Sliding Basket: An Adaptive ECC Scheme for Runtime Write Failure Suppression of STT-RAM Cache*

Xue Wang, Mengjie Mao, Enes Eken, Wujie Wen†, Hai Li, Yiran Chen
Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA
†Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33174, USA
{xuw15, mem231, ene4, hal66, yic52}@pitt.edu, †wwen@fiu.edu

***Abstract*—Write reliability is one of the major challenges in design of spin-transfer torque random access memory (STT-RAM) caches. To ensure design quality, error correction code (ECC) scheme is usually adopted in STT-RAM caches. However, it incurs significant hardware overhead. In observance of the dynamic error correcting requirements, in this work, we propose *Sliding Basket* – an adaptive ECC scheme to suppress the runtime write failures of STT-RAM cache with minimized hardware cost. Our simulation results show that compared to the STT-RAM caches with conventional ECC scheme, applying Sliding Basket can achieve up to 80.2% saving in ECC bit overhead, comparable write reliability and even better system performance.**

I. Introduction

The difficulty in further scaling the feature size of conventional memories motivated the recent investment in *emerging memory technologies* (EMTs) [1]. As a promising candidate of EMTs, *spin-transfer torque random access memory* (STT-RAM) features many attractive characteristics, including near-zero standby power, nanosecond access time, small footprint, *etc*. These properties make STT-RAM perfectly suitable for the applications that are subject to limited power and area budgets, *i.e.*, on-chip cache [2].

A major challenge in STT-RAM design is the access reliability issue, *e.g.*, high write error rate. A write error in an STT-RAM cell occurs as the write pulse is removed before the completion of the resistance switching of its data storage device – *magnetic tunneling junction* (MTJ) [1]. The parametric variability of MOS transistor and MTJ [3] as well as the thermal-induced randomness in MTJ switching process [4] induce a large variation of MTJ switching property, making the write reliability control very difficult.

Error correction code (ECC) has been widely adopted in STT-RAM to ensure the access reliability. For example, Xu *et al.* proposed to apply different ECC codes to STT-RAM cells according to their error correcting requirements so as to balance the write robustness and performance [5]. Considering the asymmetric erroneous probabilities of programming 1 and 0, Wu *et al.* developed an asymmetric ECC scheme that strengthens the protection for '0→1' switchings [6]. Nonetheless, these ECC schemes were designed for the worst-case scenario that rarely happens in real applications, *e.g.*, ignoring the variability of data error rate across different memory blocks and program segments. Such pessimistic designs require to reserve considerable design margin, introducing significant hardware cost and performance overheads.

*This work is partially supported by NSF ECCS-1202225.

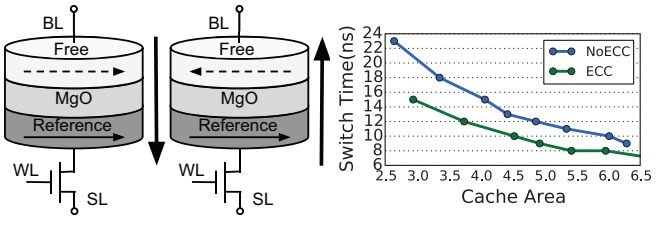
In light of the dynamic error correcting requirement in the application of STT-RAM caches, in this work, we propose *Sliding Basket* – an adaptive ECC scheme to suppress runtime write failures. In Sliding Basket, the cache is partitioned into regions (*baskets*) protected by different ECCs. The error rate of a data is speculated on-the-fly and the data is allocated to a partition that provides the needed error correcting capability. Moreover, to accommodate the time-varying error correcting requirements of runtime data, the thresholds that determine data's destination cache partition will be adaptively adjusted. Our experimental results show that compared to conventional ECC scheme, Sliding Basket can save up to 80.2% ECC bit overhead with slightly degraded write reliability of the STT-RAM cache. Moreover, the detailed analysis shows that through simultaneous optimization in cache access patterns and reducing STT cell programming workload, Sliding Basket outperforms conventional ECC design in power and energy consumptions.

