ENDURA : Enhancing Durability of Multi Level Cell STT-RAM based Non Volatile Memory Last Level Caches

Yogesh Kumar, S. Sivakumar and John Jose
MARS Lab, Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, India

***Abstract*—With high packing density and low leakage power, Spin Transfer Torque Random Access Memories (STT-RAM) are a promising alternative to replace traditional memory technologies such as SRAM and DRAM. Applications will continue to demand more memory for processing in the coming decades. To achieve higher cell density, Multi-Level Cell STT-RAM (MLC STT-RAM) that can store two or more bits in a single memory cell is preferred over Single Level Cell STT-RAM (SLC STT-RAM). But their multistep read and write operations lead to significant read and write latency. The multistep write operations are also affecting the durability of MLC STT-RAM. Specialised wear levelling techniques are not available for MLC STT-RAM. We propose ENDURA, a technique that could improve the lifetime and latency of MLC STT-RAMs. ENDURA extends the lifetime of 2MB and 4MB MLC STT-RAM L2 caches by 2.05x and 2.59x on a single-core system and reduces write latency by 9.6% and 8.06% respectively with minimal overhead.**

***Index Terms*—Non Volatile Memory, Multi Level Cell, Wear leveling, Lifetime improvement, Write variation**

I. Introduction

The amount of data handled by the processing elements is increasing every day. Because of this trend, the processing elements from handheld devices to supercomputers, require higher processing capability as well as significant on-chip and off-chip memories. Unfortunately, conventional memory technologies are not scaling up to the rate of processing speed. Conventional memory technologies such as SRAM and DRAM have very low packaging density and high leakage power, which is a major drawback in realising large volume of on-chip memory. This scenario motivated to replace these traditional technologies with emerging non-volatile memory technologies such as STT-RAM [1], PCM [2] and ReRAM [3], which have very high packing density and zero leakage power [4]. However, these technologies have limited write endurance and high write latency compared to traditional memory technologies. Among the emerging memory technologies, STT-RAM has a better lifetime and write latency, there by making it the most suitable choice for realising on-chip and off-chip cache memories [5]. The on-chip area used for memory can be further efficiently utilised by using MLC STT-RAMs, which can save more than one bit of information in a single memory cell [6] [7].

Unlike SLC, in MLC, there are two Magnetic Tunneling Junctions (MTJ) as shown in Fig. 1. Smaller MTJ is known as soft bit, and larger MTJ is known as hard bit. Soft bit

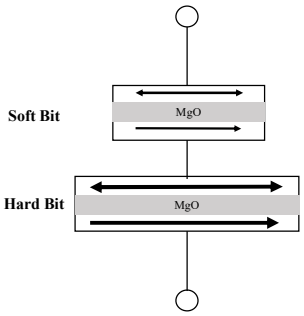

Fig. 1. Multi Level Cell STT-RAM

cell is easier to flip than hard bit. Write to hard bit may result in overwriting of the soft bit. This phenomenon is called write disturbance [8]. As feature size decreases, the difference between read and write current decreases significantly and hence reading a hard bit can flip the soft bit, which is called read disturbance. To mitigate read and write disturbances, the soft bit is read before read and write operations and later it is restored after completion of the same [9]. This immediate restore scheme results in more writes on the soft bit and further reduces its lifetime.

For an MLC-based cache memory, we have two cell arrangement options: direct mapping and cell split mapping. In direct mapping, the cache block consists of both soft and hard bits, whereas in a cell split mapping, the cache block will be either hard or soft bits as shown in Fig. 2. An application with a non-uniform write pattern can result in early wear out of memory cells. In order to overcome this issue, a proper wear leveling technique, which could distribute the writes across the memory is necessary. The state of the art wear leveling techniques designed for SLC memories do not distinguish between hard and soft ways. We believe a customised wear leveling technique can significantly improve the lifetime as well as the latency of MLC STT-RAM when used as last-level caches.

