A Reliability-Aware Replacement Policy for STT-MRAM Caches in Server-Class Processors

Abdollah Mohammadi, Elham Cheshmikhani, and Hossein Asadi, *Senior Member, IEEE*

***Abstract*—Spin-Transfer Torque Magnetic RAM (STT-MRAM) has several advantages over conventional SRAM technology in on-chip caches such as low leakage, soft error tolerance, and high density and scalability. These advantages make it the most promising non-volatile memory for SRAM replacement. However, STT-MRAM cache memory faces two main reliability challenges in emerging *nano-scale* technology nodes, i.e., *retention failure* and *read disturbance*. Because of the lower data access rate in the *Last-Level Caches* (LLCs) compared to the higher cache levels, as well as the higher contribution of read accesses, these two reliability challenges have become severe in the LLCs of server-class processors. The existing approaches to overcome these challenges impose significant area and performance overhead or adversely affect the other failure types. In this paper, we first investigate the parameters that affect the reliability of STT-MRAM-based LLCs due to the retention failure and read disturbance. Our investigation shows that a) the duration of *dead dirty blocks* is the main contributor to the retention failure rate of STT-MRAM LLCs while b) the high number of *risky* reads, i.e., those that can affect the cache reliability, in the dirty blocks is the main contributor to the read disturbance.Based on these observations, we propose a simple yet effective cache replacement policy, called *Retention failure and read disturbance reduction* (Reference), to decrease the length of dirty intervals and the number of reads, which results in a significant reduction in retention failure and read disturbance rate. Our evaluations demonstrate that the proposed replacement policy cuts down the probability of retention failure and the number of risky reads per dirty block by 56% and 61%, respectively. The area overhead of this scheme is negligible (0.2%) with no adverse effect on the energy consumption.**

***Index Terms*—Cache Memory, Reliability, Retention Failure, Read Disturbance, Replacement Policy, STT-MRAM.**

I. Introduction

By technology down-scaling, *Static Random-Access Memory* (SRAM) technology faces several challenges such as high power consumption, low density, low reliability, and low scalability [1], [2]. Recent efforts to address SRAM issues have led to the introduction of emerging *Non-Volatile Memory* (NVM) technologies such as *Spin-Transfer Torque Magnetic RAM* (STT-MRAM) [3]–[5]. The main advantages of STT-MRAM are non-volatility, high scalability and density, low leakage, and immunity to soft errors [6]. However, it faces some reliability challenges such as *retention failure*, *read disturbance*, and *write failure* [7]–[10]. When a cell is idle (no current or voltage is applied) and the cell content unintentionally changes, retention failure occurs [11], [12] and read disturbance refers to an unintentional cell content change during read access [13], [14]. The last failure type, i.e., write failure, refers to the unsuccessful switching of cell content in a write operation [15]–[18]. To employ emerging NVMs on the shelf, these failure types need to be carefully addressed.

With the down-scaling of manufacturing technology, retention failure has become the main source of STT-MRAM failures [19] as shown in Fig. 1. This failure occurs while the cell is idle. Since *Last-Level Cache* (LLC) has a low access rate and its blocks are idle for a longer time in comparison with higher cache levels [5], retention failure has become the main source of LLC failures in server-class processors. On the other hand, by increasing the number of read accesses, the probability of read disturbance grows. Since all blocks in a set are accessed in parallel to improve read performance, the increased number of read accesses leads to a higher probability of read disturbance.

Several studies attempted to reduce the retention failure and read disturbance rate in STT-MRAM caches. Utilizing refresh and scrubbing methods [19], [20], increasing thermal stability factor [21], [22], employing *Error-Correction Codes* (ECCs) [19], [23]–[26], reducing read and write currents in conjunction with a more precise sense amplifier [27]–[30], and manipulating physical dimensions of cells are the main approaches for enhancing the STT-MRAM reliability. Some of these approaches impose a high overhead on cache memory [19], [23], [24], [31]–[33], while others show an adverse impact on the other failure types [21], [22], [31].

In addition to reliability challenges, STT-MRAM caches suffer from high write energy and long latency, which are caused by asymmetric bit switching and the stochastic nature of write operation in STT-MRAM cells. To address these challenges, many previous studies conducted before are mainly categorized into reducing the thermal stability factor, using ECCs [34], [35], adaptive write pulses and write current [36]–[38], and different cache structures [39]–[41] such as multi-retention STT-MRAM caches [42]. Some of these approaches can negatively affect reliability challenges such as retention failure; however, adapting the approach presented in this paper could minimize their effects.

The probability of write failure decreases in the lower technology nodes (Fig. 1) while the rate of the other two failures increases. Under a technology node of 15nm, the contribution of write failure to the total failure rate is negligible. Therefore, an effective method to decrease the retention failure and read disturbance without imposing overheads or negative effects on the write failure is *missing* in the previous work. Given the gap in the previous work, we investigate the effective

A. Mohammadi and H. Asadi are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mails: {abd.mohammadi, asadi}@sharif.edu
E. Cheshmikhani is with the Department of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.
E-mail: e_cheshmikhani@sbu.ac.ir
Manuscript received July 22, 2024.

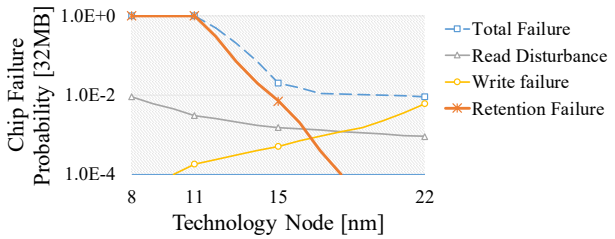

Fig. 1. Trend of failure rates for different technology nodes [19].

parameters on retention failure and read disturbance. Based on this study, mitigating the retention failure and read disturbance rates of a memory cell is based on the following major sources/parameters, which should be carefully monitored and addressed: a) cell *idle time*, b) *number of read accesses*, and c) *thermal stability factor*. As our focus is on architectural parameters, we focus on *only* the two former parameters in our proposed method, since altering the thermal stability factor requires physical changes in the manufacturing.

In this paper, we first investigate a detailed study, which divides the idle intervals of cache blocks into two categories and refers to them as *live* intervals and *dead dirty* intervals. Live intervals indicate the time between two consecutive accesses to a certain block if the current access to this block is a read access. Additionally, dead dirty intervals indicate the time between the last access and the eviction time of dirty blocks. We evaluate the impact of idle intervals on the probability of retention failure. Our experimental results show that the length of the dead dirty intervals is significantly longer than the length of live intervals. Hence, the probability of retention failure in dead dirty intervals is more significant than in live intervals. These observations indicate that decreasing dead dirty idle intervals would have a large impact on reducing retention failure rate.

To reduce read disturbance rate, we examine the number of read accesses to the cache blocks. To do so, we define *risky* and *safe* read accesses. Risky read access is one in which data would be corrupted if a read disturbance occurs during a read operation. However, the occurrence of a read disturbance during a read operation that does not lead to data loss is called safe read access. According to our observation, the number of risky read accesses in dirty blocks is much greater than in clean blocks. Thus, dirty blocks have a significant impact on the probability of read disturbance. Hence, by reducing the duration of dirty intervals (the time between a block being dirty and its eviction), the number of risky reads from dirty blocks and the probability of read disturbance will be reduced.

Based on these observations, we propose a simple yet effective cache replacement policy called *Retention Failure and read Disturbance reduction* (Reference). The proposed cache policy reduces the idle time between the last access and the eviction time of the dirty block. The *Reference* policy early evicts error-prone dead dirty blocks by detecting these blocks. We add a bit to each cache block, which allows us to evict dead dirty blocks at certain times. Hence, *Reference* reduces the length of dead dirty intervals by evicting dirty blocks that are not as old as the *Least Recently Used* (LRU) evicted blocks. As a result, *Reference* cuts down the probability of retention failure and read disturbance rate in STT-MRAM LLCs.

Our evaluation based on workloads from the SPEC CPU2006 [43] benchmark suite running on the gem5 full-system simulator [44], [45] shows that the *Reference* policy reduces the probability of retention failure of STT-MRAM caches by 56% and 69% and decreases the number of risky reads per dirty block by 61% and 66% compared to LRU and RRIP [46] (the widely-used methods for re-referenced block prediction in SRAM caches). This scheme imposes only 0.2% area overhead and no increase in energy consumption.

Briefly, we offer the following **original** contributions:

- We propose a novel classification for cache blocks of an STT-MRAM LLC based on their access patterns and show that dead dirty blocks are the main contributor to retention failure and read disturbance for the cache vulnerability.
- We investigate the read access patterns to the cache blocks and reveal that dirty blocks have a large contribution to the read disturbance rate in STT-MRAM LLCs.
- Based on our observations, we propose a novel replacement policy, which needs only a minor modification on cache management to prioritize more vulnerable blocks for eviction in the cache replacement policy. The proposed policy significantly enhances cache reliability with negligible performance and area overheads.

The organization of the paper is as follows. A review of STT-MRAM memory is provided in Section II. Previous works are investigated in Section III. Section IV introduces the motivation for this work, while the proposed schemes are introduced in Section V. Results and simulation setup are presented in Section VI. Finally, Section VII concludes the paper.

II. STT-MRAM Background

A. STT-MRAM Basic Concepts

A cell in STT-MRAM for storing data and accessing the data uses *Magnetic Tunnel Junction* (MTJ) and an NMOS transistor. An MTJ (Fig. 2) consists of three layers; a reference layer and a free layer are made up of ferromagnetic material, with fixed and changeable magnetization directions, respectively, and a layer placed between these ferromagnetic layers called the oxide barrier layer [47].

STT-MRAM uses magnetic property instead of electric charging to store data; hence, the stored data is represented by its resistance. An MTJ has two resistance states: *high*, indicating logic '1', and *low*, representing '0' [48], [49]. When the magnetization of the two layers is *parallel* (P), MTJ has low resistance (Fig. 2.a). In contrast, the *anti-parallel* (AP) magnetization leads to the high resistance of MTJ (Fig. 2.b), which represents logic '1' for the stored bit [50].

