Improving Lifetime of Non-Volatile Memory Caches by Logical Partitioning

S. Sivakumar
sivakumar@iitg.ac.in
Dept. of CSE,
Indian Institute of Technology
Guwahati, India

T.M. Abdul Khader
abdulkhadermagnetta@iitg.ac.in
Dept. of Physics,
Indian Institute of Technology
Guwahati, India

John Jose
johnjose@iitg.ac.in
Dept. of CSE,
Indian Institute of Technology
Guwahati, India

ABSTRACT

We are in an era of highly data-intensive applications, and the existing memory technologies are inadequate to meet their challenges. Non-Volatile Memories (NVMs) have emerged as a cost-effective alternative to the conventional SRAM based Last Level Caches (LLC) and DRAM-based main memories; however, they suffer from limited write endurance. Applications having non-uniform writes will cause heavily written blocks to fail faster than lightly written blocks, thereby reducing the lifetime of NVMs. Most of the modern processors use split organization in the first level cache and unified organization in the subsequent cache levels. Our proposed approach, ViSC (Virtually Split Cache) explores the write variation across the data and instruction blocks by virtually splitting unified LLC for wear-leveling. The logical mapping of LLC ways into instruction and data is interchanged periodically to distribute the writes uniformly. Our experimental results show that ViSC reduces the write variations significantly and improves the lifetime of NVMs by 1.94, 2.06, and 1.72 times for unicore, dual-core, and quad-core, respectively, by incurring negligible power and area overheads.

CCS CONCEPTS

• **Computer systems organization → Embedded hardware**.

KEYWORDS

Cache partitioning; Write variation; Intra-set variation; lifetime enhancement

ACM Reference Format:
S. Sivakumar, T.M. Abdul Khader, and John Jose. 2021. Improving Lifetime of Non-Volatile Memory Caches by Logical Partitioning . In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3453688.3461488

1 INTRODUCTION

We live in an era where technology is seamlessly integrated to our daily lives. With the emergence of multi-core processors, powerful handheld devices with high computing capability gained more popularity. These devices can smoothly run multiple applications in parallel. The data-intensive nature of many of the modern applications demands cost-effective and energy-efficient memory systems. Conventional memory technologies (SRAM/DRAM) suffer from low packaging densities, and high leakage power [1]. In this context, non-volatile memory (NVM) technologies like STT-RAM (Spin Transfer Torque RAM), PCM (Phase Change Memory), and ReRAM (Resistive RAM) are explored to cater to the growing demands of applications and devices. They exhibit more stable mechanisms for storing data compared to SRAM and DRAM based memories. NVMs intrinsically have higher latency and consume more energy to overwrite existing data. Even though they have zero leakage power and very high packing densities, they have limited write endurance. Write endurance can be defined as the maximum number of writes a memory cell can withstand before it is worn out. Table 1 shows typical write endurance values for various memory technologies [1].

Table 1: Write endurance for different memory technologies

SRAM	$> 10^{15}$
HDD	$> 10^{15}$
SLC Flash	$10^4 - 10^5$
DRAM	$> 10^{15}$
PCM	$10^8 - 10^9$
STT-RAM	$> 10^{15}$
ReRAM	10^{11}

If NVM is used for implementing cache memory, applications with non-uniform write patterns can cause some portion (sets or ways) of the cache memory to be heavily written compared to others [2]. We can broadly classify write variations in a cache as inter-set and intra-set variations. The write variation across the sets of cache memory is called inter-set variation, while the write variation across the ways of a given set is called intra-set variation. Along with the limited write endurance of NVMs, write variations generated by the applications also affect the performance of NVMs. Researchers have explored different ways to address these write variations to minimise its impact on memory lifetime. These approaches can be broadly classified into write overhead minimization [1] [3] and wear-leveling approaches [4] [5]. Write overhead minimization approaches reduce the write traffic and hence increases the lifetime of memory cells. Wear-leveling approaches make the

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461488

write distribution more uniform and thereby reduce intra-set and inter-set variations.

