2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP) | 978-1-6654-2701-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASAP52443.2021.00032

ABACa: Access Based Allocation on Set Wise Multi-Retention in STT-RAM Last Level Cache

Sukarn Agarwal
Dept. of CSE, IIT (BHU), Varanasi, India
sukarn.cse@iitbhu.ac.in

Shounak Chakraborty
Dept. of Computer Science, NTNU, Trondheim, Norway
shounak.chakraborty@ntnu.no

***Abstract*—Exhibition of potential advantages of high density, non-volatility, and low static power consumption makes STT-RAM a credible successor to SRAM in caches. However, higher write energy and latency of the STT-RAM limit its potential towards commercial usage. Relaxation of STT-RAM's retention time can be a viable solution to alleviate these obstacles by reducing both write time and energy. However, significant reduction in retention time might lead to premature expiry of the blocks requiring frequent refreshes or write-backs, which can incorporate unnecessary stalls along with the increased miss-rate.**

This paper proposes *ABACa*, an approach that logically bifurcates a cache set-wise for two different retention times where cache blocks are segregated upon their arrival and placed in the corresponding set, accordingly. In particular, if a block's arrival is triggered by a read miss, the block is placed into a set with a higher retention time, called as *read-set*. On the other hand, the block is placed into a *write-set* having a lower retention time, if the block's arrival is caused by a write miss. Our empirical analysis shows that, *ABACa* achieves a significant improvement of 40.75% in miss-rate and 61.35% EDP (Energy Delay Product) gain compared to baseline multi-retention STT-RAM-based and SRAM-based last level caches, respectively.

***Index Terms*—Multi-Retention Time, STT-RAM, Cache Memory, Block Placement**

I. Introduction

The low density, poor scalability, and high leakage power consumption of the SRAM in sub-32-nm technology have motivated designers and architects to look into other emerging avenues of memory technologies. Among the different memory technologies, Non-Volatile Memories (NVMs), such as Spin-Transfer Torque RAM (STT-RAM) [1], Phase Change Memory (PCM) [2], Flash memory, etc. are the most promising ones. Out of all these NVMs, STT-RAM is the most potential SRAM alternative due to its low leakage power, high endurance, and comparable read access times. However, the existing downsides of STT-RAM include high write energy and high write latency compared to SRAM, which can be further mitigated by the relaxation of retention time. Such mitigation is usually implemented by reducing the magnetic cell's surface area composing a unit of STT-RAM. In addition, the current needed to write into the memory cell is curtailed, which lowers the write latency and energy [3]. However, a lower retention time of only a few milliseconds is a potential trade-off, but such a lower value can incur the overheads of early write-backs to the lower level memory or refreshing of the cache blocks due to premature expiry. While refreshing, the block is first written into a buffer and then rewritten back into the cache [4]. The usage of such Dynamic Refresh Schemes (DRSs) might impose extra performance and energy overheads [5] due to bank contention and refresh energy.

Solving the problems of STT-RAM by employing DRSs or write-backs is possible at the cost of higher energy along with a fall in the hit rate. Most of the prior work have instead chosen a hybrid SRAM/STT-RAM setup [6] or used a multi-retention based STT-RAM [7], [8], where a cache is composed of multiple partitions with various retention times. In both cases, DRS and write-back induced overheads are avoided within a hybrid structure by migrating data from the lower retention areas to the SRAM or to the higher retention zones of the STT-RAM. Since such hybrid caches draw a large amount of energy and have a low density due to their SRAM counterpart, in this paper, we select a multi-retention STT-RAM. Our approach, *ABACa*, logically bifurcates an STT-RAM cache set-wise based on two different retention times (read and write retention times). Depending upon the types of misses, the fetched blocks are placed into the corresponding set. In other words, if a read miss triggers a block's arrival, it will be placed into a *read-set* that has a higher retention time. Otherwise, the block is placed into a *write-set*, having a lower retention time, if a write miss causes the block fetching.