To write '1' to a cell of the STT-MRAM cache, a write current flows from the free layer to the reference layer. In this case, the electrons with different magnetic orientations are unable to pass through the reference layer due to its strong magnetic field. Instead, they are pushed back to the free layer, which leads to a change in the magnetic orientation of the free

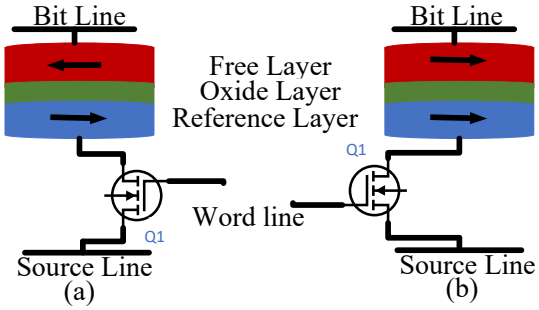

Fig. 2. STT-MRAM cell representing the value a) "1" and b) "0".

layer to the anti-parallel with the reference layer. In contrast, a write current is applied from the reference layer to the free layer to write the value '0' into a cell. To read a cell, a small current is applied and the amount of the current is sensed to distinguish the stored value [7], [12], [13].

B. Retention Failure in STT-MRAM

Retention failure is a major reliability issue in a STT-MRAM cell, as reported in Fig. 1 [19], [20]. This type of failure is occurred by unintentional change of the cell state when it is idle. Equation (1) [48] is used to calculate the retention failure probability for a cell, where t and Δ are the cell idle interval and the thermal stability factor, respectively [17], [51]. The thermal stability factor of a cell is calculated according to Equation (2) [48], where E_b, K_B, and T are the barrier energy, the Boltzmann constant, and the temperature in Kelvin [52].

$$P_{retention-failure} = 1 - \exp\left(-t \times \exp\left(-\Delta\right)\right) \quad (1)$$

$$\Delta = E_b/(K_B \times T) \quad (2)$$

Based on (1), we extract two parameters that affect the retention failure, which are the cell idle time and its thermal stability factor. As mentioned, increasing Δ for probability reduction of retention failure exacerbates the write failure rate. In addition, Δ adjustment needs to manipulate the manufacturing process, which is beyond the scope of this work.

C. Write Failure in STT-MRAM

The probability of write failure is calculated based on Equation (3) [50]. In this equation, I_{write}, c, e, m, p, μ_β, and t_{write} are write current, Euler constant, electron charge, magnetic momentum of the free layer, tunneling spin polarization, Bohr magneton, and write pulse duration, respectively [7]. The energy barrier (E_b) scales down linearly when scaling STT-MRAM cells, which causes a reduction in the thermal stability factor (Δ). As shown in Equation (3), a decrease in the thermal stability factor exponentially affects the probability of write failure.

$$P_{write-failure} = \exp(-t_{write} \times \frac{2 \times \mu_\beta \times p \times (I_{write} - I_{C_0})}{c + \log_e(\pi^2 \times \frac{\Delta}{4}) \times (e \times m \times (1 + p^2))}) \quad (3)$$

D. Read Disturbance in STT-MRAM

To write "0" or "1" in a cell of STT-MRAM cache, two opposite directions are used for current flowing. As the read and write operations share the same path in the STT-MRAM cell, applying a read current to a cell may cause flipping the cell value [52], [53]. The read disturbance probability is calculated based on Equation (4), in which τ is attempt period, t_{read} is equal to the reading interval, I_{read} and I_{C_0} are read current and critical current (write current in 0^oK), respectively [48], [50].

$$P_{read-disturbance} = 1 - \exp\left(\frac{-t_{read}}{\tau} \times \exp\left(\frac{\Delta(I_{C_0} - I_{read})}{I_{C_0}}\right)\right) \quad (4)$$

In addition to the physical parameters in (4), the frequency of read operations considerably affects the read disturbance rate. Equation (5) shows how the read requests contribute in the read disturbance rate. According to (5), the number of read accesses has a decisive role in the read disturbance occurrence.

$$P = 1 - (1 - P_{read-disturbance})^{number\_of\_read\_accesses} \quad (5)$$

In summary, the idle time of a cache block is the main player in the retention failure rate. In addition, the read disturbance rate increases with a higher number of read accesses. Thus, we focus on these parameters to enhance the reliability of STT-MRAM LLCs.

III. Related Work

As our proposed policy increases the reliability of STT-MRAM caches by predicting the dead dirty blocks, we study the related work from two aspects. We first explore the methods that addressed the retention failure and read disturbance in STT-MRAM caches and then discuss the methods that tried to predict the dead blocks in the SRAM/STT-MRAM caches.

A. Fault-tolerant STT-MRAM Cache Schemes

In [21], retention failure rate is reduced by increasing Δ, without taking the negative impact of their method on the other types of failures into account. Although retention failure rate is reduced, the higher value of Δ increases the write failure rate. In [19], *Error-Correction Codes* (ECCs) are proposed to tolerate retention failure, while using these codes imposes area and performance overhead due to encoding and decoding.

Some studies proposed DRAM-style refreshing to reduce the probability of retention failure. However, DRAM-style refreshing refers to reading blocks and writing them back [19], [20], which imposes more cost in terms of energy consumption and performance. On the other hand, scrubbing schemes have been proposed to address the DRAM-style refreshing limitation [20]. To reduce the rate of data loss due to retention failure, a refresh method over periodic intervals is proposed in [31].

In [54], the memory is classified into four sections, each with a dedicated retention time.

The sections are determined based on prediction and history tables while imposing area and performance overhead for each access. In [55] and [31], cell dimensions are reduced to improve write latency and reduce write energy. This leads to

retention time reduction and an increase in the occurrence rate of retention failure. In [22], to mitigate read disturbance, critical current and the Δ are increased. However, by increasing the thermal stability factor, the write failure rate increases. Using accurate sense amplifiers is another way to reduce the read disturbance rate, which causes high energy and area overhead [27]–[29], [56]. To mitigate errors caused by read disturbance, previous studies have employed ECCs [9], [23]. [23] uses ECCs to correct data and write it back to the cache. In [49] and [57], different transitions ($1\rightarrow0$, $0\rightarrow1$) in STT-MRAM cells were examined, and two replacement policies were proposed to overcome the high error rate. However, these approaches impose a high overhead.

Some studies investigate the effect of temperature on the STT-MRAM memories failure rate. In [58], the effect of temperature on the switching of STT-MRAM cells was examined, and they showed that fixed voltage pulses at high temperatures are more error-prone. In [24], the change in the read disturbance rate with temperature was examined. This work shows that by increasing the temperature, the read disturbance probability is increased as well. To reduce the read disturbance rate, they use strong ECCs for the high-temperature blocks. In [14], the authors examined the effect of write operations on temperature and proposed a policy to mitigate the high error rate caused by temperature in STT-MRAM caches. Their policy distributed the heat by managing consecutive write operations in distant cache blocks. Reducing read access rate by compression was proposed in [32], [33], [59]. These approaches, however, impose a significant compression cost overhead. In conclusion, some of the previous approaches impose a high overhead on cache memory, while others show an adverse impact on the other types of failures. The method presented in [50] proposes an approach to reduce the read disturbance rate in the cache tag array.

B. Dead Block Prediction in SRAM Caches

There are some previous work on dead block prediction, which are based on SRAM technology. To identify dead blocks, The *Sampling Dead Block Prediction* (SBDP) [60] policy uses locality property of cache. The SBDP utilizes program counter (PC) in a way that if an instruction triggers a dead block, other blocks linked to that instruction will also be identified as dead blocks. SBDP uses three prediction tables, which uses a group of sets of cache for training.

The Re-Reference Interval Prediction (RRIP) [46] replacement policy utilizes a 2-bit re-reference prediction value (RRPV) for each cache block. The RRPV represents the distance of a block to *Most Recently Used* (MRU) position in an LRU stack, allowing for multiple blocks to share the same RRPV. Upon accessing a block, its RRPV is reset to 0. Blocks are classified into near-immediate, inter-immediate, and distance re-reference interval based on the RRPV value in this policy. The algorithm selects a victim block with a predefined RRPV. If no victim block is present in the cache, all cache block RRPVs are incremented until a suitable victim block is identified.

The SHiP [61] policy predicts hit accesses by the signature of blocks. The SHiP uses three types of signature, i.e., memory region signature, program counter signature, and instruction sequence signature, for hit prediction.

The Leeway [62] policy uses a live distance metric to identify dead blocks. The live distance of a block refers to the maximum stack distance observed during the block's presence in the cache. The Leeway also uses set sampling and data-code correlation to predict block's live distance based on the program counter (PC) responsible for bringing the blocks into the cache. Howkeye [63] policy utilizes future decisions by learning the behavior of a block by applying Belady's optimal algorithm to previous accesses to the cache.

In summary, none of the previous policies aimed at improving reliability. Employing some of these methods can even worsen the STT-MRAM reliability. Additionally, our proposed approaches can be integrated into many of the previous studies as future work.

IV. Motivation and Observations

Here, we first examine the idle time of cache blocks and determine which type of idle time affects the probability of retention failure. We then identify the risky and safe reads and compare them in dirty and clean blocks. Next, we discuss the access pattern of evicted dirty blocks from an LRU-cache and *Re-Reference Interval Prediction* (RRIP) policy [46] as the widely-used and most efficient predictor for re-referenced blocks in SRAM caches.

A. Reliability Optimization Model

The reliability definition and formulation of STT-MRAM cache have been presented in [7] and the key parameters and variables that contribute to the reliability have been deeply investigated. Besides the physical- and device-level parameters, it has been shown that the memory access pattern and workload behavior including the number and order of read and write operations as well as live/dead intervals considerably affect the cache reliability.

By definition, the STT-MRAM cache is considered reliable when *no* read disturbance, write failure, or retention failure occurs in any of the cache blocks during workload execution. According to the cache reliability formulations, decreasing the frequency of read/write operations or shortening the duration of idle intervals for both live and dead dirty blocks can decrease the cache error rate and improve reliability. Since the contribution of write failure in the total cache error rate is downgrading compared to read disturbance and retention failure, our proposal is to reduce the rate of the latter. In summary, the proposed scheme aims to minimize the duration and the number of dead dirty blocks with the constraint of minimum performance degradation.

B. Retention Failure

Reducing the probability of retention failure by manipulating the Δ has a negative impact on other types of failure. Thus, we focus on the cell (or block) idle time as the key parameter to reduce the retention failure rate. We investigate the effect of idle time on the retention failure of cache blocks and identify four types of idle intervals as follows (Fig. 3):

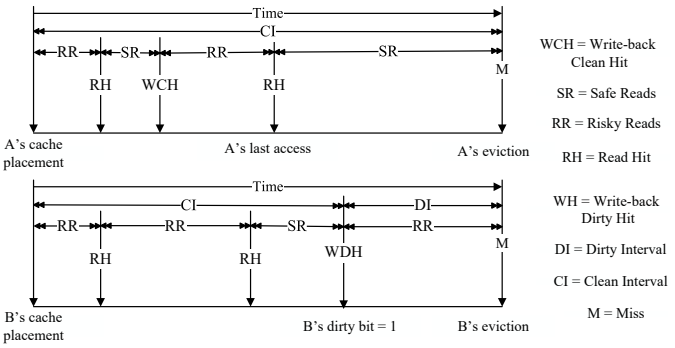

Fig. 3. Risky and safe read accesses.

