Endurance Enhancement of Write-Optimized STT-RAM Caches

Puneet Saraf
Indian Institute of Technology Madras
Chennai, Tamil Nadu, India
sarafpuneet04@gmail.com

Madhu Mutyam
Indian Institute of Technology Madras
Chennai, Tamil Nadu, India
madhu@cse.iitm.ac.in

ABSTRACT

Low density and high leakage power of SRAM are the major setbacks for its scalability. Non-volatile memory (NVM) like spin-transfer torque random access memory (STT-RAM) is a suitable replacement for SRAM at the last level cache (LLC). NVM offers high density, and near zero leakage, which are highly desired for on-chip caches. A few drawbacks of STT-RAM such as high write latency and limited endurance, have to be taken care before it can replace SRAM at the LLC. Prior works have either tried to optimize the write latency or endurance. In this paper, by considering write-optimized STT-RAM, we propose endurance improvement techniques by reducing the maximum number of writes, global write variation, and the average number of writes. We take care of low retention time of write-optimized STT-RAMs using a refresh mechanism. We employ refresh-aware cache replacement policies wherein the cache blocks that are about to expire are preferred to the recently refreshed cache blocks. This refresh-aware policy, when combined with the recency information of the cache blocks, enhances both performance and endurance of STT-RAM LLC. We show that our refresh-aware policy achieves the maximum lifetime improvement of 32.5% for single-core and 70.7% for muti-core compared to STT-RAM with no wear leveling. When we combine recency information with our refresh-aware policy, there is a slight improvement in the performance.

CCS CONCEPTS

• **Hardware → Non-volatile memory**; *Emerging architectures*; Memory and dense storage.

KEYWORDS

Emerging memory technologies, non-volatile memory, STT-RAM, cache design and optimization

1 INTRODUCTION

With the current trend of technology scaling, we need large size caches to fulfill the performance gap between processor and main memory. The SRAM technology offers low access latency, but the scalability of SRAM is limited due to its high leakage current and low storage density. A typical SRAM cell requires six transistors to store one bit of information, which leads to its low storage density, and continuous power supply is required to retain a bit in the SRAM cell, which leads to high leakage current. There is a need for an alternative like non-volatile memories (NVMs) due to the limitations of SRAM. NVMs offer a much higher density than SRAM and have near zero leakage. There are several NVMs like phase-change random access memory (PCM), resistive random access memory (ReRAM), spin-transfer torque random access memory (STT-RAM). NVMs also have their drawbacks, such as high write latency and limited endurance. Write endurance of ReRAM, PCM, and STT-RAM are 10^{11}[10], 10^{8}[14], and 4×10^{12}[20], respectively. Because of high endurance among different NVMs, STT-RAM is the most suitable candidate to be deployed in the LLC.

The read operation in STT-RAM involves reading the resistance value of the magnetic tunnel junction (MTJ) by passing a small current across the bit and source line. Whereas, in the write operation, the entire orientation of the magnetic layer may have to be changed that requires much higher current as compared to the read operation. Because of this, STT-RAM has low read latency and high write latency. Techniques have been proposed to optimize the write latency by changing the magnetic properties of the MTJ or by changing the planar area of the free layer [6][18][19]. In optimizing the write latency of STT-RAM, the data retention time of STT-RAM is reduced [18], which forces the STT-RAM to get refreshed for retaining the data. A refresh operation requires an additional write operation to the cache blocks.

Even though STT-RAM has high write endurance among the other NVMs, it is very less as compared to SRAM. The endurance value of STT-RAM indicates the number of write operations that can be performed on the MTJ. Endurance can further degrade due to the non-uniform write accesses across the STT-RAM cache. In the set-associative cache, write variation can be of two types: 1) intra-set write variation, i.e., non-uniform write accesses within a set, and 2) inter-set write variation, i.e., non-uniform write accesses across sets. These variations can also be due to the skewed accesses generated by the replacement policy. For example, the LRU replacement policy maximizes accesses to few most recently used cache block locations [12]. Write variation can cause a few cache blocks to reach the endurance limit early as compared to the other blocks, which in turn causes the STT-RAM cache to fail. The write latency optimization of STT-RAM can further degrade the endurance by generating additional writes due to refresh operation.

To improve the endurance of write-optimized STT-RAM caches, we propose a technique to minimize write variation by disassociating cache sets from cache ways. We also propose a few cache replacement policies that make use of the refresh and/or recency information of cache blocks while minimizing the number of writes to the blocks. Our main contributions are as follows:

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357538

Table 1: Cache Block Level Minimum and Maximum Number of Writes, and the Mean and Standard Deviation of the Number of Writes at the LLC of a Single-Core System for Baseline.

Parameter	Single-core Workloads																									
	As	Bz	Ca	Cl	Ga	Gc	Ge	Gr	H2	Hm	Lb	Le	Li	Mf	Mi	Na	Om	Sj	So	Xa	Ze	Ap	Bc	Bf	Co	Ss
Minimum (min_{blk})	461	2238	3560	261	16	1515	8294	237	108	807	16847	8461	7908	21922	1083	101	342	1187	1288	315	4097	908	828	469	479	809
Maximum (max_{blk})	7017	10782	18175	80139	480171	30437	168437	35883	21428	32060	29174	46055	9933	80614	13859	102843	112958	186638	78132	981630	40505	8697	7720	20934	21674	21074
Mean (μ_{blk})	1478	3246	3993	986	1817	4351	9850	1034	1090	1446	22138	10346	7952	27657	2899	639	3950	1941	1777	12674	5467	1116	1130	2607	2593	2765
Std Dev (σ_{blk})	876	757	543	2945	9907	2895	6928	1124	947	2178	1448	2335	22	6118	2377	3247	2980	4804	2433	50990	2917	107	256	266	255	271

Table 2: Cache Block Level Minimum and Maximum Number of Writes, and the Mean and Standard Deviation of the Number of Writes at the Multi-Core LLC for Baseline.

Parameter	Multi-core Workload Mixes									
	Mix1	Mix2	Mix3	Mix4	Mix5	Mix6	Mix7	Mix8	Mix9	Mix10
Minimum (min_{blk})	10	128	77	1446	84	160	258	704	124	45
Maximum (max_{blk})	65144	76855	62706	181247	271787	203940	834310	20787	9619	38471
Mean (μ_{blk})	342	1440	893	2389	1539	1894	1806	2045	387	601
Std Dev (σ_{blk})	802	1668	1240	3918	7760	2587	7634	292	281	873

- We analyze that write latency optimization generates an overhead of refresh writes that can further degrade the lifetime of STT-RAM cache.
- We propose that skewed cache organization, as compared to set-associative cache organization, reduces the write variation across the cache, which in turn can enhance the endurance of STT-RAM cache.
- Our refresh and/or recency aware policies help minimize the refresh writes without any major impact on the performance.
- Our techniques also reduce the average number of writes to the STT-RAM cache, which in turn provide dynamic energy savings compared to the state-of-the-art technique.

The rest of the paper is organized as follows: We provide background and related work in Section 2 and the motivation in Section 3. Our approach is discussed in Section 4. We provide experimental setup in Section 5 and the result analysis in Section 6. Finally, we conclude the paper in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Overview of STT-RAM

STT-RAM is a type of magnetic RAM. It comprises an information carrier magnetic tunnel junction (MTJ) and an access transistor. MTJ consists of two ferromagnetic layers and one metal oxide layer. One of the ferromagnetic layers can change its orientation with respect to the other layer that has a fixed magnetic orientation. If the magnetic orientation of layers is parallel, the STT-RAM cell is in a low resistance state (storing bit 0), whereas, if the orientation is anti-parallel, the cell is in high resistance state (storing bit 1). The threshold current required for a write operation is higher than that for a read operation as it may require to change the magnetic orientation of the free layer.

