ROCKY: A Robust Hybrid On-chip Memory Kit For The Processors With STT-MRAM Cache Technology

Mahdi Talebi*, Arash Salahvarzi*, Amir Mahdi Hosseini Monazzah*†, Kevin Skadron‡, and Mahdi Fazeli§

Abstract—STT-MRAM is regarded as an extremely promising NVM technology for replacing SRAM-based on-chip memories. While STT-MRAM memories benefit from ultra-low leakage power and high density, they suffer from some reliability challenges, namely, read disturbance, write failure, and retention failure. The write failure; storing a wrong value in an STT-MRAM cell during a write operation, is the most crucial reliability challenge. In this paper, we propose ROCKY; a robust architecture equipped with efficient replacement policies for STT-MRAM-based cache memory hierarchy to improve the robustness of STT-MRAM part against the write failures. ROCKY reduces susceptible transitions in STT-MRAM cache memories leading to more reliable STT-MRAM write operations. The simulation results through comparison with traditional cache memory hierarchy demonstrate ROCKY decreases the WER of STT-MRAM cache memories by up to 35.4% while imposing less than 1% performance overhead to the system.

Index Terms—Write Failure Rate, Replacement Policy, STT-MRAM Cache, On-Chip Memory Hirerachy.

1 Introduction

THe superiority of Non-Volatile Memory (NVM) technologies over SRAM memory technology in terms of leakage power consumption, memory density, and fault robustness has made the NVM as a promising technology for future on-chip memories. Among different NVM technologies, STT-MRAM is regarded as the most viable replacement technology for SRAM [1], [2], [3].

While STT-MRAMs benefit from low leakage power and high density, they suffer from some reliability challenges that should be addressed before being employed as on-chip memories. The most common reliability challenges of STT-MRAMs are *read disturbance* (unwanted change of the STT-MRAM cell value during the read operation), *write failure* (the value is erroneously written in the STT-MRAM cell during the write operation), and *retention failure* (unwanted change of the STT-MRAM cell value during idle time) [4], [5], [6]. Among the mentioned reliability challenges, write failure is the most serious failure challenge [7], [8].

The write operation in an STT-MRAM cell is performed by applying a directed write current to the STT-MRAM cell. Considering the stochastic switching characteristic of STT-MRAMs, the switching probability of an STT-MRAM cell can be increased by either applying the write current for a longer period of time or increasing the amplitude of the write current. Accordingly, to tackle the write failure problem, there are many studies that have proposed to modify the STT-MRAM write signal characteristics [9], [10]. On the other hand, the WER of an STT-MRAM cell is highly dependant on the pattern of values being written [8], [11], [12], [13], [14]. It means that the write failure probability during $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions are different. It is reported that the STT-MRAM write error probability in a $0 \rightarrow 1$ transition compared to a $1 \rightarrow 0$ transition is greater by two orders of magnitude [15], [16], [17].

Considering asymmetric write error of STT-MRAMs, there are some studies that suggest using Error Correction Codes (ECCs) to mitigate the write failure rate [6], [15]. However, using ECCs leads to extra energy consumption and imposes performance and area overhead to system. Another study, called Least Error Rate (LER) [14], has proposed a memory-level traffic management policy to alleviate STT-MRAM write failure challenge. This method tries to decrease the $0 \rightarrow 1$ transitions in the STT-MRAM cache by manipulating traditional replacement policy. But, unfortunately due to exorbitant read operations, LER increases the dynamic energy consumption and probability of read disturbance error.

In cache memories, the write operation is issued either by occurrence of a missed request (for filling the cache with the requested memory block) or by updating the memory block value of a hit request. In both of the mentioned situations, the replacement policies that are applied in the cache memory hierarchy play an important role in imposing write operations to the cache memories. It is obvious that an inefficient replacement policy for an STT-MRAM cache directly increases write operations by increasing the miss ratio of the cache [18]. In fact, the more the replacement policy is lucky in decreasing the number of missed requests, the less write operations are imposed to the STT-MRAM

* *Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.*
† *School of Computer Science, Institute for Research in Fundamental Sciences (IPM).*
‡ *Department of Computer Science, University of Virginia, Charlottesville.*
§ *Department of Computer Engineering, Bogazici University, Istanbul, Turkey.*
E-mails: mahdi_talebi@comp.iust.ac.ir, arash_salahvarzi@alumni.iust.ac.ir, monazzah@iust.ac.ir, skadron@virginia.edu, mahdi.fazeli@boun.edu.tr.

Manuscript received XX, XX

cache. On the other hand, the replacement policy used in the higher-level cache is also indirectly contributes in the imposed write operations to the STT-MRAM cache. Indeed, the total number of write-back requests issued by the higher-level cache is controlled by the replacement policy and may negatively affect the write operations in STT-MRAM cache.

In this paper we propose ROCKY, a robust hybrid on-chip memory scheme to improve the reliability of STT-MRAM caches with respect to the write failures. ROCKY includes a set of replacement policies for the cache memory hierarchy which are tuned for its proposed hybrid memory architecture. ROCKY considers $0 \rightarrow 1$ transition metric augmented with traditional performance-oriented metrics as the decision making parameters for ROCKY replacement policies. Briefly, the main contribution of this study is as follows:

- We perform a set of analysis to explore the effects of different replacement policies on the reliability of the cache memory hierarchy employing STT-MRAM cache.
- We propose a set of replacement policies which are efficiently coupled with each other at different cache memory levels to improve the reliability of STT-MRAM caches.
- We design and evaluate the underlying cache memory hierarchy architecture to utilize the proposed replacement policies.

We evaluate the efficiency of ROCKY in gem5 simulator [19] using SPEC CPU2006 benchmarks[20]. The simulation results show that while ROCKY improves the write failure probability of STT-MRAM cache by up to 35.4%, it only imposes less than 1% performance overhead to the system in comparison with the conventional replacement policies.

The rest of this paper is organized as follows: In Section 2 the preliminaries about STT-MRAM, its reliability characteristics, and the motivation of this study are presented. The details of ROCKY are introduced in Section 3. Then, we will explore the simulation setup and results in Section 4. The previous studies are reviewed in Section 5. Finally, we conclude the paper in Section 6.

2 PRELIMINARIES ON STT-MRAM AND MOTIVATION

In the following, we first discuss the basics of STT-MRAM technology and explore its reliability challenges. Then, we will observe the effects of cache replacement policies on the write failure rate of STT-MRAM caches and at the end of this section, we will present our motivation for performing this study and presenting ROCKY.

2.1 STT-MRAM Fundamentals

An STT-MRAM memory cell is composed of two components: A Magnetic Tunnel Junction (MTJ) and an NMOS access transistor which are connected in series as shown in Fig. 1. The MTJ is used for storing one bit data and the access transistor that is controlled by the Word Line (WL) is responsible for supplying the MTJ by proper read/write current whenever the MTJ needs to be accessed for read-/write operations. The MTJ element consists of an insulating film (e.g., MgO) surrounded by two ferromagnetic layers

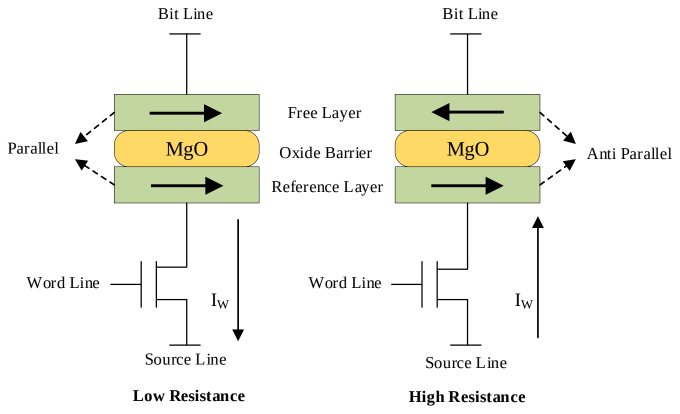

Fig. 1: Structure of an STT-MRAM cell

[1], [21]. The ferromagnetic layer with constant magnetic direction is called *pinned* or *reference* layer and the other one with flexible magnetic direction is called *free* layer.

In order to read the stored value in an STT-MRAM cell, the WL is activated and a low sensing current is applied through the cell to sense the resistance of MTJ. If the sensing current is higher than the reference current, the MTJ is in low resistance state and content of the cell is '0'. Otherwise, MTJ is in high resistance state and the cell contains '1' [22].

To write '0' in an STT-MRAM cell, the access transistor is turned on by activating the WL. Then the Bit Line (BL) and the Source Line (SL) are set to high and low, respectively. By doing so, a spin-polarized current is injected to MTJ flowing from BL to SL. In this case magnetic directions of the two ferromagnetic layers are in parallel ('P') state and the MTJ is in low resistance state which is a representative of '0' stored in the cell. If BL is set to low and SL is set to high, the current will flow from SL to BL and '1' will be written in the MTJ. This time, magnetic directions of the two ferromagnetic layers are in anti-parallel ('AP') state and the MTJ is in high resistance state which indicates '1' is stored in the STT-MRAM memory cell [23].

During a read operation, if the applied read current is higher than the critical switching current, content of the cell will be changed unwantedly, leading to an error, called *read disturbance* [5]. The probability of read disturbance occurrence in an STT-MRAM cell is calculated according to [23]:

$$P_{rd}(t_r) = 1 - exp(\frac{-t_r}{\tau} \times exp(\frac{-\Delta(I_r - I_{C_0})}{I_{C_0}})) \qquad (1)$$

Where, I_r is the applied read current, I_{C_0} is the critical switching current to write in $0°$K, t_r is duration of read pulse and τ is attempt period which is supposedly 1ns [24]. One of the device-level methods to reduce the retention failure probability in STT-MRAM based caches is to increase the thermal stability factor.