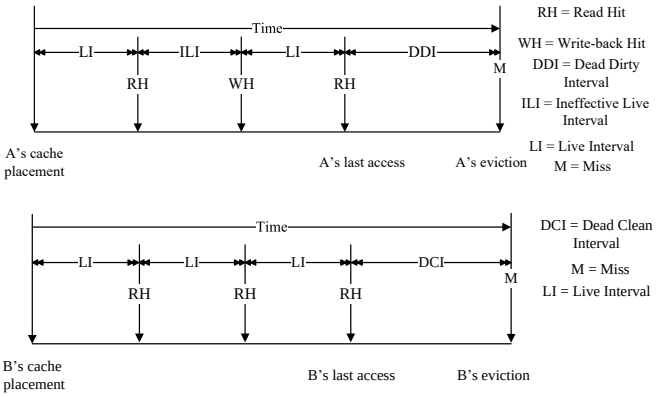

Fig. 4. Access patterns of Block A and Block B.

- **Live Interval** (LI) is the idle time between two consecutive accesses to a block while the last access is a read access. The occurrence of the retention failure in a live interval can cause *data loss*, affecting the cache reliability.
- **Ineffective Live Interval** (ILI) is the idle time between two consecutive accesses to a block while the second access is a write access. In this case, the block content is rewritten and the occurrence of the retention failure in these idle intervals may not lead to data loss. These intervals *do not affect* the cache reliability.
- **Dead Dirty Interval** (DDI) represents the idle time between the last access to the dirty blocks and their eviction time. On a dirty block eviction, the block content is written back to the next memory level. The occurrence of the retention failure in these intervals can corrupt the data block. Hence, dead dirty intervals *affect* the cache reliability.
- **Dead Clean Interval** (DCI) is the idle time between the last access and the eviction time of the clean blocks. The occurrence of the retention failure in DCIs does not lead to data loss, since there is already a copy of the cache block in the next memory level. Hence, dead clean intervals *do not affect* the cache reliability.

Next, we compare the frequency and length of the error-prone intervals, i.e., DDI and LI, as reported in Fig. 4.

The frequency of DDIs and LIs are equal to the number of dirty blocks in the cache and the read-hit accesses to the cache, respectively. Fig. 5 shows that the frequency of LIs is 1.87× greater than the number of DDIs. However, in terms of duration, the DDIs length accumulation is 2.55× greater than that of LIs, on average, as shown in Fig. 6. Therefore, the average length of each DDI is 4.77× (i.e., 1.87×2.55) of each live interval.

To compare the effect of frequency and length of intervals, we calculate the probability of retention failure in DDIs and LIs based on Eq. (6), where $P_{(RF_i)}$ is the probability of the cell retention failure in idle time with the length of i, $\texttt{blocksize}$ is the number of cells in a block, $\texttt{interv}$ is the longest idle time of the blocks, and $freq_i$ is the frequency of idle intervals with the length of i.

$$P = 1 - \left(\prod_{i=1}^{interv} (1 - P_{RF_i})^{blocksize + freq_i} \right) \quad (6)$$

The average probability of retention failure in DDIs and LIs is shown in Fig. 7. According to this figure, the probability of retention failure in DDIs for LRU and RRIP policies is 4.8× and 2.8× greater than in LIs, respectively. Therefore, the duration of intervals affects the probability of retention failure more than their frequency.

C. Read Disturbance

The main parameter which affects the probability of read disturbance is the number of read accesses to the blocks. On a read request to the cache, all blocks within the target cache set are read in parallel, which intensify the read disturbance rate in the data blocks.

As shown in Fig. 3, the read disturbance can result in data loss in case of *risky read access* while it may not result in data loss in case of *safe read access*, according to the state of the block.

Risky read access: It is a read access that can cause reliability degradation because of read disturbance. A read access to a set of cache blocks affects the probability of read disturbance of a clean block if after this access, read-hit access occurs to the same block. The occurrence of the read disturbance on this block within this interval can cause data loss, and hence reading this block affects the cache reliability. In dirty blocks, which need to be sent to the next memory level, all read accesses (miss or hit) to the set affect the probability of data loss.

Safe read access: As shown in Fig. 3, the read accesses to a set do not affect the cache reliability in two types of intervals, called safe read accesses: a) the interval between two consecutive accesses to a clean block if the last access is a write access and b) the interval between the last access to a clean block and the eviction of the block from the cache. The occurrence of a read disturbance in these intervals has *no* effect on the cache reliability because these blocks will be rewritten or evicted from the cache without sending them to the next level of memory hierarchy. Therefore, these blocks do not have any effect on the probability of read disturbance even if their data is corrupted since the content of these blocks will not be reused.

D. Dirty Blocks

For further analysis, we explore the risky read accesses of clean and dirty blocks and report the number of risky reads per

block (Fig. 8). On average, the number of risky read accesses in the dirty block of LRU and RRIP is 7× and 7.6× higher than that in clean blocks, respectively. With such motivational results and the fact that dirty blocks are more vulnerable, we further examine their access pattern. We calculate the number of accesses to the blocks before and after becoming dirty. The *Number of Accesses to a block Before being Dirty* (NABD) indicates the number of accesses to the block in the interval between the block placement in the cache and its first write access. Also, the number of accesses to a block in the time interval between its first write access and eviction is equal to the *Number of Accesses to the block After being Dirty* (NAAD). In the case of a write-back miss access, the number of accesses before dirty is zero.

By investigating the evicted dirty blocks in LRU and RRIP policies, we observe that a significant number of dirty blocks is not accessed after and before being dirty. Fig. 9 reports the number of evicted dirty blocks, the number of dirty blocks with a NAAD equal to zero, and the number of dirty blocks not accessed after and before being dirty. According to this figure, in LRU (RRIP) policy 85.6% (87.5%) of dirty blocks are not accessed after being dirty, and 99.4% (99.2%) of such blocks have an NABD equal to zero.

To conclude, the length of dead dirty intervals and the number of risky read accesses of dirty blocks are the main contributors to the probability of retention failure and read disturbance rate, respectively. Thus, dirty blocks have a significant impact on the STT-MRAM cache reliability. Therefore, we propose a scheme to reduce the length of dead dirty intervals and the read access rate to dirty blocks to overcome reliability degradation.

V. Proposed Scheme

The idle time of blocks, particularly error-prone dead dirty blocks, is a crucial factor affecting the reliability of STT-MRAM caches. Hence, any scenario that could potentially prolong idle time needs to be carefully examined. For example, in a set-associative cache, increasing the number of cache ways can extend the idle time of blocks by retaining them in the set for a longer duration. Consequently, this increase in idle time also raises the probability of retention failure and read disturbance occurrence because of the direct influence of idle time on retention failure probability as well as the escalation in the number of unnecessary reads.

In an 8-way set-associative cache with LRU replacement policy, when a dirty block is accessed for the last time, it goes into MRU position (age 0), and when its age increases to 7, it is ready to be evicted and will wait for the occurrence of a miss access. Therefore, the length of dead dirty intervals depends on the time it takes the dirty block to traverse from the MRU to the LRU position and the waiting time of the miss occurrence. **As a result, by cutting down this time interval that it takes the dirty block to go from MRU to LRU position, the duration of dead dirty intervals will be reduced**.

An ideal replacement policy to minimize the occurrence of retention failure and read disturbance is a policy that prevents unnecessary read accesses to the blocks and evicts the blocks immediately when they become dead. Thus, by reducing the interval in which the retention failure and read disturbance rate are the most, we increase the cache reliability. To achieve this goal, we propose distinctive approaches as the key piece of a comprehensive scheme.

A. Proposed Reference Policy

The dirty interval length of a block depends on the number of accesses to a set containing the block; hence, the higher the number of accesses, the longer the duration. As an example, consider a dirty block with an LRU position of 3; the more accesses to the set blocks with an LRU position of less than 3, the more time it takes for the dirty block to go to the LRU position of 7 and be ready for eviction. To reduce this interval, we use a threshold (Th_{low}) for dirty blocks. When the age of dirty blocks surpasses the threshold, the dirty block is considered as dead; i.e., these blocks have higher priority than the block in the LRU position of 7.

As mentioned earlier, 85.6% (87.5%) of evicted dirty blocks from LRU (RRIP) replacement policies are not accessed after being dirty, and 99.4% (99.2%) of these blocks are not accessed in clean intervals (CI). To identify the blocks that are not accessed in a clean interval, we add an *Access Before Dirty* (ABD) bit to each block. The ABD bit is set to '1' if the block is accessed in the clean interval (as specified in Fig. 3), otherwise, it remains '0'. Accordingly, the value of the ABD bit of a block is determined in the write-back access to the block.

In this approach, we use three scenarios to evict the dead dirty blocks: a) in the case of access to a set, if the set includes a dirty block with ABD bit equal to zero and if this block is not the requested block, the dirty block will be evicted. As the ABD bit of a block is specified in write-back access, the block (if ABD = 0) is evicted at the first access (hit or miss)

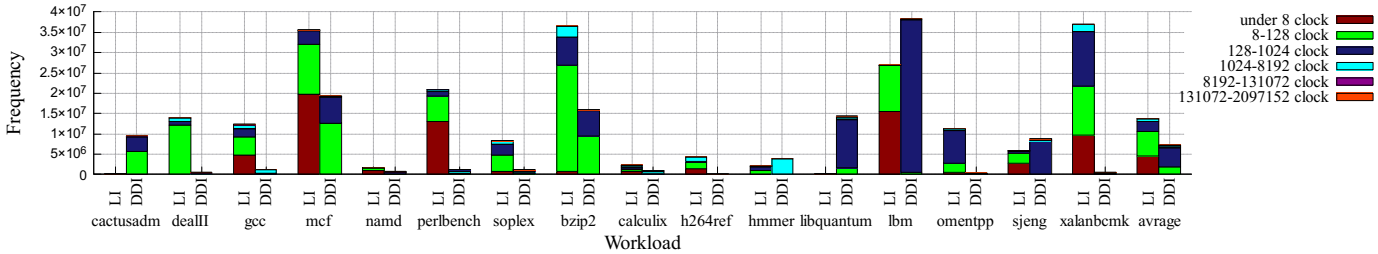

Fig. 5. Frequency of error-prone intervals (live and dead dirty intervals).