II. Preliminary

A. STT-RAM basics

In an STT-RAM cell, data is represented by the resistance state of its MTJ device. As illustrated in Fig. 1(a), an MTJ is composed of two ferromagnetic layers (*i.e.*, reference and free layers) and an oxide insulator. The magnetization direction of the reference layer is fixed while that of the free layer is switchable under a polarized write current [1]. When the magnetization directions of the two ferromagnetic layers are in parallel (anti-parallel), the MTJ is in low (high) resistance state, representing logic '0' ('1'). The write current through the MTJ is supplied by a NMOS transistor. Such an STT-RAM cell is usually denoted as '1T1J' structure.

In an STT-RAM cell, a write error happens when the write current is removed before the MTJ resistance switching



(a) (b)

Fig. 1: (a) STT-RAM basics; (b) The tradeoff between MTJ switching time and transistor size for a fixed block error rate of 3.64×10^{-12} at $22nm$. The block size is 512-bit.

process completes [7][8]. Due to the relatively lower driving strength of the NMOS transistor and the higher switching current of the MTJ, the '0→1' switching demonstrates a much higher probability of write errors than the '1→0' switching [7].

We note that due to the intrinsic randomness in process variations and thermal fluctuations, write errors of STT-RAM cannot be completely eliminated. Raising the amplitude of write current (by increasing NMOS transistor size) and prolonging write pulse width (*i.e.*, MTJ switching time) can reduce the write error rate. As shown in Fig. 1(b), a tradeoff exists between the applied write current amplitude and write pulse width under a fixed write error rate target [3]. However, the circuit-level solutions introduce extra area overhead and/or performance penalty, not even mentioning the increase in write energy consumption.

B. STT-RAM Write Error Model

As aforementioned, the write reliability of an STT-RAM cell is mainly determined by the failure rate of '0→1' switching rather than that of '1→0' switching. The write failure rate of an STT-RAM cache block, *i.e.*, the probability of having no more than t erroneous bits, can be approximated by [8]:

$$P(n_e \leq t) = \sum_{i=0}^{t} C^i_{\mathrm{FLIP}} \mathrm{BER}^i (1-\mathrm{BER})^{\mathrm{FLIP}-i}, \quad (1)$$

where FLIP denotes the number of the bits flipping from 0 to 1, which includes the flipping in ECC bits if ECC is applied; n_e is the number of error bits; and BER is the write error rate of a single bit.

STT-RAM is often adopted in latency-sensitive scenarios, so the application of ECC is usually constrained to SECDED (*single error correction, double error detection*) schemes. A cache block can be divided into several data segments, each of which is separately protected by a set of ECC check bits. In such a case, the BLER (*BLock Error Rate*) of the cache block can be approximated by:

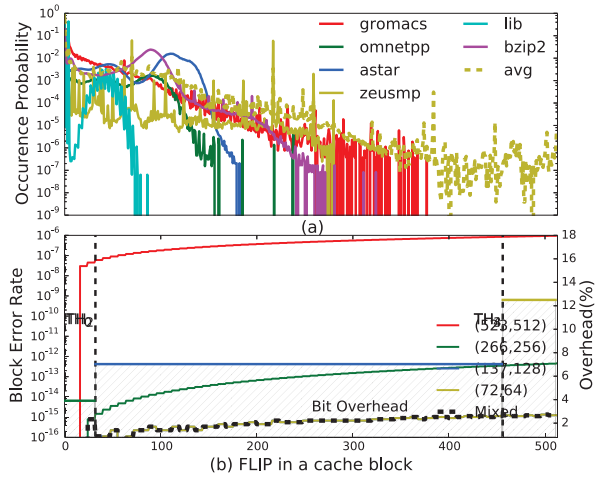


Fig. 2: (a) Distribution of the FLIP of 512-bit cache blocks in some representative benchmarks; (b) BLER and bit overhead when applying different ECC schemes.

$$P^{ecc}_{err,blk} = 1 - \prod_{i=1}^{SegNum} P_i(n_e \leq 1). \quad (2)$$

Here, $SegNum$ represents the number of the divided segments in the cache block and BLER denotes the probability of having uncorrectable error(s) in the cache block.

Fig. 2(a) summarizes the distribution of average FLIP of the 512-bit cache blocks in some representative SPEC2006 benchmarks. As the value of FLIP increases, occurrence probability reduces rapidly. In other words, it is very rare that many bits are flipping from 0 to 1 simultaneously. Here we assume the read-before-write scheme [9] is applied, *i.e.*, write operation is performed only when the data being written is different from the one stored in the targeted memory cell.