The contributions of *ABACa* can be listed as follows: (1) We propose a novel cache architecture that logically partitions an STT-RAM cache set-wise for two different retention times along with a request-type aware block allocation scheme. (2) Through a full system simulation, we further empirically validate the multiple figures of merits of *ABACa*. Our simulation result shows, *ABACa* achieves a significant improvement of 40.75% in miss-rate and 61.35% gain in energy delay product (EDP) compared to the baseline multi-retention STT-RAM-based and SRAM-based last level caches (LLCs), respectively.

II. Background

Prior work that attempted to overcome high write energy and write latency issues of the STT-RAM can be broadly classified into two categories. Techniques from the first category trim the number of writes incurred to cache by incorporating several mechanisms. One such technique, DASCA [9] predicts dead writes to those cache blocks, that are written into but are never accessed subsequently, and attempts to bypass accesses to them. The Flip-N-Write [10] targets to distinguish and flips only those selective bits of a block in the cache that need to be inverted during writing back and allocation in the non-volatile

978-1-6654-2701-2/21/$31.00 ©2021 IEEE
DOI 10.1109/ASAP52443.2021.00032

cache. Furthermore, minimizing write-counts improves the lifetime of STT-RAM, which is especially crucial in such caches, particularly to handle write-intensive benchmarks.

Another group of approaches mostly focus on the relaxation of the retention time of STT-RAM towards reducing both write latency and energy. However, the major downside of lowering retention time is that the cache block data is valid only for a limited period, which might increase the cache miss rate through the premature block evictions. To limit the reduced retention time from affecting the cache performance, many prior research attempts [11], [12] looked into DRS. In DRS, the cache entries on the verge of expiring are primarily written into a buffer and subsequently are rewritten back into the cache, which incurs the additional overhead of unnecessary migrations. To reduce such migrations, MirrorCache [12] proposed two identical caches where the data close to expiry are switched between the two caches. But, implementing MirrorCache might result in higher storage overheads that limits the density advantage of STT-RAM. However, the refresh energy in DRS can share a significant percentage (as high as **35%** [5]) of the cache's total energy requirements. Hence, *ABACa* avoids DRS and focuses primarily on write-backs.

HALLS [5] attempts to train the applications with virtual banks of distinct retention times and chooses the bank with the optimal EDP for the rest of the application's execution. But, such approach cannot account for applications' dynamic nature, which can be altered noticeably between frequent and sparse writes. However, most of the prior work partitioned the cache way-wise without considering the access-type of the loaded blocks. Furthermore, it has been experimentally observed that the access-type of the blocks inside the cache set are heavily dependent on the type of access that initially loads the blocks [13]. Hence, in this work, we propose a novel cache architecture that bifurcates the cache set-wise and loads the different cache blocks to different retention sets based on their load access types. To the best of our knowledge, *ABACa is the first work that logically partitions an STT-RAM cache set-wise to offer multi-retention time*, which makes it distinguishable from the prior work where the cache is partitioned way-wise.

III. Motivation & Proposed Technique

1) ***Motivation****:* Before proposing *ABACa*, we have empirically analyzed retention time requirements for a set of PARSEC benchmarks by simulating in gem5 [14]. Table I shows the diversities in average access retention times for both read and write accesses of the cache-sets. For all of these benchmarks, we have considered four ranges of retention times. Table I reports the percentages of cache sets that belong to the retention time range for average time taken between two consecutive *(a) Read* and *(b) Write* accesses.