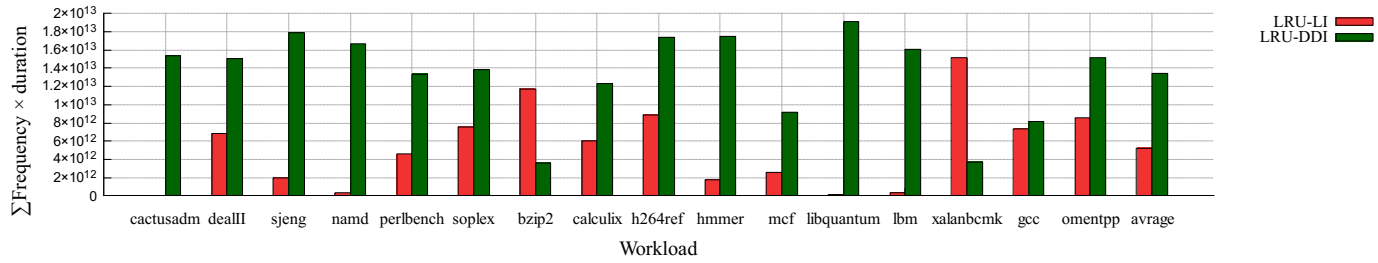

Fig. 6. Accumulation of duration of live and dead dirty intervals in LRU policy

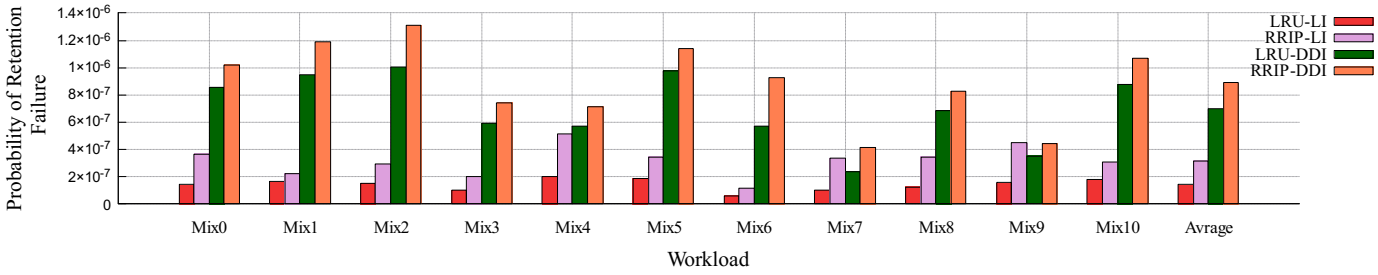

Fig. 7. Probability of retention failure in live and dead intervals.

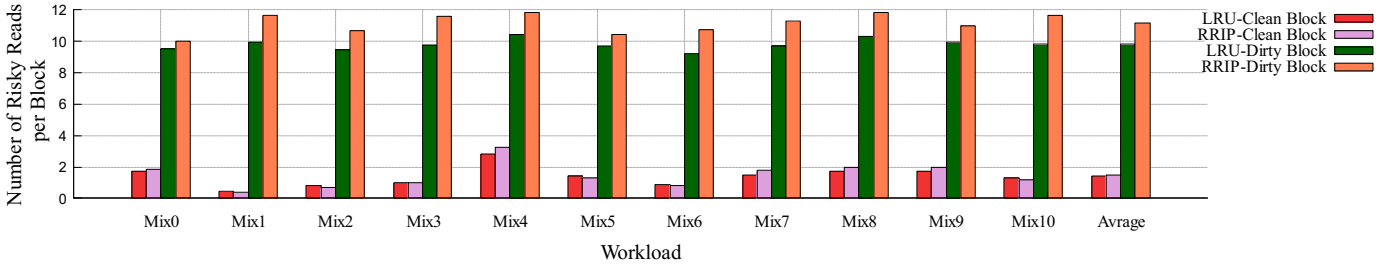

Fig. 8. Number of risky reads per dirty and clean blocks in the LRU and RRIP policies.

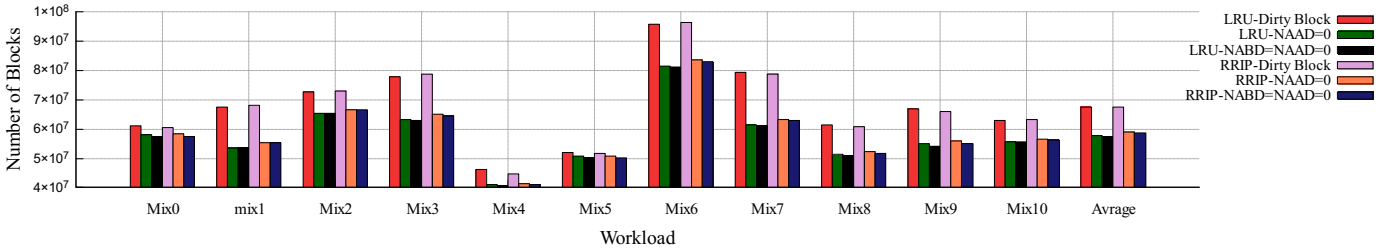

Fig. 9. Behavior of evicted dirty blocks.

after the write-back access to the set; b) in the case of a miss access, the dead dirty blocks have priority for eviction. This means if there is a dead dirty block in the set, this block will be evicted, otherwise, this approach evicts the blocks based on the LRU; and c) by predicting a miss access if a dead dirty block exists in the set, the block will be evicted.

To reduce the probability of retention failure and read disturbance, we have to reduce the duration (DDI) that error prone blocks (i.e., dead dirty blocks) are exposed to idle time and risky reads. Thus, in addition to the time we need to evict a block (miss access), predicting miss accesses could help to further reduce DDIs. If an access to a block is highly probable, its nearby blocks will be accessed in the near future (spatial locality). Thus, when a block near LRU position is accessed, it is probable that the accesses in the near future to the cache are miss accesses. We classify the cache blocks into two groups

TABLE I
SET ACCESS PATTERN (A_k *and* b_m *are accessing blocks*)

(a)	$[(a_1, a_2, ..., a_k)(b_1, b_2, ..., b_m)(a_k, ..., a_1)]^N$ $m < setsize$
(b)	$[(a_1, a_2, ..., a_k, a_k, ..., a_1)]^N$ $k > setsize$

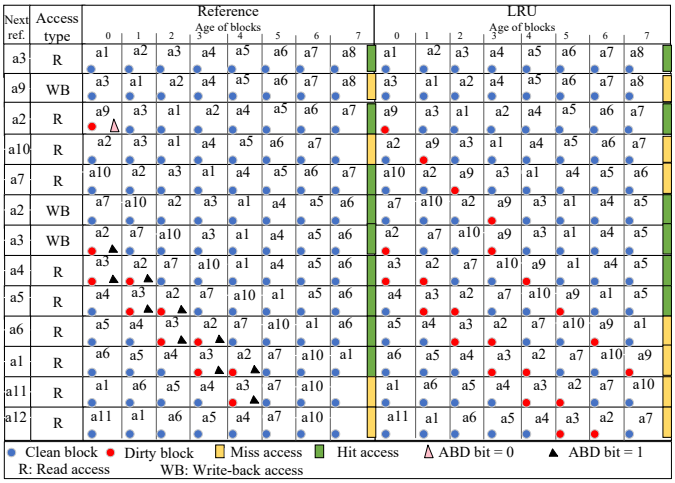

Fig. 10. Behavior of Reference and LRU replacement policy.

in terms of their access pattern: a) blocks re-referenced in the *near* future and b) blocks re-referenced in the *far* future. The blocks re-referenced in the near future are close to the MRU position, while the blocks re-referenced in the far future are near to the LRU position. When a far-future re-referenced block is accessed, it is probable that the subsequent access requests are for far-future re-referenced blocks. However, it is probable that the absence of these blocks causes miss-access in subsequent accesses, as shown in Table I.

To make it more clear, two examples of set access patterns are demonstrated in Table I. In the first example, when b_m is accessed, a_k, if not evicted from the set, is placed close to the LRU position. Thus, we consider this block and subsequent blocks $(a_{k-1}, a_{k-2}, .., a_1)$ as far-future re-referenced blocks. The result of requesting these blocks will be miss accesses. In the second example, the a_k and a_{k-1} blocks are considered as near-future re-referenced blocks and are mostly placed close to the MRU position. On the other hand, the a_1 and a_2 blocks are far-future re-referenced blocks, and if they are not evicted from the set, they are placed close to the LRU position. Hence, we set a second threshold (Th_{high}), so that when a block with an age more than this threshold is accessed, we predict subsequent miss accesses and evict a dead dirty block, if it exists.

With the given observations and definitions, we propose the *Reference* replacement policy to decrease the duration of DDIs. The *Reference* policy decreases the duration of DDIs by early eviction of dead dirty blocks. In the LRU policy, when the age of a dirty block reaches 7 and if a miss occurs, the block will be evicted. However, in this policy, some of the dirty blocks are evicted at age 0 and some of them are evicted at age more than Th_{low} (in miss prediction or miss accesses). Thus, this policy reduces the probability of retention failure and read disturbance by reducing the duration of dirty blocks.

Fig. 10 demonstrates the behavior of the *Reference* policy for multiple accesses to a set of an 8-way set associative cache, containing indicated blocks. The colors blue, red, green, yellow, pink, and black represent clean and dirty blocks, hit and miss accesses, and ABD bit equal to zero and one, respectively. The age of the blocks can also be seen in the figure. As shown, when $a9$ is accessed, the value of the ABD bit is set which is zero because $a9$ has not been accessed before it gets dirty. Thus, in the next access which is a read hit access to $a2$, $a9$ will be evicted from the cache.

When $a4$, $a5$, and $a6$ are accessed, subsequent miss accesses are predicted ($Th_{low} = Th_{high} = 3$), but there is no dead dirty block to be evicted. However, when $a1$ is accessed, subsequent miss accesses are predicted and the dead dirty block $a2$ is evicted. Also, when miss access occurs for $a11$, *Reference* evicts the dead dirty block $a3$ and replaces it with $a11$. Therefore, by cutting down the time interval for dirty blocks to go from MRU to LRU and predicting miss accesses, *Reference* reduces the length of dead dirty intervals, resulting in a reduced probability of retention failure.