Fig. 2(b) shows the change of BLER with FLIP under several SECDED ECC schemes. We use $(w,\ k)$ to denote an ECC with w codeword length and k ECC bits. Utilizing a strong ECC, *i.e.*, (72, 64), effectively suppresses the write failures but increases the ECC bit overhead. When the target failure rate is set, the normal attempt in the application of STT-RAM caches is to select ECC based on the maximum possible FLIP. Such an approach, however, induces significant design pessimism in the case of smaller FLIPs.

III. Design Methodology

A. Technical Motivations

Applying ECC can enhance the write reliability of STT-RAM, which can be further translated to memory cell area reduction and/or write performance improvement as shown in Fig. 1(b). In conventional designs, all the cache blocks are equipped with the same ECC scheme. The STT-RAM write reliability can be roughly measured by the worst BLER, which corresponds to the maximum FLIP under the protection of the strongest ECC available to the design. As shown in Fig. 2(b), the maximum BLER is 3.64×10^{-12}, which occurs when FLIP = 512 under the protection of (72, 64). If the FLIP of the cache block is smaller than the maximum FLIP, we may apply a weaker ECC to protect the block while still ensure the BLER not higher than the maximum BLER. In general, such a scheme can be represented as a pair of $[ECC_1, ..., ECC_n]$ and $[TH_1, ..., TH_n]$, where $ECC_i, i = 1, ..., n$, are different ECCs from the weakest to the strongest; and $TH_j, j = 1, ..., n$, are the maximum FLIP that can be protected by ECC_j. Here $TH_n = S$, which is the cache block size. Here we assume the error correcting strength of ECC_i increases monotonically when i increases. For illustration purpose, Fig. 2(b) compares the failure rates when applying a mixed ECC scheme of [(523, 512), (266, 256), (137, 128), (72, 64)] and [180, 256, 360, 512]. The failure rates corresponding to different FLIPs are bounded by the maximum BLER at FLIP = 512. The ECC bit overhead of every single ECC scheme is also presented in the figure for comparison purpose.

The above observation motivates us to propose *Sliding Basket* scheme that aims at reducing the ECC bit overhead by fully utilizing the error correcting strength of ECC schemes to satisfy the ever-changing needs of the cache block data.

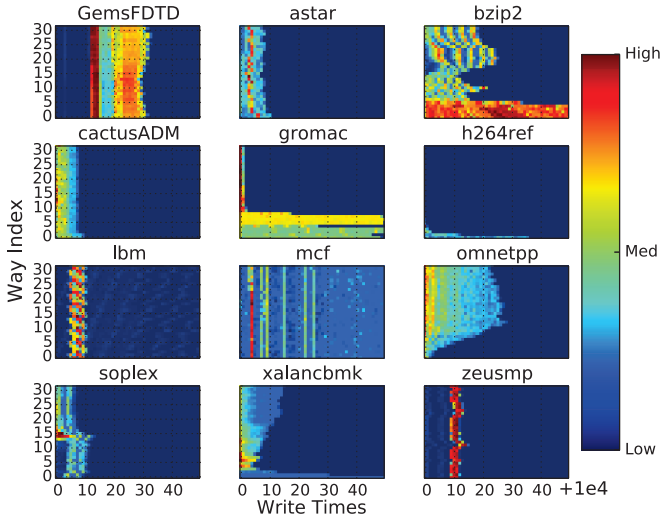


Fig. 3: Spatial and temporal variabilities of ECC strength requirement in a L2 STT-RAM cache.

B. Spatial and Temporal Variability's of ECC Requirement

Fig. 3 shows the required ECC strength of cache block that is mainly decided by the FLIP of data, varies across the 32 ways in a L2 STT-RAM cache. This property is referred to as the *spatial variability* of the ECC requirement. Also, the required ECC strength of every cache block changes over time, representing the *temporal variability*. In the following three subsections, we will show: 1) how to leverage the spatial variability of the ECC requirement to reduce the ECC bit overhead by configuring the ECC schemes of cache blocks at way level and guiding data to be stored at the corresponding cache block with proper ECC protection; and 2) managing the temporal variability of the ECC requirement through dynamically adjusting the thresholds to guide the data allocation during program execution.

C. Basic Concept of Sliding Basket

For simplicity, we assume two ECC schemes with different error correcting strengths available to our design, *i.e.*, strong and weak. All the cache blocks belonging to the same set are partitioned into two groups: Group H and Group L, which belong to the different cache ways. The cache blocks in Group H and Group L are protected by the strong ECC and the weak ECC, respectively.

