Enhancing Lifetime of Non Volatile Memory Caches by Write Aware Techniques

S.Sivakumar, Mani Mannampalli, and John Jose

Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, India
sivakumar@iitg.ac.in, mani.mannampalli@gmail.com, johnjose@iitg.ac.in

Abstract. Traditional memory technologies, such as SRAM, suffers from limited package density and high leakage power. Applications are getting increasingly memory hungry in the age of big data. Non-volatile memories such as STTRAM, PCM, and ReRAM emerged as attractive contenders to replace traditional SRAM based memories. They have high density and zero leakage power. However, they have a limited write endurance. Non-uniform write patterns in applications can shorten the life of non-volatile memories. Traditional cache block replacement strategy like LRU leads some cache blocks to be accessed more frequently than others, accelerating the wear out of the cache. We present a Write Aware Last Level Non-Volatile Cache (WALL-NVC), which improves the lifetime up to 5.84x for unicore, 3.34x for dual core and 4.11x for quad core system. It reduces intraset write variation in last-level caches upto 98.91%, 90.11%, and 94.12% for unicore, dual core, and quad-core systems, respectively using write distribution and NVM friendly replacement mechanism.

Keywords: Non Volatile Memory · Wear leveling · Lifetime improvement · Write variation.

1 Introduction

In the current world, data processing is an inevitable necessity. The amount of data that a processing unit has to handle has expanded significantly in recent years due to technological advancements, growing popularity of IoT-based devices, social media, and video streaming platforms. We expect the trend of data-intensive applications to continue for the years to come. Applications that run from handheld devices to supercomputers require greater processing power and memory than before. Because of their poor package density and high leakage power, traditional memory technologies such as SRAM are inadequate to handle this demand for large on-chip memory.

Spin Transfer Torque RAM (STT-RAM) [1], Phase Change RAM (PCRAM)[2], and Resistive RAM (ReRAM)[3] are all promising non-volatile technologies that can replace conventional memories. They feature a high package density and little leakage power, making them ideal for realising large memories [4]. However, these emerging non-volatile technologies have a significant drawback in terms of write latency and write endurance. The maximum number of writes a memory cell can withstand before it permanently wears out is termed as write endurance. When non-volatile memories are used in applications with non-uniform write patterns, some memory cells wear out faster than others. Absence of write-aware cache replacement policies can result in frequent writes to some cache blocks can also force memory cells to wear out soon. These circumstances highlight the need for a system that reduces the amount of writes or distributes them evenly when using nonvolatile memories at various levels of the memory hierarchy. Our work aims to extend the life of non-volatile memory when it is employed as a last-level cache. EqualWrites [7], a state-of-the-art wear leveling techniques which reduce the intraset write variation and improve lifetime, compare LRU and random replacement policy for cache blocks. Both of these policies, however, are not customised for NVMs. Replacement plans like Refresh Aware Replacement Policy (RFR) [8] help to extend the life of NVMs. However, they are difficult to implement because they are designed for write-optimized cache memory and do not work with traditional wear levelling schemes. We believe a more NVM-friendly replacement policy combined with a good wear levelling method can greatly boost lifetime.

In this work, we make the following major contributions:

- We analyse write variations in the last level NVM cache and draws meaningful conclusions.
- We propose Write Aware Last level Non Volatile Caches (WALL-NVC), which can reduce the intra-set variation, thereby increasing its lifetime.
- For WALL-NVC, we use an NVM-friendly replacement policy called Least Recently Used Cold Block (LRU-CB), which also contributes to increase the lifetime.
- We test WALL-NVC using SPEC 2006 [5] benchmarks on the gem5 cycle-accurate simulator [6], and find that our proposed method outperforms other state-of-the-art solutions.

2 Related Work

As previously stated, write endurance refers to the maximum number of writes that a nonvolatile memory may withstand before failing. High write variation applications, as well as malicious apps that target NVMs' limited write endurance, can significantly reduce their lifetime. Several lifetime enhancement strategies were proposed in the past which can be broadly divided into two categories: write avoidance and write distribution techniques. In write avoidance techniques, we reduce the number of writes to the NVM using write avoidance strategies such as early termination, inversion, and encoding. We try to evenly spread writes across the memory in write distribution techniques so that a few memory cells do not wear out at a faster pace than others due to frequent writing to them. Write variation can occur within a set (intraset write variation) or across sets (interset write variation). EqualWrites [7] is a strategy for reducing intraset variation by swapping frequently written hot blocks with less frequently written cold blocks within a set. EqualChance [9] is a wear levelling technique that involves changing the physical location of a frequently written data block on a regular basis. Another solution is i^2WAP [10], which reduces intraset and interset write variations through probablistic flushing of frequently written blocks and set swapping. Unlike traditional replacement policies like LRU, which focus on increasing write pressure on write-intensive blocks, replacement policies like RCR, RFR, FCR,[8] and others prioritise endurance.

3 Motivation

To study the write variations of different applications, we analyse maximum and average writes to LLC for unicore architecture. To facilitate this, we model these architectures in gem5 [6] with two levels of cache and main memory. For L1-I and L1-D caches, we use 32 KB, 4-way set associative configuration. The 512-KB unified L2 cache is 8-way set associative and we use 8 GB main memory. The block size is 64 bytes. The number of kilo writes per 1 billion instruction window for selected benchmarks from the SPEC CPU2006 suite [5] is shown in Figure 1. We plot the maximum writes per way as well as average writes across ways. Our study shows that write variations can occur within a set and also across the set. This reinforces the need for good wear leveling policy for NVM based LLC.