2.2 Addressing Write Latency Issue

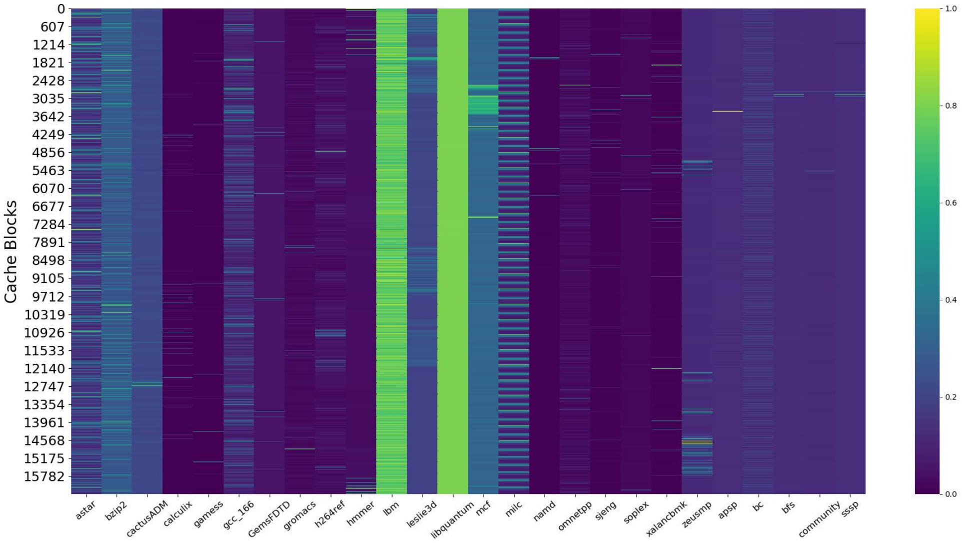
One can optimize the write latency of STT-RAM by changing several parameters of MTJ. Smullen *et al.* [18] proposed a write latency reduction technique by reducing the planar area of the cell. Sun *et al.* [19] proposed multi retention level STT-RAM design by changing different parameters of STT-RAM. Jog *et al.* [6] optimized the write latency by reducing the thickness of the free layer and lowering the saturation magnetization of STT-RAM. In [21][22], authors have proposed a hybrid cache architecture, where a cache is partitioned into multiple regions, each of different memory technology. The fast region is of SRAM type that offers low access latency, and the slow region is of STT-RAM type that offers higher density. In all the above techniques except the hybrid cache architecture, write latency optimization reduces the retention time of a cache block. Retention time is the duration for which the bit is retained without a random bit-flip. Reducing the retention time may require a refresh operation.

2.3 Addressing Endurance Issue

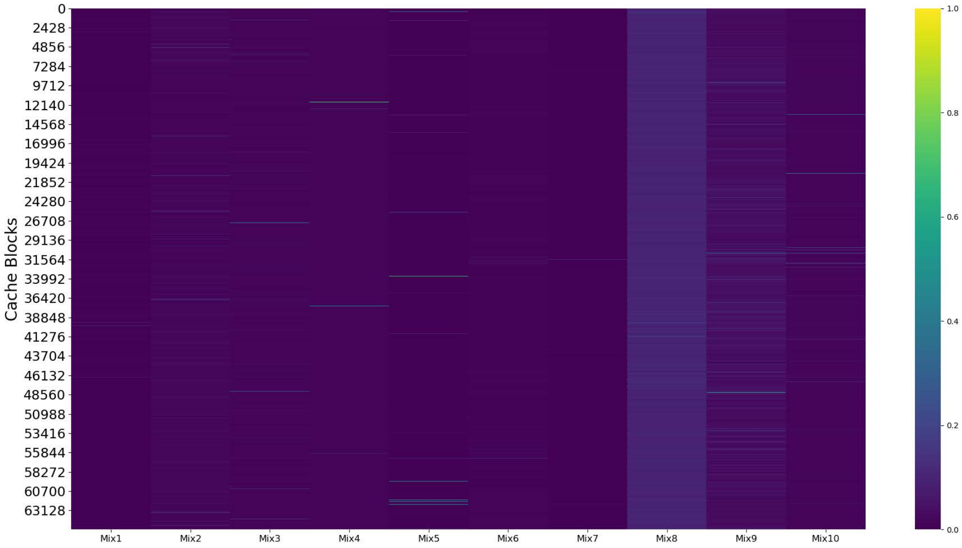
The write endurance of STT-RAM is the number of write operations that can be performed on the MTJ before the free layer stops changing its magnetic orientation. The write endurance techniques can be classified into write avoidance technique and wear leveling techniques. We discuss both the techniques below:

2.3.1 Write Avoidance Techniques: Several techniques have been proposed in the literature to reduce the number of writes. If the value to be written is the same as the value present in the cache, the write operation is terminated [24]. In addition to early termination of writes [24], before storing the data, it can be inverted if storing inverted data achieves fewer bit flips [8]. Encoding frequently written patterns to minimize the number of writes [2][9][23].

2.3.2 Write Leveling Techniques: Write leveling techniques try to make the write accesses uniform across the cache so that all the cache blocks wear out at the same rate. Inter-set write variation is handled by changing the set mapping [20]. Intra-set write variation is handled by flushing hot data probabilistically [20] or recording the number of writes to a block and then changing the location of a hot data item to a cold block location [13] or injecting random replacement policy at regular interval of instructions [16]. Sequoia [7] is the state-of-art endurance improvement technique that minimizes both intra-set and inter-set write variations. The inter-set write variation is minimized by associating a hot set with a cold set and then transferring the writes of the hot set to the associated cold set. Writes between the hot sets and their corresponding cold sets are called *swap writes.* For minimizing intra-set write variation, it swaps a hot block with a cold block within a set. Sequoia employs set level counters to measure if a block is hot. We compare our work with Sequoia and show experimental analysis in Section 6.



(a) Single-core Workloads



(b) Multi-core Workload Mixes

Figure 1: Write Access Pattern Across Different Workloads.

3 MOTIVATION

In SRAM cache, having nonuniform write access patterns across different cache blocks does not concern the endurance of the cache as its endurance is more than 10^{15}. But in the case of STT-RAM cache, whose endurance is only 4x10^{12}, nonuniform write access pattern becomes a bottleneck. A few STT-RAM cells reach their endurance limit faster than other cells leading to failure of the STT-RAM cache in the absence of any fault tolerance capability.

To check the write access patterns of various benchmarks from SPEC CPU2006 [5] and CRONO [1], we first give a few vital statistics about the write requests of single-core and multi-core workloads, build using SPEC CPU2006 and CRONO benchmarks (details about these workloads are given in Section 5), at the LLC for baseline. The statistics are collected at the cache block level granularity. Tables 1-2 show the workload-wise cache block level the minimum (min_{blk}) and the maximum (max_{blk}) number of writes across all the cache blocks, and the values of mean (μ_{blk}) and standard deviation (σ_{blk}) of the cache block level writes at the LLC of a single-core system and a multi-core system, respectively, for baseline. There are a few applications with very low σ_{blk}, which indicates that the number of writes at the cache block level is almost uniform around μ_{blk}. There are also a few applications with max_{blk} is almost two orders of magnitude higher than μ_{blk}, which gives a scope of balancing the writes across cache blocks. Compared to single-core workloads,

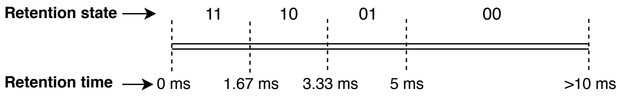


Figure 2: Representation of Retention State Based on the Retention Time.

almost all the multi-core workloads show high value of $\frac{max_{blk}}{\mu_{blk}}$, and also the capacity of the multi-core LLC is four times that of the single-core LLC (details of the experimental setup is given in Section 5), the writes can be balanced efficiently and achieve a significant reduction in the maximum number of writes at the multi-core LLC.

Figures 1a-1b show the cache block-wise write count, normalized to the maximum write count (max_{blk}) for single-core and multi-core workloads, respectively, where X-axis represents benchmarks/mixes, and Y-axis represents cache blocks. The cache blocks having higher write access count are represented by colors towards value 1 in the color bar, and the cache blocks with lower write access count are represented by colors towards value 0 in the color bar. Benchmarks such as *libquantum* and *lbm* show uniformly high write count across all the cache blocks, while benchmarks such as *hmmer* and *xalancbmk* show very high count for a few cache blocks and uniformly low write count for the remaining cache blocks. Similar trends are observed across multi-core workload mixes also. Overall, in both single-core and multi-core workloads, we can see that a few cache blocks are getting a very high number of write accesses as compared to other cache blocks. This access pattern motivates us to change the address mapping to distribute the write accesses across the cache evenly.

4 DEALING WITH WRITE-OPTIMIZED STT-RAM LLC

We first optimize the write latency of STT-RAM by reducing the cell area. Write latency optimization may increase the number of write operations due to refresh overhead. Furthermore, due to workload access patterns, there will be variations in intra-set and inter-set writes. For removing the intra-set and inter-set write variation, we disassociate the cache sets from the cache ways by changing the address mapping of the cache blocks. We then combine the address mapping technique with an efficient replacement policies to achieve uniform write accesses but without any performance degradation.