When a data is scheduled to be written into the STT-RAM cache, its required ECC strength can be first estimated based on its FLIP, as discussed in Section II-B. Because only two ECC schemes are available in the presented example, only one FLIP threshold is needed to categorize the data into two types, say, *High Flip Data* (HFD) and *Low Flip Data* (LFD). Once the estimated required ECC strength is obtained, the following

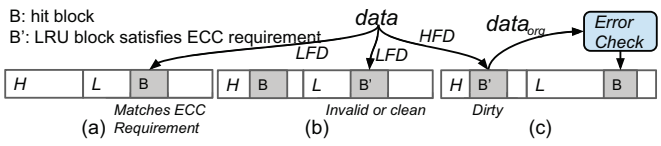


Fig. 4: Handle blocks when write hits.

Select Group based on ECC requirement
$data \leftarrow$ *incoming data*
$B \leftarrow$ *hit block if hit*
$B' \leftarrow$ *lru block from selected group*
$data_{org} \leftarrow$ *data original in B'*
Write miss:
Find LRU block *B'* in selected group
Write *data* to *B'*
Write hit:
if *B* $matches$ *ECC requirement* **then**
  Update $data$ to *B*, see Fig. 4(a)
else
  Find LRU Block *B'* in selected group
  if *B'* $is\ clean\ or \in\ weaker\ ECC\ group$ **then**
    Evict $data$ in *B*, see Fig. 4(b)
  else
    Perform error check for $data_{org}$ in *B'*
    Move $data_{org}$ to *B*, see Fig. 4(c)
  Write *data* to *B'*

Fig. 5: Data allocation flow of Sliding Basket.

data allocation procedure will be applied to guarantee the reliability of the data:

In a write miss, the HFD and the LFD are placed into Group H and Group L, respectively, by following LRU policy within each group. In a write hit, if the hit Block B belongs to a group which has the ECC exactly matching the need of the data, B is updated with the new data directly, as shown in Fig. 4(a). Otherwise, a block in the group with the ECC exactly matching the need of the data, say B', will be selected to store the data based on LRU policy. If Block B' is invalid or clean or belongs to the group with an ECC strength weaker than that of the group containing Block B (*e.g.*, in Group L as shown in Fig. 4(b)), the original data in Block B' is evicted and replaced with the new data. If Block B' belongs to a group with an ECC strength stronger (*e.g.*, Group H in Fig. 4(c)) than that of the group containing Block B, the data in Block B' is first moved to Block B and then the new data is written to Block B'. An error check and correction will be performed on the original data of Block B' before writing it to Block B. The whole procedure is depicted in Fig. 5.

The rationales of the above flow are the follows: As majority of the data have low FLIPs during program execution, the capacity of the groups with weak ECC, *e.g.*, Group L, shall be larger than that of the groups with strong ECC, *e.g.*, Group H. Hence, if the block to be replaced (*i.e.*, B', which is the least recently used block in the group) is in Group L, replacing Block B' with the new data imposes minimum impact on system performance due to the large capacity of Group L (see Fig. 4(b)). However, if the block to be replaced (*i.e.*, B') is in Group H, directly overwriting the original $data_{org}$ in Block B' may greatly affect system performance because of the limited capacity of Group H. Hence, we need to move the original data in B' to the originally hit Block B (see Fig. 4(c)).

There exists a small probability that the weak ECC associated with Block B in Group L may not be able to provide sufficient protection to the data moved from Block B'. Adding

an error check and correction step in moving the data from Block B' into Block B can effectively reduce the possibility of writing an erroneous data. In fact, as Block B' is identified in Group H based on LRU policy, the possibility of continuing to use the data moving from Block B' is already very low.