In this paper, we make the following major contributions:

- We analyse write variations in the last level NVM cache and draws meaningful conclusions.
- We study the impact of state-of-the-art wear leveling techniques on MLC NVM based Last Level Caches.
- We propose ENDURA a wear leveling technique that reduce the intraset [10] write variation in NVM LLCs and

978-1-6654-9005-4/22/$31.00 ©2022 IEEE

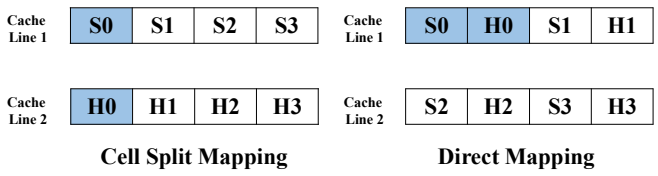

Fig. 2. Direct mapping vs Cell split mapping

improve the write latency.

- We test ENDURA using SPEC 2006 [11] benchmarks on the gem5 cycle-accurate simulator [12], and find that our proposed method outperforms other state-of-the-art solutions.

II. Related Work

Masayuki Sato et al. proposed a hybrid cache architecture based on STT-RAM [13] with its data replacement policy named Fast Region First Used (FRFU). Their technique combines a write-optimized STT-RAM to create fast and slow regions in the cache. This helps in achieving high density and low leakage power. FRFU policy tries to avoid the hot data being inserted into the slow region where the write penalty is high. The evaluation results show that their technique has 55% less energy consumption than conventional architectures. Adaptive Restore Scheme [14] reduce the energy overhead of the Immediate Restore Scheme in MLC STT-RAM for write and read disturbance restores. They introduce the concept of read reuse distance to quantify the temporal distance between two successive reads to a block. Jue Wang et al. proposed a wear-levelling technique by combining Swap Shift (SwS) and Probabilistic set Line Flush (PoLF) together to form Inter and Intra-set Write-variation Aware Policy, named i^2WAP [10]. The goal of the SwS is to reduce inter-set write variations by shifting the mapping of physical cache sets. PoLF reduces intra-set write variations by probabilistically flushing hot cache line such that next time when this data is needed, it is loaded to a cold cache line. Both of these techniques can be implemented using very minimal storage overhead. Sparsh Mittal and Jeffrey S. Vetter proposed EqualWrites [15], a wear-levelling technique used to improve the lifetime of NVM caches by minimizing intra-set write variation. EqualWrites achieves this by recording the number of writes at cache line granularity and swapping a hot cache line with a cold one using data migration within a set.

III. Motivation

Experimental studies show a significant difference between the maximum and average writes to LLC of different unicore cache architecture systems [16]. This write variation emphasises the need for a good wear leveling policy for NVM caches. MLC NVM has the advantage of higher packing density over SLC NVM. Replacing SLC with MLC NVM allows for higher capacity caches in a given area and can lower the miss rate. However, it suffers from the same issues as SLC, i.e. higher write latency and low write endurance. These issues are scaled up for writes on a hard way. Conventional wear-levelling techniques for SLC cannot be directly applied to MLC because they are oblivious to the functional and operational differences between hard and soft ways. This motivated us to explore the possibility of a customised wear leveling technique for MLC NVMs, which could improve the lifetime and write latency when used as last-level caches.

IV. ENDURA

We propose a wear leveling technique ENDURA to enhance the durability of MLC NVMs. Unlike conventional SLC wear leveling techniques, ENDURA is tailor-made for MLC based last level caches for improving lifetime and write latency. In our design, we choose cell-split mapping as it allows us to exploit the latency difference between Soft Bit Line (SBL) and Hard Bit Lane (HBL). We organize the cache such that SBL and corresponding HBL are in the same set and form a pair. In the context of our work, we define pair as a group of the hard and corresponding soft block (way) in a set. ENDURA has two significant components, a wear leveling unit and write latency reduction unit. We discuss these units in the coming sections.