On the other hand, when an STT-MRAM cell remains idle for an interval, its content might change due to its stochastic behaviour, leading to an error, called *retention failure* [4]. The probability of retention failure is expressed by the following equation [24]:

$$P_{rf}(t) = 1 - exp(\frac{-t}{exp(\Delta)}) \quad (2)$$

Where, t is the idle interval and Δ is thermal stability factor. As it can be seen, retention failure is exponentially dependent on Δ. Similar to read disturbance, increasing the thermal stability factor, makes the STT-MRAM cell less vulnerable to retention failure.

Finally, write operations in STT-MRAMs also show a stochastic behavior due to thermal variation in MTJs. If the write current signal period is shorter than the MTJ switching time, the cell is not updated with the new datum correctly and its content remains unchanged, resulting in an error, called *write failure* [13], [25]. The probability of write failure in an STT-MRAM cell is expressed as [26]:

$$P_{wf}(t_w) = exp(-t_w \times \frac{2\mu_B p(I_w - I_{C_0})}{(c + ln(\pi^2 \frac{\Delta}{4})) \times (em(1+p^2))}) \quad (3)$$

Where, I_w is write current, t_w is duration of write pulse, μ_B is Bohr magneton, p is tunneling spin polarization, e is charge of electron, m is magnetic momentum of the free layer and C is Euler constant. According to this formula, increasing the amplitude or width of the write current, diminishes the write failure rate. However, this results in extra energy consumption and increasing the probability of oxide barrier breakdown.

In STT-MRAM based memories the switching time of $0 \rightarrow 1$ transition is much more than the $1 \rightarrow 0$ transition, which means the write failure rate has an asymmetric behavior. As mentioned before, the failure probability of write operation in $1 \rightarrow 0$ transition is two orders of magnitude lower than $0 \rightarrow 1$ transition [15], [16], [17]. Therefore, $0 \rightarrow 1$ transitions are the main contributor to the write failure in STT-MRAM based memories.

2.2 Observations and Motivation

As we mentioned previously, $0 \rightarrow 1$ transitions in write operations are the main contributor of the write failures in the STT-MRAMs. On the other hand, in a write operation, the number of $0 \rightarrow 1$ transitions in a cache block are proportional to Hamming Weight (HW) which is equal to the number of ones in the incoming block. Indeed, the worst case we may experience is that the number of $0 \rightarrow 1$ transitions are equal to incoming block's HW. It should be noted that in the following of this study, we will take HW of incoming blocks as a metric for comparing the effectiveness of replacement policies in reducing the write failure rate in STT-MRAM cache.

Considering an STT-MRAM based cache memory located farther from L1 cache memory in the memory cache hierarchy, the write operations are performed due to 1) a missed request (for filling the new data) or 2) a hit write-back request to update the contents of cache block. For an STT-MRAM cache memory, it is obvious that its replacement policy directly affects the write failure probability by collecting the most suitable data and keeping the hit rate as high as possible. On the other hand, the contribution of the upper-level (closer to cpu) cache memory's replacement policy in the write failure of STT-MRAM cache could not be understood in a straightforward manner. To this end, we conducted a set of simulations using gem5 full-system simulator [19] [1]. Accordingly, we fixed the L2 cache replacement policy to Least Recently Used (LRU) and alternatively changed the L1 data cache replacement policy to observe the effects of replacement policies on the write failure.

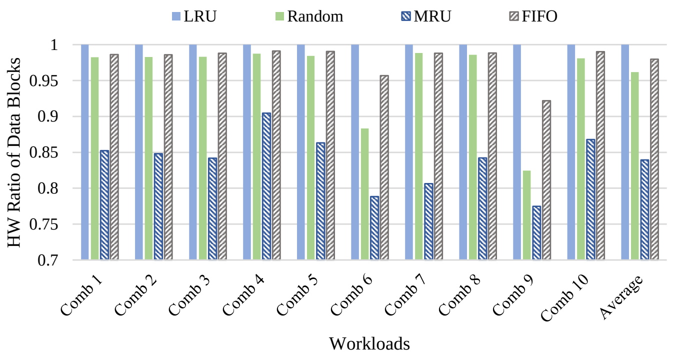

Fig. 2: HW ratio of L2 cache incoming data blocks, normalized to LRU as L1 cache replacement policy

The replacement policies that are chosen for L1 data cache are: LRU, Most Recently Used (MRU), Random, and First In First Out (FIFO). The victim block according to LRU policy is always the one which has been used less than the others in the cache set. MRU policy acts exactly opposite of LRU policy and evicts the block which was the last one read or written. Random replacement policy regardless of the usage frequency of the blocks chooses the victim block arbitrary and according to the FIFO replacement policy, the evicted block is the one which was first written in the cache set in comparison with the others.

Additionally, to have a more clear way of comparing the effect of different L1 cache replacement policies on HW of L2 cache incoming data blocks, we have chosen the HW ratio parameter, which is equal to number of one bits over the number of all bits in each incoming data block.

Fig. 2 depicts the HW ratio of L2 cache incoming write requests with respect to the four different L1 data cache replacement policies. The results in the Fig. 2 are normalized to the case that L1 replacement policy is LRU. As it can be seen in the Fig. 2, changing the L1 data cache replacement policy has a direct effect on the HW ratio of the incoming data blocks that are written in L2 cache. Accordingly, selecting LRU as the replacement policy for L1 data cache leads to higher HW ratio of write requests in comparison with other replacement policies. This shows LRU replacement policy imposes more susceptibility to write failure when it is applied in cache hierarchy.

According to Fig. 2, when FIFO and random policies are selected as L1 data cache replacement policy, compared to the case that LRU is set as L1 replacement policy, they reduce the HW ratio by 2% and 3.8%, on average, respectively. This shows that none of the FIFO or random replacement policies are much effective in reducing the write failure rate in L2 cache. On the other hand, MRU policy consid-

1. A 32-Kbyte 4-way set associative L1 SRAM data cache and 4-Mbyte 16 way set associative STT-MRAM L2 cache are considered with 512 bits blocks' width. The multi-programmed workloads are selected from SPEC CPU2006 [20] benchmarks suite.

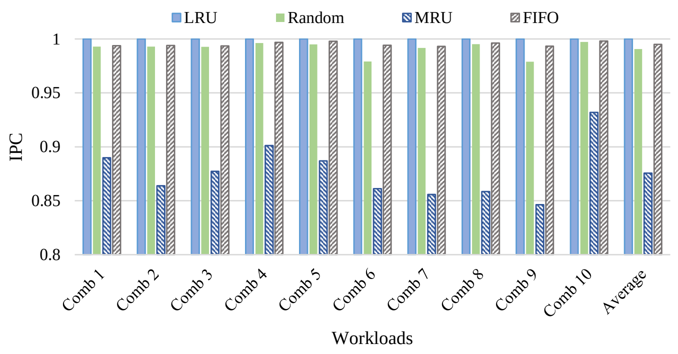

Fig. 3: IPC of selected replacement policies for level one cache, normalized to LRU as L1 cache replacement policy

erably reduce the HW ratio by 16%, on average. It should be noted that, since none of the explored L1 replacement policies consider the HW ratio of the evicted block on their algorithms, the behaviour of them completely depend on the workloads. Thus, the depicted results may vary with workloads. Indeed, in Fig. 2 we just want to clearly show that L1 replacement policies directly affect the STT-MRAM L2 cache WER.

While our observation shows that utilizing the MRU replacement policy in L1 data cache provides the most suitable write failure rate for STT-MRAM caches, there is another important consideration that should be taken for selecting a proper replacement policy in L1 data cache, which is *performance efficiency*. Accordingly, apart from the HW ratio, we have also investigated the effect of the four aforementioned replacement policies on the Instruction Per Cycle (IPC) of system when they are chosen as the L1 data cache replacement policy.

Fig. 3 depicts the IPC of system at different configurations, normalized to the case that LRU is chosen as L1 cache replacement policy. As it can be seen in Fig. 3 while MRU as L1 replacement policy, provides the most suitable configuration for a system in term of STT-MRAM L2 cache write failure rate, it noticeably degrades performance of the system. This is due to the fact that MRU always evicts the last used block which means it violates the locality of references phenomenon in the programs. Indeed, it is illustrated in the Fig. 3 that selecting MRU as the L1 data cache replacement policy leads to the worst system performance and compared to the case that LRU is selected as the L1 replacement policy, IPC is reduced by 12.5%, on average, which is a significant amount of reduction in system performance.

According to our observations, none of the aforementioned four replacement policies were able to considerably reduce the HW ratio of incoming write request and keep the system performance high at the same time. To this end, in this paper we are going to propose ROCKY; new replacement policies to alleviate the write failure challenge of STT-MRAMs while keep the performance of system as high as possible. It should be noted that, we observed selecting LRU policy as replacement policy for both level one and level two caches, always leads to the best cache performance. Therefore, in the rest of this paper and during our experiments we will take the case that replacement policies of both L1 and L2 are set to LRU as the baseline and compare ROCKY with this configuration.

3 The ROCKY Approach

In this section we will introduce ROCKY, a robust architecture for on-chip cache memory hierarchy that utilizes STT-MRAM in its design. Considering an STT-MRAM cache, ROCKY suggests a set of replacement policies for both the STT-MRAM cache and the upper-level cache to handle the WER of the STT-MRAM cache and accordingly increases the reliability of the on-chip cache hierarchy. In the following we call the upper-level cache as Leading Cache (LC) and the STT-MRAM one as STT-MRAM Cache (SC).

In Section 2 we observed that the LC (L1 data cache) replacement policy plays an important role in issuing write requests to the SC (L2 cache). Furthermore, we saw that the HW of the incoming block is the main contributor of the WER in STT-MRAMs. Thus, at the first step to control the WER of SC, we need to modify the typical replacement policy of LC considering two constraints:

1) The overall number of write operations that are imposed by LC to SC should be kept as low as possible.
2) The HW of the incoming blocks during the write operations that are imposed by LC to SC should be potentially as low as possible.