The results of this approach in terms of retention failure, read disturbance, and miss rate are calculated and presented in Fig. 11, Fig. 12, and Fig. 13, respectively. Because of two enhancements on the Reference approach in the next subsections, we name the basic Reference as $Approach1$ or $Appr1$ in the figures. According to Fig. 11, the probability of retention failure in DDIs is reduced by 91.9% and 93.6% compared to LRU and RRIP, respectively. This improvement is because of the reduced DDI duration. As depicted in Fig. 12, which shows the number of risky reads per clean and dirty block, the number of risky reads per dirty block in LRU and RRIP replacement policies is reduced by 5.8× and 6.6× for our approach (Appr1), respectively. However, the number of risky reads per clean block in the proposed approach increases by 1.52× and 1.45× compared to the LRU and RRIP replacement policies, respectively. As shown in Fig. 13, Appr1 increases the miss rate by 1.2% compared to the LRU policy.

Next, we investigate different scenarios for dead dirty block prediction and eviction used in the Reference approach for its performance enhancement. The details of these enhancements are presented in the following sections.

B. Reference Policy Enhancement

To enhance the Reference basic idea, `dead dirty blocks` are considered as those dirty blocks with an ABD bit of zero. When a set including a dead dirty block is accessed, in the enhanced Reference policy, $Approach2$ ($Appr2$), the dirty block with an ABD bit equal to zero is evicted. As mentioned, the ABD bit of a block is determined in write-back access. Thus, the dead dirty block will be evicted at the first access to the set (which includes the dead dirty block) after write-back access. If the first access is a miss, the dead dirty block will have an eviction priority; if the first access is a hit, the dead dirty block will be evicted at this access. In case of a miss access, if there is no dead dirty block in the set, this enhanced version of the Reference policy evicts a block based on LRU policy.

According to Fig. 14, Appr2 reduces the probability of retention failure in DDIs by 79.4% and 83.7% compared to the LRU and RRIP policies, respectively. Fig. 15 demonstrates that the number of risky reads per dirty block is decreased by 3.3× and 3.7× using Appr2, in comparison with LRU and

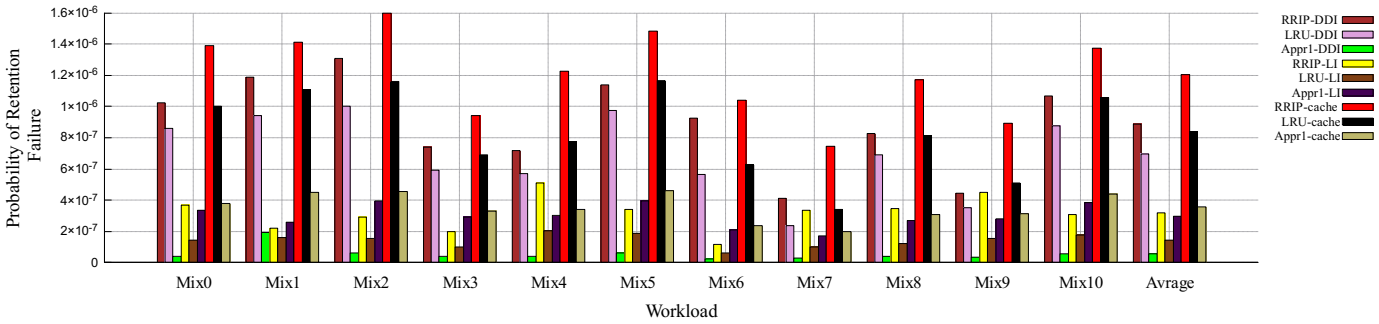

Fig. 11. Probability of retention failure in DDIs, LIs, and cache of LRU, RRIP, and Approach1 (first approach with $Th_{low} = 3$, $Th_{high} = 3$).

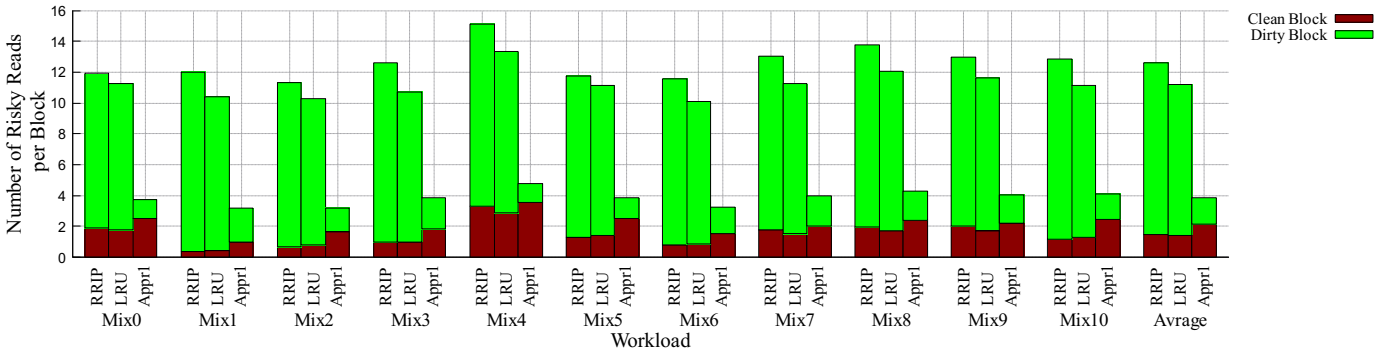

Fig. 12. Number of risky reads in LRU, Approach1, and RRIP.

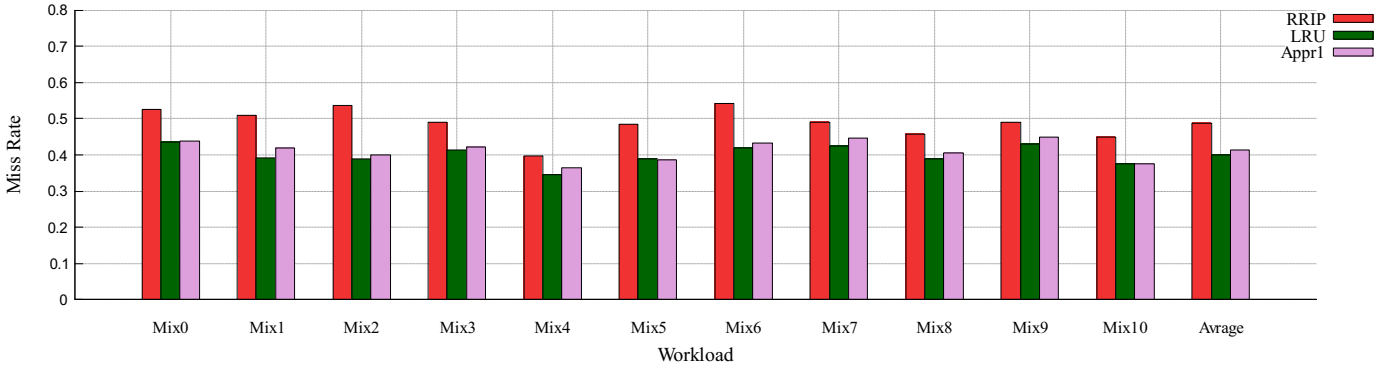

Fig. 13. Miss rate of LRU, Approach1, and RRIP.

RRIP, respectively. However, the number of risky reads per clean block in Appr2 increases by 1.35× and 1.29× compared to LRU and RRIP, respectively. Appr2 increases the miss rate by 0.66% compared to LRU (Fig. 16).

In *Appr1*, we base our observations on what is shown in Fig. 9. In this figure, we examine the number of evicted dirty blocks and their properties such as the number of accesses to dirty blocks before and after they became dirty. This observation shows that under the LRU (RRIP) policy, 85.6% (87.5%) of evicted dirty blocks have not been accessed after becoming dirty. To consider all dirty blocks, a more precise solution involves not only investigating the number of evicted dirty blocks but also including an examination of the number of accesses to dirty blocks (see Fig. 17). For example, when 85.6% of evicted dirty blocks under the LRU policy have not been accessed after becoming dirty, it is probable that there are dirty blocks that have been accessed after becoming dirty but not accessed before becoming dirty. As shown in Fig. 17, the occurrence of this situation is highly probable. This figure (Fig. 17) reports the number of accesses to the dirty blocks, the number of accesses to the evicted dirty blocks, the number of dirty blocks with NABD equal to zero and an NAAD equal to one (NAAD: number of accesses to a block after get dirty), and the number of dirty blocks not accessed before getting dirty and accessed for one time after getting dirty in the LRU position less than 3. According to the results, 79.3% of the dirty blocks not accessed before being dirty and get only one access after being dirty (NABD = 0, NAAD = 1) are accessed in the LRU position less than three after becoming dirty. This value for the *lbm* workload is 99.99%.

To undertake the above investigation and present the ultimate *Reference* policy that outperforms the second approach

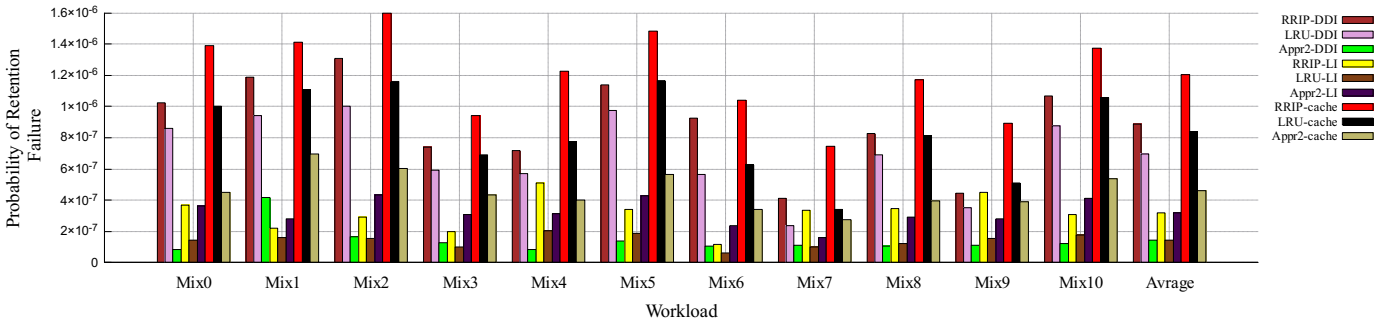

Fig. 14. Probability of retention failure in DDIs, LIs, and cache of LRU, RRIP, and Approach2.