From the table we observe that for any type of access, the average retention time for all of these benchmarks mostly lies within the range of $0-100ms$ (95.3% for read and 95.5% for write). Notably, the higher values of cache set percentage (82%) for *Write* accesses in the lowest retention group of $0-10ms$ for all of these benchmarks can potentially motivate us to make a separate group of cache sets for the write-intensive blocks. Whereas the average read access cache set percentage for the retention range of $0-10ms$ is around 69%. This practically implies that the read-intensive blocks can be placed in higher retention-time based group of cache sets, whereas write intensive ones might be placed in the group of lower ones. *The prime reason for such segregation is that the elapsed time between two consecutive writes for a particular write-intensive block might be shorter than the read-intensive ones, which requires different retention times.*

TABLE I: Average Read and Write Access Retention of different cache sets for benchmarks with various retention time.

Set Retention Time	0-1ms		1-10ms		10-100ms		100ms-1sec	
Benchmarks	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
blackscholes	0.6%	56.9%	90.9%	24.2%	6.3%	5.9%	2.2%	12.9%
bodytrack	0.01%	26.1%	8.7%	43.3%	85.7%	28.4%	5.6%	2.2%
canneal	0.63%	7.4%	84.4%	86%	9.4%	4.5%	5.6%	2.1%
dedup	7.5%	39.3%	83.9%	54%	3.1%	4.5%	5.5%	2.2%
fluidanimate	0%	0.7%	20.7%	55%	73.7%	42.2%	5.6%	2.1%
swaptions	54.8%	58.5%	36.2%	30.4%	6.1%	4.1%	2.9%	7%
x264	10.8%	31%	80.4%	62.3%	3.3%	4.5%	5.5%	1.2%
Mean	10.6%	31.4%	57.9%	50.7%	26.8%	13.4%	4.7%	4.5%

Our benchmark based analysis shows that 35.4% of the blocks are read-miss blocks (GETS/GET_INSTR) and 64.6% (GETX) of the blocks are write-miss blocks for the quad-core system. Hence, *ABACa* logically divides the cache-sets into two equal groups of different retention times when *(a)* write-intensive blocks with lower retention time will be written more frequently thereby it saves write latency and energy, and *(b)* read-intensive group will have higher retention time and thus will save more energy for data allocation.

2) ***Proposed Technique: ABACa****:* The main motive behind the *ABACa* is to place write missed blocks into lower retention period. However, subsequently after placement, if the write-intensive block becomes the read-intensive there is no harm in terms of latency and energy as evident from the Table III. Figure 1 illustrates our proposed architecture, *ABACa*, that logically bifurcates the LLC set-wise based on the retention times. The two groups of cache-sets are named as *Rd Set* and *Wr Set*. Each of these groups contains the same number of contiguous cache sets. However, *ABACa* does not incur any changes to the LLC way-wise. Note that both tag array and data array are bifurcated to cater the future requests. In Figure 1, the architecture is shown for an STT-RAM-based LLC having 8 cache sets and 8 ways. So, the cache is divided set-wise into two groups where *Wr Set* contains cache sets numbered $0-3$, and *Rd Set* group has the sets $4-7$.

The mechanism for accessing this cache for several operations will be performed in the following manner: The requests generated from L1 cache can barely be classified into two prime categories- (a) Read *(Rd)* and (b) Write *(Wr)*. The *Rd* request is generated once an L1 cache experiences a miss due to the unavailability of either an instruction block or a data block that has to be read. For data block to be read in a shared manner, the controller usually generates a *GETS* type request. Whereas, for the instruction block, the controller

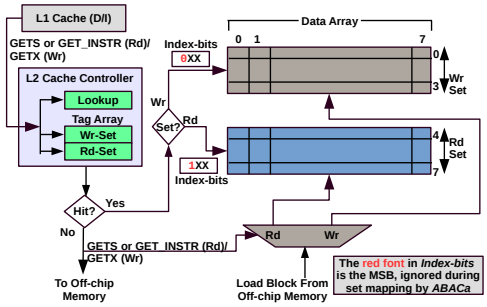

Fig. 1: *ABACa*, the proposed mechanism.