4.1 Write Optimization

One of the drawbacks of STT-RAM is its high write latency. The write latency of STT-RAM depends on the thermal barrier. A thermal barrier is a measure defined based on different properties of Magnetic Tunnel Junction (MTJ). Higher the thermal barrier, higher will be the switching current. The current required to flip the bit in an STT-RAM cell is switching current and the time duration for which switching current is applied is known as the *write pulse width*. Reducing the thermal barrier also reduces the retention time, which is the duration for which the bit is retained without a random bit-flip. So, the write latency can be optimized by reducing the thermal barrier, but in turn, it also reduces the retention time. The thermal barrier of MTJ is reduced by reducing the planar area of MTJ, changing the magnetic parameter, or reducing the thickness of the free layer [6][18][19]. Considering ten years of retention time leads to 10.2 *ns* of write latency whereas for 10 *ms* retention time, the write latency is narrowed down to 2.5 *ns*. For the latter case, if a block is there in the STT-RAM cache for more than 10 *ms*, a refresh operation has to be performed. A refresh operation performs an additional write operation to the block. We can conclude that optimizing the write latency may increase the number of write operations that can further affect the endurance of STT-RAM. We perform an empirical study on SPEC CPU2006 benchmarks and identify duration between consecutive reference to a block. On average, almost 97% of the blocks in the LLC get referenced within 10 *ms*. Based on this observation, we consider a write-optimized STT-RAM with a retention time of 10 *ms*, and hence the write latency of STT-RAM is considered as 2.5 *ns*.

4.2 Refresh Mechanism

Figure 2 shows the retention states, which are defined by partitioning the retention time. In our implementation, 10 *ms* is partitioned into four different intervals, where the last interval (represented by state 00) is for retention time greater than 5 *ms*, and the first three intervals (represented by 11, 10, and 01, respectively) are equally divided between 0 and 5 *ms*. We keep the fourth interval larger as compared to the first three intervals as a low fraction of cache blocks is retained in the STT-RAM LLC for more than 5 *ms*. We associate a 2-bit retention state with each cache block in the SRAM tag array. To refresh the cache blocks, we make use of the dynamic counter-controlled refreshing policy proposed by Sun et al. [19]. The state of every cache block is changed at the end of each interval. Only the cache blocks in state 00 are refreshed.

4.3 Disassociating Cache Sets from Cache Ways

Set-associative caches with skewed write accesses generate intra-set and inter-set write variations. The two write variations arise as a cache block that is mapped to one set cannot be mapped to a different set in the set-associative cache design; we can only map it to a different way within the same set. To reduce the write variations, we plan to disassociate cache sets from the cache ways so that a cache block can be mapped to different sets. While this idea was earlier proposed in SRAM-based skewed caches [17] to improve performance, ours is the first proposal that uses the idea for endurance improvement in STT-RAM LLC. In our cache architecture, we consider a cache way *i* (where *i* is from 0 to associativity) across all the sets collectively as *way-bank i*. As shown in Figure 3, for a 4-way set-associative cache, there will be four way-banks. A hash function is associated with each way-bank to identify a location for a given cache block. For a cache block, there are four locations after hashing its physical address with four different hash functions, all of which may lie across different sets. This address mapping may reduce variation in the number of writes across cache blocks as different hash functions are used for each way-bank.

4.4 Finding an Efficient Replacement Policy

In a set-associative cache, the LRU replacement policy is implemented by maintaining recency information among cache blocks within a set. The same cannot be implemented in our case as cache

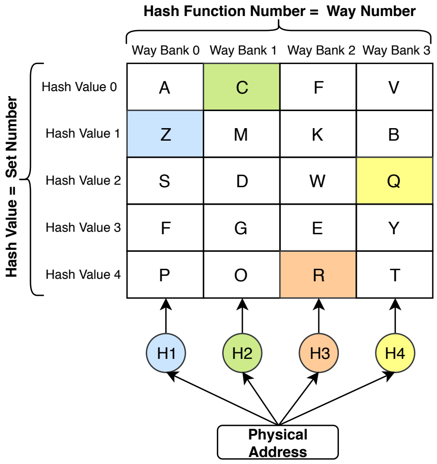


Figure 3: **Address Mapping using Hash Functions.**

ways are disassociated from cache sets unless we consider a global timestamp for each cache block, but maintaining the global timestamp incurs significant overhead. So, we need to come up with a replacement policy that should neither skew writes to a subset of cache blocks nor hurt performance. We now explore a few replacement policies with this objective.

4.4.1 Recency-Aware Replacement (RCR) Policy: We propose *recency-aware replacement* (RCR) policy, wherein, to account for the recency of the cache blocks, we introduce *recently accessed* (RA) bit with each cache block. Whenever a cache block is accessed, we set its RA bit. From among the set of potential victim cache blocks identified using the hash functions, RCR policy selects one cache block with RA bit reset as the victim cache block. If more than one cache block with RA bit reset is present, the policy randomly selects a block among them. If none of the blocks with RA bit reset, the policy randomly selects a block with its RA bit set. The policy prefers the eviction of a clean block over a dirty block.

Once the RA bit of a cache block is set, there is no possible way to reset it. It may so happen that for all the cache blocks, the RA bits are set after some time. To deal with such scenarios, we reset RA bit of all the cache blocks after a certain number of accesses to LLC. We perform an empirical study to find an optimal reset interval that provides better values for the maximum number of writes and instructions-per-cycle (IPC). Figure 5 shows the reduction in the maximum number of writes and IPC improvement for different reset intervals, normalized to baseline. Reset interval of $2K$ accesses to the LLC shows the highest reduction in the maximum number of writes, and it also improves IPC compared to other reset intervals. Thus, we consider $2K$ reset interval for our evaluations.

4.4.2 Refresh-Aware Replacement (RFR) Policy: RCR policy prefers performance enhancement to endurance improvement. If a few blocks in a set are repeatedly written, RCR policy will not select them as victim blocks, and hence the write count for those blocks increases, which can potentially limit the endurance. We now propose a new replacement policy, wherein we prefer endurance to performance.

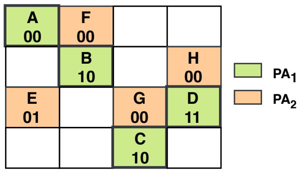


Figure 4: **Cache Block A is Selected as Victim for PA_1 and Randomly Among F, G, H for PA_2.**

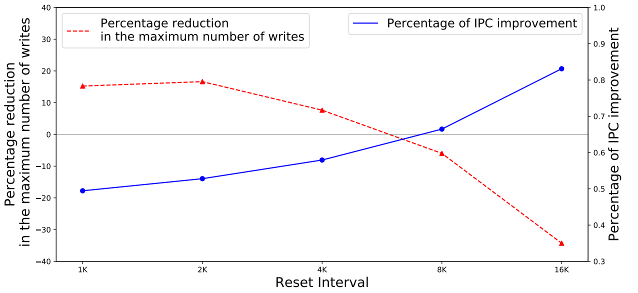


Figure 5: **Reduction in the Maximum Number of Writes at the LLC Vs IPC Improvement for Various Reset Intervals (Normalized to Baseline).**

In the refresh mechanism discussed earlier (refer to Section 4.2), we maintain retention state for selectively refreshing the cache blocks that are about to expire soon. We make use of the same information for selecting victim cache blocks in our *refresh-aware replacement* (RFR) policy. While selecting a victim from the set of cache blocks identified by hashing the physical address, the policy selects the cache block that is nearest to its expiry. If more than one block is in 00 retention state, the policy selects the victim randomly among these blocks. If none of the blocks is in retention state 00, the policy selects the block with the lowest retention state value. This replacement policy tries to minimize refresh writes. Figure 4 shows an example using RFR policy. Each cache block maintains a 2-bit retention state information. Cache blocks A, B, C, and D, with retention states as 00, 10, 10, and 11, respectively, are identified as the four possible victim candidates, from which the policy selects block A as the victim candidate.

A block will expire if it has not been written for approximately the retention time duration, i.e., 10 *ms*. But the retention states do not capture any information about the read accesses. Hence, this replacement policy may enhance the endurance of STT-RAM LLC, by minimizing the number of writes, but may not decrease the miss rate (and improve the performance) as it does not capture recency information of the blocks.