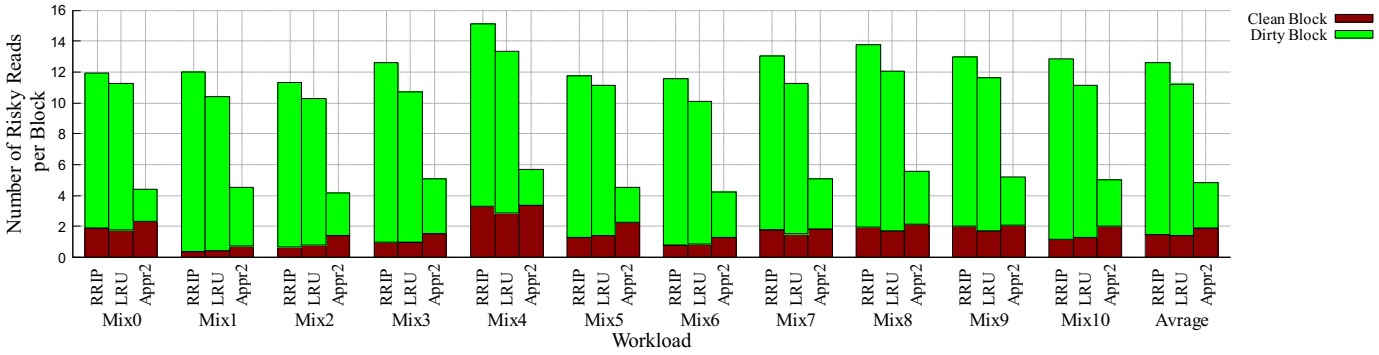

Fig. 15. Number of risky reads in LRU and Approach2.

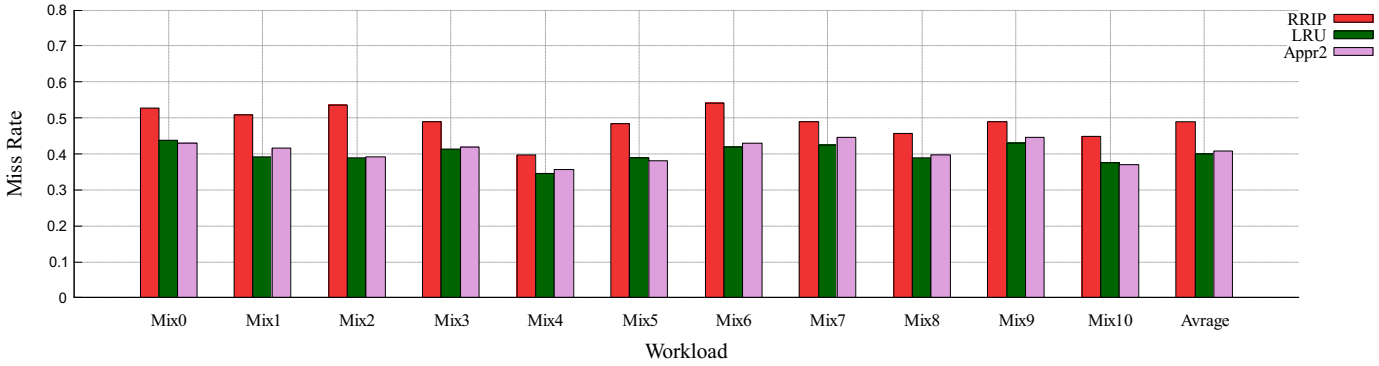

Fig. 16. Miss rate of LRU, Approach2, and RRIP.

(*Appr*2) in terms of reliability and performance, the final dead dirty block prediction and eviction scenarios are specified in the following section.

C. Further Enhancement on Reference Policy

To further reduce the miss rate, we use a threshold (Th_{low_2}) for the dirty blocks with an ABD bit equal to zero in *Approach3 (Appr3)*. When the age of a dirty block with an ABD equal to zero exceeds the threshold, the block will be assumed to be dead. To reduce the duration of DIs (dirty intervals), another threshold (Th_{high_2}) is used. When the age of a dirty block exceeds the second threshold (Th_{high_2}), the block will be assumed to be dead. Typically, Th_{low_2} is smaller than Th_{high_2}, and Th_{high_2} is used for dirty blocks with an ABD bit equal to one. Thus, in Appr3, `dead dirty blocks` refer to either the dirty blocks with an ABD bit equal to zero whose age is more than Th_{low_2}, or the dirty blocks with an ABD bit equal to one, whose age is more than Th_{high_2}.

With the given definitions, this approach evicts the dead dirty blocks under two circumstances: 1) if a set is accessed and it includes a dead dirty block with an ABD bit equal to zero (the block age is greater than Th_{low_2}); and 2) if a miss-access occurs, the dead dirty blocks have eviction priority, so that if there is a dead dirty block with an ABD bit equal to zero in the set, this block will be evicted; otherwise, a dirty block with an age higher than Th_{high_2} will be evicted. If a miss access occurs and there is no dead dirty block, Appr3 evicts a block based on the LRU policy.

VI. Experimental Setup and Results

To evaluate the proposed replacement policy, we model an ARM processor with four cores and two levels of on-chip caches in the gem5 simulator [44], [45]. The details of cache structures are given in Table II. We use the SPEC CPU2006 benchmark suite [43] as our workloads and extract the results by executing two billion instructions. To compromise between

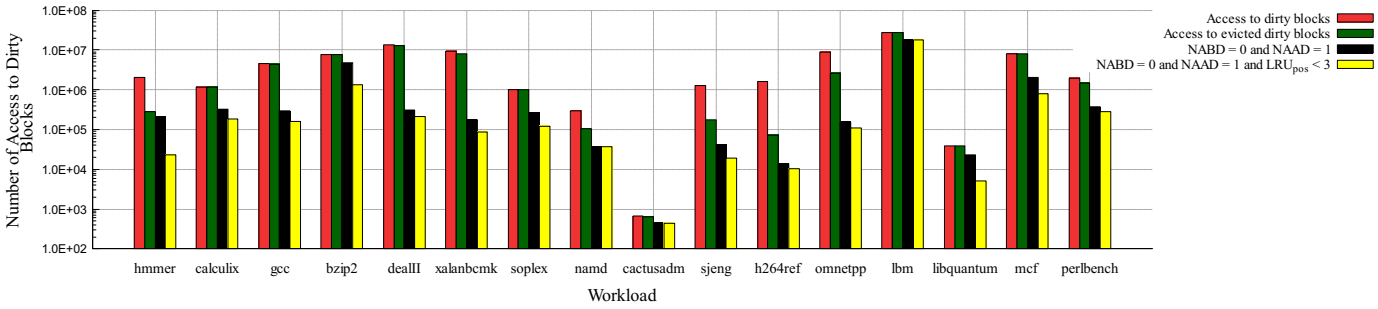

Fig. 17. Access pattern of dirty blocks in LRU policy.

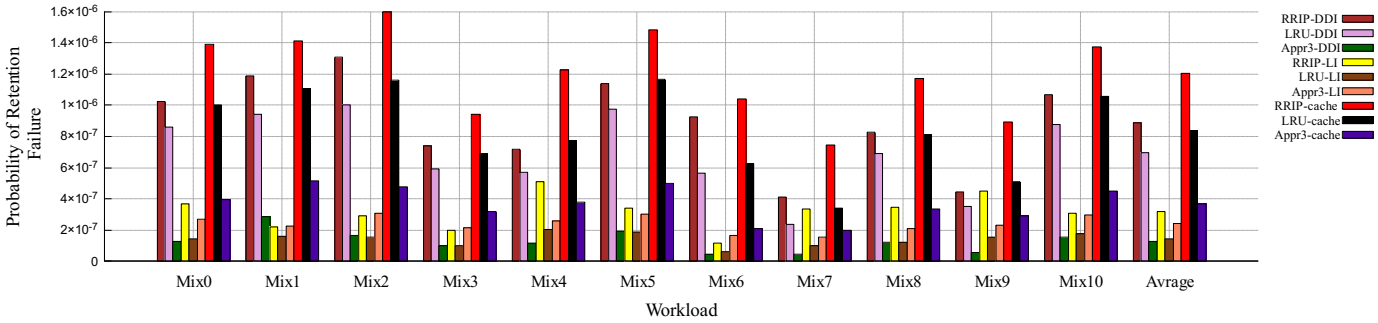

Fig. 18. Probability of retention failure in DDIs, LIs, and cache of LRU, RRIP, and Approach3 policies

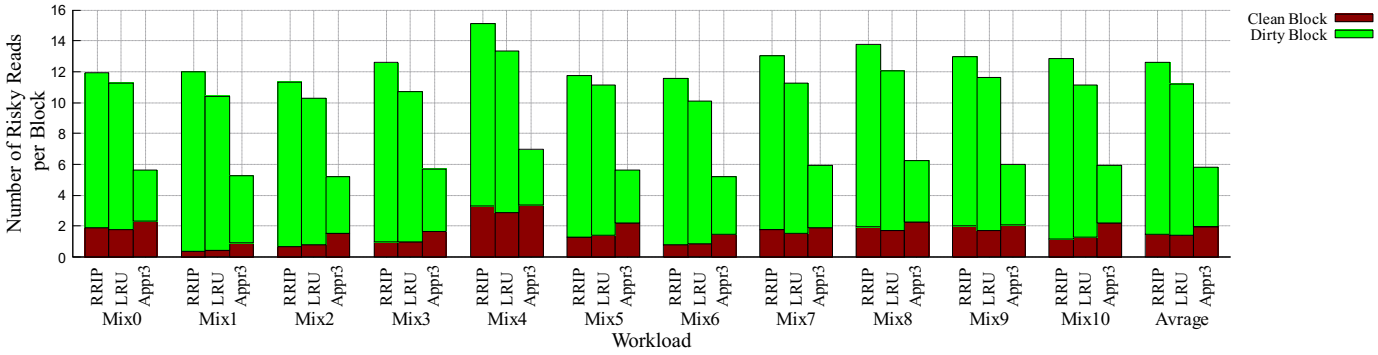

Fig. 19. Number of risky reads in LRU, Approach3, and RRIP

TABLE II
CONFIGURATION OF ON-CHIP CACHES

L1 I-cache	32KB, 4-way set associative, 64B block size write-back, SRAM
L1 D-cache	64KB, 4-way set associative, 64B block size write-back, SRAM
L2 cache	1MB, 8-way set associative, 64B block size write-back, STT-MRAM

the performance overhead and the reliability improvement, for $Appr1$, we set Th_{low} and Th_{high} to 3, and for $Appr3$, we set Th_{low_2} and Th_{high_2} to 2 and 4, respectively.

The previous schemes for STT-MRAM reliability enhancement are not comparable to our proposed policy because they are either circuit-level schemes [19], which increase the STT-MRAM cell robustness, or addressed read disturbance in the tag array [50], which is not applicable to the data array. Among the previous schemes, there are some replacement policies that either focus on cache endurance [64] or write failure [14]. To the best of our knowledge, there is no replacement policy that specifically concentrates on cache reliability due to retention failure and read disturbance. In addition to the conventional LRU replacement policy, we have included the comparison with the state-of-the-art re-reference interval prediction policy (RRIP) [46], which outperforms other replacement policies in dead-dirty block reduction.