generates a *GET_INSTR* type request. On the other hand, once a write miss is encountered at the L1 cache, that needs to be performed exclusively, the controller generates a *GETX* request. However, based on the type of the received requests (i.e., *GETS/GET_INSTR/GETX*) from the L1 cache(s), the cache controller will look into the respective portions of the tag array. For instance, the *GETS/GETX* request will be handled by looking into the tag arrays for *Rd Set* as well as for *Wr Set*. To determine the set-id for a data block, *ABACa* uses the conventional cache mapping technique but ignores the most significant bit (MSB) (0 of the index bits (ref. Figure 1)). With *GET_INSTR* request, the cache controller will only look into the read set of tag array and for this, the MSB of the index-bits is always considered as 1. If L2 cache (LLC in our case) contains the data, i.e., a cache hit is detected at the tag array, the request will be served by sending the block from its respective location in the data array. In case of an L2 miss, the request will be forwarded to the off-chip memory like conventional memory systems, and the data will be fetched subsequently from the (off-chip) memory. Once the data is received at the L2 from the memory, it will be loaded inside the L2 based upon the request type for which the data has been requested for access. That is, if the data is requested due to a read miss (i.e. *GETS/GET_INSTR*) it will be loaded inside the *Rd Set* of L2, else it will be placed in the *Wr Set* (for *GETX*).

IV. Simulation setup & Evaluation

The proposed cache design of *ABACa* has been simulated with a full system simulator, gem5 [14]. Our cache-based policy has been implemented in the Ruby module of the gem5. Table II details about the system configuration used in our simulations. Towards evaluation, we modeled different retention times, and energy parameters mentioned in Table III (obtained from CACTI-STT [15]). To compare the efficacy of *ABACa*, we implemented baseline SRAM and random allocation having two retention groups in baseline STT-RAM without access-type aware block placement. In particular, the read and write missed blocks can be placed anywhere based on the cache set index. The PARSEC 2.1 benchmark suite [16] with medium input set is used in all simulations. In *ABACa*, both read and write sets are searched during the requested block's lookup operation. We simulated with extra latency of 1 cycle (obtained from CACTI-STT [15]) for the tag lookup.

While simulating, we set the retention times of write and read cache sets to 10ms and 1 second, respectively. Later, we describe the justification of these retention time choices.

TABLE II: *ABACa*: System Configuration

System Component	Configuration
CPU	X86 Quad-core, @2GHz
L1 SRAM I/D-Cache	32KB, 64B cache block, 4-way, LRU, MESI
L2 STT-RAM Cache	4MB, 64B cache block, 16-way, LRU, MESI Retention Times: 1ms, 10ms, 100ms, 1s, 10s (with cache sets $1-2048$, $2049-4096$, respectively)
Main Memory	8GB DRAM

*1) **Write-backs and MPKI**:* Figure 2 shows normalized total write-backs due to expiry of the blocks for random allocation and *ABACa*. We have observed an average reduction of write-backs by 72.7% for *ABACa* over random allocation. The reason behind the improvement is the access-type aware block placement to different retention sets by *ABACa*.

Figure 3 represents the normalized Misses Per Kilo Instructions (MPKI) for random allocation, *ABACa*, and SRAM. *ABACa* reduces MPKI by an average of 40.75% over the random allocation. This improvement is basically due to fewer premature expiration of the blocks by *ABACa* compared to the random allocation. However, compared to SRAM, the random allocation increases the MPKI by 55%, whereas, in *ABACa* the MPKI is worsened by 14.2%. This is due to lesser premature write-back expiration of the blocks in SRAM.

*2) **Cycles Per Instruction (CPI)**:* Figure 4 depicts normalized CPIs of all three techniques. *ABACa* improves CPI by an average of 3% over the random allocation due to comparatively smaller MPKI. Over SRAM, due to a large number of misses, the random allocation degrades CPI by an average of 4%. However, concerning *ABACa*, the CPI degrades only by a margin of 0.95% over SRAM. This is due to SRAM's lower write latency and no write-backs block expiration.