4.4.3 Refresh and Recency-Aware Replacement (FCR) Policy: In the above two replacement policies, RCR policy prefers performance to endurance, while RFR policy prefers endurance to performance. To capture the benefits offered from both the replacement policies, we propose a new replacement policy, i.e., *reFresh and reCency-aware Replacement* (FCR) policy.

FCR policy maintains a three-bit information per cache block where the first bit represents the RA bit, and the other two bits

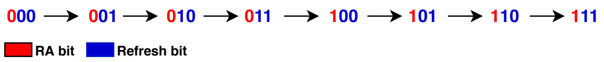


Figure 6: Order in Which Cache Blocks Get Evicted in FCR Policy. 000 is the Most Suitable Candidate while 111 is the Least Suitable Candidate.

represent the retention state. A cache block with its RA bit set indicates that it is recently accessed, while its RA bit reset indicates that it is not recently accessed. The most recently refreshed cache block will have its refresh bits as 11 while the least recently refreshed cache block will have its refresh bits as 00. FCR policy prefers "not recently accessed blocks" to "not recently refreshed blocks" as victim blocks to avoid any performance degradation. Figure 6 shows the order in which cache blocks get evicted based on the three-bit information. When there are multiple potential victim cache blocks with the same state, FCR policy selects one of them randomly as a victim by preferring clean blocks to dirty blocks. We observe that selecting victim blocks based on the recency and retention information reduces the number of maximum writes without degrading performance.

4.5 Lifetime Improvement

In an ideal case, all the cache blocks in the cache should get an equal number of write accesses so that the lifetime of each cache block degrades at the same pace. But in the actual scenario, caches exhibit non-uniform write accesses that creates write variation in the cache. In the set-association cache, due to the set and way organization, there can be intra and inter-set variation [20], but in our technique, as there is no notion of set and way, we consider the write variation of a cache block concerning all the remaining cache blocks. We compute this write variation by measuring the coefficient of write variation. We term this coefficient of write variation as *global write variation* (*GlobalV*).

$$GlobalV = \frac{1}{W_{avg}} \sqrt{\frac{\sum_{i=1}^{N} (W_i - W_{avg})^2}{N-1}}$$

where N is the number of cache blocks in the LLC, W_i is the total number of writes to a cache block i, and W_{avg} is the average number of writes across all the cache blocks.

GlobalV computes how far the write count of a cache block variate with respect to the average number of writes to all the cache blocks. For an ideal case, where all the cache blocks have an equal number of writes, *GlobalV* will be 0. *GlobalV* is inversely proportional to the endurance of STT-RAM LLC. Lower the *GlobalV*, higher the endurance of STT-RAM LLC. The maximum number of writes to a cache block may not give the correct information about the endurance enhancement of STT-RAM LLC as write variation information is not captured by it. There can be a scenario where a technique that reduces the maximum number of writes may increase the average number of writes, which indirectly affects the endurance of STT-RAM LLC. Wang et al., [20] proposed a lifetime improvement (LI) parameter for set-associative cache based on the average number of writes, intra- and inter-set write variations. We propose a similar parameter using *GlobalV* write variations.

Table 3: Simulation Parameters and Workloads.

	Simulation Parameters
Core(s)	4GHz, x86, out-of-order, 8-wide issue
SRAM L1 Cache	private: L1-I cache, 32KB, 4-way, 2 cycles private: L1-D cache, 32KB, 4-way, 2 cycles
STT-RAM LLC	single-core: 1MB, 8-way, 6 cycles (Read), 10 cycles (Write)
	4-core: 4MB (shared), 8-way, 12 cycles (Read), 13 cycles (Write)
	Workloads
1-core	SPEC CPU2006
	As (astar), Bz (bzip2), Ca (cactusADM), Cl (calculix), Ga (gamess), Gc (gcc), Ge (GemsFDTD), Gr (gromacs), H2 (h264ref), Hm (hmmer), Lb (lbm), Le (leslie3d), Li (libquantum), Mf (mcf), Mi (milc), Na (namd), Om (omnetpp), Sj (sjeng), So (soplex), Xa (xalancbmk), Ze (zeusmp)
	CRONO
	Ap (apsp), Bc (bc), Bf (bfs), Co (community), Ss (sssp)
4-core	Mix1: namd-hmmer-h264ref-gamess Mix2: gromacs-milc-omnetpp-soplex Mix3: libquantum-h264ref-lbm-omnetpp Mix4: astar-bzip2-cactusADM-GemsFDTD Mix5: astar-bzip2-soplex-xalancbmk Mix6: namd-hmmer-bzip2-mcf Mix7: leslie3d-gamess-astar-sjeng Mix8: cactusADM-sssp-apsp-bzip2 Mix9: astar-bc-apsp-h264ref Mix10: omnetpp-sssp-apsp-hmmer

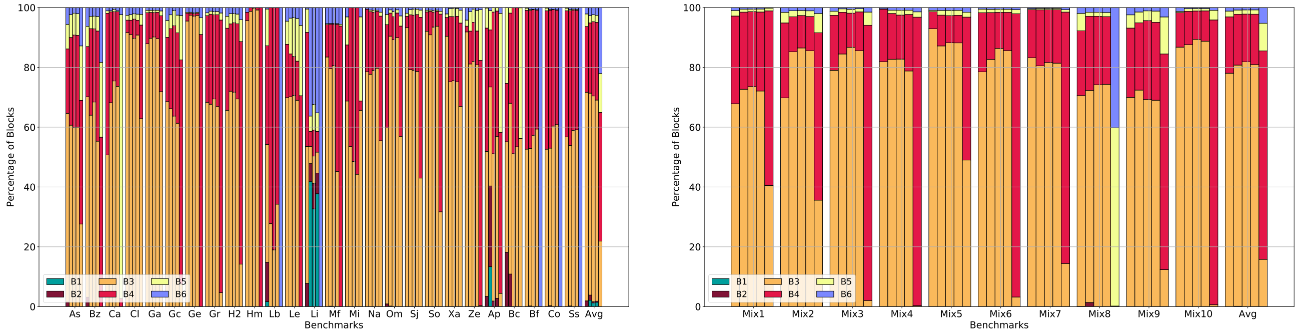
$$LI = \frac{W_{avg\_base}(1 + GlobalV_{base})}{W_{avg\_imp}(1 + GlobalV_{imp})} - 1$$

where W_{avg_base} is the average number of writes for the baseline, W_{avg_imp} is the average number of writes for the technique for which we compute the lifetime improvement. Similarly, $GlobalV_{base}$ and $GlobalV_{imp}$ are the coefficient of write variation for the baseline and the technique to be compared with, respectively.

To improve the lifetime of STT-RAM LLC, one has to reduce the average number of writes to the LLC and/or the maximum number of writes, which in turn reduces global write variation. We evaluate our techniques with baseline and Sequoia, a state-of-the-art endurance enhancement technique, in terms of reduction in the maximum number of writes, the average number of writes, and the lifetime improvement.

5 EXPERIMENTAL SETUP

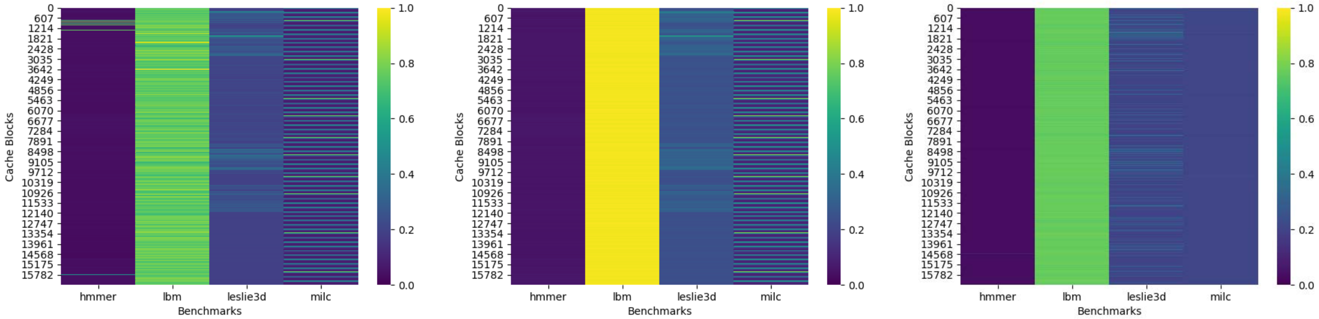
We use GEM5 simulator [4] to model and evaluate the proposed techniques. We use CACTI [3] to calculate the L1 SRAM cache latencies and NVSim [11] simulator to calculate STT-RAM LLC latency. We consider the tag array of the LLC designed using SRAM



(a) 1-core

(b) 4-core

Figure 7: Percentage of Cache Blocks Belong to Different Write-Count Bins for Baseline (1st Bar), RCR (2nd Bar), RFR (3rd Bar), FCR (4th Bar), and Sequoia (5th Bar).