From the first constraint point of view, the number of write operations in SC is inversely proportional to the hit ratio of LC. The more hit ratio of the LC, the less write operations are imposed to SC. From many years ago, the LRU replacement policy has been the first choice of the system designers to achieve the most efficient hit ratio in the computer systems [27]. Accordingly, LRU would be a tough competitor and we will have not enough room to improve the hit ratio of LRU in our new LC replacement policy to decrease the number of SC write operations. Thus, we should just be careful to provide a comparable hit ratio in our new LC replacement policy in comparison with LRU.

Unlike the good performance of LRU in restricting the number of imposed write operations from LC to SC, we observed its challenging performance from the HW perspective in Fig. 2. Accordingly, here we observe significant room to decrease the HW of the incoming blocks write requests that arrive at SC from LC.

In ROCKY we introduce a new replacement policy for LC cache which tries to decrease the overall HW ratio of data blocks in SC cache. To this end, ROCKY LC replacement policy augments the HW parameter with the traditional parameters used in LRU to conduct both HW efficiency and performance efficiency in a system that utilizes SC on-chip memory.

Fig. 4, depicts the ROCKY architecture. In the left side of Fig. 4 we can see the abstraction view of the cache memory hierarchy of a system and the target LC and SC positions which are considered as level (i-1) and level (i) caches, respectively. It should be mentioned that ROCKY can be applied at any cache level with the *i* value above 2. Furthermore, it is worthy to mention that we considered LC and SC to be implemented by SRAM and hybrid STT-MRAM-SRAM technologies, respectively. Accordingly, as depicted in Fig. 4, ROCKY's SC cache is constructed as a Non-Uniform Cache Architecture (NUCA). Access to different parts of this cache may be handled by the same methods that are considered for the NUCA caches like the one mentioned in [28].

The goal of proposing a hybrid cache which is composed of STT-MRAM and SRAM technologies is to benefit from STT-MRAM's high scalability and low leakage power while increasing the reliability of cache by employing SRAM as part of the SC cache memory.

To control the HW ratio of incoming data blocks that are imposed to SC by LC, we modified the typical architecture of LC in Fig. 4. Three main modifications that should be applied in LC are: 1) adding a HW counter module, 2) adding the HW flag bit, and 3) updating the cache controller. In the following, we introduce the details of the modified parts of LC.

■ **HW counter:** This module is responsible for counting the HW of each block that arrives at LC and providing the cache controller with this information. The details of the employed HW counter unit can be found in [29].

■ **HW flag:** We modified one of the LRU flag bits in tag array of LC to keep the basic information about the HW of each block. This flag provides the ROCKY cache controller with the required information about the HW of blocks.

■ **Cache controller:** This module is responsible for implementing the ROCKY replacement policy in the LC, based on the information provided by the HW counter and other controlling flags in tag array including HW flag.

Algorithm 1 depicts the ROCKY LC controller and replacement policy procedure that is called after each access to the LC cache. The inputs of algorithm 1 are *miss_signal* which shows the availability of the requested block in LC (1: miss and 0: hit), *write_signal* which shows if the recently arrived request at LC was write (1) or not (0), *new_block* which includes the content of the new incoming block that should be written in the LC, *HW* of the new incoming block and *target_set* that indicates target set of the requested block. In case of miss occurrence and the *victim_block* being dirty, the outputs of algorithm 1 are *write_back_req* and *target_block*. Otherwise, if a hit occurs or the *target_block* is not dirty, the output of the ROCKY LC algorithm is *null* and it does not need to perform a write-back operation.

If the request that is arrived at LC is a missed request, at the first step algorithm 1 tries to find the highest value of *M:HW* among the blocks in the target set (line 5). The Most Significant Bit (MSB) of *M:HW* value, i.e., *M* shows the characteristic of each block from the recently usage perspective. As it is depicted in Fig. 4, ROCKY keeps this information for each block of LC in the provided *M* bit flag. The '1' value in this flag shows that this block has not been accessed recently and if '0' value is stored in the flag, it means this block has been used lately. Note that the mechanism which is used to work with *M* flag is exactly the same as traditional LRU mechanism.

On the other hand, the Least Significant Bit (LSB) of *M:HW* value, i.e., *HW* represents the characteristic of each block from the HW point of view. The '0' value in this flag shows that this block may impose high $0 \rightarrow 1$ transitions if it is evicted from LC to SC and the '1' value in this flag indicates that this block will not impose considerable $0 \rightarrow 1$ transitions in case of eviction from LC to SC.

Algorithm 1: ROCKY LC controller and replacement policy algorithm.

```
1  Input: miss_signal, write_signal, new_block, HW, target_set
2  Output: write_back_req, victim_block
3  if $miss\_signal$ then
4      ▷ A miss occurred in LC
5      $M:HW \leftarrow$ Max($target\_set_{M:HW}$);
6      for All the Blocks in $target\_set$ do
7          if $M:HW == block^i_{M:HW}$ then
8              $victim\_list \leftarrow$ AddToList($block_i$);
9          end
10     end
11     $target\_block \leftarrow$ RandomEvict($victim\_list$);
12     $victim\_block \leftarrow target\_block$;
13     $target\_block \leftarrow new\_block$;
14     ▷ Updating M:HW
15     if $HW \leq HW_{thr}$ then
16         $target\_block_{M:HW} \leftarrow 0b01$;
17     else
18         $target\_block_{M:HW} \leftarrow 0b00$;
19     end
20     if $victim\_block_{dirty}$ then
21         return write_back_req, victim_block;
22     else
23         return $null$
24     end
25 else
26     ▷ A hit occurred in LC    ▷ Updating M:HW
27     for All the Blocks in $target\_set$ do
28         if $block^i \neq target\_block$ then
29             $block^i_{M:HW} \leftarrow block^i_{M:HW}$ OR $0b10$;
30         else
31             $block^i_{M:HW} \leftarrow block^i_{M:HW}$ AND $0b01$;
32         end
33     end
34     if $write\_signal$ then
35         if $HW \leq HW_{thr}$ then
36             $target\_block_{M:HW} \leftarrow 0b01$;
37         else
38             $target\_block_{M:HW} \leftarrow 0b00$;
39         end
40     end
41     return null;
42 end
```

Accordingly, for each block in LC, the higher the *M:HW* value, the better candidate it is for eviction in ROCKY LC algorithm. After finding the maximum value of *M:HW* among the blocks in *target_set* (line 5), ROCKY constructs a *victim_list* in line 6 to 10 of Algorithm 1. ROCKY needs this list since in many times there are more than one block in a *target_set* that have maximum *M:HW* value. Each one of the blocks in the *victim_list* would be a good candidate for eviction since their possible eviction will not threaten system neither from performance point of view nor from HW point of view. This is because '0' is stored in the most frequently used block's *M* flag, and *HW* flag of the block whose HW is greater than the threshold is also '0'.

To introduce a victim block in algorithm 1, ROCKY randomly selects a victim block from the constructed *victim_list* (line 11). Then, the contents of the *target_block* is stored on a buffer for a possible write back to SC (line 12). The *new_block* will be stored in the *target_block* in line 13 and its *M:HW* flags will be updated in line 15 to 19 of algorithm 1. To this end, ROCKY makes the *M* flag '0' to show that this block is recently used. For updating the *HW* flag, we

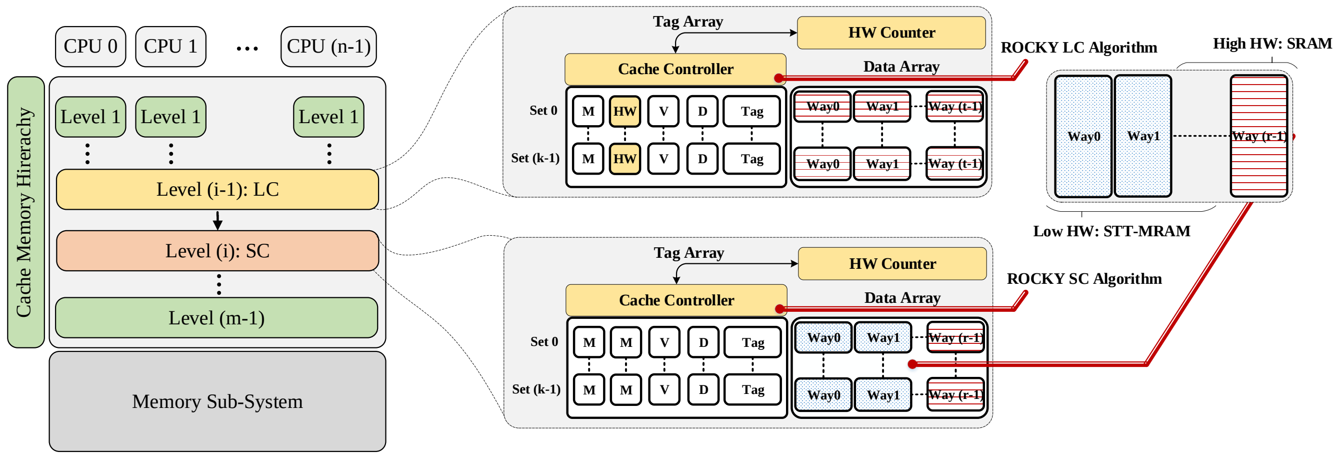

Fig. 4: Rocky Architecture

considered a HW$_{thr}$ in ROCKY and compare the HW of each block with this value. HW$_{thr}$ can be determined with a set of profiling analysis. If the calculated HW is greater than HW$_{thr}$, ROCKY will store '0' in *HW* flag to keep this block in the LC as much as possible (line 15 to 16). Otherwise, ROCKY stores '1' in *HW* flag to provide better possibility for this block to become a victim candidate in future (line 17 to 19). Finally, to finish a miss occurrence process, if the *victim_block* is dirty, a *write_back_req* signal and the contents that should be written back will be returned in line 20 to 22, otherwise if the *victim_block* is not dirty, *null* will be returned (line 22 to 24).