A. Reliability

As explored, dead dirty intervals are the main challenge for the cache reliability in facing retention failure. As shown in Fig. 11, the proposed replacement policy reduces the retention failure rate and improves the cache reliability by reducing the idle time of dirty blocks. The first approach (*Appr1*) in comparison to LRU (RRIP) policy reduces the probability of the cache retention failure and DDIs by 57.6% (70.5%) and 91.9% (93.6%), respectively. According to Fig. 14, *Appr2* decreases the probability of the cache retention failure and

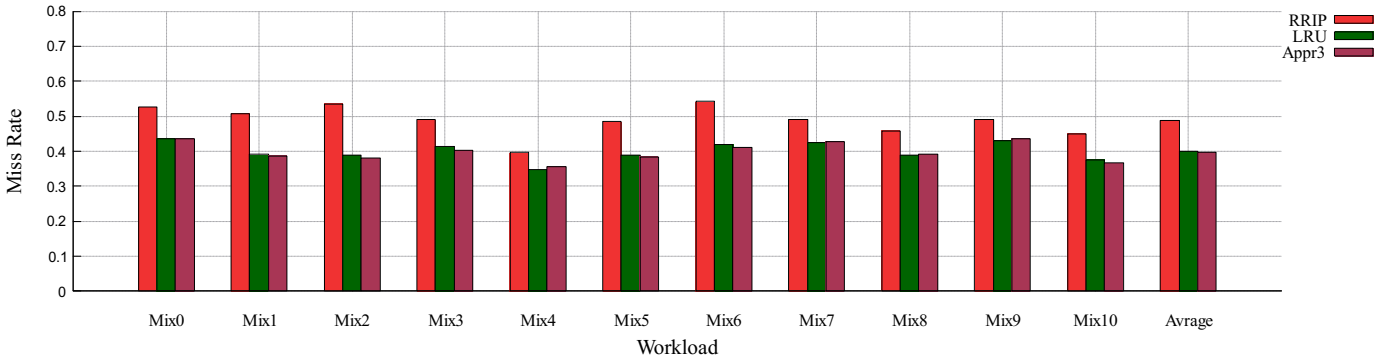

Fig. 20. Miss rate of LRU, Approach3, and RRIP.

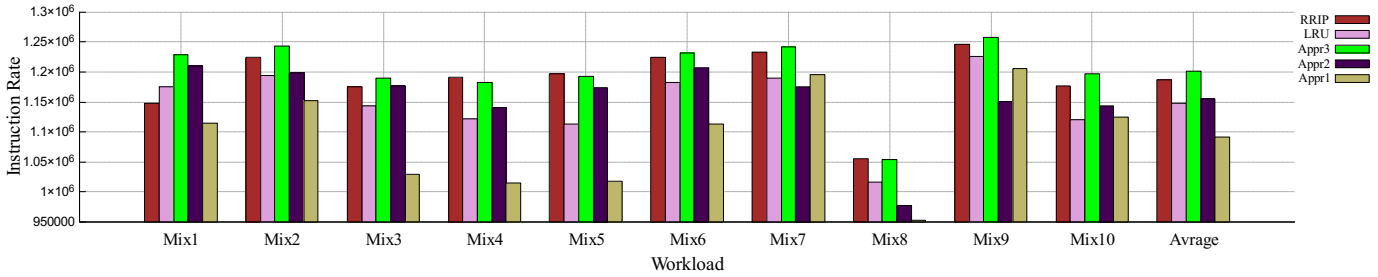

Fig. 21. Instruction Rate for the proposed approaches vs. LRU and RRIP.

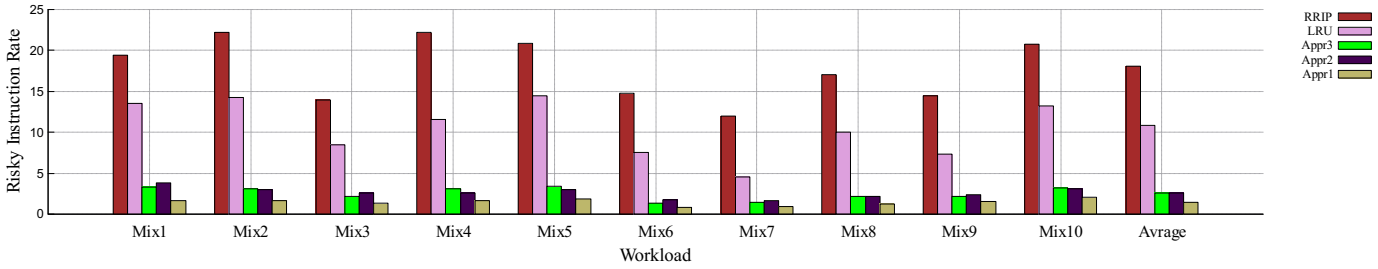

Fig. 22. Risky Instruction Rate for the proposed approaches vs. LRU and RRIP.

DDIs by 44.9% (61.6%) and 79.4% (83.9%), respectively, compared to the LRU (RRIP) policy. Fig. 18 shows the probability of retention failure in *Appr3*, LRU, and RRIP policies. *Appr3* reduces the probability of the cache retention failure and DDIs by 56.0% (69.3%) and 81.4% (85.5%), respectively, compared to the LRU (RRIP) policy.

As discussed earlier, dirty blocks have a low access rate (85.6% of dirty blocks are not accessed after being dirty). Thus, the high read access rate per dirty block is because of miss read accesses to the set, including this block or hit read access to the other blocks of the set. To reduce the read access rate of dirty blocks, the proposed policies decrease the duration of DIs. The results in terms of the number of risky reads per block are depicted in Fig. 12, Fig. 15, and Fig. 19. As shown in these figures, the number of risky reads per dirty block in LRU (RRIP) is 5.8× (6.6×), 3.3× (3.7×), and 2.6× (2.9×) using *Appr1*, *Appr2*, and *Appr3* policies, respectively. The results show that our enhanced approach (*Appr3*) leads to fewer reads and consequently, reduces the number of read accesses per dirty block; hence, the probability of read disturbance decreases as well. In the best case, the Reference (*Appr1*) replacement policy reduces the probability of retention failure in the cache and DDIs by 57.6% (70.5%) and 91.9% (93.6%), respectively, compared to the LRU (RRIP) replacement policy.

B. Performance

To investigate the performance of the proposed approach, we calculate the miss rate of all workloads. As shown in Fig. 13 and Fig. 16, *Appr1* and *Appr2* increase the miss rate by 1.2% and 0.7% compared to LRU, respectively. According to Fig. 20, *Appr3* decreases the miss rate by 0.2% compared to LRU. This can be explained by the observation reported in Fig. 9, which shows an enormous number of dirty blocks are not accessed, resulting in a considerable decrease in the probability of retention failure and read disturbance, as well as an improvement in the miss rate.

The instruction rate of the presented approaches and the LRU and RRIP replacement policies are shown in Fig. 21. As shown in this figure, the instruction rate in the third approach increases by 4.7% and 1.3% compared to LRU and RRIP, respectively. In the second approach, the instruction per

second increases (decreases) by 0.7% (2.6%) compared to the LRU (RRIP) replacement policy. The performance overhead in terms of instruction rate for the first approach compared to the LRU and RRIP is 5.3% and 8.1%, respectively. Considering the logic complexity of the proposed scheme, the cache controlling logic checks ABD bit on a cache miss to prioritize the dead blocks for eviction. Since the victim block selection on a miss is not on the critical path and the modifications required for controlling logic are minimal, no latency overhead is imposed in this regard.

To address reliability challenges in STT-MRAM caches, many studies have been conducted. However, to the best of our knowledge, there is no presented replacement policy that reduces the effect of read disturbance and retention failure on the reliability of STT-MRAM. Therefore, to better understand our contribution and ensure fairness to all replacement policies, we use a new metric called *Risky Instruction Rate*. The risky instruction rate indicates the number of instructions within a cycle for which the occurrence of read disturbance and retention failure is highly probable. This metric combines the performance and reliability of STT-MRAM caches. Fig. 22 shows the risky instruction rate for RRIP, LRU, *Appr3*, *Appr2*, and *Appr1*. According to this figure, the risky instruction rate for the LRU (RRIP) replacement policy is 4 (7) times greater than that of Appr3, on average.

C. Area and Energy Consumption

As explained earlier, our analysis showed that 85.6% of evicted dirty blocks from the LRU policy are not accessed after being dirty. Additionally, 99.4% of these blocks are not accessed before being dirty. In the proposed policy, we identified these blocks as `dead blocks`. To determine these blocks, a bit (ABD bit) is added to each block. Based on our analysis, the added bit in *Reference* policy imposes less than 0.2% overhead to the cache area, which is negligible. The energy consumption overhead is related to the required energy for writes and reads on the ABD bit. However, the proposed replacement policy aims to evict error-prone dead dirty blocks to prevent unnecessary reads and ideal time (Fig. 19). Thus, by reducing the number of unnecessary reads, the proposed policy decreases energy consumption in addition to avoiding read disturbance.

VII. Conclusion

Retention failure and read disturbance are the main sources of the failure rate in the last level of the STT-MRAM caches due to long error-prone idle intervals and high risky reads rate. This paper evaluated the idle time and read accesses of low-accessed cache blocks and proposed the *Reference* replacement policy to reduce the length of dead dirty intervals and the number of risky reads. The *Reference* policy reduces the probability of retention failure by 56.0% (69.3%) and reduces the number of risky reads per dirty block by 61.2% (65.9%) compared to the LRU (RRIP) policy. This reliability improvement is achieved with a negligible area and no energy consumption and performance overheads.

References

[1] T. Na, S. H. Kang, and S.-O. Jung, "Stt-mram sensing: a review," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[2] H. Farbeh, A. M. H. Monazzah, E. Aliagha, and E. Cheshmikhani, "A-cache: Alternating cache allocation to conduct higher endurance in nvm-based caches," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 7, pp. 1237–1241, 2018.

[3] T. Huynh-Bao, A. Veloso, S. Sakhare, P. Matagne, J. Ryckaert, M. Perumkunnil, D. Crotti, F. Yasin, A. Spessot, A. Furnemont *et al.*, "Process, circuit and system co-optimization of wafer level co-integrated finfet with vertical nanosheet selector for stt-mram applications," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[4] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, M. H. Cilasun, J. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "MOUSE: inference in non-volatile memory for energy harvesting applications," in *IEEE/ACM International Symposium on Microarchitecture, MICRO , Athens, Greece, October 17-21*. IEEE, 2020, pp. 400–414.