The L1 cache uses split organization (separate I and D caches) to avoid memory conflicts when instruction and data are accessed at the same clock cycle. It is seen that the unified caches offer a higher hit rate than a split cache of the same size because the number of instructions and data that can be accommodated in a unified cache is dynamically adjusted based on the access pattern. Most of the modern processors use private split cache architecture for L1 caches and shared unified design for the higher-level caches. However, few processors have a private split cache in L2 also. When NVMs are used in L1 caches, D-cache wears out faster than I-Cache as the data blocks are heavily written compared to the instruction blocks. Studies show that in an L1 split cache, on an average, D-cache has 472 times more writes than I-cache [6]. This clearly shows the heavy write variation between data and instruction. As mentioned before, instructions and data are kept together in unified last-level caches in multi-core processors. We observe that the blocks that store data are heavily written compared to blocks that store instruction.

In this work, we make the following major contributions:

- We analyse write variations in the last level unified NVM cache in run time and draws meaningful conclusions.
- We propose Virtually Split Cache (ViSC), which can reduce the intra-set variation, thereby increasing its lifetime. This is done by logically splitting the ways of unified last-level NVM cache into I and D caches and periodically interchange its logical mapping to attain wear-leveling.
- We evaluate ViSC on gem5 cycle-accurate simulator [7] using SPEC 2006 benchmarks [8]. Our experiments show that ViSC improves 72%, 106% and 94% average relative lifetime for quad-core, dual-core, and uni-core processor based systems, respectively.

The rest of the paper is organized as follows. Section 2 covers the related work in write variation management in NVMs, followed by motivation for the proposed work in Section 3. Our proposed technique ViSC is described in Section 4. We discuss the simulation framework and results in Section 5 and conclude the paper in Section 6.

2 RELATED WORK

Write variations occur at different levels of the cache memory hierarchy (from L1 to LLCs) and also at different granularity (ways to sets). i^2WAP (inter/intra-set Write variation-Aware cache Policy) has two features; namely, Swap-Shift to reduce inter-set variation and Probabilistic Set Line Flush to reduce intra-set variations [9]. In the Swap-Shift feature, whenever the number of writes to a particular set exceeds a threshold value, the contents of that set are swapped with the neighbouring set by exchanging the set IDs and invalidating their data. Probabilistic Set Line Flush uses a global counter to measure the total write count to the cache memory and invalidates the line whose write counter is saturated. This is performed under the assumption that the probability of a hotline saturating the counter is high. This invalidation further allows the hotline to be replaced by a cold line and migrate to a different location. EqualWrites [10] is another technique that addresses intra-set

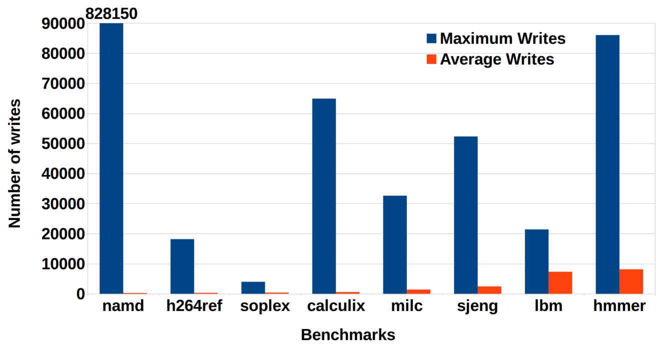

Figure 1: Average and maximum writes per way of various SPEC CPU 2006 benchmarks: Height difference between bars of a given benchmark indicates intensity of write level variations.

variations happening in non-volatile caches. It records the number of writes to each cache block. It identifies the location of the hot data blocks and redirects the future writes to a cold block.

The concept of write restriction window for reducing intra-set variations is a promising approach despite its hardware overhead [5]. Static write restricted window divides the cache into different logical windows, and one among them is selected as write restricted window, serving only read operation. All the write operations to the write restricted window is redirected to other windows that are not write restricted. Selection of the write restricted window is made periodically in a round-robin fashion. If the cache ways that are already heavily written are not included in the write restricted window, they may get penalised again. Dynamic windows based on write count and threshold value give better performance [2].

3 MOTIVATION

To understand the write variations in LLC, we model a unicore processor in gem5 [7] with two levels of cache and 8 GB main memory. We use 32 KB, 4-way set associative L1-I and L1-D caches. The 512 KB L2 cache (LLC) is 8-way set associative. Both L1 and L2 caches use 64 B blocks. Simulations are run using various SPEC CPU 2006 benchmarks [8]. We record the average and the maximum number of writes to each way of LLC and plot the results in Figure 1. We can observe that there is a considerable difference between average and maximum writes per way of LLC. We can see that except for few applications like *soplex*, the average maximum writes in a way is much more than the average writes in a way. This evident variation shows that if we use an NVM-based LLC the heavily written ways will wear out fast due to the limited endurance of NVMs, thereby reducing the entire LLC lifetime. Therefore, we need techniques to facilitate wear-leveling by distributing the writes of such heavily written ways among the other lightly written ways.