D. ECC Requirement Driven Cache Partition

By assuming the data can be perfectly allocated to the cache group with exactly matched ECC strength, at a specific time, an ideal partition of the Sliding Basket cache is:

$$N_i = \begin{cases} \lceil \sum_{j=TH_{i-1}+1}^{S} P_j \times A \rceil, i = n \\ \lceil \sum_{j=TH_{i-1}+1}^{S} P_j \times A \rceil - \sum_{j=i+1}^{n} N_j, i < n \end{cases}. \quad (3)$$

Here S denotes the size of cache block; A is the associativity of the cache; P_j is the occurrence probability when FLIP $= j$; N_i is the number of ways protected by the ECC_i; TH_0 = 0. The case of $i = n$, in which the strongest ECC is needed, is handled separately to make sure to cover the worst-case. The above "ideal" partition guarantees to provide the least necessary ECC protection to the data with minimum ECC bit overheads, which can be calculated by:

$$O_{ECC}^{verhead} = \sum_{i=1}^{n} N_i \times O_{ECC_i}^{verhead}. \quad (4)$$

Here $O_{ECC_i}^{verhead}$ denotes the bit overhead of ECC_i to cover one block in the corresponding data group.

TABLE I summarizes the bit overheads of the four concerned ECC schemes and their FLIP thresholds for the 512-bit cache block. The average occurrence probability of the FLIPs between two adjacent thresholds over all the benchmarks are also depicted in the 5^{th} line of the table. The corresponding "ideal" cache partition of this FLIP distribution is shown in the bottom line of the table. Based on Eq. (3), the data with the required ECCs of (266, 256) and (137, 128) are covered by (72, 64). Thus, no cache ways are assigned to these two ECC schemes in the cache partition in TABLE I. This partition incurs a 2.47% ECC bit overhead as calculated by Eq. (4).

To accommodate the temporal variability of the ECC requirement, we introduce some margins to the cache partition by raising the number of cache ways assigned to the strong ECC from the average case. More details on the design tradeoff between the cache partition and system performance/reliability will be discussed in Section IV.

Obtaining FLIP of a cache block needs to compare both the incoming data and the data originally stored in the cache block. In the Sliding Basket flow presented in Fig 5, however, deciding the destination cache block also requires to know FLIP. To solve this "chicken-and-egg" problem, we propose using HW (*Hamming Weight*), which denotes the number of '1's in the data, to approximate the FLIP of the destination cache block to guide the operation of Sliding Basket. In [3], Bi *et al.* proved that there is a strong correlation between HW and FLIP of the data. Indeed, HW is a good pessimistic approximation (upper bound) of FLIP. Hence, the basic Sliding Basket scheme presented in Section III-C can be safely modified by using HW (instead of FLIP) as the threshold to allocate the data into different cache groups.

TABLE I: Bit overheads of different ECC schemes.

Data bits	512	256	128	64
ECC bits	11	10	9	8
Overhead	2.15%	3.91%	7.03%	12.5%
TH_i	180	256	360	512
$\sum_{j=TH_{i-1}+1}^{TH_i} P_j$	99.16%	0.74%	0.06%	0.04%
N_i^{avg}	31	0	0	1

E. Dynamic Sliding

Obviously a fixed cache partition (even with margin to the group with strong ECC) cannot perfectly accommodate the temporal variability of the ECC requirement. Besides the possible write failures caused by temporarily inadequate capacity of the groups with the needed ECC strength, the inflexibility of the fixed cache partition could also harm system performance by over- or under-utilizing some particular cache groups. To solve this issue, we propose to dynamically adjust the thresholds that are used to allocate the data to overcome the shortcoming of the fixed cache partition: A miss rate counter is added to monitor the usage pattern of each group. The TH of a group will be reduced (raised) when a significant miss rate increase (decrease) is detected.

IV. Experimental Results

A. Experiment Setup

TABLE II summarizes the baseline system configuration used in our experiments. The timing and energy parameters of the STT-RAM cache are extracted from NVSim [10] at $22nm$ technology. Here the baseline access latency of the STT-RAM cache has already taken into account the ECC encoding and decoding latency of (72, 64), which is set to 1 clock cycle according to [11]. To be conservative, we assume that the ECC encoding and decoding of (523, 512) take one more cycle to finish than that of (72, 64). All the simulations are performed on GEM5 simulator [12] with 64-bit Alpha instruction set.

We select 15 representative benchmarks from SPEC CPU 2006 suite [14] in our evaluations. SimPoints [15] is used to extract a single simulation point of benchmarks. Each simulation point contains 500 million instructions. The caches and memory system are warmed up with 100 million instructions before jumping into a simulation point.

To evaluate the impact of cache partitioning on the system performance and reliability, we include not only the cache partition of the average case ("Opt 4" in TABLE III) but also some partitions with larger Group H ("Opt 1-3") in our

TABLE II: System configuration.