*3) **Dynamic Energy and EDP**:* Figure 5 shows the dynamic energy of all the evaluated schemes normalized to the random allocation. Over random, *ABACa* improves the dynamic energy by 15.5%. This improvement is due to access-type based placement of *ABACa* that reduces the number of misses and entails with decreasing number of write allocations and reducing overall write energy. However, due to lower write energy and fewer number of misses in SRAM, the dynamic energy consumption by random allocation is 60.7% over SRAM. Whereas, with *ABACa*, this dynamic energy degradation is brought down to 45.2%.

Figure 6 presents the EDP values of all the evaluated schemes normalized to random. *ABACa* achieved an average EDP reduction of 5% over random allocation due to lesser CPI and dynamic energy. Compared to SRAM, we observed a substantial amount of EDP improvement of 61.35% by *ABACa*. This is due to STT-RAM's lower leakage energy over SRAM.

*4) **Comparative Analysis**:* Table IV presents the comparative analysis of varying retention times of read and write sets over different metrics. On reducing the write set's retention time to $1ms$, the premature expiration of blocks has been substantially increased, which further increases overall EDP and MPKI. On the other hand, the number of premature expiration of the block and MPKI are decreased due to the rise in the write set's retention time ($100ms$). But, the costly write operations increase dynamic energy here.

TABLE III: Timing and Energy values for SRAM and STT-RAM cache configurations ($4MB$, $64B$ block, 16-way) at different retention times

Memory Device	SRAM	STT-RAM				
Retention Time	-	1ms	10ms	100ms	1s	10s
Wr Energy (pJ) (per access/bit)	0.948	4.194	5.265	5.808	6.320	6.926
Rd Energy (pJ) (per access/bit)	0.948	0.441	0.452	0.461	0.484	0.486
Leakage Power	1315.5 mW	475.5 mW				
Rd Latency (cycles)	3	3	3	3	3	3
Wr Latency (cycles)	3	5	6	7	10	11

TABLE IV: Comparative Analysis between retention of read and write set (normalized to *ABACa*) for all considered PARSEC applications

Parameter Comparison	Write Retention	Read Retention	CPI	Dynamic Energy	EDP	MPKI	Expired Blocks
ABACa	10 ms	1 sec	1.00	1.00	1.00	1.00	1.00
Write Set	1 ms	1 sec	1.02	1.045	1.04	1.57	4.44
	100 ms	1 sec	1.00	1.040	1.00	0.88	0.211
Read Set	10 ms	100 ms	1.00	1.020	1.00	1.23	1.57
	10 ms	10 sec	1.04	1.035	1.03	0.99	0.92

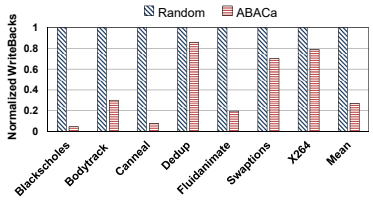

Fig. 2: write-backs normalized (norm.) to random allocation (lesser is better)

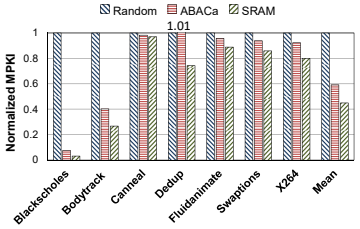

Fig. 3: MPKI norm. to random allocation (lesser is better)

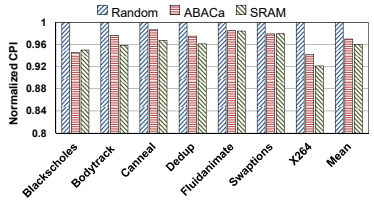

Fig. 4: CPI norm. to random allocation (lesser is better)