State-of-the-art wear-leveling techniques can be broadly classified into proactive and reactive approaches. Reactive approaches trigger wear-leveling when the write count of a memory cell surpasses a predetermined threshold value [9] [4] [5][11] while proactive approaches have their wear-leveling mechanism always active.

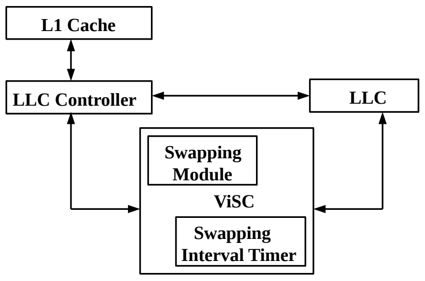

Figure 2: L1 cache - LLC interaction through the proposed ViSC module

They use runtime analysis on writes and use a counter-based hardware logic to restrict the writes happening to the hot lines or swap the contents of hot and cold lines. Write restriction mechanisms affect the performance of the memory access and incur additional hardware overheads to overcome the endurance limitations. This motivates us to explore the low overhead proactive approaches further with the goal to optimize it.

4 PROPOSED WORK

We observe that the cache blocks holding instruction are less written than the cache blocks holding data. To alleviate intra-set write variation in unified caches, we propose a wear-leveling approach, ViSC (Virtually Split Cache). Here, we make necessary architectural modifications on a unified cache to behave like a split cache. We split the NVM-based unified LLC virtually into the instruction and data cache space at the way level granularity. Hence, ways in each set are partitioned for instruction and data. For an *m*-way set associative cache, for each set, *k* ways can be reserved for storing instruction and rest (*m*-*k*) ways for storing data.

The block diagram of the proposed ViSC architecture is given in Figure 2. We can see that the proposed ViSC module is attached to the LLC controller. Every L1 cache miss that reaches the LLC controller is forwarded to the ViSC module for necessary background check and update. The ViSC module has a swapping interval timer that is used to make instruction ways to be marked as data ways. This process is coordinated and controlled by the swapping module. It generates necessary signals to mark the current *k*-ways used for storing instruction as data ways and new *k*-ways to store instruction. This reassignment and swap of ways between instruction and data is known as set reorganization. The swapping module triggers set reorganization at regular intervals using a suitable timer. When there is a write request to LLC, it checks whether the time elapsed since the last set reorganisation is greater than a predetermined threshold value. If so, the swapping module is invoked. Given that data ways are heavily written (hot ways), changing them to less written instruction ways (cold ways) curtails the number of writes and reduces the write variation. Algorithm 1 gives an overview of various operations done in the ViSC module.

L1 caches generally use the split organization to reduce structural hazards in the instruction pipeline while accessing memory for

Algorithm 1: Operational steps in ViSC module

```
instr_start = 0 \\first way of instruction ways;
data_start = 3 \\first way of data ways;
cache_assoc = 8 \\cache associativity;
num_inst_ways =3 \\number of instruction ways;
$t_p$: time elapsed since last set reorganisation, increments
  every clock cycle;
threshold = 100000;
List instruction_way_list : List of ways reserved for
  instructions sequentially from instr_start. Size = 3;
List data_way_list : List of ways reserved for data;
  sequentially from data_start. Size = 5;
repeat
for every L2 cache request R and block B do
    if R==read then
        normal read operation;
    else
        if $t_p$ > threshold then
            swap(instruction_way_list,data_way_list);
            instr_start = (instr_start+num_inst_ways) %
              cache_assoc;
            data_start = (data_start+num_inst_ways) %
              cache_assoc;
            instruction_way_list = {instr_start, instr_start+1,
              instr_start+2 };
            data_way_list = {data_start, data_start+1,
              data_start+2, data_start+3, data_start+4 };
            normal write operation;
        else
        end
        normal write operation;
    end
end
until end of execution
swap(instruction_way_list,data_way_list)
{
foreach instruction_way_list[i], 0 <= i < num_inst_ways
 do
    temp=instruction_way_list[i];
    instruction_way_list[i]=data_way_list[i];
    data_way_list[i] =temp ;
    } $t_p$=0;
end
```

instruction and data in the same clock cycle. Other levels of cache have unified organization where the instruction and data coexist in the same cache. We now discuss our proposed architecture in detail. Consider an 8-way set associative L2 cache with a virtual splitting of three instruction ways (k=3) and five data ways (8-3=5). Unlike L1 split caches that usually have equal-sized instruction and data caches, ViSC may have unequal partitioning for the L2 cache. This is motivated by the fact that in a unified cache, more space is needed for data as memory footprint of data is large. In ViSC, due to the logical partitioning, each set has three instruction ways