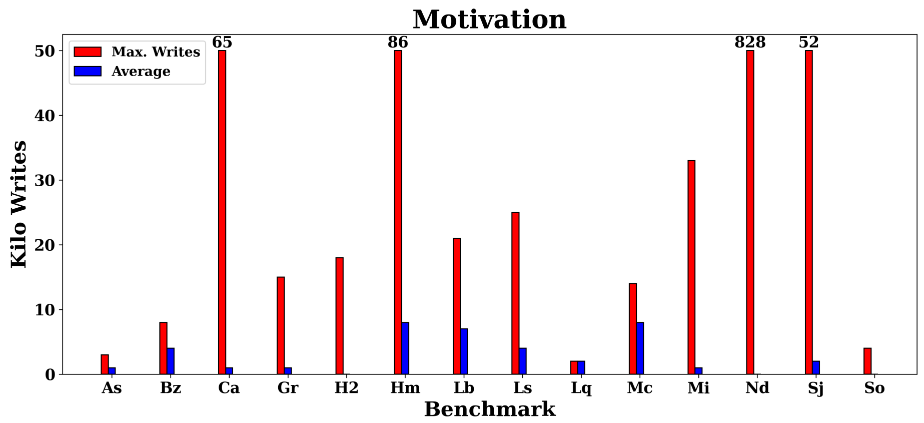

Fig. 1: Average and maximum writes per way (Kilo writes per 1 billion instructions) of various SPEC CPU 2006 benchmarks: Height difference between bars of a given benchmark indicates intensity of write level variations.

The write variation that occurs within a set is is quantified using the coefficient of intraset variation ($IntraV$). Write variation that occurs across sets is quantified using the coefficient of interset variation ($InterV$). $IntraV$ and

$InterV$ are given below.

$$IntraV = \frac{100}{N.Write_{avg}} \sum_{k=1}^{N} \sqrt{\frac{\sum_{l=1}^{M} (W_{k,l} - \sum_{m=1}^{M} \frac{W_{k,m}}{M})^2}{M-1}} \tag{1}$$

$$InterV = \frac{100}{Write_{avg}} \sqrt{\frac{\sum_{k=1}^{N} (\sum_{l=1}^{M} \frac{W_{k,l}}{M} - W_{avg})^2}{N-1}} \tag{2}$$

where N is number of sets in cache.
M is the number of ways in a set.
$W_{k,l}$ is the write count in set k and way l.
$Write_{avg}$is average write count given by

$$Write_{avg} = \frac{\sum_{k=1}^{N} \sum_{l=1}^{M} W_{k,l}}{N.M} \tag{3}$$

Low $IntraV$ and $InterV$ values suggest more evenly distributed writes within and across the cache sets, respectively. The $Write_{avg}$ shows the average number of writes in the cache memory. Popular cache replacement policies, such as Least Recently Used (LRU), Pseudo LRU, and others, consider the most recent use of a cache block into account while choosing a victim block for replacement. However, in non-volatile memories where write endurance is a major concern, the number of writes to the victim block can impact the cache memory's lifetime. To the best of our knowledge, state-of-the-art wear-leveling techniques do not study the role of replacement policy in improving the lifetime. EqualWrites which reduces the intraset write variation compare between LRU and Random replacement policy. However, both of these policies are not custom made for NVMs. This inspires us to investigate how to effectively combine a wear levelling technique and a better replacement policy tailor made to enhance the endurance of NVM caches.

4 Write Aware Last Level Non-Volatile Cache

To improve the lifetime of NVM while running applications having non-uniform writes and protecting them against targeted malicious attacks by repeated writes to specific blocks, we propose Write Aware Last Level Non-Volatile Cache (WALL-NVC). Unlike most of the state-of-the-art wear leveling techniques, WALL-NVC is a dual-stage wear leveling technique. The first stage is a new Least Recently Used Cold Block (LRU-CB) replacement policy, that takes care of selecting a better victim block for cache replacement in NVMs. The second stage employs a traditional write distribution strategy that works in tandem with LRU-CB to increase lifetime. The following sections discuss these stages in detail.

4.1 LRU-CB Replacement Policy

To the best of our knowledge, the impact of cache block replacement policy on the write endurance of NVM based caches is not explored so far by any wear leveling mechanisms. A good cache replacement policy for NVMs should enhance write endurance and reduce intra-set write variation. Ideally it should preserve the hottest blocks and prevent frequent evictions while reducing the write variation across blocks. When the cache hit rate is high, the number of replacements is less; hence the impact of replacement policies is minimal. Traditional replacement policies like LRU and Psuedo LRU do not consider the write count of the block while selecting a victim block. To meet these objectives, we propose a simple NVM friendly cach block replacement policy called as Least Recently Used Cold Block (LRU-CB).

The basic concept of LRU-CB policy is to choose a block from the set that is less frequently written as the victim block, thereby making writes to the set more uniform. To ensure that the blocks are not evicted frequently, a weighted aggregate average of each block's LRU age and write index is calculated. The block with the lowest aggregate average is picked as the victim block. To facilitate this, we attach a write counter to each block. When one of the write counters in a set reaches its saturation value, the write counters of blocks of that set are bitwise right-shifted. This downgrading of the counter value forces the least significant bit (LSB) of the counter to be lost, resulting in a minor loss of precision. This ensure that the respective values of write counters are downgraded (with a small error margin) before getting wrap around. We study the impact of LRU-CB by running different benchmarks and find that LRU-CB marginally improves the lifetime of NVM caches.This marginal improvement demonstrates the necessity for a complementary technique to LRU-CB in order to increase its performance.