For a hit occurrence in LC, Algorithm 1 should update the *M:HW* flags of the *target_set* so that the future decisions on evicting blocks will be feasible. Accordingly, to update *M* flag of the LC blocks in this set ROCKY makes the *M* flag of the hit block '0' to show that this block is used recently and makes the other blocks' *M* flags '1' to demonstrate that they are not accessed recently (line 27 to 33). Finally, to update the *HW* flag of the hit block for a write request, ROCKY compares the calculated HW of the *new_block* with HW$_{thr}$ and updates the *HW* flag correspondingly (line 34 to 40).

As explained, in ROCKY, we intend to decrease the WER in SC cache. Although, the changes in LC architecture and replacement policy contributes in reducing the WER in SC cache, there might be still some high HW data blocks which arrive at SC from lower level memories or from LC cache in situations that HW of all candidate blocks in *victim_list* of algorithm 1 are more than HW$_{thr}$.

In order to make the SC cache as robust against write failures as possible, we have made a number of changes in the architecture and replacement policy of the SC cache, As shown in Fig. 4. The major adjustments in SC cache include: 1) HW counter, 2) cache controller, 3) a hybrid cache composed of STT-MRAM and SRAM technologies. The details of each adjustment is explained in the following.

■ **HW counter:** Similar to LC cache architecture, this unit is employed in SC cache to calculate the HW of incoming data blocks and updates the cache controller with this information.

■ **Cache controller:** This module is responsible for redirecting the traffics between SRAM and STT-MRAM parts of the SC cache based on the information retrieved from HW counter. To this end we implemented Algorithm 2 in this module.

■ **Hybrid Cache:** To handle the high WER of incoming blocks with HW higher than HW$_{thr}$, we suggested a hybrid SC architecture.

As it is shown in Fig. 4, in the ROCKY, STT-MRAMs cover a large fraction of SC while SRAMs constitute a small part of it. The reason for using SRAM technology as part of SC cache is that SRAMs are robust against write failure. Therefore, allocating a small part of each set to SRAM technology, gives us an opportunity to greatly reduce the WER for high HW blocks in SC cache, while we can still benefit from major advantages of STT-MRAMs over SRAMs, i.e., low leakage power and high density.

It is noteworthy to mention that, since in most of the cases, ROCKY LC cache replacement policy evicts the blocks with a HW of less than HW$_{thr}$, allocating just a few ways of each set to SRAM memory is enough to cover the blocks whose HW are more than HW$_{thr}$.

Algorithm 2 illustrates the ROCKY SC replacement policy which is triggered by the cache controller as soon as a new data block arrives at SC cache. The inputs and outputs of the SC algorithm is the same as LC algorithm, except that two new variables are added to the inputs, i.e., STT-MRAM_ways and SRAM_ways which represent the physical locations of STT-MRAM and SRAM based ways of the SC cache, respectively. Since STT-MRAM is vulnerable to high HW write operations, all the blocks with low HW are mapped to the STT-MRAM based ways of the SC, while high HW blocks are written in the ways which are made of SRAM technology.

When a new data block arrives to be written in SC as a read miss from Level (i+1) cache memory or memory sub-system in Fig. 4 (line 3 to 13), HW of the new block is compared to the HW$_{thr}$. If the new block's HW is less than the threshold, the LRU replacement policy will be triggered by the cache controller to find the target block's location from the STT-MRAM_$ways$ and stores it in $target\_block\_loc$ (line 5 to 6). Otherwise, the LRU replacement policy up-

Algorithm 2: ROCKY SC controller and replacement policy algorithm.

```
Input: miss_signal, write_signal, new_block, HW, target_set, STT-MRAM_ways, SRAM_ways
Output: write_back_req, victim_block
if $miss\_signal$ then
    ▷ A miss occurred in SC
    if $HW \leq HW_{thr}$ then
        $target\_block\_loc \leftarrow$ LRUEvict($[target\_set]$[STT-MRAM_$ways$]);
    else
        $target\_block\_loc \leftarrow$ LRUEvict($[target\_set]$[SRAM_$ways$]);
    end
    $victim\_block \leftarrow target\_block$;
    $target\_block \leftarrow new\_block$;
    if $victim\_block_{dirty}$ then
        return write_back_req, victim_block;
    else
        return $null$
    end
else
    ▷ A hit occurred in SC
    if $write\_signal$ then
        if $HW \leq HW_{thr}$ AND FoundedWay($new\_block$, $STT\text{-}MRAM\_ways$) then
            return null;
        else if $HW > HW_{thr}$ AND FoundedWay($new\_block$, $SRAM\_ways$) then
            return null;
        else
            if $HW \leq HW_{thr}$ then
                $target\_block \leftarrow$ LRUEvict($[target\_set]$[STT-MRAM_$ways$]);
            else
                $target\_block \leftarrow$ LRUEvict($[target\_set]$[SRAM_$ways$]);
            end
            $victim\_block \leftarrow target\_block$;
            $target\_block \leftarrow new\_block$;
            Invalidate($new\_block\_loc$);
            if $victim\_block_{dirty}$ then
                return write_back_req, victim_block;
            else
                return $null$
            end
        end
    end
end
```

dates $target\_block\_loc$ with the location of a block from the SRAM_$ways$ (line 7 to 9).

In lines 10 to 12, the contents of the target block is written into a write buffer, called $victim\_block$. Then $target\_block$ is updated with the new_block and finally the $write\_back\_req$ signal and $victim\_block$ are sent to the lower level memory if the *victim_block* is dirty (line 12 to 14), or *null* will be returned if the *victim_block* is not dirty (line 14 to 16). When a write-back comes from LC, it is either a write-back miss or a write-back hit. If write-back miss occurs it will be handled similar to a read miss as explained, but a write-back hit must be handled more carefully.

On a write-back hit, ROCKY should check the HW of the updated block which is going to be written in SC to avoid any HW write violations according to ROCKY SC algorithm rules. To this end, ROCKY first checks the HW boundaries of the hit block's way with the updated block's HW to find out if it can perform the write operation in SC or not. If the updated block's HW still obeys the STT-MRAM_ways or SRAM_ways HW threshold (line 20 to 24) ROCKY does not need any block movement and it should not victim any block.

On the other hand, if the updated block's HW violates the ROCKY STT-MRAM_ways or SRAM_ways HW threshold, ROCKY should evict a block from an appropriate place in SC to free a new space for the hit block and invalidate the previous location of the hit block in SC for future usage (line 25 to 38).

TABLE 1: Simulations Configurations

Component	Parameter	Value
Processor	Architecture	ARM
	Frequency	1GHz
	Number of Cores	4
	Type	Homogeneous
	Model	detailed
	Warmup Cycles	100 millions cycles
L1 Cache	Memory Technology	SRAM
	Size	32KB
	Associativity	4
	Block Size	64B
	Access Time	2 cycles
	Replacement Policy	ROCKY LC
	Energy per Access	0.240 nJ
L2 Cache (Hybrid)	Memory Technology	STT-MRAM+SRAM
	Size	4MB (3584KB +512KB)
	Associativity	16 (14 way + 2 way)
	Block Size	64B
	STT-MRAM Access Time	Write 20 and Read 5 cycles
	SRAM Access Time	5 cycles
	Replacement Policy	ROCKY SC
	STT-MRAM Energy per Read	0.210 nJ
	STT-MRAM Energy per Write	1.01 nJ
	SRAM Energy per Access	0.240 nJ
Main Memory	Memory Technology	DRAM DDR3
	Size	4GB
	Access Time	100 cycles

4 System Setup and Results

In this section, we first introduce the simulation setup which is used to evaluate ROCKY. Next, we will discuss the evaluation results from the reliability, performance, energy consumption, and area perspectives.

4.1 Experimental Setup

We have used gem5 cycle-accurate simulator [19] to evaluate ROCKY. To model the latency, area, dynamic and static energy consumption of both STT-MRAM and SRAM memories, NVSim [30] is used. We also synthesized the employed peripheral circuits in Synopsis Design Compiler® [31] to extract the information about their latency, area and energy consumption. The 45nm technology library is considered in both NVSim and Synopsis Design Compiler®. The detailed mode of ARM processor provided by gem5 is selected through this study. During our experiments we considered L1 cache as ROCKY's LC and L2 shared cache as ROCKY's SC. The details of the simulations configuration are outlined in Table 1. As can be seen in Table 1, we considered a NUCA L2 cache for our evaluations. In our simulations, the read access latencies of SRAM and STT-MRAM parts are the same, while the STT-MRAM part imposes noticeable write access time overhead compared to the SRAM part.

Ten different combinations as multi-programmed workloads are chosen from SPEC CPU2006 benchmarks suite [20]. Table 2 includes the details of all combinations used in the simulations. To find-out the HW_{thr} value discussed in Section 3, we conducted a set of profiling experiments. Fig.