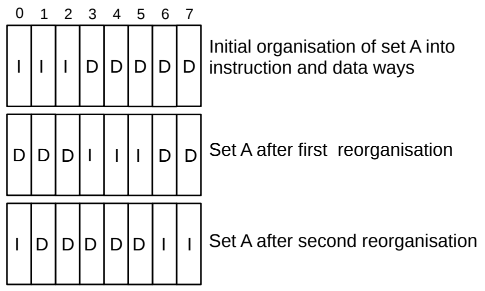
0 1 2 3 4 5 6 7
I I I D D D D D
Initial organisation of set A into instruction and data ways
D D D I I I D D
Set A after first reorganisation
I D D D D D I I
Set A after second reorganisation

Figure 3: Organisation of a Set A

Table 2: Simulation parameters

CPU	2 GHz, Uni-core, Dual-Core, Quad- Core, ALPHA
L1 Cache	Private, 32KB, SRAM Split cache, 64 B block, 4-way set associative
L2 Cache	Shared 512KB, 8-way set associative, 64 B block
Main Memory	8 GB

Table 3: Benchmark categories

Category	Benchmark
Low	namd, soplex, calculix, astar, gromacs
Mid	milc, libquantum, sjeng, bzip2
High	leslie3d, lbm, mcf, hmmer

where the instructions brought from the main memory are placed and data in the remaining five ways.

Figure 3 shows the instruction and data ways mapping in the initial phase and after subsequent reorganisations. Initially, when the cache warms up, Way 0, Way 1, and Way 2 are instruction ways. Instructions blocks mapped to this set are stored to any of these ways and data are be stored to the rest of the ways. We fix a threshold time to 100,000 clock cycles. After the expiry of threshold time, a reorganization will happen, and ways 3, 4 and 5 will be the instruction ways as shown in the figure. Similarly, at regular intervals, instruction ways and data ways are sliding across the given 8-ways in a circular manner.

All read requests to LLC are serviced normally. However, for write requests, ViSC checks whether the time since last cache reorganisation has crossed the threshold value. In such a case, set reorganisation process is initiated by swapping the contents of the current instruction/data way to the corresponding new data/instruction way. This swapping and copying process in the L2 cache happens in the background and is not in the critical path for instruction execution. The processor is still executing by accessing instruction and data from the L1 cache. We know that the instruction ways in ViSC are written only when a block is copied from the main memory to the L2 cache. However, the data ways are written not only during an L2 cache miss and subsequent bringing of the new block from main memory to L2 but also during write back and write through operations from L1 cache. L1-I cache block evictions do not create a write operation on the L2 cache. We observe that the dirty cache block evictions from the L1 cache have a significant share of writes on data ways of the L2 cache of ViSC. Since we reorganize I and D ways in ViSC, the number of writes in each way of a cache set gets balanced. As mentioned before, for an m-way set associative cache, different possible logical split up of instruction ways (k) and data ways (m-k) are possible. For an 8-way set associative cache we found that k=3 exhibits the best performance.

5 SIMULATION SETUP AND RESULT ANALYSIS

We model ViSC on gem5 [7], a cycle-accurate simulator and evaluate its performance using SPEC CPU 2006 benchmark suite [8]. Ruby module is used to simulate memory module and MESI protocol models cache coherence operations. LRU policy is adopted for cache replacement. We simulate ViSC for quad, dual and uni-core systems using the configuration given in Table 2. Based on WPKI values (Writes Per Kilo Instructions), we categorise our benchmarks into low, mid, and high as shown in Table 3. We study the impact on a lifetime under various threshold time values and plot the results in Figure 4. Threshold time of 100,000 cycles gives better performance for benchmarks with very high write variation and very low variation, and this reasoned us to fix the threshold time as 100,000 cycles. We analyse the lifetime of ViSC and Dynamic Window Write Restriction (DWWR) [2] and compare them with the un-optimized baseline system. We consider a normal NVM-based L2 cache that does not have any write balancing approach as the baseline. DWWR divides sets into equal-sized windows and dynamically enforce write restriction on heavily written ways. We consider only the DWWR technique for comparison as it had better performance than techniques such as PoLF, WAD, Equal chance, and SWRR [2].

