Table XI compares the strength of CPPC, RAID-6, 2DP (512 line group), and SuDoku. SuDoku is at least 10^6 times as strong as these schemes.

Table XI
COMPARING CPPC, RAID-6, 2DP WITH SUDOKU

	CPPC + CRC-31	**RAID-6 + CRC-31**	**2DP ECC-1 + CRC-31**	**SuDoku**
FIT-Rate	1.69×10^{14}	571×10^{3}	2.8×10^{8}	1.05×10^{-4}

Error correction using parity groups is also employed in LPDC codes. In contrast to block codes like ECC codes that have error-correcting guarantees (such as six-error correction), LDPC codes lack any error-correcting guarantees. Contrary to LPDC codes, SuDoku provides error correction guarantees in the same spirit as the ECC codes.

B. Mitigating Failures in STTRAM

Prior studies [6] have looked at using multi-bit ECC to mitigate errors in STTRAM. Smullen et al. [10] proposed a refresh policy that reads every line of the cache and writes it back again within the retention time. Unfortunately, for the error rate we target, having only ECC-1 with each line is insufficient. Naeimi et al. [5] suggested using 5EC6ED for a 64MB STTRAM-based cache to guarantee a failure rate of 10 FIT for Delta of 32.

A low Δ can also increase the write error rate (WER) [55]. SuDoku does not differentiate between write errors and retention errors. Therefore, even if WER is nearly the same as retention BER (using reasonable values of write timings from [55]), SuDoku will provide similar reliability.

C. Tolerating Runtime Failures

Sazeides et. al. [56] investigated using ECC-1 codes to store metadata. We can also compress and store ECC in the space obtained from compression [57]–[59]. Revive [60] looked at software supported low-cost check-pointing. Prior work [61] has also investigated fixing cache failures using strong ECC and storing this ECC in main memory. These schemes are orthogonal to SuDoku and can be used in combination to further enhance reliability.

IVEC [62] and Synergy [63] leverages the message authentication codes (MACs) of secure memories for also improving reliability. To tolerate faults, they perform a sequential check of every single bit in the cache line by flipping it and recomputing the MAC to see if it matches with the stored MAC. Such naive searching becomes impractical for tolerating multi-bit failures.

Several studies [64]–[70] have also investigated mechanisms to mitigate large granularity failures in DRAM. However, these designs are effective for tolerating a small number of large granularity failures, whereas we are interested in tolerating a very high rate of bit failures. This has a significant impact on both the granularity at which we perform RAID as well as the effectiveness of schemes such as SDR. Therefore, these solutions are not directly applicable for tolerating high-rate of transient faults to enable scalable STTRAM.

Another technique, called Hi-ECC tries to provision ECC at a larger granularity of 1KB, instead of 64B cachelines [71]. This reduces the overheads of ECC-6 to 0.9%. Unfortunately, Hi-ECC increases the number of bits per line that the ECC needs to protect by 16x. Due to this, Hi-ECC has a higher FIT-Rate as compared to SuDoku. Table XII shows the FIT-Rate of SuDoku as compared to a Hi-ECC scheme that uses ECC-6 over a 1KB region of data.

Table XII
SUDOKU VS HI-ECC

Scheme	FIT-Rate
SuDoku	1.05×10^{-4}
Hi-ECC	1.47

D. Mitigating Permanent Faults

Enabling cache operation at high error rates has been a very active area of research in the last decade, primarily as a means of allowing caches to operate at low Vmin [12]. These proposals [12], [21], [22] rely on knowing the location of fault bits. A priori knowledge of the location of a bit that will encounter a transient fault is impractical, so such solutions are not viable. Several proposals [14], [15] have looked at mitigating a high error rate in DRAM due to scaling related faults by sparing faulty cells. Again, these solutions are effective for permanent faults, and are not applicable to transient errors.

E. Mitigating Retention Failures

Retention failure is a problem in other technologies as well. For example, RAIDR [38] was proposed to reduce the retention overheads in DRAM system. It relies on a priori knowledge of weak cells. Awasthi et al. [72] propose adaptive scrub mechanisms to reduce the scrub overheads in phase change memories. Efficient scrub schemes are orthogonal to our solution. Retention failures can be problem for SRAM too. However, SRAM cells are order of magnitude less susceptible to transient faults than STTRAM [73], and therefore a large number of bits failing within a cache line is not currently a concern for SRAM.

IX. Conclusions

This paper investigates cost-effective solutions for tolerating a high rate of transient failures. As transient failures occur at run-time, they cannot be identified with design time testing and cannot be mitigated using sparing mechanisms. We study the challenge of efficiently tolerating high-rates of transient faults, using STTRAM as an example. For STTRAM, transient failures due to random thermal noise is considered as a critical obstacle to scaling. We investigated a regime where the error rates of STTRAM were 5.3 ppm over a scrub period of 20ms. An effective means of tolerating transient failure is to do periodic scrubbing and employ per-line ECC. Unfortunately, at our target error rate, we would need to ECC-6 per line, which incurs significant storage overheads. Ideally, we would like to tolerate a high rate of transient failures while avoiding the overheads of strong error correction. To that end, this paper proposes SuDoku, a technique that optimizes for the common case of 1 bit failure and provisions each line with only ECC-1 along with a strong multi-bit error detection code (CRC-31). SuDoku provides 874x higher reliability than ECC-6, while incurring 30% lower storage overhead, and providing within 0.1% of an idealized performance of a fault-free baseline.