TABLE 2: SPEC CPU2006 Benchmarks Combinations as Multi-Programmed Workloads in the Evaluation of a Quad-Core Processor

Workloads	**Benchmarks (Input) (Size)**			
	Core1	**Core2**	**Core3**	**Core4**
Comb1	perlbench (checkspam.pl) (8.4 KB)	bzip2 (control) (98 B)	mcf (inp.in) (3.3 MB)	soplex (ref.mps) (280 MB)
Comb2	perlbench (checkspam.pl) (8.4 KB)	bzip2 (control) (98 B)	omnetpp (omnet.ini) (2.7 KB)	xalancbmk (t5.xml) (59 MB)
Comb3	perlbench (checkspam.pl) (8.4 KB)	mcf (inp.in) (3.3 MB)	omnetpp (omnet.ini) (2.7 KB)	xalancbmk (t5.xml) (59 MB)
Comb4	bzip2 (control) (98 B)	mcf (inp.in) (3.3 MB)	soplex (ref.mps) (280 MB)	xalancbmk (t5.xml) (59 MB)
Comb5	gcc (200.in) (1.8 MB)	bwaves (bwaves.in) (212 B)	mcf (inp.in) (3.3 MB)	cactusADM (benchADM.par) (1KB)
Comb6	namd (namd.input) (7.6 MB)	dealII (-) (-)	soplex (ref.mps) (280 MB)	hmmer (retro.hmm) (38 KB)
Comb7	perlbench (checkspam.pl) (8.4 KB)	gcc (200.in) (1.8 MB)	mcf (inp.in) (3.3 MB)	namd (namd.input) (7.6 MB)
Comb8	perlbench (checkspam.pl) (8.4 KB)	namd (namd.input) (7.6 MB)	soplex (ref.mps) (280 MB)	xalancbmk (t5.xml) (59 MB)
Comb9	bwaves (bwaves.in) (212 B)	dealII (-) (-)	namd (namd.input) (7.6 MB)	lbm (100_100_130_ldc.of) (1.3 MB)
Comb10	gcc (200.in) (1.8 MB)	bwaves (bwaves.in) (212 B)	soplex (ref.mps) (280 MB)	xalancbmk (t5.xml) (59 MB)

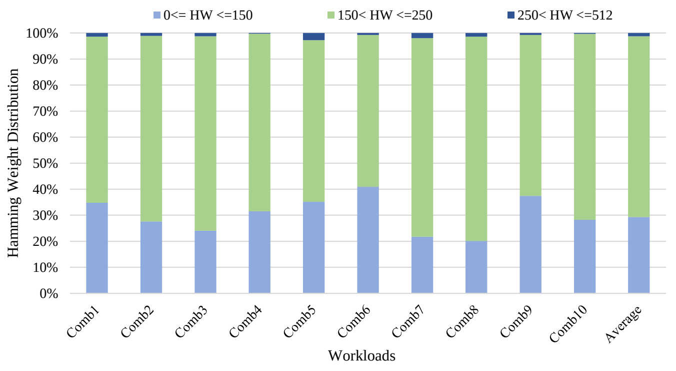

Fig. 5: HW Distribution of the Incoming Blocks Write Operations in L2 cache

5 depicts the HW distribution of the workloads introduced in Table 2.

The HW_{thr} can be considered as a knob in ROCKY. To find-out the proper value for the HW_{thr}, designers should perform a set of profiling evaluations. During this profiling we find-out that choosing high HW_{thr} leads to high write error-rate in L2 cache since most of the writes will serve from the STT-MRAM part of the L2 cache and SRAM part of L2 cache is not appropriately utilized. On the other hand, choosing the low HW_{thr} leads to high L1 cache miss since the in L1 cache the ROCKY algorithm tries to keep the blocks with higher than HW_{thr}. Accordingly, the victim select-ability of L1 cache according to HW will be reduced and the blocks are evicted more randomly which leads to high miss rate of L1 cache and noticeable performance penalty. Accordingly, we have a trade-off here selecting a suitable value for HW_{thr} which simultaneously keeps the WER low and performance high.

According to Fig. 5, for the average of all combinations, HW of a large fraction of write operation are in the range of 150 up to 250 which shows they are the dominant contributors in write operations and normally WER. Accordingly, choosing HW_{thr} in this interval would be more effective since a huge amount of write operations would be affected. To this end, our profiling results confirmed that 180 would be the most efficient value for HW_{thr}.

After tuning the HW_{thr}, we should decide on the number of SRAM and STT-MRAM ways in the architecture of ROCKY. To this end, we should perform a set of profiling experiments to find-out the appropriate number of SRAM ways in ROCKY's hybrid SRAM/STT-MRAM SC cache. Please note that putting extra SRAM ways in ROCKY's SC hybrid cache may lead to under-utilization of SRAM parts and impose noticeable static energy consumption to the design. On the other hand, putting less than required SRAM ways in the ROCKY's SC hybrid cache may lead to frequent misses in the SRAM ways and significantly affect the performance of the system.

4.2 Results

In the following we first discuss the effects of the ROCKY on L2 cache WER, then the results are discussed from performance, energy consumption and area perspectives. We compared all the obtained results from ROCKY with two configurations:

1) Baseline configuration in which the cache hierarchy does not benefit from any write failure reduction technique and both L1 and L2 caches are equipped with LRU replacement policy. In this configuration the L1 cache implemented with pure SRAM technology and the L2 cache implemented with pure STT-MRAM technology.
2) LER [14] as a state-of-the-art STT-MRAM cache replacement policy which tries to place the new incoming block in a cache line that causes the least $0 \rightarrow 1$ transitions. More details on LER can be found in Section 5. In this configuration the L1 cache implemented with pure SRAM technology and the L2 cache implemented with pure STT-MRAM technology.

4.2.1 *Reliability*

We calculated the HW ratio of the incoming data blocks in STT-MRAM based part of ROCKY-equipped L2 cache. Fig. 6 depicts the HW ratios normalized to the baseline. Considering baseline configuration, Fig. 6 illustrates that ROCKY reduces the HW ratio of the incoming data blocks by about 31.3%, on average. Since LER only decreases the $0 \rightarrow 1$ transitions and does not have any control on number of ones in the incoming block, the HW ratio of incoming data blocks that arrive at L2 cache is not affected by LER and stays the same as baseline.

Fig. 7, shows the total number of $0 \rightarrow 1$ transitions normalized to the baseline. According to Fig 7, ROCKY reduces the number of $0 \rightarrow 1$ transitions by 30.5%, on average, while LER reduces the number of $0 \rightarrow 1$ transitions by only 19%, on average. Although we mentioned that the probability of write failure in $0 \rightarrow 1$ transitions is much more than the probability of write failure in $1 \rightarrow 0$ transitions,

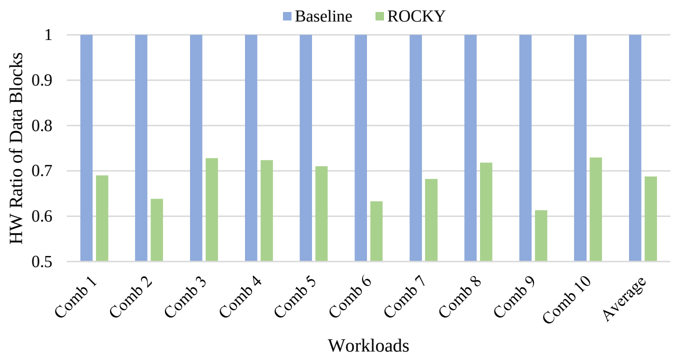

Fig. 6: HW ratio of L2 cache, normalized to the baseline.

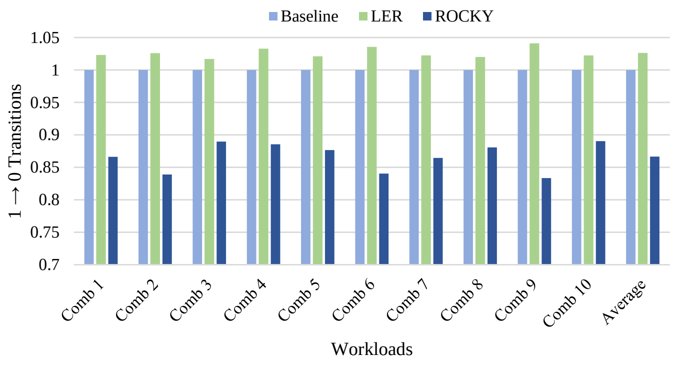

Fig. 8: $1 \rightarrow 0$ transitions in L2 cache, normalized to the baseline.

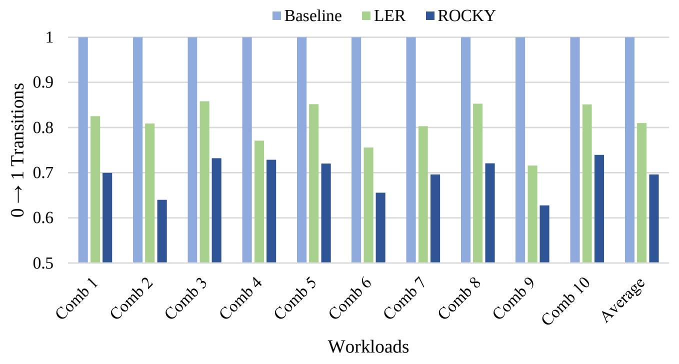

Fig. 7: $0 \rightarrow 1$ transitions in L2 cache, normalized to the baseline.

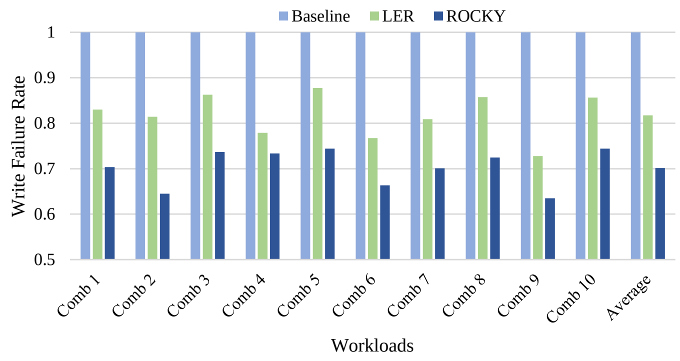

Fig. 9: WER in L2 cache, normalized to the baseline.

to compute the final WER more precisely, we also counted the number of $1 \rightarrow 0$ transitions in L2 cache. As illustrated in Fig. 8, on average, ROCKY reduces the total number of $1 \rightarrow 0$ transitions by 11.7% compared to the baseline, while as a side effect of LER the number of $1 \rightarrow 0$ transitions are increased by 2.6% in comparison with the baseline since LER increases the total number of accesses to L2 cache by its significantly higher miss rate.