We estimate relative lifetime by using the inverse of maximum write count to a block of memory. Figure 5, Figure 6 and Figure 7 shows the relative lifetime comparison in uni-core, dual-core and quad-core systems, respectively. We observe that for uni-core and dual-core systems, across all benchmarks, ViSC significantly improves NVM-based LLC's lifetime by its logical partition of instruction and data ways. In multi-core systems, the write imbalance created by one core may get automatically balanced by the writes of another core. This restricts the performance of ViSC to few workloads in a quad-core system.

To quantify the write variation, we use two parameters, namely; coefficient of intra-set variation (IntraV) and coefficient of inter-set variation (InterV) defined as follows,

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

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

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.

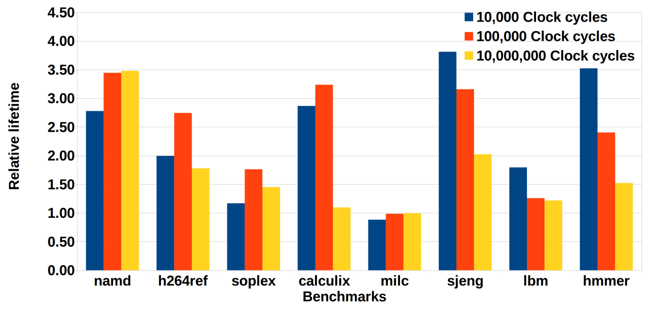

Figure 4: Life time improvement of various benchmarks for different threshold values (taller the bar, the better)

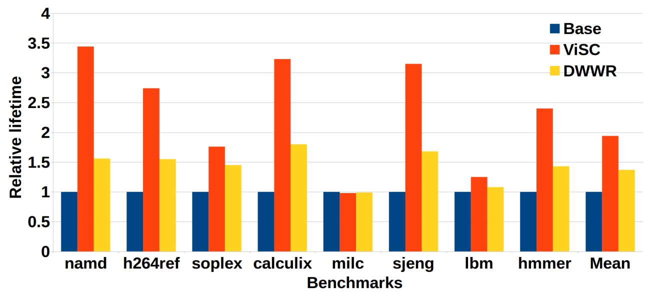

Figure 5: Comparison of lifetime improvement for DWWR and ViSC for unicore systems normalised to base configuration (taller the bar, the better)

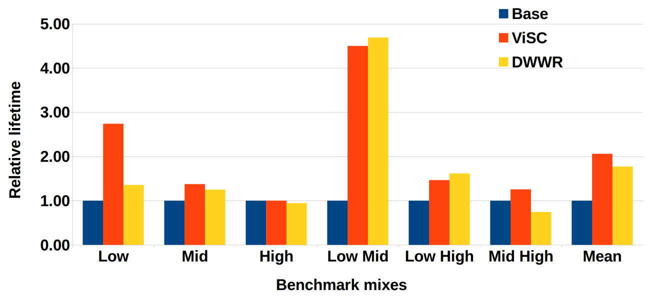

Figure 6: Comparison of lifetime improvement for DWWR and ViSC for dual-core system normalised to base configuration (taller the bar, the better)

$Write_{avg}$is average write count given by

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

IntraV is defined as the coefficient of variation of the average write count within cache sets and InterV is defined as the average of the CoV of the write counts across a cache set. Lower the value of IntraV and InterV better the write distribution within and across the sets. $Write_{avg}$ shows the average number of writes taking place in the cache memory

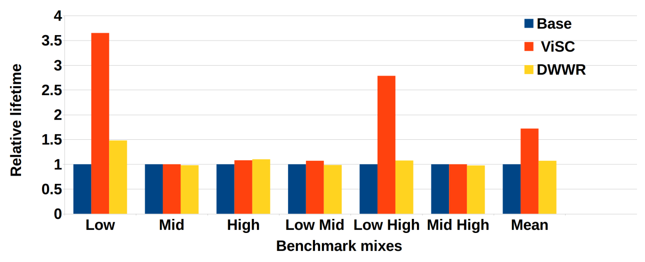

Figure 7: Comparison of lifetime improvement for DWWR and ViSC for quad-core system normalised to base configuration (taller the bar, the better)