A. *SpH Wear leveling unit*

To protect the cache memory from early wear out from non-uniform write patterns and targeted repeated writes on selected cache lines, ENDURA uses $SpH(SplusH)$ wear leveling unit. SpH works in similar lines with the EqualWrites [15] technique but is customised to MLC cache lines. Algorithm 1 gives an overview of SpH wear leveling unit. Consider a pair of hard way and soft way in a cache set. Let s be the number of writes to soft way, and h be the number of writes to the hard way from the requester (processor or L1 cache). Recall that when there is a write to hard way, it results in a write to both hard and soft way. After accounting for this write disturbance, there will be $(s+h)$ writes to soft way and h writes to hard way. Due to this higher number of writes to the soft ways, they degrade faster. The total number of writes to a pair $(s + h)$ determines the lifetime of the cache and not based on the ratio these writes are distributed. Hence, we should perform wear-levelling on the number of writes to a pair rather than soft and hard way separately. Our proposed technique SpH wear-levelling incorporates this idea. We maintain counters for each pair, and on each write to that pair, we increment that counter and swap two pairs when the difference between them reaches a threshold. Let M_1 be the number of bits used for the sph counter. We define $\Omega_1 = 2^{M_1}$ as the sph threshold. All counters are initialised to $\Omega_1/2$. On a write-hit to a cache line, if the current value of the corresponding sph counter is less than $\Omega_1 - 1$, we increment it. Otherwise, we search for another pair with sph counter value 0. If such a pair exists, we swap it with the current pair. Otherwise, we decrement all other sph counters in that set by 1. Since hard writes can overwrite the soft ways, we follow a definite order for swapping. Let (S_1, H_1) be the first pair and (S_2, H_2) be the second pair. The swapping operation on pairs can be performed in 3 steps,

1) Read S_1, S_2, H_1, H_2 in parallel

2) Write to H_1 and H_2in parallel
3) Write to S_1 and S_2 in parallel

We write to hard ways first followed by write to soft ways in the order described above. This way we ensure that hard writes do not interfere with soft writes.

Algorithm 1: Algorithm for SpH wear-levelling

```
Let w be the way index of the write hit block
Let p = w/2                                   ▷ integer division
if Sph[i][p] ≠ Ω1 - 1 then
    Write new data to Data[i][w], mark dirty, update LRU-stack
    Sph[i][p] = Sph[i][p] + 1
else
    Let target = NULL                         ▷ Target for write-redirection
    for all pair-locations q ( ≠ p) do
        if Sph[i][q] == 0 then
            target = q
            Break from for loop
    if target ≠ NULL then                     ▷ Candidate for write-redirection exists
        Let soft1 = NULL
        Let hard1 = NULL
        Let soft2 = Read Data[i][2 * target]
        Let hard2 = Read Data[i][2 * target + 1]
        if w is soft way then
            soft1 = new data
            hard1 = Read Data[i][2 * p + 1]
        else
            soft1 = Read Data[i][2 * p]
            hard1 = new data
        Write hard1 to Data[i][2 * target + 1]
        Write hard2 to Data[i][2 * p + 1]
        Write soft1 to Data[i][2 * target]
        Write soft2 to Data[i][2 * p]
        Sph[i][p] = Sph[i][target] = Ω1/2
    else
        for all pair-locations q ( ≠ p) do
            Sph[i][q] = Sph[i][q] - 1
        Write new data to Data[i][w], mark dirty, update LRU-stack
```

B. Hard Write Predictor for improving Write latency

The SpH unit balances writes between pairs. This results in an improvement in cache lifetime. ENDURA use the Hard Write Predictor (HWP) unit to improve write latency. HWP exploits the latency difference between soft and hard ways. Table I shows the latency difference between soft bits and hard bits [14]. Since soft bit writes are approximately two times faster than the hard bits writes, redirecting the write intensive blocks to softways will improve the overall write latency.