In order to calculate the WER in L2 cache, first $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions impact on WER must be discovered. The probability of write failure rate is expressed by the following equation [16]:

$$Pr_{WF} = 1 - (1 - Pr_{bit(0\rightarrow1)})^{N_{0\rightarrow1}}(1 - Pr_{bit(1\rightarrow0)})^{N_{1\rightarrow0}} \quad (4)$$

where $Pr_{bit(0\rightarrow1)}$ and $Pr_{bit(1\rightarrow0)}$ are write failure probabilities of an STT-MRAM cell in $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions, respectively. $N_{0\rightarrow1}$ is the aggregate number of $0 \rightarrow 1$ transitions and $N_{1\rightarrow0}$ is the aggregate number of $1 \rightarrow 0$ transitions. Please note that since STT-MRAM is not yet a mature technology, it is hard to judge its WERs (or $Pr_{bit(0\rightarrow1)}$ and $Pr_{bit(1\rightarrow0)}$). Considering SRAMs would be replaced by STT-MRAMs, in ROCKY we assume that the base WER is no more than 10^{-9}, which appears feasible for SRAM based on studies such as [32], [33]. For the sake of calculations and to keep the two order of magnitude difference between $Pr_{bit(0\rightarrow1)}$ and $Pr_{bit(1\rightarrow0)}$ as considered in [15], [17], we assumed 10^{-7} as the nominal WER for $0 \rightarrow 1$ transitions and 10^{-9} as the nominal WER for $1 \rightarrow 0$ transitions.

Fig. 9, demonstrates the impact of ROCKY and LER on the WER for all workloads normalized to the baseline. As it can be seen, ROCKY outperforms LER and reduces the WER by about 28.7%, on average, while the error rate reduction gained by LER is 17.5%, on average.

To clarify the contribution of ROCKY in reducing the WER lets consider an example. Assume that the WER of a STT-MRAM cache is 7×10^{-2}. It means that from each 100 write operations, seven of them have suffered from write errors. Considering 28.7% WER improvement in ROCKY, the number of write errors can be improved to five from each 100 write operations. Please note that, as also mentioned in [8], [11], [12], WER is an important reliability challenges for STT-MRAMs and there is not a unique approach that can completely resolve this challenge. Thus, different approaches should be coupled with each others to effectively reduce it. In this regard, as one of the promising error protection technique, ROCKY can easily coupled with other approaches like ECCs and even better the WER significantly.

It must be noted that in our approach, the write current that is applied for write operation is strong enough to satisfy the reliability constraints and our method contributes to the improvement of the reliability of L2 cache memory by reducing the probability of write failure.

4.2.2 *Performance*

Due to increase in cache miss rate, ROCKY imposes performance overhead to system. The main reasons behind increase in cache miss rate are the changes in replacement policies of L1 cache and L2 cache. Since ROCKY L1 cache replacement policy always tries to evict the block with a HW

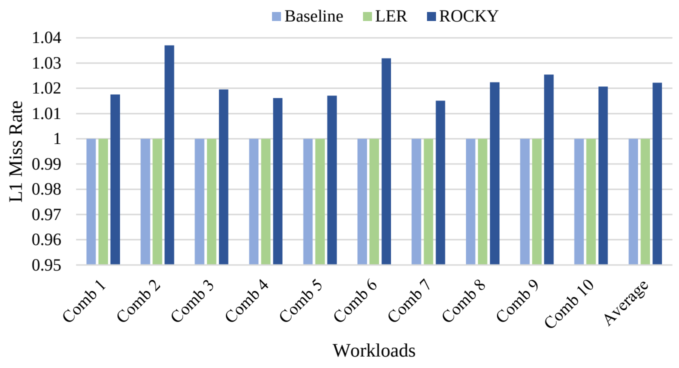

Fig. 10: Miss rate of L1 cache, normalized to the baseline.

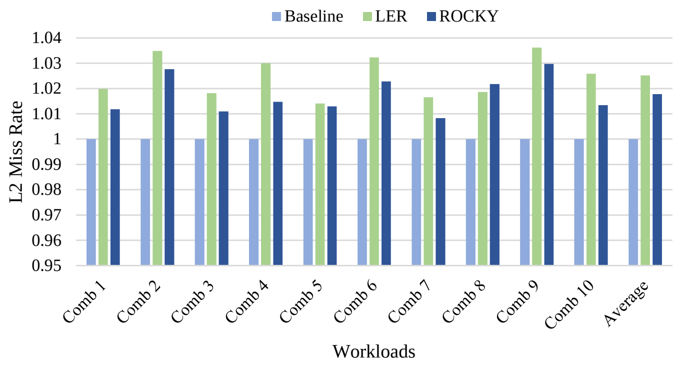

Fig. 11: Miss rate of L2 cache, normalized to the baseline.

of less than 180 which is not necessarily the least recently used block, it might have a negative effect on cache miss rate considering LRU replacement policy as the baseline. Partitioning L2 cache is another reason for cache miss rate increase in ROCKY. Because of allocating just two ways for blocks whose HW are more than 180, the LRU replacement policy for the SRAM_$ways$ in Algorithm 2 has limited options for block eviction which might lead to higher cache miss rate.

Fig. 10 and Fig. 11 depicts the miss rates of L1 cache and L2 cache for three configurations including ROCKY. Considering the L1 cache, as it can be seen in Fig. 10, LER and baseline have the same miss rate since the L1 cache replacement policies in both of them are LRU. On the other hand, since the replacement policy of ROCKY in L1 cache is modified it imposes about 2.2% miss rate overhead to the system, on average. For the L2 cache, as it can be seen in Fig. 11, both LER and ROCKY impose noticeable miss rate overhead to the system since both of them use their specific replacement policies in L2 cache rather than traditional LRU which is used in baseline. Indeed, on average, LER and ROCKY impose about 2.4% and 1.8% miss rate overheads to the system comparing with the baseline.

We have chosen IPC as the metric for showing system performance. Fig. 12, shows the IPC overhead after applying ROCKY normalized to the baseline. As explained above, ROCKY increases the cache miss rate of the workloads. Accordingly, as shown in Fig. 12, these increases in the miss rates of the workloads lead to IPC overhead of less than 0.9%, on average. Considering LER, besides its higher miss rate comparing with baseline, since it requires several extra

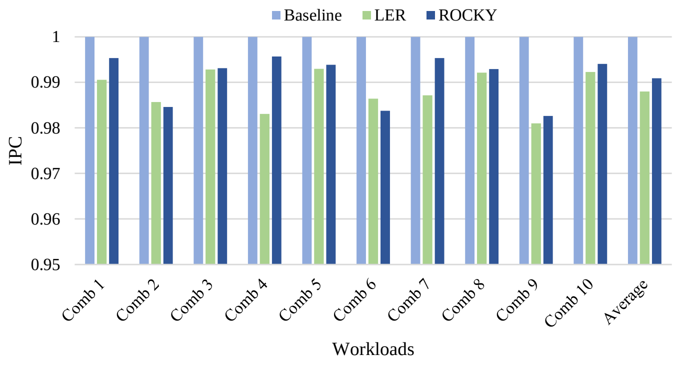

Fig. 12: Normalized IPC to the baseline.

read operations prior to each block eviction in the L2 cache, it imposes 1.2% IPC overhead to the system, on average.

It is noteworthy to mention that, although allocating more than two ways to SRAM in the L2 hybrid cache might help reliability and performance of the system, since SRAM memories suffer from high leakage power consumption and low scalability issues, it will lead to significant area and static power overhead.

4.2.3 *Energy Consumption*

In ROCKY there are three key factors that affect the dynamic energy consumption: 1) the increase in number of cache misses causes more accesses to L2 cache for read and write operations which in turn results in extra dynamic energy consumption, 2) the employed HW counter unit also slightly increases the total dynamic energy consumption, 3) on the other hand, we allocated two SRAM ways for ROCKY L2 cache which imposes less dynamic energy consumption comparing with pure STT-MRAM baseline.

Fig. 13 shows the dynamic energy consumption for workloads after applying ROCKY. Our simulation results show that the third mentioned factor dominates the other two and as it is shown in Fig. 13, in comparison with baseline, ROCKY not only does not impose energy overhead to the system, but also decreases the overall dynamic energy consumption by 3.55%, on average. As mentioned before, in LER, before evicting the block from L2 cache, contents of all ways in a set are read in order to find the block which incurs minimum transitions. Accordingly, LER increases the total dynamic energy consumption by 1.9%, on average, comparing with baseline.

In ROCKY, the downside of allocating two ways of L2 cache to SRAM technology is the higher leakage power compared to the baseline and LER, where all the ways of L2 cache are made of STT-MRAM technology. As illustrated in Fig. 14, ROCKY increases the total static energy by 1.3%, on average, while LER increases the total static energy consumption of L2 cache by only 0.2%, on average. It is noteworthy to say that, the HW counter's effect on static energy consumption of both L1 and L2 caches is totally negligible, while it increases the dynamic energy consumption of L1 cache by 0.3%.

4.2.4 *Area*

We have employed almost the same peripheral circuits in L1 cache since we have used the LRU controlling flags in

ROCKY. The only extra element in L1 cache is the HW counter which imposes a negligible amount of 0.25% area overhead to L1 cache. Allocating two ways to SRAM in L2 cache, increases this cache's area by 2.1% while the HW counter unit only imposes a minor amount of 0.08% area overhead to this cache. On aggregate, the total area of L2 cache is increased by 2.18% compared to the baseline.

5 Related Work

There have been several studies in order to reduce the WER in STT-MRAM caches. There are different approaches at different levels of abstraction to improve the WER of STT-MRAMs. In some approaches, the physical characteristics of STT-MRAM cells are manipulated to trade off different properties of standard STT-MRAM cells, i.e., retention, performance, energy consumption to improve the WER [34], [35], [36], [37]. Considering ROCKY, which is proposed in this paper, it can be coupled with any of these approaches to further improve the WER of the STT-MRAM caches.