(a) Baseline

(b) Sequoia

(c) RFR Policy

Figure 8: Write Access Pattern for Four Different Benchmarks.

and the data array of the LLC designed using STT-RAM. Caches are miss allocate and mostly inclusive. Table 3 provides the simulation parameters used in our experiments. For optimized STT-RAM write latency of 2.5 *ns*, we consider the retention time of STT-RAM as 10 *ms*. We warm up the system for 1 billion instructions and then simulate 5 billion instructions for the single-core applications and 1 billion per workload for the multi-core applications. For multi-core workloads, if an application finishes its 1 billion instruction, then it will keep on running until all the other applications finish their 1 billion instructions.

We consider 21 benchmarks from SPEC CPU2006 [5] and five from CRONO [1], as shown in Table 3, for our experimental analysis. We use acronyms to represent each workload for the simplicity of presenting it in the graphs. We also create 4-core mixes out of these workloads for multi-core simulations.

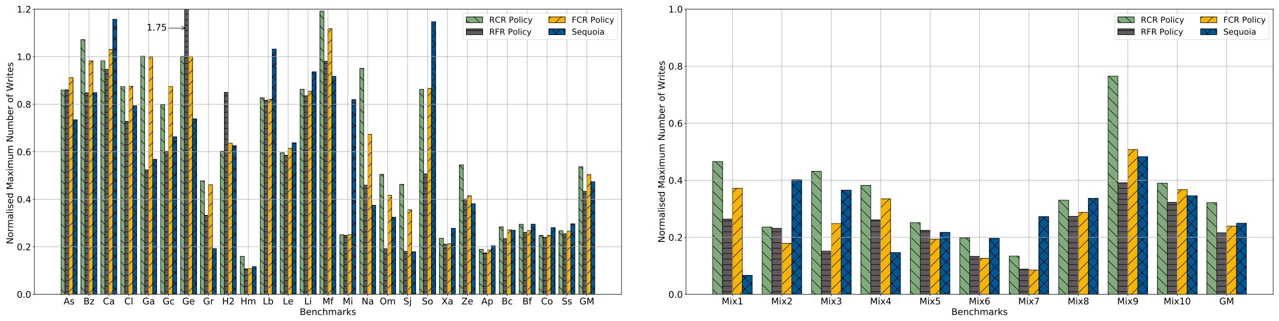
In our baseline setup, we consider SRAM L1 cache and STT-RAM LLC and use set-associative mapping at both the levels with LRU replacement policy. In our technique, we consider the address mapping technique as given in Skewed Cache [17] for the STT-RAM LLC and use three different replacement policies for the evaluations. In this work, we consider only two levels of cache because the endurance of STT-RAM is not much of a concern at L3 cache as most of the writes are filtered by L1 and L2 caches. But our technique is applicable for any levels of the cache hierarchy.

6 RESULT ANALYSIS

We discuss the results for single-core systems followed by multi-core systems. For our analysis, we consider the following parameters: the maximum number of writes, the average number of writes, endurance enhancement, misses per kilo instructions (MPKI) at the LLC, and instruction per cycle (IPC). For multi-core workloads, IPC value for an application is recorded when it completes its 1 billion instructions. We compare our techniques with baseline and Sequoia [7] based on these parameters.

6.1 Write-Count Distribution at the LLC

To get an insight into the number of writes at the LLC, we consider six write-count bins with each bin having a specific write-count interval, B1:$\leq (\mu_{blk} - 2\sigma_{blk})$; B2:$(\mu_{blk} - 2\sigma_{blk}, \mu_{blk} - \sigma_{blk}]$; B3:$(\mu_{blk} - \sigma_{blk}, \mu_{blk}]$; B4:$(\mu_{blk}, \mu_{blk} + \sigma_{blk}]$; B5:$(\mu_{blk} + \sigma_{blk}, \mu_{blk} + 2\sigma_{blk}]$; and B6:$> (\mu_{blk} + 2\sigma_{blk})$, where μ_{blk} and σ_{blk} are the mean and standard deviation of the cache block level writes at the LLC for baseline, respectively. A cache block blk is said to belong to a write-count bin X, if the total number of writes to blk is within the



(a) 1-core

(b) 4-core

Figure 9: Normalized Maximum Number of Writes at the LLC with respect to Baseline (Lower is Better).

interval of X. The write-count also includes refresh writes for all the techniques with additional swap writes for Sequoia.

Figure 7 shows the percentage of cache blocks belong to different write-count bins for single-core and multi-core workloads across five different techniques. For each workload in the figure, there are five bars, where the first and the last bars represent baseline and Sequoia, and the middle three are for our policies, RCR, RFR, and FCR, respectively. As different policies handle victim selection differently, from the figure, we see the different write-count distribution for each of these policies. A higher percentage of cache blocks belong to small write-count bins will be beneficial for reducing the maximum write-count and average write-count. Because of skewed cache organization considered in our techniques, our techniques spread writes across different blocks, and as seen from the figure, our techniques achieve a higher percentage of cache blocks belong to low write-count bins compared to Sequoia and baseline. On the other hand, due to swap writes, Sequoia increases the percentage of cache blocks belong to large write-count bins. Compared to single-core workloads, multi-core workloads show a meager percentage of cache blocks belong to B5 and B6 for all the techniques except Sequoia.

By considering four benchmarks, *hmmer*, *lbm*, *leslie3d* and *milc*, we show the write access patterns for baseline, RFR policy and Sequoia in Figure 8. The block-wise write count of all the benchmarks is normalized with respect to the maximum write count of a block of the baseline.

In *hmmer*, most of the blocks have a uniform write access pattern except a few blocks; we can see green color peaks in Figure 8a. As there are a very few peaks (refer to Figure 8a) and the mean number of writes to a block at the LLC is close to the minimum number of writes (refer to Table 1), both RFR policy and Sequoia can make the write accesses uniform, as shown in Figures 8c and 8b, respectively.

For *lbm*, in Figure 8a, we can observe that the majority of the blocks have a very high write count, in turn, making the write access pattern uniform. In Figure 8c, RFR technique can redirect those yellow peaks to the block having less write count and make the write accesses uniform. Whereas, if we observe Sequoia in Figure 8b, due to a high number of swap writes, the write count of each block increases uniformly across all the blocks. Furthermore, in Sequoia, for blocks numbered between 6032 and 6047, the write count is greater than the maximum write count of a block in the baseline. As these high write count values are represented in white color, and since there are more than 16k blocks in the heatmap, hence the color is not visible.

For *leslie3d* and *milc*, the write access patterns of baseline and Sequoia are shown in Figure 8a and 8b, respectively. We can observe the same write access pattern for both the techniques as they both use the set-associative cache organization. But due to associating multiple sets, Sequoia reduces the write intensity for the maximum number of writes. On the other hand, due to the skewed cache organization in RFR policy, we can observe that not only peaks are neutralized (refer to Figure 8c) but also the write access pattern is quite different from baseline and Sequoia. From the above analysis, we can conclude that the skewed cache organization coupled with our replacement policies perform much better in reducing the maximum number of writes.

6.2 Maximum Number of Writes at the LLC

If the total number of writes to a cache block reaches the write endurance limit of STT-RAM, it may fail the entire cache. By reducing the maximum number of writes to cache blocks, we can enhance the endurance of STT-RAM LLC. Figure 9a shows the maximum number of writes, normalized to baseline, for different techniques on a single-core system. Among all the benchmarks, *hmmer* achieves a significant reduction in the maximum number of writes across all the techniques compared to baseline. This can also be seen in Figure 7a for *hmmer*, where cache blocks from write-count bins B5 and B6 are missing for all the techniques except baseline. All the CRONO benchmarks as well as *xalancbmk* also show a very high reduction in the maximum number of writes across all the techniques compared to baseline.

The benchmarks that show high write variation in baseline achieve better write leveling using our techniques. Overall, for 19 out of 26 benchmarks, at least one of our three techniques (i.e., RFR, RCR, and FCR) outperforms baseline and Sequoia [7]. We also observe from the figure that for 7 out of 26 benchmarks, Sequoia outperforms our techniques. The reason for this behavior is the way Sequoia is leveling the writes. Sequoia uses counters to track the total number of write accesses to a set, and it swaps blocks between hot and cold sets using these counter values, whereas our techniques level writes only while finding a victim block. So if a benchmark has a low miss rate, our techniques cannot get enough opportunity to write leveling.