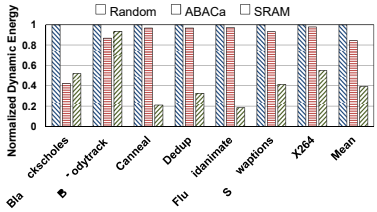

Fig. 5: Dynamic Energy norm. to random allocation (lesser is better)

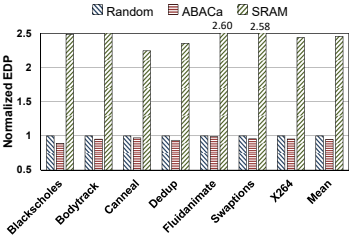

Fig. 6: EDP norm. to random allocation (lesser is better)

With a smaller retention time ($100ms$) of the read set, the number of premature evictions and MPKI have been increased. This results into an increase in the dynamic energy due to the higher number of allocations. Whereas, in case of read set with higher retention time ($10sec$), the number of write-backs due to premature expiration has been decreased. However, with larger write latency and energy, the EDP, dynamic energy, and CPI have been increased.

V. Conclusions and Future Directions

The potential advantages of high density, non-volatility, and low static power consumption make STT-RAM a potential memory technology for next-generation caches. But, higher write energy and latency of the STT-RAM limits its potential towards commercial cache designs. To mitigate such design obstacles, we propose *ABACa*, that logically bifurcates an STT-RAM-based cache set-wise based on retention times, where blocks are grouped and placed depending upon the causes of their arrivals (read/write miss). On a read miss, the block is fetched and placed into the group with a higher retention time, whereas the block is placed into the group having a lower retention time in case of a write miss. Empirical analysis shows, *ABACa* achieves significant improvements of 61.35% in EDP and 40.75% in MPKI compared to the baseline SRAM-based and multi-retention baseline STT-RAM-based LLCs with random reallocation, respectively. We intend to extend *ABACa* towards *(a)* dynamic resizing and/or construction of the groups of cache-sets to reduce capacity and conflict misses, together with *(b)* request-types aware block migrations between different retention zones, based on the dynamic phase-based analysis of the applications.

Acknowledgment

Marie Curie Individual Fellowships from *EU* have been granted to *Shounak Chakraborty*.

References

[1] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *ACM JETC*, 2013.

[2] M. K. Qureshi *et al.*, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, 2011.

[3] Z. Sun *et al.*, "STT-RAM cache hierarchy with multiretention MTJ designs," *IEEE TVLSI*, 2014.

[4] ——, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *MICRO*, 2011.

[5] K. Kuan and T. Adegbija, "HALLS: An energy-efficient highly adaptable last level STT-RAM cache for multicore systems," *IEEE TC*, 2019.

[6] S. Agarwal and H. K. Kapoor, "Reuse-distance-aware write-intensity prediction of dataless entries for energy-efficient hybrid caches," *IEEE TVLSI*, vol. 26, no. 10, pp. 1881–1894, 2018.

[7] A. Jog *et al.*, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *DAC*, 2012.

[8] J. Park *et al.*, "A novel integration of STT-MRAM for on-chip hybrid memory by utilizing non-volatility modulation," in *IEDM*, 2019.

[9] J. Ahn *et al.*, "DASCA: Dead write prediction assisted STT-RAM cache architecture," in *HPCA*, 2014.

[10] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *MICRO*, 2009.

[11] K. Kuan and T. Adegbija, "Energy-efficient runtime adaptable L1 STT-RAM cache design," *IEEE TCAD*, 2020.

[12] K. Kuan and T. Adegbija, "MirrorCache: An energy-efficient relaxed retention L1 STT-RAM cache," in *GLS-VLSI*, 2019.

[13] X. Wu *et al.*, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *DATE*, 2009.

[14] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH CAN*, 2011.

[15] S. Arcaro *et al.*, "Integration of STT-MRAM model into CACTI simulator," in *IDT*, 2014.

[16] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications." PACT, 2008.