Among all the approaches at different levels of abstraction, focusing on the architectural-level where ROCKY proposed, employing ECCs is a very prevalent method to keep the error rate low in STT-MRAM caches [11], [38], [39]. However, the facts that WER has an asymmetric behavior and data contents of cache blocks play a major role in WER, have not been considered in most of them [4], [7], [25], [40], [41], [42]. Since the majority of data bits that are written in cache blocks are zeros, large portion of the caches are not needed to be protected by strong ECCs, therefore employing ECCs with constant high protection power which is only useful for the worst-case scenario (most of the incoming data bits are ones) imposes energy, area and performance overhead.

In the study presented in [4], it has been stated that write operations are one of the main sources of heat accumulation and temperature increase in cache memories. This in turn leads to significant increase in error rate. To alleviate the negative effect of temperature on error rate, Cheshmikhani et al. [4] have proposed Thermal-Aware Least-Recently Written (TA-LRW) replacement policy for STT-MRAM caches. Given that high percentage of consecutive write operations are mapped to neighbouring cache blocks which may result in temperature augmentation, TA-LRW maps successive write operations to non-neighbouring cache blocks in a way that each two successive write operations are mapped to cache blocks with at least three blocks distance. By using this replacement policy, TA-LRW dispense the heat all over the cache blocks which leads to lower error rate. Overlooking the effect of data contents of the cache in write operations may impose energy and performance overhead.

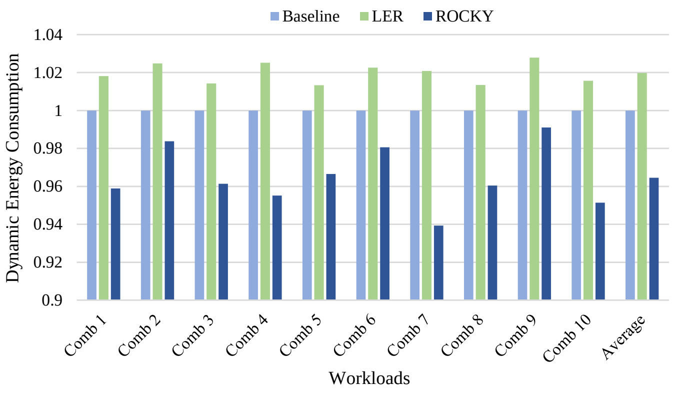

Fig. 13: Dynamic energy consumption of L2 cache, normalized to the baseline

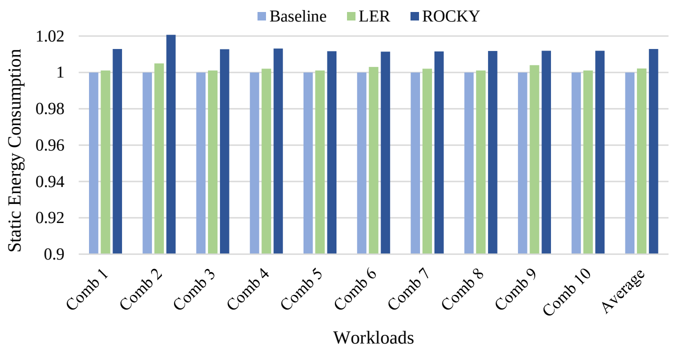

Fig. 14: Static energy consumption of L2 cache, normalized to the baseline.

Some of the studies have tried to improve the reliability of STT-MRAM caches by changing the amplitude or width of the applied current [9], [25]. The study in [9], shows that in order to complete a write process without any error, the time margin for write operation must be up to 20X more than the average switching latency. This problem is aggravated when effects of process variation is taken into account as well, and the margin could be further increased by up to 40 times of average switching latency. To tackle this problem, a second controller is employed inside the memory array which works by handshaking protocol. Every time a data block is sent to be written in memory, the controller inside the memory array, monitors the memory constantly to see whether all data bits of the block are written correctly. The main controller cuts the write signal off, as soon as the data block is written in memory flawlessly and the ack signal is sent to it by the memory array. This approach complicates the design and imposes area overhead to the system due to using an extra controller.

One of the common methods is to shorten the width of write pulses and performing the write operation in multiple steps [43]. In each step a short write pulse is applied to a cache cell and then content of the cell is read, if the magnetization direction of the free layer is not changed the whole process is repeated. Due to excessive read operations these techniques increase read disturbance error rate. On the other hand, [10] and [44] have proposed increasing the amplitude of write current to overcome the write failure problem. In this circuit-level scheme the width of the access transistor is augmented to improve the current driving ability of the access transistor. Consuming extra energy and enhancing the probability of oxide breakdown are side effects of this scheme.

In [6], [14], [15], [45], [46] the asymmetric behavior of write failure have been taken into consideration. As mentioned in Section 4, Monazzah et al. [14] have proposed a replacement algorithm, named LER to reduce the WER in STT-MRAM caches. This algorithm, at first calculates the Hamming Distance (HD) of all cache lines of a set by comparing the contents of current cache lines with the incoming block. The line which has minimum HD is replaced by the new block. Because of several read operations before each write, this method causes energy overhead and increases the probability of read disturbance error.

Another protection technique, called Adaptive Way Allocation for Reconfigurable ECCs (AWARE) is suggested by [6]. AWARE employs lightweight ECCs to keep an acceptable level of reliability in all cache lines. In case the number of ones in the upcoming data block are more than a specific threshold the whole cache set is protected by strong ECC and one of the lines of the set is dynamically reconfigured to store the ECC bits. This technique aggravates miss rate due to using one of the cache ways for storing ECC bits.

A similar method to [6] has been proposed in [46] to protect data blocks with non-uniform ECC bits. In this work, called Sliding Basket, cache ways are partitioned into multiple baskets, each one guarded by a different level of ECC protection. Whenever an erroneous data arrives at cache, depending on its error rate, it is mapped to the basket which can satisfy its required level of protection. This method also imposes area overhead to system due to using extra ECC bits for the strong protection scenario.

6 Conclusion

SRAM technology will lose its place in the next generation of the on-chip caches due to its high leakage power and low density. STT-MRAM technology benefits from a number of advantages such as: non-volatility, extremely low leakage power and high integration density which makes it the most suitable candidate for replacing SRAMs in cache memories. Besides its advantages, STT-MRAM suffers from reliability challenges, where the write failure is the most critical one. $0 \rightarrow 1$ transitions in write patterns are the main contributors to the write failure. In order to reduce the number of one bits and subsequently number of $0 \rightarrow 1$ transitions in STT-MRAM cache, we have proposed ROCKY in this paper. Both STT-MRAM cache and its cpu-side neighbouring cache in ROCKY are manipulated in a way that the HW ratio of the incoming data blocks that are written in STT-MRAM cache is reduced so that the write failure rate in STT-MRAM cache is decreased as well, while the system performance is kept as high as possible. Our experiments show that ROCKY reduces the write failure rate by 28.7%, on average. ROCKY also decreases the dynamic energy consumption of SC cache by 3.55%, on average, while it imposes minor overheads to the system from IPC, static energy consumption and area point of view.

References

[1] T. Endoh, H. Koike, S. Ikeda, T. Hanyu, and H. Ohno, “An Overview of Nonvolatile Emerging Memories— Spintronics for Working Memories,” *IEEE JETCAS*, 2016.

[2] E. Chen, D. Apalkov, A. Driskill-Smith, A. Khvalkovskiy, D. Lottis, K. Moon, V. Nikitin, A. Ong, X. Tang, S. Watts, R. Kawakami, M. Krounbi, S. A. Wolf, S. J. Poon, J. W. Lu, A. W. Ghosh, M. Stan, W. Butler, T. M. S. Gupta, C. K. A. Mewes, P. B. Visscher, and R. A. Lukaszew, “Progress and Prospects of Spin Transfer Torque Random Access Memory,” *IEEE TMAG*, 2012.

[3] A. Jadidi, M. Arjomand, and H. Sarbazi-Azad, “High-Endurance and Performance-Efficient Design of Hybrid Cache Architectures through Adaptive Line Replacement,” in *Proc. of ISLPED*, 2011.

[4] E. Cheshmikhani, H. Farbeh, S. G. Miremadi, and H. Asadi, “TA-LRW: A Replacement Policy for Error Rate Reduction in STT-MRAM Caches,” *IEEE TC*, 2019.

[5] W. Kang, Z. Li, K. J.-O, Y. Chen, Y. Zhang, D. Ravelosona, C. Chappert, and W. Zhao, “Variation-Tolerant and Disturbance-Free Sensing Circuit for Deep Nanometer STT-MRAM,” *IEEE TNANO*, 2014.

[6] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi, “AWARE: Adaptive Way Allocation for Reconfigurable ECCs to Protect Write Errors in STT-RAM Caches,” *IEEE TETC*, 2017.

[7] E. Cheshmikhani, H. Farbeh, and H. Asadi, “ROBIN: Incremental Oblique Interleaved ECC for Reliability Improvement in STT-MRAM Caches,” in *Proc. of ASP-DAC*, 2019.

[8] Y. Zhang, W. Wen, and Y. Chen, “STT-RAM Cell Design Considering MTJ Asymmetric Switching,” *SPIN Journal*, 2012.

[9] A. Ahari, M. Ebrahimi, F. Oboril, and M. Tahoori, “Improving Reliability, Performance, and Energy Efficiency of STT-MRAM with Dynamic Write Latency,” in *Proc. of ICCD*, 2015.

[10] E. I. Vatajelu, R. Rodriguez-Montañés, S. Di Carlo, M. Indaco, M. Renovell, P. Prinetto, and J. Figueras, “Power-Aware Voltage Tuning for STT-MRAM Reliability,” in *Proc. of ETS*, 2015.

[11] Z. Azad, H. Farbeh, and A. M. H. Monazzah, “ORIENT: Organized interleaved ECCs for new STT-MRAM caches,” in *Proc. of DATE*, 2018.