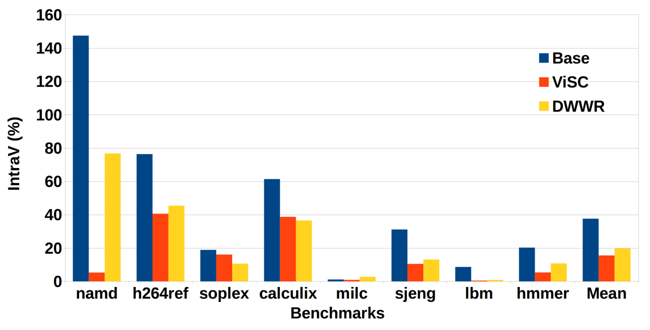

Figure 8: Comparison of intra-set variation for DWWR and ViSC for uni-core system (shorter the bar, the better)

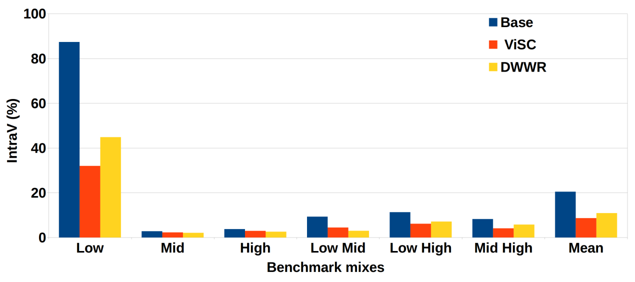

Figure 9: Comparison of intra-set variation for DWWR and ViSC for dual-core system (shorter the bar, the better)

Figure 8, Figure 9 and Figure 10 shows the intra-set write variations in uni-core, dual-core, and quad-core systems, respectively. ViSC gives the best performance for benchmarks and workloads having high intra-set variations whereas gives reasonably good performance compared to DWWR for workloads having medium and low intra-set write variations. For data-intensive applications, ViSC gives the best performance owing to the fact that some ways of LLC are reserved only for instructions. In write restriction strategies such as DWWR, a portion of LLC is inaccessible for the write operation, whereas in ViSC, entire cache memory is available for the write operation. Because of the non-restrictive write approach, ViSC is able to minimize the number of writes per way and hence the improvement in lifetime.

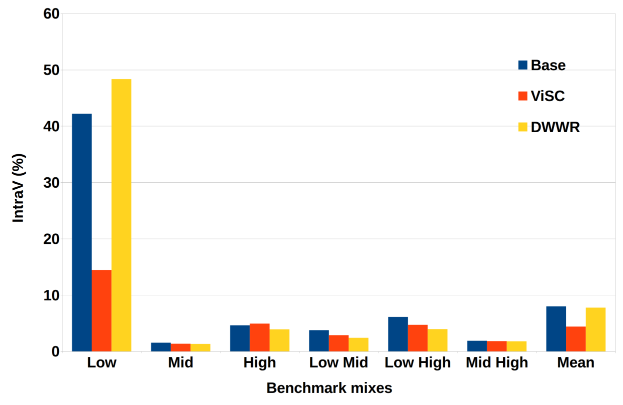

Figure 10: Comparison of intra-set variation for DWWR and ViSC for quad-core system (shorter the bar, the better)

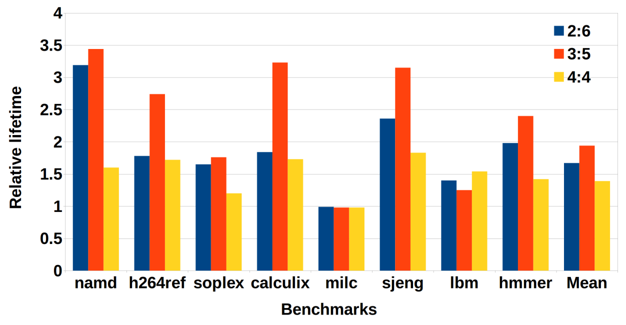

Figure 11: Comparison of relative lifetime for varying the count of I and D ways in the L2 cache of uni-core system (higher the bar, the better)