[5] K. Korgaonkar, I. Bhati, H. Liu, J. Gaur, S. Manipatruni, S. Subramoney, T. Karnik, S. Swanson, I. Young, and H. Wang, "Density tradeoffs of non-volatile memory as a replacement for SRAM based last level cache," in *ACM/IEEE Annual International Symposium on Computer Architecture, ISCA Los Angeles, CA, USA*, M. Annavaram, T. M. Pinkston, and B. Falsafi, Eds., 2018, pp. 315–327.

[6] H. Li, M. Bhargav, P. N. Whatmough, and H.-S. P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *ACM/IEEE design automation conference (DAC)*, 2019, pp. 1–6.

[7] E. Cheshmikhani, H. Farbeh, and H. Asadi, "A system-level framework for analytical and empirical reliability exploration of stt-mram caches," *IEEE Transactions on Reliability*, vol. 69, no. 2, pp. 594–610, 2019.

[8] H. Asadi, E. Cheshmikhanikhanghah, and H. Farbeh, "Preventing read disturbance accumulation in a cache memory," Jun. 18 2022, uS Patent App. 16/798,451.

[9] E. Cheshmikhani, H. Farbeh, and H. Asadi, "Enhancing reliability of stt-mram caches by eliminating read disturbance accumulation," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 854–859.

[10] G. Hu, "Spin-transfer torque mram-status and outlook," in *IEEE International Magnetic Conference-Short Papers (INTERMAG Short Papers)*, 2023, pp. 1–2.

[11] M. Hadizadeh, E. Cheshmikhani, M. Rahmanpour, O. Mutlu, and H. Asadi, "Copa: Cold page awakening to overcome retention failures in stt-mram based i/o buffers," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[12] N. Mahdavi, F. Razaghian, and H. Farbeh, "An architectural-level reliability improvement scheme in stt-mram main memory," *Microprocessors and Microsystems*, vol. 90, p. 104462, 2022.

[13] H. Asadi and E. Cheshmikhanikhanghah, "Reducing read disturbance error in tag array," 30 2022, uS Patent 17,204,957.

[14] E. Cheshmikhani, H. Farbeh, S. G. Miremadi, and H. Asadi, "Ta-lrw: A replacement policy for error rate reduction in stt-mram caches," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 455–470, 2018.

[15] S. Sethuraman, V. K. Tavva, and M. Srinivas, "Techniques to improve write and retention reliability of stt-mram memory subsystem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[16] E. Cheshmikhani, H. Farbeh, and H. Asadi, "Robin: Incremental oblique interleaved ecc for reliability improvement in stt-mram caches," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 173–178.

[17] S. Han, J. Lee, K. Suh, K. Nam, D. Jeong, S. Oh, S. Hwang, Y. Ji, K. Lee, Y. Song *et al.*, "Reliability of stt-mram for various embedded applications," in *IEEE International Reliability Physics Symposium (IRPS)*, 2021, pp. 1–5.

[18] D. C. Worledge, "Write-error-rate of spin-transfer-torque mram," in *2023 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2023, pp. 1–4.

[19] 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.

[20] N. Sayed, S. M. Nair, R. Bishnoi, and M. B. Tahoori, "Process variation and temperature aware adaptive scrubbing for retention failures in stt-mram," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 203–208.

[21] X. Bi, H. Li, and J.-J. Kim, "Analysis and optimization of thermal effect on stt-ram based 3-d stacked cache design," in *IEEE Computer Society Annual Symposium on VLSI*, 2012, pp. 374–379.

[22] R. Bishnoi, M. Ebrahimi, F. Oboril, and M. B. Tahoori, "Read disturb fault detection in stt-mram," in *International Test Conference*, 2014, pp. 1–7.

[23] S. M. Seyedzadeh, R. Maddah, A. Jones, and R. Melhem, "Leveraging ecc to mitigate read disturbance, false reads and write faults in stt-ram," in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 215–226.

[24] B. Wu, B. Zhang, Y. Cheng, Y. Wang, D. Liu, and W. Zhao, "An adaptive thermal-aware ecc scheme for reliable stt-mram llc design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1851–1860, 2019.

[25] X. Zhong, K. Cai, P. Kang, G. Song, and B. Dai, "Deep learning-based adaptive error-correction decoding for spin-torque transfer magnetic random access memory (stt-mram)," *IEEE Transactions on Magnetics*, 2023.

[26] M.-S. Wu, Y.-L. Chua, J.-F. Li, Y.-T. Chuan, and S.-H. Huang, "Fault-aware ecc scheme for enhancing the read reliability of stt-mrams," in *2023 IEEE International Test Conference in Asia (ITC-Asia)*. IEEE, 2023, pp. 1–6.

[27] T. Na, J. P. Kim, S. H. Kang, and S.-O. Jung, "Read disturbance reduction technique for offset-canceling dual-stage sensing circuits in deep submicrometer stt-ram," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 6, 2016.

[28] D. Zhang, L. Zeng, Y. Zhang, J. O. Klein, and W. Zhao, "Reliability-enhanced hybrid cmos/mtj logic circuit architecture," *IEEE Transactions on Magnetics*, vol. 53, no. 11, pp. 1–5, 2017.

[29] J. W. Kwak, A. Marshall, and H. Stiegler, "28nm stt-mram array and sense amplifier," in *International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2019, pp. 1–4.

[30] V. Tavva, M. Srinivas, and S. Sethuraman, "Techniques to improve write and retention reliability of stt-mram memory subsystem," 2022.

[31] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *IEEE International Symposium on High Performance Computer Architecture*, 2011, pp. 50–61.

[32] S. Mittal, "Mitigating read disturbance errors in stt-ram caches by using data compression," in *Nanoelectronics*. Elsevier, 2019, pp. 133–152.

[33] I. Alam, S. Pal, and P. Gupta, "Compression with multi-ecc: Enhanced error resiliency for magnetic memories," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 85–100.

[34] N. Sayed, M. Ebrahimi, R. Bishnoi, and M. B. Tahoori, "Opportunistic write for fast and reliable stt-mram," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 554–559.

[35] N. Sayed, R. Bishnoi, and M. B. Tahoori, "Fast and reliable stt-mram using nonuniform and adaptive error detecting and correcting scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1329–1342, 2019.

[36] N. Sayed, R. Bishnoi, F. Oboril, and M. B. Tahoori, "A cross-layer adaptive approach for performance and power optimization in stt-mram," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 791–796.

[37] R. Bishnoi, M. Ebrahimi, F. Oboril, and M. B. Tahoori, "Improving write performance for stt-mram," *IEEE Transactions on Magnetics*, vol. 52, no. 8, pp. 1–11, 2016.

[38] A. M. H. Monazzah, A. M. Rahmani, A. Miele, and N. Dutt, "Cast: Content-aware stt-mram cache write management for different levels of approximation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4385–4398, 2020.

[39] Y. Gupta and L. Bhargava, "Write energy reduction of stt-mram based multi-core cache hierarchies," *International Journal of Electronics Letters*, vol. 7, no. 3, pp. 249–261, 2019.

[40] O. Coi, G. Patrigeon, S. Senni, L. Torres, and P. Benoit, "A novel sram—stt-mram hybrid cache implementation improving cache performance," in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2017, pp. 39–44.

[41] K.-W. Kwon, S. H. Choday, Y. Kim, and K. Roy, "Aware (asymmetric write architecture with redundant blocks): A high write speed stt-mram cache architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 712–720, 2013.

[42] N. Sayed, L. Mao, R. Bishnoi, and M. B. Tahoori, "Compiler-assisted and profiling-based analysis for fast and efficient stt-mram on-chip cache design," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 4, pp. 1–25, 2019.

[43] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[44] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[45] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, 2011.

[46] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 60–71, 2010.

[47] N. Sayed, L. Mao, and M. B. Tahoori, "Dynamic behavior predictions for fast and efficient hybrid stt-mram caches," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 1, pp. 1–21, 2021.

[48] M. Hadizadeh, E. Cheshmikhani, and H. Asadi, "Stair: High reliable stt-mram aware multi-level i/o cache architecture by adaptive ecc allocation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[49] M. Rahbari and H. Farbeh, "Crp: Conditional replacement policy for reliability enhancement of stt-mram caches," *IEEE Transactions on Magnetics*, vol. 58, no. 7, pp. 1–13, 2022.

[50] E. Cheshmikhani, H. Farbeh, and H. Asadi, "3rset: Read disturbance rate reduction in stt-mram caches by selective tag comparison," *IEEE Transactions on Computers*, 2021.

[51] T. Endoh, H. Honjo, K. Nishioka, and S. Ikeda, "Recent progresses in stt-mram and sot-mram for next generation mram," in *IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[52] M. Talebi, A. Salahvarzi, A. M. H. Monazzah, K. Skadron, and M. Fazeli, "Rocky: A robust hybrid on-chip memory kit for the processors with stt-mram cache technology," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2198–2210, 2020.

[53] X. Jiang, J. Bao, L. Zhang, and L. Bai, "A novel dual-reference sensing scheme for computing in memory within stt-mram," *Microelectronics Journal*, vol. 121, p. 105355, 2022.

[54] K. Kuan and T. Adegbija, "Lars: Logically adaptable retention time stt-ram cache for embedded systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 461–466.

[55] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *DAC Design Automation Conference*, 2012.

[56] J. Fu, L. Sun, X. Tong, B. Liu, and H. Cai, "High sensing margin sensing amplifier with improved reliability for stt-mram," in *IEEE 23rd International Conference on Nanotechnology (NANO)*. IEEE, 2023, pp. 550–555.

[57] A. M. H. Monazzah, H. Farbeh, and S. G. Miremadi, "Ler: Least-error-rate replacement algorithm for emerging stt-ram caches," *IEEE Transactions on Device and Materials Reliability*, vol. 16, no. 2, pp. 220–226, 2016.

[58] T. Hadámek, S. Selberherr, W. Goes, and V. Sverdlov, "Modeling thermal effects in stt-mram," *Solid-State Electronics*, vol. 200, p. 108522, 2023.

[59] T. A. Nguyen and J. Lee, "Improving bit-error-rate performance using modulation coding techniques for spin-torque transfer magnetic random access memory," *IEEE Access*, vol. 11, pp. 33 005–33 013, 2023.

[60] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 175–186.

[61] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 430–441.

[62] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017, pp. 180–193.

[63] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 78–89.

[64] S. G. Ghaemi, I. Ahmadpour, M. Ardebili, and H. Farbeh, "Sleepy-lru: Extending the lifetime of non-volatile caches by reducing activity of age bits," *The Journal of Supercomputing*, vol. 75, no. 7, pp. 3945–3974, 2019.