TABLE I
COMPARISON OF SOFT AND HARD WAYS

	Soft bit	Hard bit
Read Latency	6.73 cycles	9.80 cycles
Write Latency	25.31 cycles	56.50 cycles
Read Energy	0.22 nJ	0.43 nJ
Write Energy	0.842 nJ	2.50 nJ

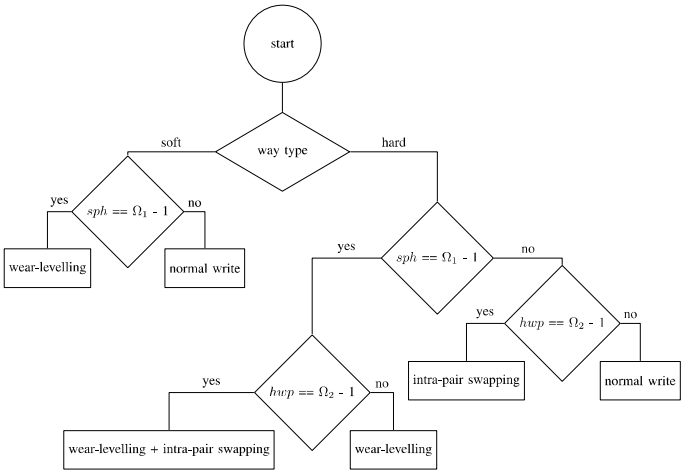

Fig. 3. Flowchart of write operation

As discussed before, degradation of a storage cell depends on the total writes (sum of soft and hard writes) in a pair. Consider case A, where we have s writes to soft way, and h writes to the hard way and case B, where we have $s+t$ writes to soft way and $h-t$ writes to the hard way. In both of these scenarios, the total number of writes is the same, i.e. $(s+h)$. So, in both cases, the impact on cache line lifetime should be the same. But in case B, since more writes are to soft way, which has lower write latency than hard way, average write latency is lower than case A.

HWP aims to shift some writes from the hard ways to soft ways. When the application writes more data to hard ways, HWP will swap it with its corresponding soft way in that pair. The key idea is that with the overhead of just one write; we can save more hard writes in future. We use hwp counter for each pair to count the number of consecutive writes to the hard way. When this counter saturates, we will swap ways in that pair on the next hard write. Let Ω_2 be the consecutive hard write threshold. We increment the corresponding hwp counter in the event of a hard write, if the current hwp value is less than $\Omega_2 - 1$. If it is already at the threshold, then we perform intra-pair swapping. Soft way writes trigger counter reset. Like inter-pair swapping, we follow prefixed sequential writes. Let (S_1, H_1) be a pair. We can complete intra-pair swapping in three steps.

1) Read S_1
2) Write to H_1 (value from S_1)
3) Write to S_1 (new data)

Fig. 4 shows the a cache set of 4 way set associative cache which uses our proposed technique. Depending on values of sph counter, hwp counter and type of write as shown in Fig. 3, we have following different operations.

1) Normal write - Increment sph counter. Change hwp counter based on type of write.
2) Wear leveling - Depending on whether there is a pair with sph counter 0, we either swap or decrement other sph counters. Change hwp counter based on type of write.

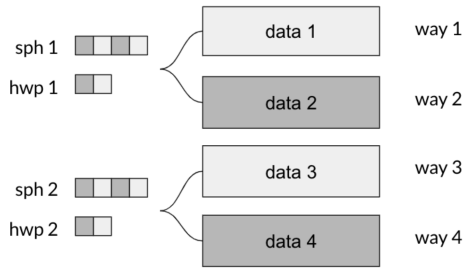

Fig. 4. Organisation of an ENDURA enabled 4 way set associative NVM cache

TABLE II
SYSTEM CONFIGURATION

CPU	Uni-core, ALPHA, Out of order
L1 Cache	Private, 64 KB, SRAM split cache, 64 B block, 2-way set associative
L2 Cache	Shared 512KB/1MB/2MB/4MB, NVM unified cache 64 B block, 8-way set associative
Main Memory	4 GB
Benchmark	SPEC CPU2006

3) Intra-pair swapping - Swap soft and hard way in that pair. Reset hwp counter. Increment sph counter.
4) Wear leveling + intra-pair swapping - This combines (2) and (3). Change sph counter accordingly. Reset hwp counter.

V. EXPERIMENTAL SETUP AND RESULT ANALYSIS

To evaluate our technique we have used gem5 [12], a cycle-accurate event based open source architectural simulator. We use Ruby model and MESI protocol for simulating cache memory model and maintaining cache coherence. Details of the system configuration are given in Table II.