4.2 Impact of LRU-CB with Write Distribution

Write-aware replacement policies have a limited impact on the endurance of NVM caches while running applications with high L1 cache hit rates as they trigger fewer evictions. Write distribution policies increase lifetime by distributing writes evenly. Lifetime of NVM caches can be extended by combining a good wear levelling policy with a write-aware replacement policy rather than doing so separately.

We compare the effectiveness of EqualWrites technique with the pseudo LRU policy and LRU-CB in order to verify the impact of LRU-CB when used in conjunction with a standard state-of-the-art wear levelling technique. Figures 2, 3 and 4 show the relative lifetime, intraset variation and hit rate of NVM caches, respectively while running different benchmarks in SPEC CPU 2006 suite. From the graphs we can observe that EqualWrites with LRU-CB increases the lifetime of L2 cache upto 1.39x than the combination of EqualWrites with psuedo LRU. LRU-CB reduces the intraset variation upto 83.08% without affecting the hit rate. This improvement is visible across all benchmarks thereby ascertaining that LRU-CB is a better cache block replacement algorithm for NVM caches.

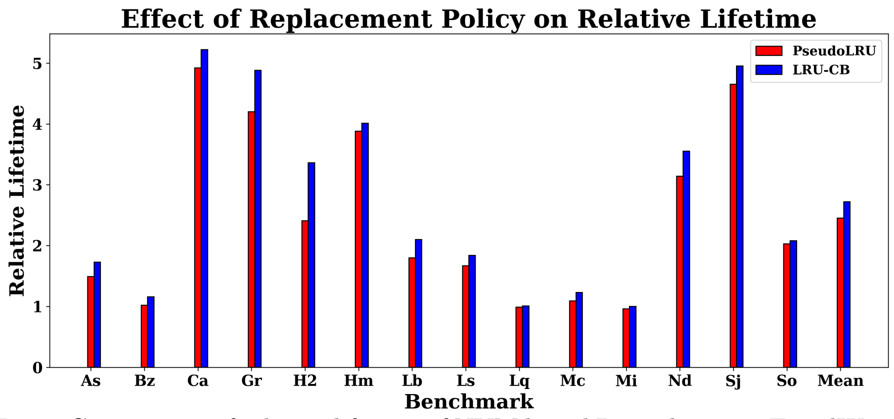

Fig. 2: Comparison of relative lifetime of NVM based L2 cache using EqualWrites with Pseudo LRU and LRU-CB replacement policies.

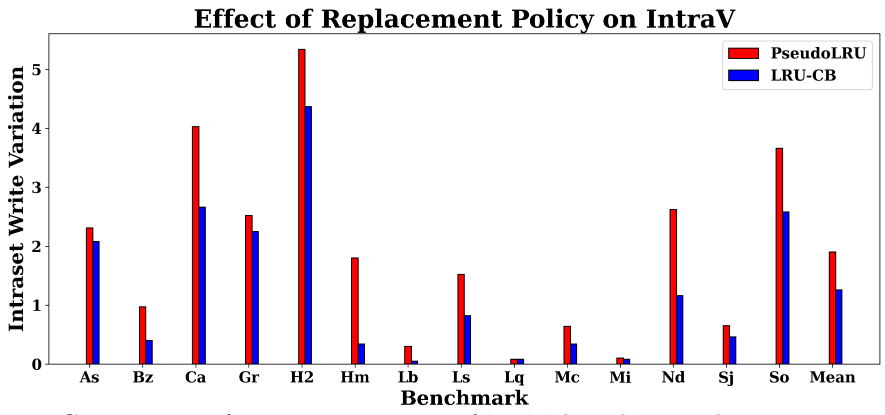

Fig. 3: Comparison of Intraset variation of NVM based L2 cache using Equal-Writes with Pseudo LRU and LRU-CB replacement policies.

4.3 Write Distribution in WALL-NVC

LRU-CB policy improves the performance of EqualWrites technique. But this comes with a high overhead. This is because LRU-CB needs extra counters apart from ones used for EqualWrites. This motivated us for the need for wear leveling policy that reduces the intraset variation and improves lifetime and synergizes with LRU-CB. Like other popular wear leveling techniques, WALL-NVC also works on the principle of redirection of the writes from hot blocks to cold blocks.