Core	4GHz, OOO 8-width, 64-energy LSQ, 64-entry instruction queue, 192-enrty ROB
Caches	32KB L1I, 2-way, 2-cycle, 64B line
	32 KB L1D, 4-way, 2-cycle, 64B line, write back
	8MB STT-RAM L2 cache, single-core, 32-way, 16 banks, 1 port, 64B line, write back, 20 MSHRs Read energy 0.241nJ, Read delay 12 Cycles, Write energy 0.882nJ, Write delay 30 Cycles
Main memory	DDR3-1600, 2-channel, open-page, FR-FCFS [13]

TABLE III: Scheme configuration.

Scheme Configuration	ECC bits*	Overhead	Cache Area
No ECC	0	0	6.02 mm^2
SECDED(72, 64)	64	12.5%	4.53 mm^2
SECDED(523, 512)	11	2.15%	4.14 mm^2
Opt 1: 26(523, 512)+6(72, 64)	20.93	4.09%	4.22 mm^2
Opt 2: 27(523, 512)+5(72, 64)	19.28	3.76%	4.20 mm^2
Opt 3: 28(523, 512)+4(72, 64)	17.62	3.44%	4.19 mm^2
Opt 4: 31(523, 512)+1(72, 64)	12.65	2.47%	4.16 mm^2

*ECC bits denote the average ECC bits number for a 64B line in a 32-way cache set.

simulations. Since the occurrence probability of the data with high FLIP/HW is very low (refer TABLE. I), only two ECC schemes – (523, 512) and (72, 64), are adopted in Sliding Basket scheme. A threshold TH_1 is used to guide the data allocation between the two groups and set to 180 at beginning of execution. The miss rate counter is checked every one million cycles and TH_1 changes by 10 if the variation of the miss rate is larger than 5%.

B. ECC Overheads

For comparison purpose, the bit overheads of SECDED (523, 512) and (72, 64) are also included in TABLE III, representing the best and the worst design overheads of conventional ECC schemes, respectively. Also, the areas of the L2 caches with and without ECC protections extracted from NVSim [10] are listed in TABLE III including different cache partitions of Sliding Basket. The BER of STT-RAM cells is set to 1.5×10^{-8}, which leads to a BLER of 3.64×10^{-12} when (72, 64) is applied. The write pulse width is set to $10ns$. To achieve the same BLER without applying ECC, the size of NMOS transistor in the STT-RAM cell must be increased to supply a larger write current, resulting in 32.9% more cache area compared to that of (72, 64) (*i.e.*, $6.02mm^2$ vs. $4.53mm^2$).

The ECC bit overheads for Sliding Basket configurations "Opt 1-4" vary between 2.47% and 4.41%, which are only 17.8% to 32.7% of that (12.5%) of (72, 64). Note that the area estimations have included the contribution of ECC logic and other support circuits of these schemes.

C. Reliability Enhancement

Fig. 6 compares the *Mean Time Between Failure* (MTBF) of all the ECC schemes and cache configurations. A larger MTBF implies a better system reliability. On average, the MTBF achieved by Sliding Basket is 20.0% ("Opt 4") $\sim$ 32.9% ("Opt 2") of the one of (72, 64), that is, $5.67\times \sim 9.35\times$ of that of (523, 512). All the data are normalized to the one of (72, 64). The results show that raising the size of Group H from the cache partition based on the average case ("Opt 4") can effectively enhance the system reliability, reaching the highest MTBF at "Opt 2". Continuing to increase the size of Group H, however, does not further enhance the system reliability.

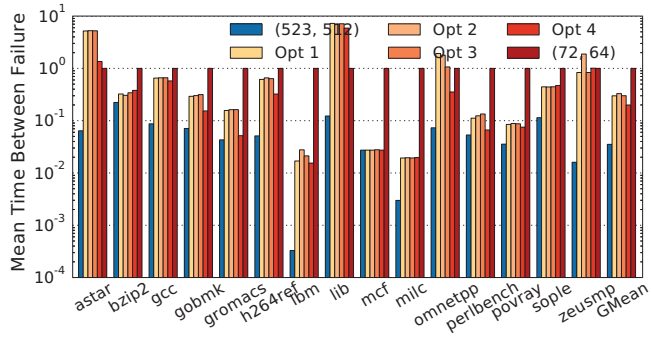


Fig. 6: Comparison of Mean Time Between Failure.