TABLE III
BENCHMARK CLASSIFICATION BASED ON WPKI

Category	Benchmark
Low	namd (Nd), soplex (So), gromacs (Gr), href264(Hf)
Mid	milc (Mi), libquantum (Lq), sjeng (Sj), bzip2 (Bz)
High	leslie3d (Ls), lbm (Lb), hmmer (Hm)

We execute one billion instructions from selected benchmarks from SPEC CPU 2006 benchmark suite to evaluate the performance of our proposed architecture and state of the art technique, EqualWrites [15] for 512KB, 1MB, 2MB and 4MB MLC NVM L2 cache memories. We categorise the benchmark programs into Low, Mid and High categories based on their Writes Per Kilo Instruction (WPKI) values. We evaluate the performance of ENDURA with different sizes of sph and hwp counters and found that sph counter size of 4-bits and hwp counter size of 2-bits (swap after 3 consecutive hard writes) gives the optimal performance. The following sections discuss the impact on different performance metrics for an baseline cache (MLC NVM without any optmization), EqualWrites and ENDURA. We estimate relative lifetime by using the inverse of maximum write count to a cache memory block.

A. Performance Analysis

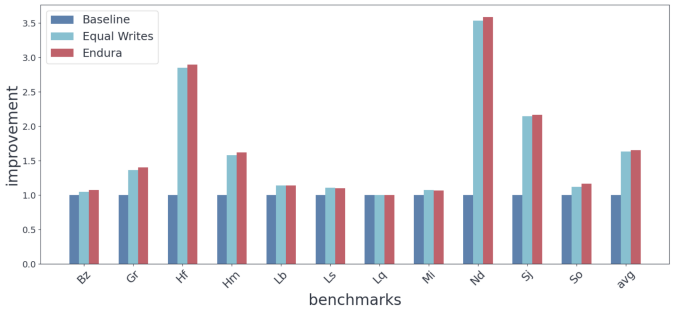

Fig. 5. Comparison of Lifetime for various 512KB L2 MLC NVM cache architectures in unicore system. (taller the bar, the better)

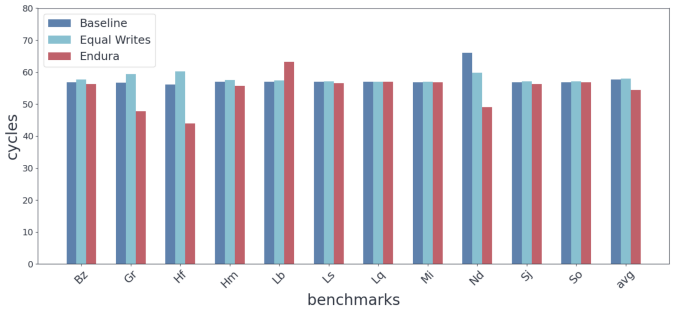

Fig. 6. Comparison of Average Write Latency for various 512KB L2 MLC NVM cache architectures normalised to base configuration in unicore system. (shorter the bar, the better)

Fig. 5 shows the comparison of relative life (with respect to baseline) of different NVM architectures for 512 KB L2 cache in unicore system. We can see that our proposed architecture improves lifetime for all benchmarks. The impact of our proposed technique is more evident for benchmarks like *namd*, and *h264ref*. These benchmark have low average write and high write variations. Whereas for benchmarks like *libquantum* and *lbm* the relative life time improvement is not high because of the fact that these applications follow a much more uniform write distribution pattern. A similar trend can be seen for average write latency of various benchmark programs shown in Fig. 6. Benchmarks with non-uniform write patterns show major improvement in average write latency also because our technique could redirect more writes from hard way to soft way whose scope is limited for applications with uniform write patterns. We run our technique for 1 MB, 2 MB and 4 MB cache sizes. ENDURA gives 1.89 times better lifetime and 7.90% reduction in write latency when compared to baseline architecture and 7.90% reduction in write latency for 1MB L2 cache as shown in Fig. 7 and 8.