Each set of an n-way set associative WALL-NVC has $(n+1)$ counters: one set counter and n block counters. For each write hit to WALL-NVC, the corresponding set and block counter are updated. Once the set counter reaches a prefixed threshold T, it selects a write redirection target among blocks of that set. Block with least writes is preferred as the redirection target, and hence it selects the block with zero write count. Swapping is initiated between the ac-

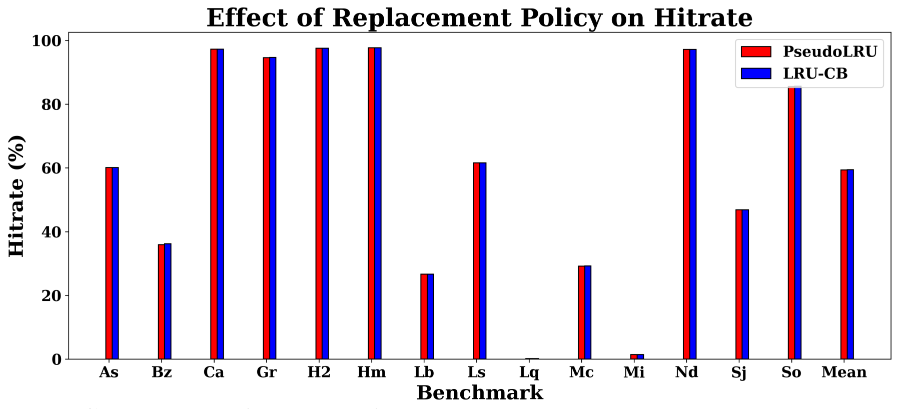

Fig. 4: Comparison of hit rate of NVM based L2 cache using EqualWrites with Pseudo LRU and LRU-CB replacement policies.

cessed block and redirection target blocks, if such block is available. If target block is invalid, instead of swapping, it writes to the target block and invalidates hot line. If a block with zero write count is unavailable, If the target is not found, all counters (including set counter) are decreased by the value of the least written block, which delays write redirection for a few more writes. This is done to prevent unnecessary write redirections when the write pattern to the set is more uniform. Note that decreasing the block counter value does not affect its functionality. Moreover, decreasing the counter also delays the need for the bitwise right shift operation of counters for replacement victim selection, which improves the precision of the technique.

To understand the working of WALL-NVC we use an illustration of a four-way set associative cache block of WALL-NVC having a threshold value, T=50. Each cache set is associated with a set counter and four block counters which keep track of the write count of each set and block, respectively. Let us consider a specific set A whose four blocks are $B0, B1, B2$, and $B3$. Consider an instance where the values of set counter and block counters of A are shown as in the first row in Figure 5. Write hit in a block increments block counter and A's set counter. Once the set counter reaches the threshold value (50), it searches for a write redirection target for heavily written block ($B2$). Since there is no target block with zero counts, the values of all block counters and the set counter are decremented by the value of least count (here it is 2 for $B3$) to create a block with zero counts. After the decrement operation, the cache is operated normally by incrementing the counters on write hits. When the set counter reaches the threshold again and write redirection is initiated; the redirection takes place by swapping the contents of most written block ($B2$) with least written block ($B3$) by swapping their contents using swap module. As $B2$ and $B3$ are valid blocks, write redirection results in an extra write in $B2$ and $B3$ and set counter is reset.

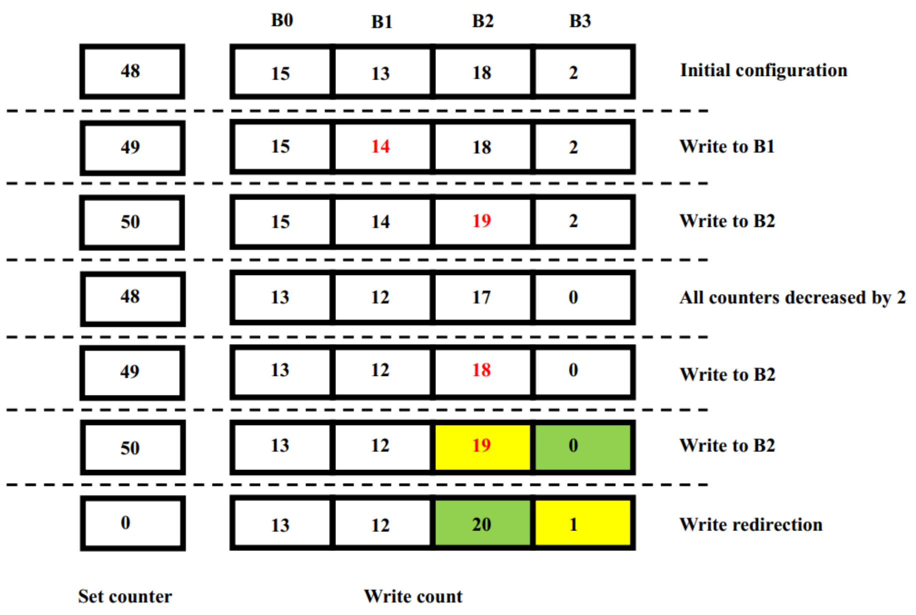

Fig. 5: Sample counter updating of WALL-NVC for threshold value, T=50

Table 1: System Configurtaion

CPU	1 GHz, Uni-core, Dual-Core, Quad- Core, ALPHA
L1 Cache 64 B block	Private, 32 KB, SRAM based split cache, 4-way set associative
L2 Cache	Shared 512 KB, NVM based unified cache 64 B block, 8-way set associative
Main Memory	8 GB

5 Experimental Setup and Result Analysis

We use gem5 [6], a cycle-accurate event based open source architectural simulator, to model and evaluate the performance of the last level WALL-NVC cache on unicore, dual-core and quad-core system architecture. Ruby module is used for simulating memory module, and MESI protocol is used for maintaining cache coherence. Details of the system configuration are given in Table 1.