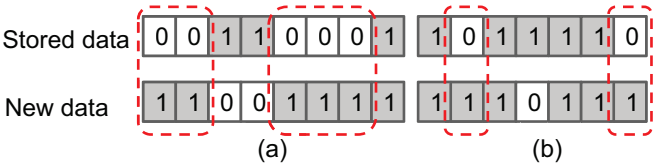


Fig. 7: Illustration of bit matching for reducing FLIP.

Interestingly, we found that in some benchmarks, such as `astar`, `omnetpp`, `lib`, and `zeusmp`, *the highest MTBFs achieved by Sliding Basket are even* $1.13\times \sim 6.81\times$ *better than that of the strongest ECC scheme (72, 64)!* A detailed analysis on this interesting observation shall be given in the next subsection.

D. Bit Matching Effect in Sliding Basket

The principle of Sliding Basket is allocating the data with similar ECC requirement (indexed by Hamming Weight) to the group with the matched ECC strength. Besides, an important byproduct of Sliding Basket – *the potential reduction of* FLIP that naturally enhances the write reliability of STT-RAM cache – can be observed.

Fig. 7 explains the reason for the FLIP reduction. In Sliding Basket, the data with similar HW (*i.e.*, the number of '1's) are allocated to the same group. Considering the locality of the cache data, such an operation potentially increases the probability for these '1's to show up at the same location of the new data and the stored data. Since read-before-write scheme is applied, the overlapped bits with the same value will not be updated during the write operation. In fact, in Group H, since the data are all with very high HW (more '1's), the overlapping rate of '1's between the new data and the stored data is even more prominent. Thus, compared to the data allocation in conventional cache design, a smaller average FLIP can be expected in Sliding Basket and a better write reliability may be achieved.

To verify our hypothesis, we compared the average FLIP of the cache data (not including the ECC bits) under different

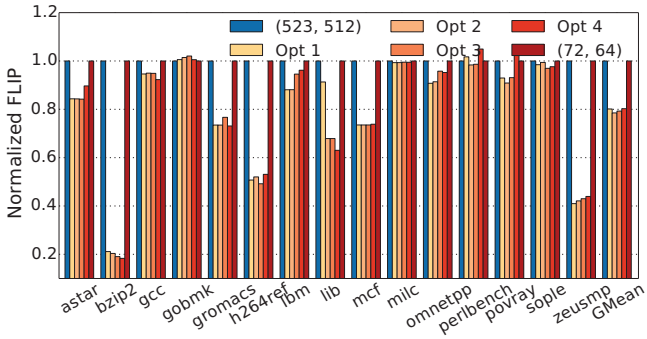


Fig. 8: Comparison of average FLIP.

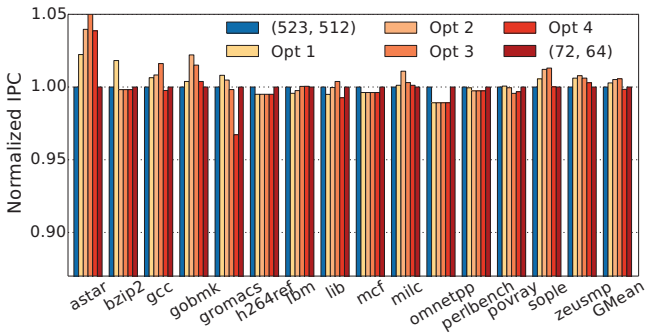


Fig. 9: Comparison of Instruction Per Cycle (IPC).

ECC schemes and cache partitions, as shown in Fig. 8. Compared to the level of (72, 64), Sliding Basket can reduce the FLIP up to 21.5% ("Opt 2") across all 15 benchmarks. The highest FLIP reduction (80.2%) is observed at `bzip2`.

E. Performance and Energy

Fig. 9 compares the performance of all the simulated ECC schemes and cache partitions. Our results show that Sliding Basket achieves a system performance generally comparable to the conventional ECC scheme across the 15 benchmarks. The largest performance degradation of Sliding Basket (mere 0.16%) occurs at `gromacs` with the configuration of "Opt 4" due to the increased miss rate in Group H. In fact, except for "Opt 4", the IPCs (*Instructions Per Cycle*) of "Opt 1-3" are all improved from the (72, 64) baseline by 0.28%, 0.51% and 0.57%, respectively, even the 1-cycle extra ECC encoding/decodeing latency has been included in the simulations. Our detailed analysis shows that in these configurations, their L2 cache miss rates all decrease from the baseline. A possible reason is that the HW based data allocation flow improves the data eviction effectiveness by the enhancing the correlation of the new data and the data stored in the cache block.