On the other hand, for some benchmarks, both Sequoia and our techniques incur higher maximum number of writes compared to that of baseline: Sequoia for *cactusADM*, *lbm*, and *soplex*; RFR policy for *GemsFDTD*; RCR policy for *bzip2* and *mcf*; and FCR policy for *cactusADM* and *mcf*. After analyzing the write access patterns of *cactusADM* and *soplex*, we observe that there are a few blocks that are highly written compared to the other blocks (refer to Figure 1a), and Sequoia cannot handle such behavior efficiently. In the case of *lbm*, as min_{blk} and max_{blk} at the LLC is very close (refer to Table 1), Sequoia is not able to find a suitable cold set for each hot set, as a result, it increases the number of writes (and all the cache blocks fall into B6 bin, refer to Figure 7a) when a set is associated and disassociated again and again. Both RCR and FCR policies consider recency information when selecting victim candidates. But as some of the cache blocks in *mcf* and *bzip2* are repeatedly accessed, they will not be selected as victim candidates by these two replacement policies, which in turn increases the number writes to these cache blocks. GemsFDTD has a high number of writes compared to reads with few blocks frequently written, RFR policy tries to retain these block, and hence, it increases the number of writes to these cache blocks.

If we consider geometric mean across all single-core workloads, all the techniques, including Sequoia, achieve a significant reduction in the maximum number of writes over baseline. As RFR policy exploits refresh-awareness in selecting victim candidates, it outperforms all the other techniques and achieves 56.64% reduction in the maximum number of writes, followed by Sequoia, which achieves 52.6% reduction. Lack of refresh awareness costs RCR policy losing out in the competition, while recency and refresh awareness ensure FCR policy performs better than RCR policy.

Both Sequoia and our techniques can efficiently utilize large cache sizes for balancing the writes by spreading the writes across different cache blocks. This is evident from the multi-core workload results shown in Figure 9b. Because of balancing the writes, we see that all the techniques achieve a significant reduction in the maximum number of writes at the LLC with RFR being the most efficient one. Here also, RFR policy achieves the highest reduction of 78.4%, whereas Sequoia achieves a reduction of 75%.

6.3 Average Number of Writes at the LLC

While enhancing the endurance of STT-RAM LLC by reducing the maximum number of writes, Sequoia drastically increases the average number of writes at the LLC. We can observe from Figure 7 that Sequoia increases the number of cache blocks in write-count bins B4, B5, and B6, as compared to baseline and our techniques, which in turn increases the total number of writes, and thus the average number of writes also increases. The average number of writes at the LLC affect the dynamic write energy of the LLC. Figure 10 shows workload-wise the average number of writes on single-core and multi-core systems for different techniques normalized to baseline. Due to a large number of swap operations that Sequoia performs for write leveling, it increases the average number of writes significantly compared to baseline. But, our techniques perform almost the same as that of baseline in most of the benchmarks (a slight variation in the average number of writes compared to baseline is due to the skewed cache organization considered in our techniques). In the case of *hmmer* and *omnetpp*, due to the skewed cache organization considered in our techniques, the writes are spread across different cache blocks, due to which we see a reduction in the miss rate (refer to Figure 12a), the average number of writes (refer to Figure 10a), and the maximum number of writes (refer to Figure 9a) at the LLC. Reduction in the number of writes can also be observed from Figure 7a.

We observe the similar behavior for multi-core mixes also (refer to Figure 10b). On average (geometric mean), compared to baseline, RFR policy achieves a reduction in the average number of writes by 4.18% for single-core and 7.17% for multi-core, while Sequoia increases the average number of writes by 27% for single-core and 37.1% for multi-core systems.

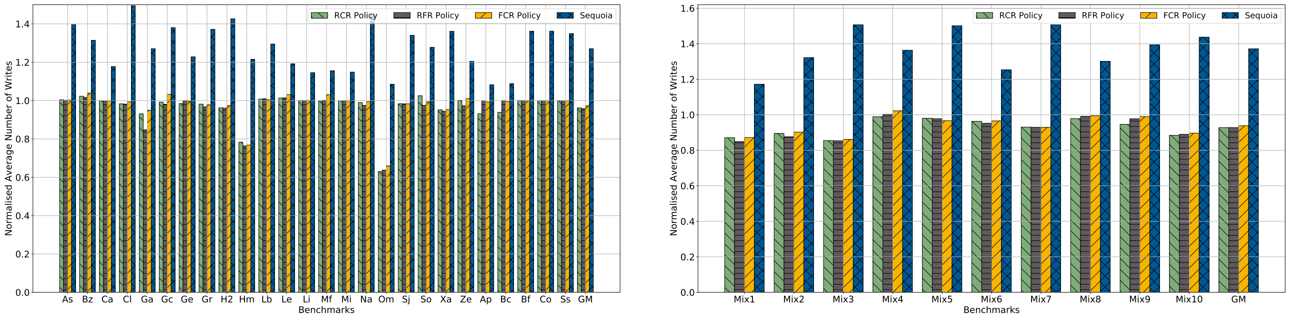
6.4 Endurance Enhancement

Endurance of the STT-RAM LLC is enhanced by increasing *Lifetime Improvement* (LI) parameter (refer to Section 4.5). Endurance enhancement signifies that if a cache lasts for x years with w writes performed on the cache, after applying the endurance enhancement technique the same cache may last for $x + \delta$ years with $w + \gamma$ writes performed on the cache, where δ and γ are some positive values.

Unlike the endurance enhancement that is captured by a reduction in the maximum number of writes, LI captures endurance enhancement based on the maximum number of writes, coefficient of write variation and the average number of writes. Figure 11 shows the LI of our policies and Sequoia with respect to baseline. Our techniques outperform Sequoia across different workloads, except for *calculix* and *gamess*. For these two benchmarks, as the MPKI (misses per kilo instructions) values are low (refer to Figure 12a), our techniques are less effective in balancing writes across cache blocks compared to Sequoia. We can also observe from Figures 9a and 10a that the gap between the maximum and the average number of writes is higher for our techniques compared to Sequoia, due to which variation in the write counts across cache blocks will be high. As *GlobalV* is affected by an inefficient balancing of writes, our techniques incur lifetime degradation compared to Sequoia.

We observe that there is a huge improvement in the lifetime of *hmmer*, *milc*, *namd*, *omnetpp*, *sjeng*, and *xalancbmk* for our techniques. The reason for such a huge improvement is the reduction in the maximum number of writes, which reduces the GlobalV parameter.

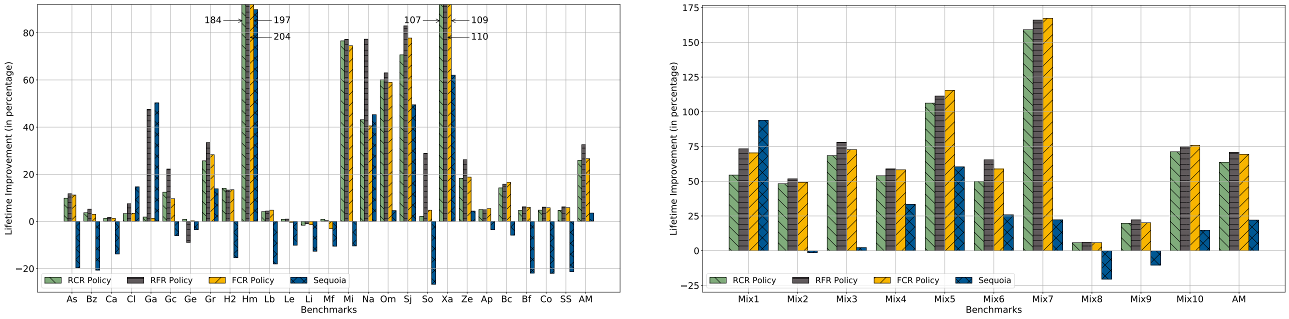
There are a few benchmarks where the lifetime degrades for our techniques compared to baseline. In the case of *libquantum*, even though the average number of writes for our techniques is equal to that of baseline (refer to Figure 10a), due to the skewed cache organization considered in our techniques, a large number of cache blocks are present in write-count bins B1 and B6 (refer to Figure 7a). Whereas in the case of baseline, a large number of cache blocks belong to B3 and B4, which in turn increases the *GlobalV* parameter for our techniques, and hence the degradation in the lifetime. In the case of *GemsFDTD*, due to a considerable increase in the maximum number of writes for RFR policy (refer to Figure 9a), the *GlobalV* parameter value increases, which in turn degrades the lifetime. In the case of *mcf*, both the maximum number of writes (refer to