We use the SPEC CPU 2006 benchmarks [5] to evaluate the performance of state-of-the-art techniques as well as WALL-NVC. Based on number of Writes Per Kilo Instruction (WPKI) to last level cache, these benchmarks are classified into three categories: Low (WPKI $\leq$ 9), Mid (10 $\leq$ WPKI $\leq$ 29), and High (WPKI $\geq$ 30) as shown in Table 2. This classification helps us to understand the impact of the existing and proposed techniques for applications having different

Table 2: Benchmark Classification based on WPKI

Category	Benchmark
Low	namd (Nd), soplex (So), calculix (Ca), astar (As), gromacs (Gr)
Mid	milc (Mi), libquantum (Lq), sjeng (Sj), bzip2 (Bz)
High	leslie3d (Ls), lbm (Lb), mcf (Mc), hmmer (Hm)

write characteristics. For architectural modeling in unicore system, we assign one benchmark instance to the core. For dual-core and quad-core architectures we create workloads by mixing instances of two and four benchmarks, respectively. We run these benchmarks and workload categories on an unoptimised NVM LLC (baseline), two state-of-the-art write balancing approaches: EqualWrites technique and EqualChance technique and the proposed WALL-NVC with a threshold value T=50 (WALL-NC50).

5.1 Performance Analysis

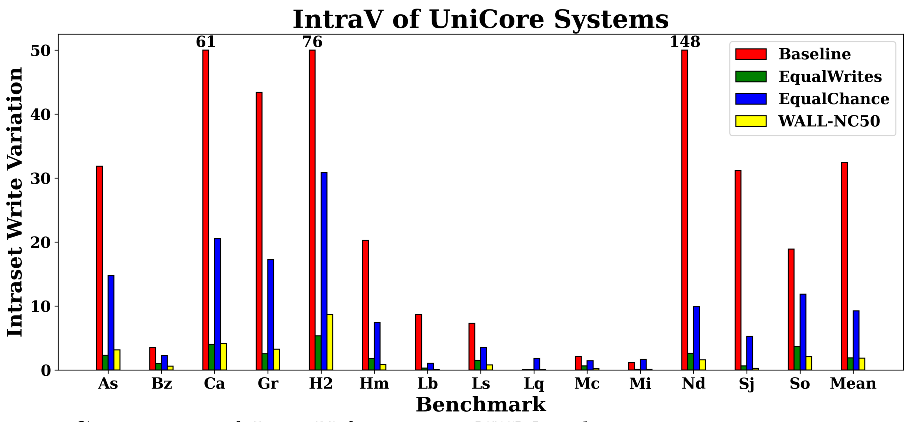

Fig. 6: Comparison of $IntraV$ for various NVM architectures in unicore system. (shorter the bar, the better)

Figure 6 shows the comparison of intraset variation ($IntraV$) for various architectures under study in unicore systems. We can clearly see that for benchmarks *leslie3d*, *lbm*, *mcf*, *milc* and *bzip2* the writes are happening more or less uniformly across different ways of a set there by having lower $IntraV$ for all architectures. This behavior is more prominent in benchmarks with Mid and High WPKI. In some low WPKI benchmarks like *namd, calculix* and *gromacs* there is a great improvement in $IntraV$ that we could achieve. So we conclude

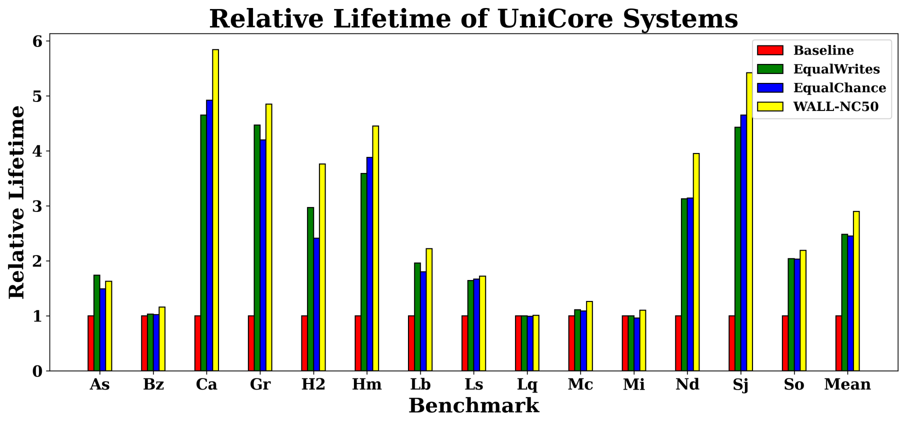

Fig. 7: Comparison of lifetime for various NVM architectures normalised to base configuration in unicore system. (taller the bar, the better)

that the write variance depends a lot on pattern of write hits on a given set than number of writes over a period of time. However, we can see that WALL-NC50 gives a low write variance irrespective of benchmarks classification that makes it suitable add-on to NVM caches.

Figure 7 shows comparison of lifetime for various architectures under study normalized to baseline in unicore systems. The inverse of the maximum write count to an LLC block is used to calculate lifetime. As expected we observe that benchmarks with high WPKI have limited improvement in lifetime due to the heavy writes to LLC. Compared to the baseline architecture, on an average WALL-NVC (T=50) improves lifetime by 2.90x and show 1.16x and 1.18x improvement compared to EqualWrites and EqualChance, respectively.