The results in Fig. 9 also show that raising the capacity of Group H effectively improves the system performance and reaches the highest performance at "Opt 3". Continuing to increase the capacity of Group H, however, does not offer a better performance.

An energy consumption lower than the one of conventional ECC schemes is achieved in almost every Sliding Basket configurations in the 15 benchmarks, as shown in Fig. 10. Among all cache partitions, the largest average energy saving is obtained by "Opt 4" because of its largest capacity of (523, 512) among all partitions. For "Opt 4", the largest energy saving is achieved at `gromacs`, or 13.6% lower than that of (72, 64). Besides the slightly improved IPC, such a wide energy saving achieved by Sliding Basket is mainly resulted by the reduction of flipping bits caused by: 1) the bit matching effect discussed in Section IV-D; and 2) the less number of bits that actually need to be written by applying weak ECC (and hence, less ECC bits) to the majority of the cache blocks.

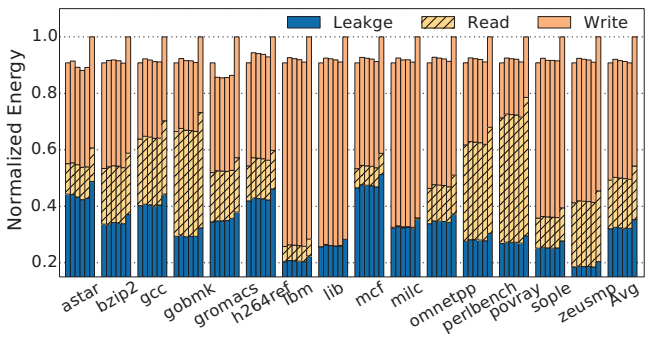


Fig. 10: Comparison of energy consumption.

V. Conclusion

In this work, we examine the dynamic needs of STT-RAM cache for ECC protections across different data blocks and program segments and propose Sliding Basket. It is an adaptive ECC scheme that can allocate every data to a cache group with the just needed ECC strength. As such, the associated hardware cost can be minimized. Our simulations show that Sliding Basket can save up to 80.2% ECC bit overheads with slightly degraded runtime reliability, compared to conventional ECC schemes. System performance and cache energy efficiency are also improved, benefiting from the enhanced data eviction effectiveness and the reduced bit flipping rate.

References

[1] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *IEDM Technical Digest*, 2005.

[2] 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*, 2012.

[3] X. Bi, Z. Sun, H. Li, and W. Wu, "Probabilistic design methodology to improve run-time stability and performance of stt-ram caches," in *ICCAD*, 2012.

[4] X. Bi, H. Li, and J.-J. Kim, "Analysis and optimization of thermal effect on stt-ram based 3-d stacked cache design," in *VLSI*, 2012.

[5] W. Xu, Y. Chen, X. Wang, and T. Zhang, "Improving stt mram storage density through smaller-than-worst-case transistor sizing," in *DAC*, 2009.

[6] 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 *ICCAD*, 2013.

[7] Y. Zhang, X. Wang, Y. Li, A. K. Jones, and Y. Chen, "Asymmetry of mtj switching and its implication to stt-ram designs," in *DATE*, 2012.

[8] W. Wen, Y. Zhang, Y. Chen, Y. Wang, and Y. Xie, "Ps3-ram: a fast portable and scalable statistical stt-ram reliability analysis method," in *DAC*, 2012.

[9] K.-J. Lee, B.-H. Cho, W.-Y. Cho, S. Kang, B.-G. Choi, H.-R. Oh, C.-S. Lee, H.-J. Kim, J.-M. Park, Q. Wang, *et al.*, "A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput," *IEEE Journal of Solid-State Circuits*, 2008.

[10] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, 2012.

[11] B. Zimmer and M. Zimmer, "Maximizing energy-efficiency through joint optimization of l1 write policy, sram design, and error protection," 2012.

[12] 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," *SIGARCH*, 2011.

[13] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, *Memory access scheduling*. 2000.

[14] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH*, 2006.

[15] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," in *SIGMETRICS*, 2003.