Acknowledgment

We thank the anonymous reviewers and our shepherd for their valuable feedback and directions. This work was supported in part by the Center for Future Architectures Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

References

[1] E. Chen *et al.*, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 1873–1878, June 2010.

[2] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (stt-mram)," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.

[3] S. A. Wolf *et al.*, "The promise of nanomagnetics and spintronics for future logic and universal memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2155–2168, Dec 2010.

[4] W. Zhao, Y. Zhang, T. Devolder, J. Klein, D. Ravelosona, C. Chappert, and P. Mazoyer, "Failure and reliability analysis of stt-mram," *Microelectronics Reliability*, vol. 52, no. 9, pp. 1848 – 1852, 2012.

[5] H. Naeimi, C. Augustine, A. Raychowdhury, S. l. Lu, and J. Tschanz, "Sttram scaling and retention failure," *Intel Technology Journal*, vol. 17, no. 1, 2013.

[6] B. Del Bel *et al.*, "Improving stt-mram density through multibit error correction," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 182:1–182:6.

[7] K. C. Chun *et al.*, "A scaling roadmap and performance evaluation of in-plane and perpendicular mtj based stt-mrams for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb 2013.

[8] A. Driskill-Smith *et al.*, "Non-volatile spin-transfer torque ram (stt-ram): An analysis of chip data, thermal stability and scalability," in *2010 IEEE International Memory Workshop*, May 2010, pp. 1–3.

[9] N. D. Rizzo *et al.*, "Thermally activated magnetization reversal in submicron magnetic tunnel junctions for magnetoresistive random access memory," *Applied Physics Letters*, vol. 80, 2002.

[10] C. W. Smullen *et al.*, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 50–61.

[11] Z. Sun *et al.*, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44, 2011, pp. 329–338.

[12] C. Wilkerson *et al.*, "Trading off cache capacity for reliability to enable low voltage operation," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 203–214.

[13] H. Duwe *et al.*, "Rescuing uncorrectable fault patterns in on-chip memories through error pattern transformation," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 634–644.

[14] 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.

[15] Y. H. Son *et al.*, "Cidra: A cache-inspired dram resilience architecture," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 502–513.

[16] W. Peterson and D. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[17] M. Manoochehri, M. Annavaram, and M. Dubois, "Cppc: correctable parity protected cache," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, pp. 223–234.

[18] J. Kim *et al.*, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proceedings of the 40th annual IEEE/ACM international symposium on microarchitecture*. IEEE Computer Society, 2007, pp. 197–209.

[19] R. Dorrance *et al.*, "Scalability and design-space analysis of a 1t-1mtj memory cell for stt-rams," *IEEE Transactions on Electron Devices*, vol. 59, no. 4, pp. 878–887, April 2012.

[20] K. C. Chun *et al.*, "A scaling roadmap and performance evaluation of in-plane and perpendicular mtj based stt-mrams for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb 2013.

[21] M. K. Qureshi and Z. Chishti, "Operating secded-based caches at ultra-low voltage with flair," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–11.

[22] Z. Chishti *et al.*, "Improving cache lifetime reliability at ultra-low voltages," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 89–99.

[23] A. R. Alameldeen *et al.*, "Energy-efficient cache design using variable-strength error-correcting codes," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 461–471.

[24] P. Wang *et al.*, "A thermal and process variation aware mtj switching model and its applications in soft error analysis," in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2012, pp. 720–727.

[25] G. A. Gibson, "Redundant disk arrays: Reliable, parallel secondary storage," 1992.

[26] A. Thomasian and J. Menon, "Raid5 performance with distributed sparing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 6, pp. 640–657, 1997.

[27] H. Noguchi *et al.*, "7.2 4mb stt-mram-based cache with memory-access-aware power optimization and write-verify-write / read-modify-write scheme," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 132–133.

[28] Y. Kim *et al.*, "Write-optimized reliable design of stt mram," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 3–8.

[29] P. Koopman. Crc polynomial zoo. [Online]. Available: https://users.ece.cmu.edu/~koopman/crc/crc31.html

[30] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 331–342.

[31] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[32] A. R. Alameldeen *et al.*, "Energy-efficient cache design using variable-strength error-correcting codes," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 461–471.

[33] A. Gebregiorgis *et al.*, "A cross-layer analysis of soft error, aging and process variation in near threshold computing," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 205–210.

[34] C. Slayman, "Soft error trends and mitigation techniques in memory devices," in *OPS Alacarte, Santa Clara*.

[35] M. Qazi, M. Sinangil, and A. Chandrakasan, "Challenges and directions for low-voltage sram," *IEEE Design Test of Computers*, vol. 28, no. 1, pp. 32–43, Jan 2011.

[36] K. K. Chang *et al.*, "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.