We also analyse the performance of our technique on dual-core and quad-core systems as well. WALL-NC50 shows an average lifetime improvement of NVM by 2.25x and 1.63x compared to baseline systems for dual-core and quad-core, respectively, as shown in Figure 10 and Figure 11. It improves 1.07x on dual-core systems when compared to EqualWrites and 1.02x when compared to Equalchance. Lifetime improvement of 1.10x and 1.02x are achieved for quad-core systems, respectively. Similarly we plot the intraset variation in dual-core and quad-core systems in Figure 8 and Figure 9, respectively. Since multicore framework needs more than one benchmark to run depending on number of core, we create workload consisting of benchmark mixes. Depending on WPKI values of the constituent benchmarks we create workloads marked as Low, Mid, Low-High, Mid-High etc. On careful analysis we find that due to the multiple applications accessing the shared NVM based LLC, the write variation created by writes of one core gets reduced by writes from other core. Hence, we could achieve only minor improvement in the lifetime improvement using Mid-High workloads in dual core and quad-core systems.

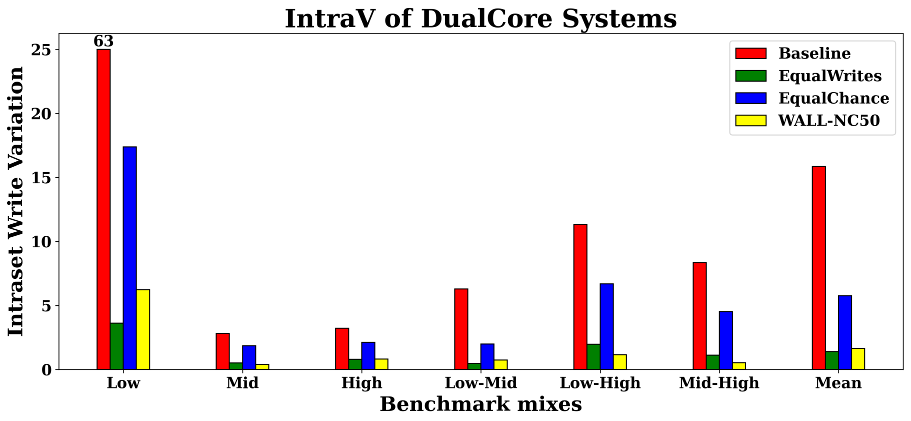

Fig. 8: Comparison of $IntraV$ for various NVM architectures in dual-core system. (shorter the bar, the better)

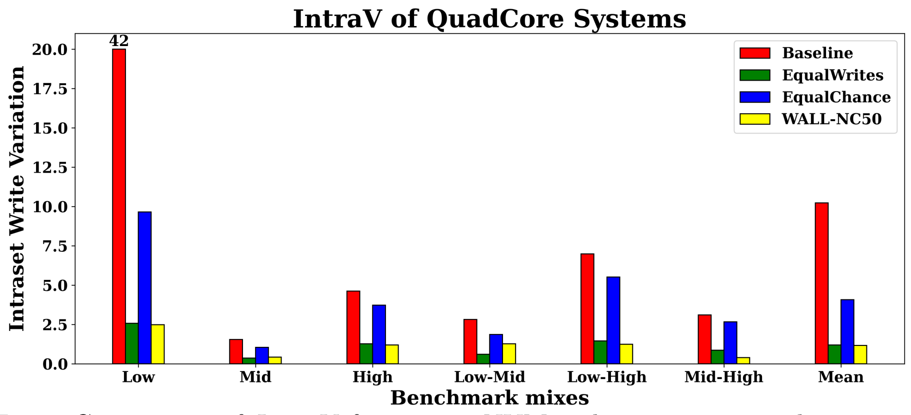

Fig. 9: Comparison of $IntraV$ for various NVM architectures in quad-core system. (shorter the bar, the better)

5.2 Sensitivity Analysis

We study the impact of the threshold value (T) by experimenting with five different values, T=$\{10, 30, 50, 70, 100\}$. Based on the endurance improvement and associated overhead, we fix the default value of T as 50. We also conduct a detailed sensitivity analysis on various threshold values. Table 3 shows the mean lifetime improvement with respect to baseline and intraset write variation, using different threshold values in WALL-NVC.

Another parameter that can impact the performance of the proposed architecture is the weightage given to LRU-CB while selecting a victim block for cache replacement. As discussed earlier, we compute the weighted aggregate average of each cache block using its LRU age and write count index. We explore two variants: (a) 80% for LRU age and 20% for write count index-0.2W and (b) 60% for LRU age and 40% for write count index - 0.4W. We compare these two vari-

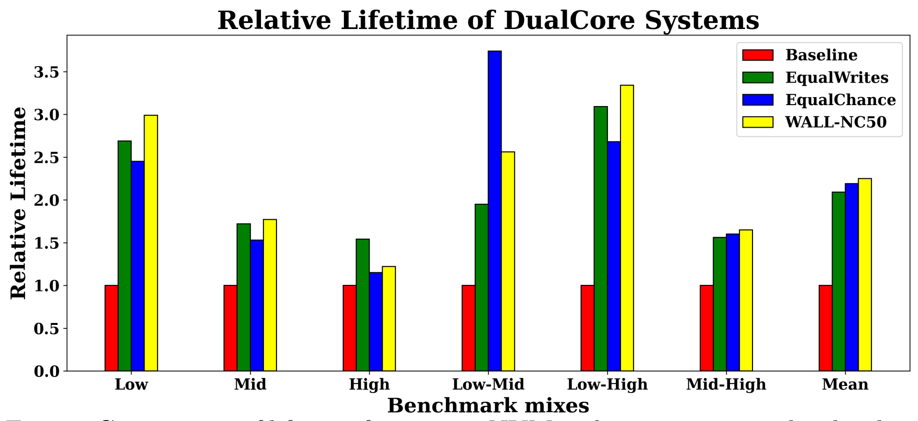

Fig. 10: Comparison of lifetime for various NVM architectures normalised to base configuration in dual-core system. (taller the bar, the better)