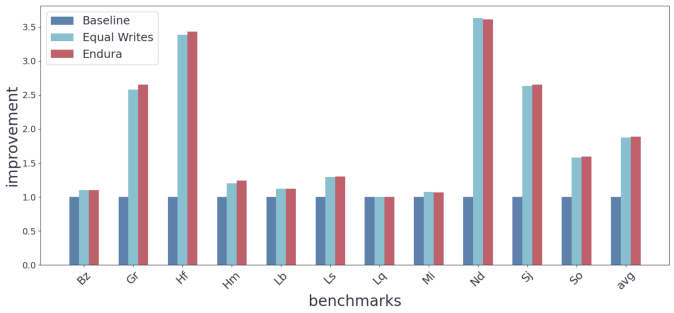

Fig. 7. Comparison of Lifetime for various 1MB L2 MLC NVM cache architectures in unicore system. (taller the bar, the better)

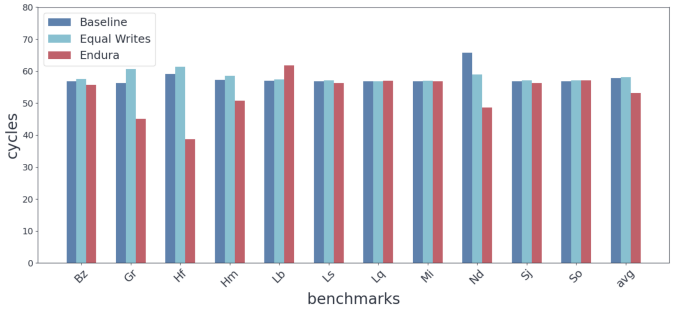

Fig. 8. Comparison of Average Write Latency for various 1MB L2 MLC NVM cache architectures normalised to base configuration in unicore system. (shorter the bar, the better)

Fig. 9 and 10 shows the experimental results for 2MB cache architectures. ENDURA shows average lifetime improvement of 2.05x for 2 MB cache when compared to baseline architecture, which is almost same as 2.04x improvement in EqualWrites with much less storage overhead. We also observe 9.61% reduction in average write latency when compared to baseline whereas EqualWrites increases latency by 1.29%.

Similarly, as shown in Fig. 11 and 12 ENDURA gives a relative lifetime improvement of 2.55 times and reduction in write latency of 8.06% with respect to baseline system and 13.61% reduction in write latency when compared to EqualWrites technique for 4 MB cache size.

Fig. 13 shows the distribution of writes among hard and soft ways for different cache architectures with capacity of 2MB. Write distribution policy in EqualWrites technique is based on write count to each block. While redirecting the writes, it does not consider the latency of writes, i.e. whether the target block is a hardway or softway. However, ENDURA distributes the writes in softway-hardway pair granularity; hence, it can redirect write-intensive blocks within and across the pair, giving a better performance.

B. Overhead Analysis

ENDURA uses two groups of counters: sph and hwp. It also needs a swapping module to swap the contents between hard and soft ways. Let S, A, B, T, N, M denote the number of sets, set associativity, block size, tag size, number of swap

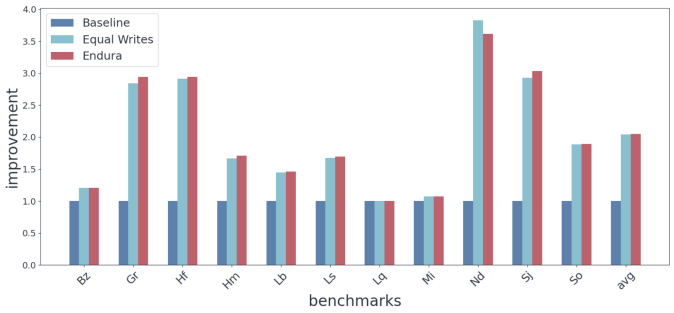

Fig. 9. Comparison of Lifetime for various 2MB L2 MLC NVM cache architectures in unicore system. (taller the bar, the better)