[37] R. Fan *et al.*, "The detection and investigation of sram data retention soft failures by voltage contrast inspection," in *2015 China Semiconductor Technology International Conference*, March 2015, pp. 1–3.

[38] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *Proceedings of the 39th International Symposium on Computer Architecture*, 2012.

[39] M. K. Qureshi *et al.*, "Avatar: A variable-retention-time (vrt) aware refresh for dram systems," in *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, June 2015, pp. 427–437.

[40] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in dram memories," *Computer Architecture Letters*, vol. 14, no. 1, pp. 9–12, Jan 2015.

[41] P. J. Nair *et al.*, "Reducing read latency of phase change memory via early read and turbo read," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 309–319.

[42] Y. Cai *et al.*, "Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery," in *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, June 2015, pp. 438–449.

[43] A. Jaleel *et al.*, "Cmpsim: A pin-based on-the-fly multi-core cache simulator," in *Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA. 2008*.

[44] N. Chatterjee *et al.*, "Usimm: the utah simulated memory module," *University of Utah, Tech. Rep*, 2012.

[45] "Spec cpu2006 benchmark suite," in *Standard Performance Evaluation Corporation*. [Online]. Available: http://www.spec.org/cpu2006/

[46] C. Bienia, "Benchmarking modern multiprocessors," in *Ph.D. Thesis, Princeton University*, 2011.

[47] K. Albayraktaroglu *et al.*, "Biobench: A benchmark suite of bioinformatics applications," vol. 0, 2005, pp. 2–9.

[48] (2012) Memory scheduling championship (msc). [Online]. Available: http://www.cs.utah.edu/~rajeev/jwac12/

[49] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *Acm sigplan notices*, vol. 40, no. 6. ACM, 2005, pp. 190–200.

[50] D. A. Roberts and P. J. Nair, "Faultsim: A fast, configurable memory-resilience simulator," in *The Memory Forum: In conjunction with ISCA-41*, Jun 2014.

[51] S. Wang *et al.*, "Memres: A fast memory system reliability simulator," in *SELSE 2015*, 2015.

[52] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Faultsim: A fast, configurable memory-reliability simulator for conventional and 3d-stacked systems," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 44:1–44:24, Dec. 2015.

[53] J. Zhan, O. Kayıran, G. H. Loh, C. R. Das, and Y. Xie, "Oscar: Orchestrating stt-ram cache traffic for heterogeneous cpu-gpu architectures."

[54] C. Chou, P. Nair, and M. K. Qureshi, "Reducing refresh power in mobile devices with morphable ecc," in *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, June 2015, pp. 355–366.

[55] S. Wang *et al.*, "Comparative evaluation of spin-transfer-torque and magnetoelectric random access memory," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 134–145, June 2016.

[56] Y. Sazeides *et al.*, "Implicit-storing and redundant-encoding-of-attribute information in error-correction-codes," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 160–171.

[57] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Cop: To compress and protect main memory," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 682–693.

[58] S. Hong *et al.*, "Attaché: Towards ideal memory compression by mitigating metadata bandwidth overheads," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 326–338.

[59] V. Young, S. Kariyappa, and M. Qureshi, "Enabling transparent memory-compression for commodity memory systems," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2019, pp. 570–581.

[60] M. Prvulovic, Z. Zhang, and J. Torrellas, "Revive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ser. ISCA '02, 2002, pp. 111–122.

[61] D. H. Yoon and M. Erez, "Flexible cache error protection using an ecc fifo," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–12.

[62] R. Huang and G. E. Suh, "Ivec: Off-chip memory integrity protection for both security and reliability," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, 2010, pp. 395–406.

[63] G. Saileshwar *et al.*, "Synergy: Rethinking secure-memory design for error-correcting memories," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 454–465.

[64] J. Kim, M. Sullivan, and M. Erez, "Bamboo ecc: Strong, safe, and flexible codes for reliable computer memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 101–112.

[65] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Citadel: Efficiently protecting stacked memory from large granularity failures," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 51–62.

[66] X. Jian, V. Sridharan, and R. Kumar, "Parity helix: Efficient protection for single-dimensional faults in multi-dimensional memory systems," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 555–567.

[67] D. W. Kim and M. Erez, "Relaxfault memory repair," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 645–657.

[68] J. Kim *et al.*, "All-inclusive ecc: Thorough end-to-end protection for reliable computer memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 622–633.

[69] S.-L. Gong *et al.*, "Clean-ecc: High reliability ecc for adaptive granularity memory system," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48, 2015, pp. 611–622.

[70] P. J. Nair, V. Sridharan, and M. K. Qureshi, "Xed: Exposing on-die error detection information for strong memory reliability," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 341–353.

[71] C. Wilkerson *et al.*, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 83–93.

[72] M. Awasthi *et al.*, "Efficient scrub mechanisms for error-prone emerging memories," in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, ser. HPCA '12, 2012, pp. 1–12.

[73] P. Dubey *et al.*, "A 500 mv to 1.0 v 128 kb sram in sub 20 nm bulk-finfet using auto-adjustable write assist," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Jan 2014, pp. 150–155.