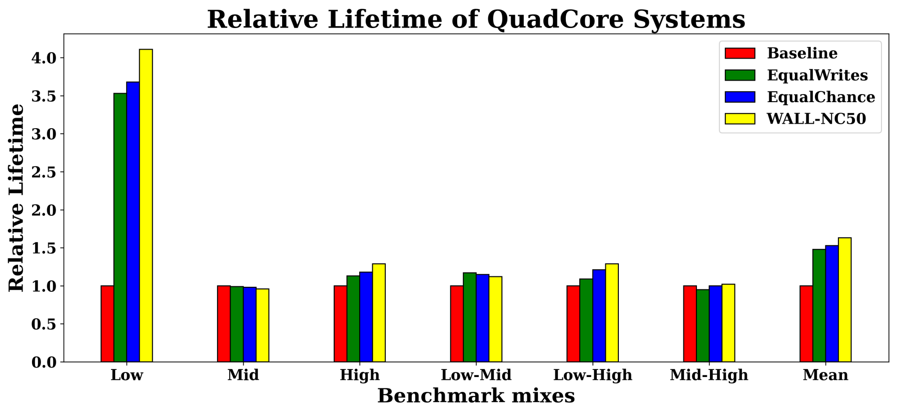

Fig. 11: Comparison of lifetime for various NVM architectures normalised to base configuration in quad-core system. (taller the bar, the better)

ants in a unicore system. The results for $IntraV$, relative lifetime wit respect to baseline and LLC hit rate are given in Figure 12, 13, and 14, respectively. We can see that 0.2W gives 1.13 time lifetime improvement and 2.16% improvement in $IntraV$ than 0.4W. We could also observe that there is not much impact on hit rate for these two variants in any benchmarks. Since 0.2W and 0.4W are better than baseline, we propose that a minimum weightage to LRU-CB (0.2W) should be given to get a suitable victim block. At the same time overemphasize to write count index (0.4W) diminishes the role of LRU age.

5.3 Overhead Analysis

WALL-NVC employs two types of counters: a set counter for each set and a block counter for each block. It also necessitates the use of a swapping module to swap the contents of hot and cold data blocks. The swapping module has 64

Table 3: Relative lifetime improvement (LT) and $IntraV$ of WALL-NVC for different threshold values

	Unicore		Dual-core		Quad-core	
	LT	$IntraV$	LT	$IntraV$	LT	$IntraV$
Baseline	1	32.41	1	15.85	1	10.22
10	2.32	6.55	2.08	7.28	1.31	6.58
30	2.54	2.46	2.63	1.48	1.68	0.34
50	2.90	1.85	2.25	1.65	1.63	1.17
70	2.56	4.08	2.23	2.59	1.59	1.43
100	2.57	4.20	2.35	2.26	1.53	1.52

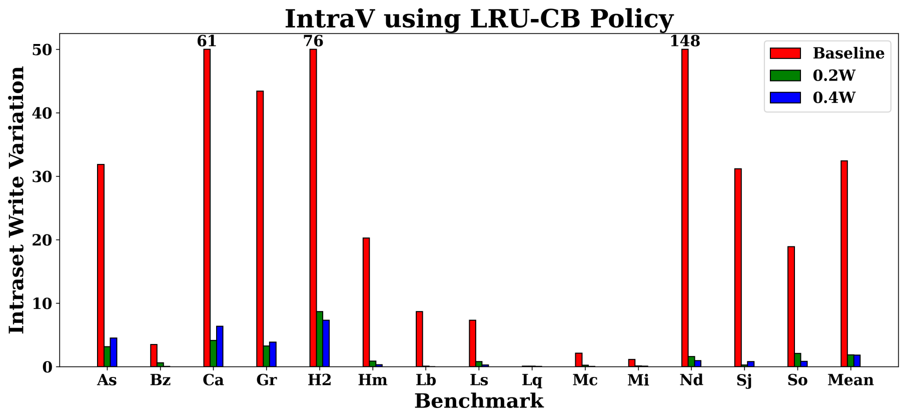

Fig. 12: Comparison of $IntraV$ for WALL-NC50 variants in uni-core system. (shorter the bar, the better)

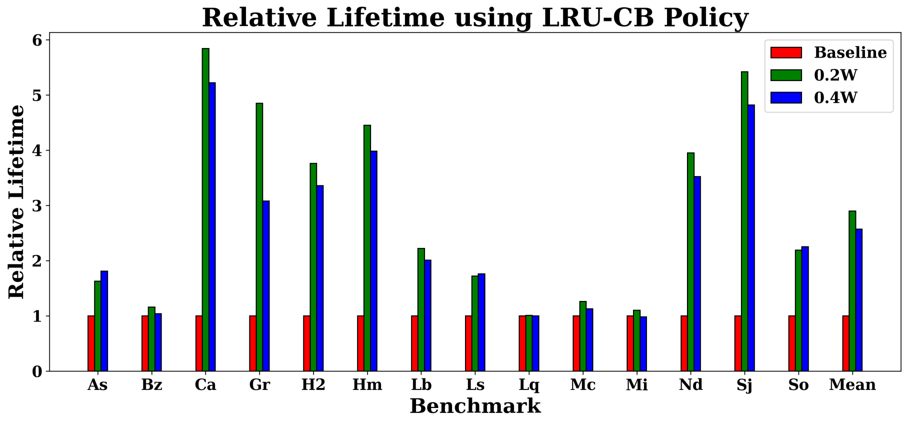

Fig. 13: Comparison of relative lifetime for WALL-NC50 variants in uni-core system. (taller the bar, the better)