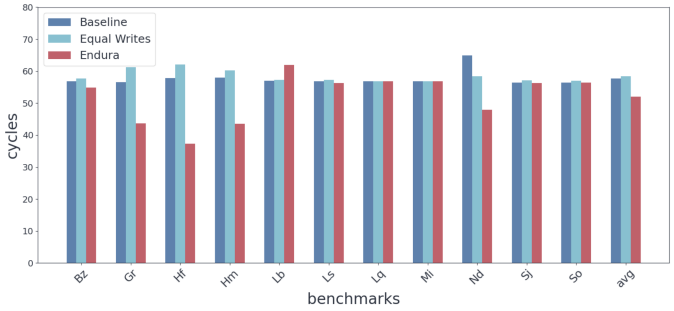

Fig. 10. Comparison of Average Write Latency for various 2MB L2 MLC NVM cache architectures normalised to base configuration in unicore system. (shorter the bar, the better)

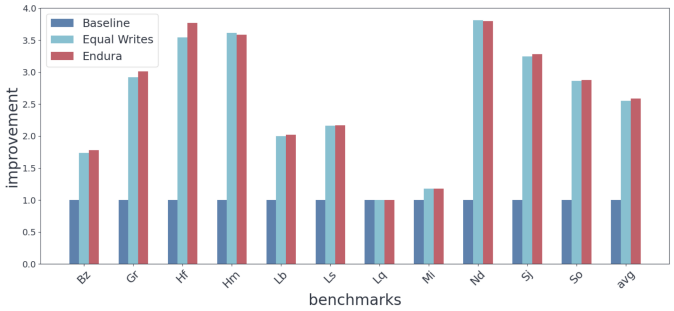

Fig. 11. Comparison of Lifetime for various 4MB L2 MLC NVM cache architectures in unicore system. (taller the bar, the better)

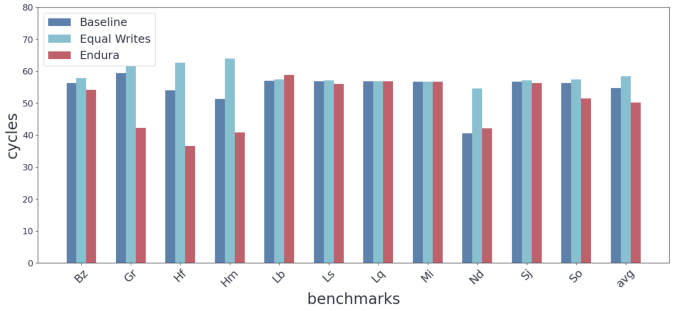

Fig. 12. Comparison of Average Write Latency for various 4MB L2 MLC NVM cache architectures normalised to base configuration in unicore system. (shorter the bar, the better)

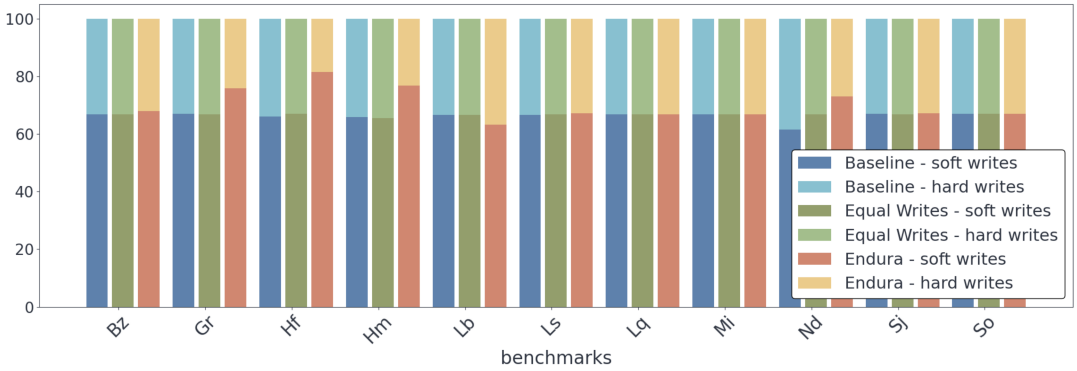

Fig. 13. Comparison of Write distribution for various 2MB L2 MLC NVM cache architectures.