(a) 1-core

(b) 4-core

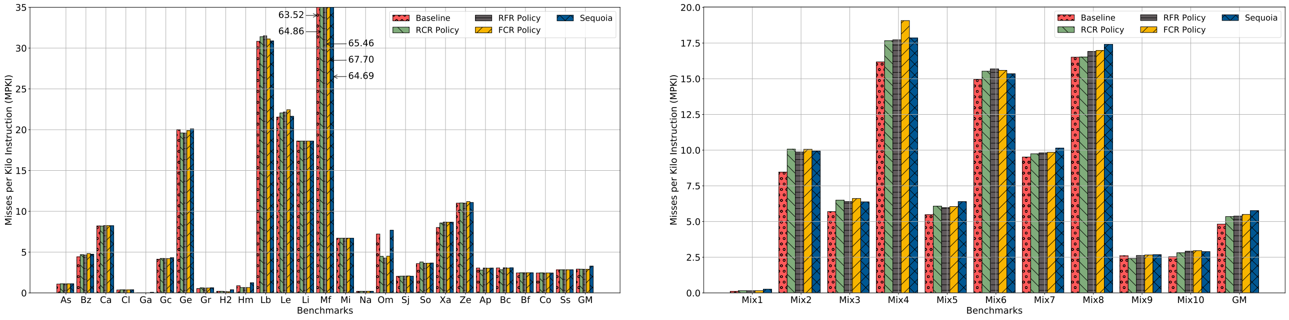
Figure 10: Normalized Average Number of Writes at the LLC with respect to Baseline (Lower is Better).



(a) 1-core

(b) 4-core

Figure 11: Lifetime Improvement with respect to Baseline (Higher is Better).



(a) 1-core

(b) 4-core

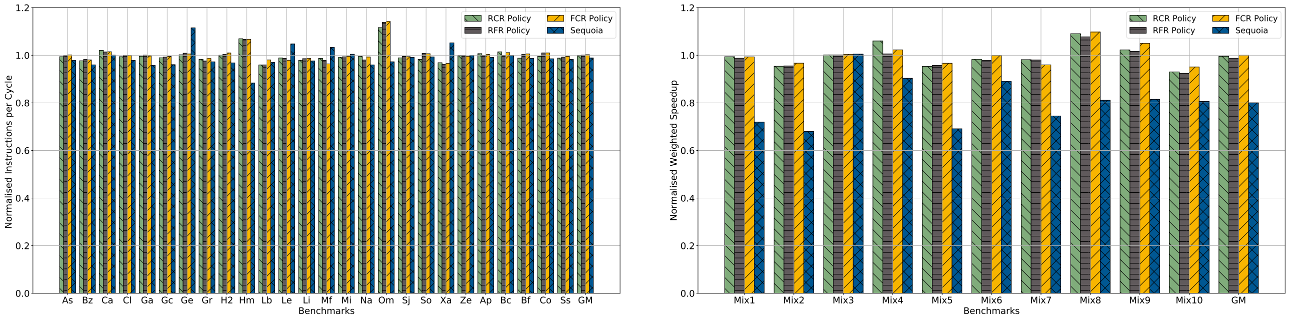
Figure 12: Misses per Kilo Instruction (MPKI) at the LLC (Lower is Better).

Figure 9a) and the average number of writes (refer to Figure 10a) for FCR policy are higher than that of baseline. Furthermore, around 45% of the cache blocks each belong to write-count bins B3 and B4 for FCR policy while it is 85% and 5% for baseline (refer to Figure 7a), which effects the *GlobalV* parameter. Because of these reasons, the lifetime is degraded for FCR policy compared to baseline. On the other hand, for Sequoia, the lifetime degrades for 17 out of 26 single-core workloads and 3 out of 10 multi-core mixes.

On average (arithmetic mean), RFR technique achieves the maximum LI of 32.58% and 70.76% for single-core and multi-core workloads, respectively, while, Sequoia achieves only 3.55% and 22.05%, respectively.

6.5 Misses per Kilo Instructions at the LLC

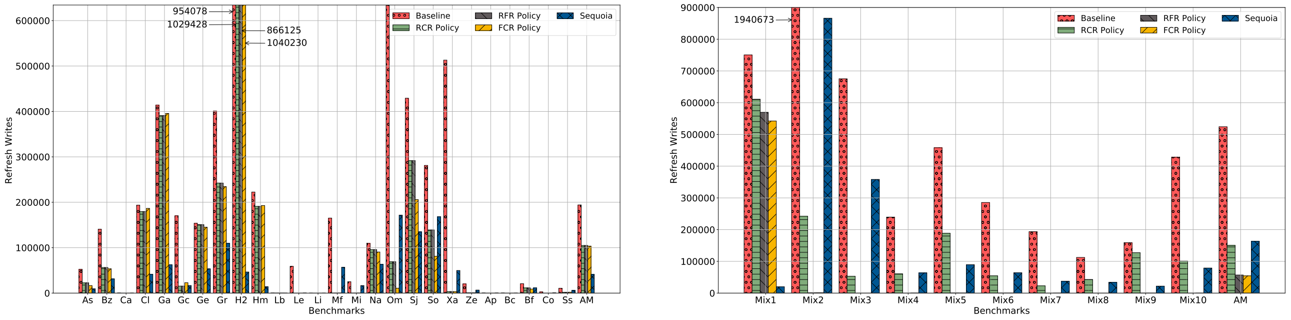
Miss rate depends on two factors: cache organization and replacement policy. Baseline and Sequoia use the set-associative cache organization with LRU replacement policy. In our techniques, we



(a) 1-core

(b) 4-core

Figure 13: Normalized Performance with respect to Baseline (Higher is Better).



(a) 1-core

(b) 4-core

Figure 14: Refresh Writes at the LLC (Lower is Better).

consider the skewed cache organization. RFR policy selects a victim candidate based on the retention state information of the block, RCR policy selects a victim candidate based on the recency information of the block provided by RA bit of the block, and FCR policy finds the most suitable replacement block using the retention and recency state information. Figure 12a shows the comparison of misses per kilo instructions (MPKI) for all the techniques. A few benchmarks such as *omnetpp*, *GemsFDTD*, *hmmer*, *lbm*, *leslie3d*, *bzip*, and *xalancbmk* show a different MPKI values compared to baseline and Sequoia, primary reason for this behavior is the skewed cache organization in our techniques, where data mapping in our techniques can be different from that of baseline and Sequoia. On average (geometric mean), our techniques achieve negligible reduction (less than 1.2%) in the miss rate compared to baseline for single-core systems. Sequoia associates and disassociates a set with another set to reduce the maximum number of writes, and during association of two sets, sometimes it may not be possible to apply the LRU policy for victim selection [7], due to which Sequoia incurs 12.8% higher miss rate compared to baseline.

In the case of multi-core scenario, as shown in Figure 12b, on average (geometric mean) our techniques incur around 10%-14% higher miss rate compared to baseline, while Sequoia incurs around 20% higher miss rate compared to baseline.

6.6 Instructions per Cycle

While enhancing the write endurance of STT-RAM LLC, the performance of the system should not be affected significantly. As both RFR and FCR policies consider refresh state information while selecting a victim, there is a possibility of evicting a block that is going to be required shortly. In the case of Sequoia, there is a situation where it is not possible to apply the LRU replacement policy because of which the performance may degrade. Figure 13a shows single-core performance comparison with respect to baseline. Our techniques achieve almost the same performance (−0.14% to 0.25%) as that of baseline, while Sequoia incurs 1% performance degradation compared to baseline.

For multi-core workloads, we consider weighted speedup (WS) values [15] for comparing our techniques with baseline for 4-core configuration. The weighted speedup is defined as follows:

$$WS = \sum_{i=1}^{N} \frac{IPC_i^{shared}}{IPC_i^{alone}}$$

where N is number of benchmarks per mix, IPC_i^{shared} is the IPC value of a benchmark i, when it completes 1 billion instructions while executing along with other benchmarks in the mix. IPC_i^{alone} is the IPC value of a benchmark i when it executes in isolation.