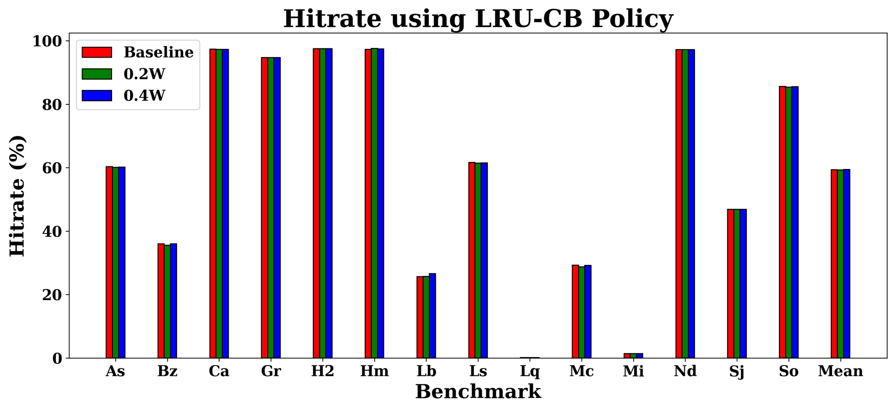

Fig. 14: Comparison of LLC hit rate for WALL-NC50 variants in uni-core system. (taller the bar, the better)

buffers, each of which is 64 bytes in size that incurs a total storage overhead of 2%. The SRAM based counters and swap buffers incurs a maximum power and area overhead of 0.47% and 1.47%, respectively when compared with baseline configuration. The cache block replacement policy, LRU-CB uses the same counters for victim selection; hence it does not incur any additional overhead.

6 Conclusion

Limited write endurance of NVM is always a critical challenge. In this paper, we proposed a new architecture called as WALL-NVC that used a write distribution policy and an NVM friendly Least Recently Used Cold Block cache replacement policy to improve lifetime of NVM caches. We observed that both the write distribution policy as well as the write aware replacement policy contributed equally to the improved performance. Experimental results showed that with minimal area and power overhead, our technique improved the lifetime for unicore, dual-core, and quad-core systems. To achieve further lifetime improvement, we look forward to incorporate a dynamic adaptive replacement policy based on run time inputs. Dynamic power-gating can be also be applied to adapt diverse write patterns of applications.

References

1. P. Chi, S. Li, Yuanqing Cheng, Yu Lu, S. H. Kang and Y. Xie, "Architecture design with STT-RAM: Opportunities and challenges," 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 109-114, doi: 10.1109/ASP-DAC.2016.7427997.
2. H. -. P. Wong et al., "Phase Change Memory," in Proceedings of the IEEE, vol. 98, no. 12, pp. 2201-2227, Dec. 2010, doi: 10.1109/JPROC.2010.2070050.

3. H. Akinaga and H. Shima, ”Resistive Random Access Memory (ReRAM) Based on Metal Oxides,” in Proceedings of the IEEE, vol. 98, no. 12, pp. 2237-2251, Dec. 2010, doi: 10.1109/JPROC.2010.2070830.
4. S. Mittal and J. S. Vetter, ”A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems,” in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, pp. 1537-1550, 1 May 2016, doi: 10.1109/TPDS.2015.2442980.
5. John L. Henning, ”SPEC CPU2006 benchmark descriptions”, SIGARCH Comput. Archit. News 34, 4 (September 2006), 1–17. doi:https://doi.org/10.1145/1186736.1186737
6. Nathan Binkert et al., ”The gem5 simulator”.SIGARCH Comput. Archit. News 39, 2 (May 2011), 1–7. DOI:https://doi.org/10.1145/2024716.2024718
7. S. Mittal and J. S. Vetter, ”EqualWrites: Reducing Intra-Set Write Variations for Enhancing Lifetime of Non-Volatile Caches,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 1, pp. 103-114, Jan. 2016, doi: 10.1109/TVLSI.2015.2389113.
8. Puneet Saraf and Madhu Mutyam, ”Endurance enhancement of write-optimized STT-RAM caches”, In Proceedings of the International Symposium on Memory Systems (MEMSYS ’19). Association for Computing Machinery, New York, NY, USA, 101–113, 2019 doi:https://doi.org/10.1145/3357526.3357538
9. S. Mittal and J. S. Vetter,”EqualChance: Addressing Intra-set Write Variation to Increase Lifetime of Non-volatile Caches”, 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14),Oct 2014 .
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,” 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), 2013, pp. 234-245, doi: 10.1109/HPCA.2013.6522322.
11. X. Dong, C. Xu, Y. Xie and N. P. Jouppi, ”NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, no. 7, pp. 994-1007, July 2012, doi: 10.1109/TCAD.2012.2185930.