buffers and number of bits for counters per block, respectively. Storage overhead can calculated as

$$\theta = \frac{M \times S \times A + N \times B}{S \times A(B+T)} \times 100$$

We assume B = 64B and memory address to be 48 bit wide (for calculating T). We have chosen counter size to be 4 bits for sph and 2 bits for hwp. We are using four swap buffers of size 64B each to read two pairs in step 1 of inter-pair swapping algorithm. This will cause a storage overhead of 0.565% for a 2 MB cache. EqualWrites on the other hand with 4 bit counter cause a storage overhead of 0.923%. For a 4MB cache architecture ENDURA has a storage overhead of 0.560% and EqualWrites has a storage overhead of 0.832%. This shows that ENDURA has 38.79% and 32.69% less storage overhead than EqualWrites for 2MB and 4MB caches respectively.

VI. Conclusion and Future Scope

Emerging memory technologies are gaining popularity as they have the potential to substitute or work in conjunction with conventional memory technologies. NVM offers a low power and high packing density solution. However, it cannot be used directly due to its poor write endurance and latency. We have gone through some of the existing literature which tries to improve these drawbacks of NVM. We have also proposed a solution to perform wear-levelling and latency reduction in MLC. We have compared our work with EqualWrites and found that it performs better in terms of cache lifetime with 13.61% less average write latency and 32.69% less storage overhead. Some future works can be developed, like minimizing hard reads to reduce average read latency. The second idea could be to partition the cache into two regions - instruction and data. Former constitute only hard ways and later will have both. Our technique can be used on data regions after some modifications.

References

[1] E. Chen, D. Lottis, A. Driskill-Smith, D. Druist, V. Nikitin, S. Watts, X. Tang, and D. Apalkov, "Non-volatile spin-transfer torque ram (stt-ram)," in *68th Device Research Conference*, 2010, pp. 249–252.

[2] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[3] H. Akinaga and H. Shima, "Resistive random access memory (reram) based on metal oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, 2010.

[4] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, 2016.

[5] P. Chi, S. Li, Y. Cheng, Y. Lu, S. H. Kang, and Y. Xie, "Architecture design with stt-ram: Opportunities and challenges," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 109–114.

[6] X. Chen, J. Wang, and J. Zhou, "Promoting mlc stt-ram for the future persistent memory system," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2017, pp. 1180–1185.

[7] H. Luo, L. Shi, Q. Li, C. J. Xue, and E. H.-M. Sha, "Energy, latency, and lifetime improvements in mlc nvm with enhanced wom code," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 554–559.

[8] Y.-S. Hsieh, Y.-H. Chen, Y.-P. Tang, and P.-C. Huang, "A selective write strategy for the elimination of write disturb errors on nonvolatile memory caches," in *2019 8th International Conference on Innovation, Communication and Engineering (ICICE)*, 2019, pp. 10–13.

[9] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen, "Multi-level cell stt-ram: Is it realistic or just a dream?" in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 526–532.

[10] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i2wap: 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)*, 2013, pp. 234–245.

[11] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, p. 1–17, Sep. 2006. [Online]. Available: https://doi.org/10.1145/1186736.1186737

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011.

[13] M. Sato, X. Hao, K. Komatsu, and H. Kobayashi, "Energy-efficient design of an stt-ram-based hybrid cache architecture," in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2020, pp. 1–3.

[14] X. Chen, N. Khoshavi, R. F. DeMara, J. Wang, D. Huang, W. Wen, and Y. Chen, "Energy-aware adaptive restore schemes for mlc stt-ram cache," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 786–798, 2017.

[15] S. Mittal and J. S. Vetter, "Equalwrites: Reducing intra-set write variations for enhancing lifetime of non-volatile caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 103–114, 2016.

[16] S. Sivakumar, T. Abdul Khader, and J. Jose, *Improving Lifetime of Non-Volatile Memory Caches by Logical Partitioning*. New York, NY, USA: Association for Computing Machinery, 2021, p. 123–128. [Online]. Available: https://doi.org/10.1145/3453688.3461488