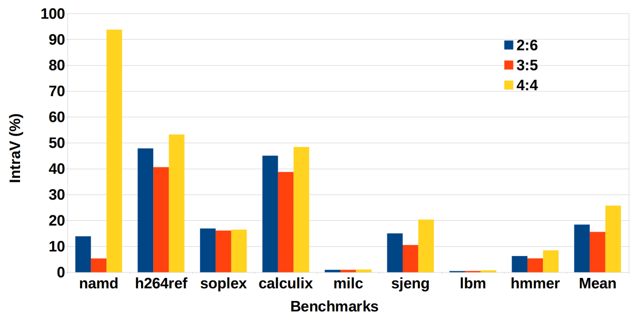

Figure 12: Comparison of intra-set variation for varying the count of I and D ways in the L2 cache of uni-core system (shorter the bar, the better)

ViSC has very minimal power and area overhead, 0.20% and 0.15%, respectively, owing to additional threshold timer and 67B memory that is used to swap the contents of instruction and data ways during the way reorganisation. We conduct a study by varying the number of ways for instruction (k) and data (8-k) as shown in Figure 11 and Figure 12. Based on this, we fix k as 3.

6 CONCLUSION

Write variations at the inter-set level and intra-set level substantially reduce the lifetime of NVM caches. We proposed ViSC, a technique that efficiently reduces the intra-set variation occuring within the sets of LLC. The key contribution of this paper is the concept of virtually splitting the L2 cache into data and instruction ways. Since data cache is heavily written and instruction cache is written exiguously, change in the logical mapping of instruction and data ways periodically is proposed in order to distribute the writes uniformly. Our technique shows significant improvement of the intra-set variation with negligible hardware and power overheads over the baseline architecture and the state-of-the-art DWWR technique. To improve ViSC further, whenever write intensive applications run, cache reorganisation can be made more frequent and vice versa by adjusting the switching interval. ViSC technique has a huge potential in enhancing the lifetime of NVM-based LLCs in modern multi-core processors.

REFERENCES

[1] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems", IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 5, pp. 1537-1550, May 2015.

[2] S. Agarwal and H. K. Kapoor, “Improving the lifetime of non-volatile cache by write restriction,” IEEE Transactions on Computers, pp. 1–1, January 2019.

[3] P. Huang, G. Wan, K. Zhou, M. Huang, C. Li, and H. Wang, “Improve effective capacity and lifetime of solid state drives,” in Proc. Int. Conf. Netw., Archit. Storage, 2013

[4] S. Mittal and J. S. Vetter, “EqualWrites: Reducing intra-set write variations for enhancing lifetime of non-volatile caches,” IEEE Trans. Very Large Scale Integr. Syst., vol. 24, no. 1, pp. 103–114, Jan. 2016.

[5] S. Agarwal and H. K. Kapoor, “Towards a better lifetime for nonvolatile caches in chip multiprocessors,” in Proc. 30th Int. Conf. VLSI Des. 16th Int. Conf. Embedded Syst., Jan. 2017, pp. 29–34.

[6] E. A. Hamed Farbeh, Amir Mahdi Hosseini Monazzah and E. Cheshmikhani, “A-cache: Alternating cache allocation to conduct higher endurance in nvm-based caches,” IEEE Transactions on Circuits and Systems II: Express Briefs, pp. 1–1, November 2018.

[7] 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, pp. 1–7, Aug. 2011

[8] J.L. Henning, “SPEC CPU2006 benchmark descriptions,” SIGARCH Comput. Archit. News, vol. 34, no. 4, pp. 1–17, Sep. 2006

[9] Jue Wang, Xiangyu Dong, Yuan Xie, Norman P. Jouppi, “i2wap: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations,” 19th International Symposium on High Performance Computer Architecture (HPCA), pp. 234 – 245, February 2013.

[10] S. Mittal and J. S. Vetter, “EqualChance: Addressing intra-set write variation to increase lifetime of non-volatile caches,” in Proc. Workshop Interactions NVM/Flash Operating Syst. Workloads, 2014

[11] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad, “Sequoia: A high-endurance NVM-based cache architecture,” IEEE Trans. Very Large Scale Integr. Syst., vol. 24, no. 3, pp. 954–967, Mar. 2016.

[12] X. Dong, C. Xu, S. Member, Y. Xie and N.P. Jouppi, "NVSim: a circuit-level performance energy and area model for emerging nonvolatile memory", IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 31, no. 7, pp. 994-1007, 2012.