Due to a higher miss rate as reported in Figure 12b, Sequoia incurs a penalty of 19.93%, while it is around 1.25% for our techniques as compared to baseline (refer to Figure 13b). The reason for the high degradation in the performance of Sequoia is because of the extra latency spent on finding a cold set among 64 different sets.

6.7 Refresh Writes at LLC

We need to perform a write on a block that is going to be expired to retain the validity of the block, and this comes at the cost of a refresh write. Figure 14a shows the number of refresh writes for different techniques in single-core systems, here we have not counted the refresh writes for a block which is not accessed after its allocation at LLC. Since baseline and RCR are refresh unaware, hence they will be less efficient in reducing the refresh writes. RFR and FCR policies aim at minimizing the refresh writes at the time of victim selection. Benchmarks that have high MPKI give more room for minimizing the number of refresh writes. On the other hand, if a benchmark has a low MPKI, it becomes difficult to minimize the refresh writes; benchmarks that show this trend are *calculix*, *gamess*, *gromacs*, *h264ref*, *hmmer*, *namd*, *sjeng*, *soplex*, and *bfs*. In RFR policy, if all the victim candidates are about to expire, i.e., they are in retention state 00, then a victim is selected randomly. As FCR policy has additional information in terms of refresh awareness for selecting the most suitable candidate, it performs slightly better than RFR policy in reducing the refresh writes for some benchmarks. In the case of Sequoia, whenever the counter of a set saturates, the next accessed block is written to its associated set. Due to the regular shifting of the block from a set to its associated set, the number of refresh writes is minimized.

Figure 14b shows the number of refresh writes for different techniques on multi-core systems. In the case of *Mix1*, which comprises of *namd*, *hmmer*, *h264ref*, *gamess*, as all the benchmarks show a very low MPKI, there is not enough opportunity for minimizing the refresh writes for both RFR and FCR policies. Sequoia incurs a large number of refresh writes in *Mix2* as the number of read hits in *Mix2* is the highest among all mixes; therefore, most of the blocks are repeatedly shifted to MRU position, and these blocks will require a refresh write. On average (arithmetic mean), for FCR policy, we observe a reduction of 67.19% and 92.75% for single-core and multi-core workloads, respectively.

7 CONCLUSION AND FUTURE WORK

We observed that optimizing write latency may adversely affect the endurance of STT-RAM cache. The endurance enhancement of write-optimized STT-RAM is more challenging as it comes with the overhead of refresh writes, which further impact the endurance of STT-RAM. To improve the endurance of STT-RAM last level cache (LLC), we focussed on reducing the write variation across cache blocks. Instead of conventional set-associative cache architecture, we considered the skewed cache architecture and proposed a family of replacement policies that are refresh and recency aware. We showed through experimental analysis that without considering refresh awareness, the recency-aware replacement policy is effective in improving the endurance. The refresh-aware replacement policy is effective in reducing the average number of writes, reducing the maximum number of writes, and enhancing the lifetime of STT-RAM LLC, which in turn improve the endurance. As the refresh-aware policy reduces the average number of writes, it also results in the reduction of the dynamic energy consumption. While lowering the maximum number of writes, Sequoia, the state-of-art endurance improvement technique, significantly increases the average number of writes, which can affect the lifetime of STT-RAM LLC, and increases the dynamic energy consumption. Our refresh-aware policy improves the lifetime of STT-RAM LLC by 32.5% for single-core and 70.7% for muti-core compared to STT-RAM LLC without wear-leveling. To the best of our knowledge, ours is the first technique to study the effect of write optimization on the write endurance of STT-RAM.

Reducing the retention time reduces the write efforts in terms of latency and write energy, but it introduces the overhead of refresh writes. This aspect can be explored as a future work where one can analyze the trade-off between the write effort and cost of refresh writes.

REFERENCES

[1] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan. 2015. CRONO: A Benchmark Suite for Multithreaded Graph Algorithms Executing on Futuristic Multicores. In *2015 IEEE International Symposium on Workload Characterization*. 44–55. DOI: http://dx.doi.org/10.1109/IISWC.2015.11

[2] S. Asadi, A. M. H. Monazzah, H. Farbeh, and S. G. Miremadi. 2017. WIPE: Wearout Informed Pattern Elimination to Improve the Endurance of NVM-based Caches. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 188–193. DOI: http://dx.doi.org/10.1109/ASPDAC.2017.7858318

[3] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (June 2017), 25 pages. DOI: http://dx.doi.org/10.1145/3085572

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. DOI: http://dx.doi.org/10.1145/2024716.2024718

[5] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17. DOI: http://dx.doi.org/10.1145/1186736.1186737

[6] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das. 2012. Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs. In *DAC Design Automation Conference 2012*. 243–252. DOI: http://dx.doi.org/10.1145/2228360.2228406

[7] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad. 2016. Sequoia: A High-Endurance NVM-Based Cache Architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 3 (March 2016), 954–967. DOI: http://dx.doi.org/10.1109/TVLSI.2015.2420954

[8] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie. 2010. Energy- and endurance-aware design of phase change memory caches. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*. 136–141. DOI: http://dx.doi.org/10.1109/DATE.2010.5457221

[9] Jun Yang, Youtao Zhang, and R. Gupta. 2000. Frequent value compression in data caches. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*. 258–265. DOI: http://dx.doi.org/10.1109/MICRO.2000.898076

[10] Y. Kim, S. R. Lee, D. Lee, C. B. Lee, M. Chang, J. H. Hur, M. Lee, G. Park, C. J. Kim, U. Chung, I. Yoo, and K. Kim. 2011. Bi-layered RRAM with unlimited endurance and extremely uniform switching. In *2011 Symposium on VLSI Technology - Digest of Technical Papers*. 52–53.

[11] S. Li, L. Liu, Peng Gu, C. Xu, and Yuan Xie. 2016. NVSim-CAM: A circuit-level simulator for emerging nonvolatile memory based Content-Addressable Memory. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–7. DOI: http://dx.doi.org/10.1145/2966986.2967059

[12] Sparsh Mittal and Jeffrey S. Vetter. 2014. EqualChance: Addressing Intra-set Write Variation to Increase Lifetime of Non-volatile Caches. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/inflow14/workshop-program/presentation/mittal

[13] S. Mittal and J. S. Vetter. 2016. EqualWrites: Reducing Intra-Set Write Variations for Enhancing Lifetime of Non-Volatile Caches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 1 (Jan 2016), 103–114. DOI: http://dx.doi.org/10.1109/TVLSI.2015.2389113

[14] M. K. Qureshi, S. Gurumurthi, and B. Rajendran. 2011. *Phase Change Memory: From Devices to Systems*. Morgan Claypool. https://ieeexplore.ieee.org/document/6813358

[15] M. K. Qureshi and Y. N. Patt. 2006. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 423–432. `DOI`:http://dx.doi.org/10.1109/MICRO.2006.49

[16] Elizabeth Reed, Alaa R. Alameldeen, Helia Naeimi, and Patrick Stolt. 2017. Probabilistic Replacement Strategies for Improving the Lifetimes of NVM-based Caches. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '17)*. ACM, New York, NY, USA, 166–176. `DOI`:http://dx.doi.org/10.1145/3132402.3132433

[17] André Seznec. 1993. A Case for Two-way Skewed-associative Caches. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA '93)*. ACM, New York, NY, USA, 169–178. `DOI`:http://dx.doi.org/10.1145/165123.165152

[18] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 50–61. `DOI`:http://dx.doi.org/10.1109/HPCA.2011.5749716

[19] Z. Sun, X. Bi, H. Li, W. Wong, Z. Ong, X. Zhu, and W. Wu. 2011. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 329–338.

[20] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi. 2013. i^{2}WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 234–245. `DOI`:http://dx.doi.org/10.1109/HPCA.2013.6522322

[21] Z. Wang, D. A. JimÃĺnez, C. Xu, G. Sun, and Y. Xie. 2014. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 13–24. `DOI`:http://dx.doi.org/10.1109/HPCA.2014.6835933

[22] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. 2009. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 34–45. `DOI`:http://dx.doi.org/10.1145/1555754.1555761

[23] S. Yazdanshenas, M. R. Pirbasti, M. Fazeli, and A. Patooghy. 2014. Coding Last Level STT-RAM Cache for High Endurance and Low Power. *IEEE Computer Architecture Letters* 13, 2 (July 2014), 73–76. `DOI`:http://dx.doi.org/10.1109/L-CA.2013.8

[24] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. 2009. Energy reduction for STT-RAM using early write termination. In *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*. 264–268.