[12] X. Bi, Z. Sun, H. Li, and W. Wu, “Probabilistic Design Methodology to Improve Run-time Stability and Performance of STT-RAM Caches,” in *Proc. of ICCAD*, 2012.

[13] Y. Zhang, X. Wang, Y. Li, A. K. Jones, and Y. Chen, “Asymmetry of MTJ Switching and Its Implication to STT-RAM Designs,” in *Proc. of DATE*, 2012.

[14] A. M. H. Monazzah, H. Farbeh, and S. G. Miremadi, “LER: Least-Error-Rate Replacement Algorithm for Emerging STT-RAM Caches,” *IEEE TDMR*, 2016.

[15] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi, “An Efficient Protection Technique for Last Level STT-RAM Caches in Multi-Core Processors,” *IEEE TPDS*, 2017.

[16] E. Aliagha, A. M. H. Monazzah, and H. Farbeh, “REACT: Read-/Write Error Rate Aware Coding Technique for Emerging STT-MRAM Caches,” *IEEE TMAG*, 2019.

[17] W. Wen, M. Mao, X. Zhu, S. H. Kang, D. Wang, and Y. Chen, “CD-ECC: Content-Dependent Error Correction Codes for Combating Asymmetric Nonvolatile Memory Operation Errors,” in *Proc. of ICCAD*, 2013.

[18] S. Mirbagher Ajorpaz, E. Garza, S. Jindal, and D. A. Jiménez, “Exploring Predictive Replacement Policies for Instruction Cache and Branch Target Buffer,” in *Proc. of ISCA*, 2018.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” *ACM SIGARCH Comput. Archit. News*, 2011.

[20] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Comput. Archit. News*, 2006.

[21] L. Zhang, A. Todri, W. Kang, Y. Zhang, L. Torres, Y. Cheng, and W. Zhao, “Quantitative Evaluation of Reliability and Performance for STT-MRAM,” in *Proc. of ISCAS*, 2016.

[22] J. Yang, P. Wang, Y. Zhang, Y. Cheng, W. Zhao, Y. Chen, and H. Li, “Radiation-Induced Soft Error Analysis of STT-MRAM: A Device to Circuit Approach,” *IEEE TCAD*, 2016.

[23] W. Kang, L. Chang, Z. Wang, W. Lv, G. Sun, and W. Zhao, “Pseudo-Differential Sensing Framework for STT-MRAM: A Cross-Layer Perspective,” *IEEE TC*, 2017.

[24] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, “STT-MRAM Scaling and Retention Failure,” *Intel Technology Journal (ITJ)*, 2013.

[25] N. Sayed, M. Ebrahimi, R. Bishnoi, and M. Tahoori, “Opportunistic Write for Fast and Reliable STT-MRAM,” in *Proc. of DATE*, 2017.

[26] Z. Pajouhi, X. Fong, A. Raghunathan, and K. Roy, “Yield, Area, and Energy Optimization in STT-MRAMs using Failure-Aware ECC,” *ACM JETC*, 2016.

[27] T. S. B. Sudarshan, R. Mir, and S. Vijayalakshmi, “Highly Efficient LRU Implementations for High Associativity Cache Memory,” in *Proc. of ICACCS*, 2004.

[28] C. Kim, D. Burger, and S. W. Keckler, “An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated on-Chip Caches,” in

Proc. of 10th International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.
[29] S. Veeramachaneni, A. Lingamneni, M. Kirthi-Krishna, and M. B. Srinivas, "Novel Architectures for Efficient (m, n) Parallel Counters," *ACM GLSVLSI*, 2007.
[30] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr.*, 2012.
[31] [AccessedMay.20,2019]. [Online]. Available: http://www.synopsys.com.
[32] C. Wilkerson, H. Gao, A. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *Proc. of ISCA*, 2008.
[33] C. Yan and R. Joseph, "Enabling Deep Voltage Scaling in Delay Sensitive L1 Caches," in *Proc. of DSN*, 2016.
[34] R. Bishnoi, M. Ebrahimi, F. Oboril, and M. Tahoori, "Improving Write Performance for STT-MRAM," *IEEE TMAG*, 2016.
[35] W. S. Zhao, T. Devolder, Y. Lakys, J. O. Klein, C. Chappert, and P. Mazoer, "Design considerations and strategies for high-reliable STT-MRAM," *Microelectronics Reliability*, 2011.
[36] T. Inokuchi, H. Yoda, Y. Kato, M. Shimizu, S. Shirotori, N. Shimomura, K. Koi, Y. Kamiguchi, H. Sugiyama, S. Oikawa, K. Ikegami, M. Ishikawa, B. Altansargai, A. Tiwari, Y. Ohsawa, Y. Saito, and A. Kurobe, "Improved read disturb and write error rates in voltage-control spintronics memory (VoCSM) by controlling energy barrier height," *Applied Physics Letters*, 2017.
[37] A. Salahvarzi, A. M. H. Monazzah, M. Fazeli, and K. Skadron, "NOSTalgy: Near-Optimum Run-time STT-MRAM Quality Energy Knob Management for Approximate Computing Applications," *IEEE TC*, 2020.
[38] P. Skoncej, "ECC with Increased Hard Error Correction Capability for Memory Reliability Improvement," in *Proc. of NVMTS*, 2014.
[39] T.-Y. Hsieh, T.-L. Chih, and M.-J. Wu, "Cost-Effective Enhancement on Both Yield and Reliability for Cache Designs Based on Performance Degradation Tolerance," *IEEE TVLSI*, 2017.
[40] K.-W. Kwon, X. Fong, P. Wijesinghe, P. Panda, and K. Roy, "High-Density Robust STT-MRAM Array through Device/Circuit/Architecture Interactions," *IEEE TNANO*, 2015.
[41] W. Kang, L. Zhang, J.-O. K. W. Zhao, Y. Zhang, D. Ravelosona, and C. Chappert, "Yield and reliability improvement techniques for emerging nonvolatile STT-MRAM," *IEEE JETCAS*, 2015.
[42] N. Onizawa and T. Hanyu, "Redundant STT-MTJ-Based Nonvolatile Flip-Flops for Low Write-Error-Rate Operations," in *Proc. of NEWCAS*, 2016.
[43] Y. Lakys, W.-S. Zhao, T. Devolder, J.-O. Klein, D. Ravelosona, and C. Chappert, "Self-Enabled "Error-Free" Switching Circuit for Spin Transfer Torque MRAM and Logic," *IEEE TMAG*, 2012.
[44] H. Lee, A. Lee, S. Wang, F. Ebrahimi, P. Gupta, P.-K. Amiri, and K. Wang, "A Word Line Pulse Circuit Technique for Reliable Magnetoelectric Random Access Memory," *IEEE TVLSI*, 2017.
[45] Q. Zeng and J.-K. Peir, "Content-Aware Non-Volatile Cache Replacement," in *Proc. of IPDPS*, 2017.
[46] X. Wang, M. Mao, E. Eken, W. Wen, H. Li, and Y. Chen, "Sliding Basket: An Adaptive ECC Scheme for Runtime Write Failure Suppression of STT-RAM Cache," in *Proc. of DATE*, 2016.

Mahdi Talebi received the B.Sc. degree in Computer Hardware engineering from Tabriz University, Tabriz, Iran, in 2015. He received the M.S degree in Computer Architecture engineering from Iran University of Science and Technology (IUST), Tehran, Iran, in 2019. He is a member of Dependable Systems and Architectures Laboratory (DSA) from 2016. As a visiting researcher, he was with the Laboratoire de Conception et d'Intégration des Systèmes (LCIS), University of Grenoble, Valence, France, in 2018. His research interests include Low Power Design, non-volatile memory systems, hardware security and fault-tolerant embedded system design.

Arash Salahvarzi received the M.S degree in computer engineering from Iran University of Science and Technology (IUST), Tehran, Iran, in 2018. He is currently a researcher in Dependable Systems and Architecture Laboratory (DSA Lab) at IUST. His research interests include low power design, emerging nonvolatile memory technologies, fault-tolerant embedded system design, and storage systems.

Amir Mahdi Hosseini Monazzah received the Ph.D degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2017. He was a member of the Dependable Systems Laboratory from 2010 to 2017. As a Visiting Researcher, he was with the Embedded Systems Laboratory, University of California, Irvine, CA, USA from 2016 to 2017. As a postdoc fellow he was with the school of computer science, institute for research in fundamental sciences (IPM), Tehran, Iran from 2017 to 2019. He is currently a faculty member of the School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. His research interests include investigating the challenges of emerging nonvolatile memories, hybrid memory hierarchy design, and IoT applications.

Kevin Skadron is the Harry Douglas Forsyth professor and chair of the Department of Computer Science at the University of Virginia, where he has been on the faculty since 1999. His research focuses on heterogeneous architecture, design and applications of novel hardware accelerators, and design for physical constraints such as power, temperature, and reliability. Skadron and his colleagues have developed a number of open-source tools to support this research, including the HotSpot thermal model, the Rodinia GPU benchmark suite, the ANMLZoo automata benchmark suite, the MNCaRT automata processing design framework, and the RAPID programming language. Skadron is a Fellow of the IEEE and the ACM, and recipient of the 2011 ACM SIGARCH Maurice Wilkes Award.

Mahdi Fazeli received the M.Sc and Ph.D. degrees in computer engineering both from the Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively. He was with the department of computer engineering, Iran University of science and technology (IUST) from 2011 to 2019 as an associate professor. He is currently an associate professor at the department of computer engineering, Bogazici University, Istanbul, Turkey. He has established and chaired two research laboratories at IUST since 2012, namely Dependable Systems and Architectures Laboratory (DSA) and Networked and Embedded System Laboratory (NESL). He has authored and co-authored more than 50 papers in reputable journals and conference proceedings. His research interests include reliable issues in VLSI circuits and emerging technologies, dependable computer architectures, Low power circuits and systems, fault tolerant computer architectures, Fault injection and reliability modeling and evaluation.