Improving the Lifetime of Non-Volatile Cache with Modified ViSC and Write-Aware Swapping

1st Deepak Kumar
Department of Computer Science
Indian Institute of Technology (IIT)
Ropar, India
2022csm1002@iitrpr.ac.in

2nd Hrishikesh Bodkhe
Department of Computer Science
Indian Institute of Technology (IIT)
Ropar, India
2022csm1006@iitrpr.ac.in

***Abstract*—Currently, we are in a time where there are many applications that require a significant amount of data, and the current memory technologies are not sufficient to handle the challenges they present. To address this, Non-Volatile Memories (NVMs) have emerged as a viable and cost-effective alternative to the traditional SRAM-based Last Level Caches (LLC) and DRAM-based main memories. However, NVMs have a limited write endurance, which means that blocks that are heavily written will fail faster than blocks that are written lightly. This can reduce the lifespan of NVMs when dealing with applications that have non-uniform writes. Many modern processors use a split organization for the first level cache and a unified organization for subsequent cache levels. As proposed in [10], ViSC (Virtually Split Cache) explores the write variation across the data and instruction blocks by virtually splitting unified LLC for wear-levelling. This was done in periodic intervals. Here, in our work, we propose a slight modification by eliminating the periodic swapping of data and instruction ways. For every write request, the swapping of the instruction and data ways will happen by observing the write variation between them.**

***Index Terms*—Intra-Set variation, Inter-Set Variation, Lifetime enhancement, Non-Volatile Memory (NVM), Last Level Cache (LLC)**

I. Introduction

As technology is scaling, different applications such as multimedia, virtualization software, AI/ML algorithms, gaming, etc. are demanding huge amounts of memory and thus, require large-sized Last Level Cache (LLC). Generally, these LLCs are SRAM-based. With the increase in the capacity of LLC, SRAM incurs various disadvantages such as high leakage power and limited density. To solve this, researchers are exploring Non-Volatile Memories (NVMs) as a potential solution for LLCs. Recent NVM technologies such as Spintronics (e.g. Spin-Transfer-Torque (STT-RAM)), Phase-Change Memory (PCM), and Resistive RAM (ReRAM) can improve energy efficiency for computing. They also have higher density and lower power leakage than SRAMs, making them a good option for building large last-level caches. However, these technologies have long write latencies that can slow down performance and cause congestion at the cache. Further, these NVMs have endurance issues – the write endurance is in several orders of magnitude, e.g., for STTRAM it is is 4×10^{12} [1], for ReRAM it is 10^{11} & for PCM it is 10^{9} [1], which is lower than the conventional SRAMs, which has endurance above 10^{15}.

Some other problem is related to blocking access to an application. The cache management techniques which are with us at present only focus on the performance improvement of the cache and are not taking care of the write endurance issue. For example, if an application is exploiting the temporal locality, then the number of accesses to some specific blocks will increase. Thus, the write variation across different blocks can be large with some blocks experiencing more writes than others. So, this large write variation combined with the limited endurance problem will result in crossing the endurance limit and hence will cause the failure of the device.

To address the endurance issues, suitable wear-levelling techniques like uniform distribution of writes should be followed, which will ultimately help increase the expected lifetime of NVMs and make them a reliable option for LLCs.

II. Motivation for Solution

Generally, in the memory hierarchy, at the L1 level we have a split organisation of the cache: a separate I-cache and a separate D-cache. This is followed to avoid memory conflicts and overcome structural dependency when instruction and data are accessed simultaneously. However, at the last level, we have a unified cache, which means the instructions and data blocks are contained in the same cache in a scattered manner. Typically, there will be more blocks containing data than those containing instructions. Thus, it means that there will be more writes for the data blocks than for the instruction blocks, leading to heavy write variation between data and instruction. So in an NVM-based LLC, the data blocks will wear out faster than the instruction blocks, resulting in the failure of the LLC. To address this issue, it is important to distribute the writes uniformly so that the write variation is reduced and early wearing out of cells can be prevented.

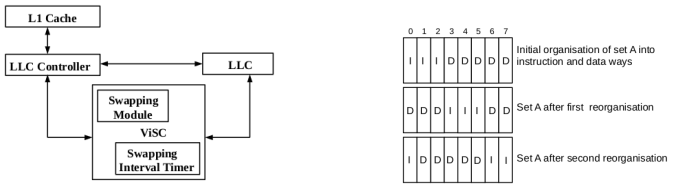
An approach discussed in [10], talks about Virtually Split Cache (ViSC), in which the unified NVM-based LLC is logically split into instruction ways and data ways. This logical mapping is periodically interchanged to achieve wear-levelling and uniform write distribution. The periodic interval is 100000 cycles, termed the threshold. Whenever a write request comes for a set, it is checked whether we have crossed the threshold

and if yes then only the set reorganisation, that is the swapping of instruction and data ways takes place. The major drawback of this approach is that for a given way, the reorganisation may take more time to happen, and the way might wear out. So, to overcome this a rapid movement of the instruction window can be explored.

III. Proposed Idea

A. Prior Approach

By observing that the cache blocks containing instructions are written less frequently than those containing data, [10] proposes a method called Virtually Split Cache (ViSC) for unified caches that address the unevenness in writing. The NVM-based LLC is split into two virtual spaces for instruction and data at the way level. This means that ways in each set are divided into two groups for instruction and data. In an m-way set-associative cache, k ways can be designated for storing instruction and the remaining (m-k) ways can be used for storing data in each set. Whenever a miss reaches the LLC controller, it is forwarded to the ViSC module, which contains a Swapping module and a Swapping Interval Timer. The Swapping module is responsible for Set reorganization. The set reorganisation is defined as the reassignment and swap of ways between instruction and data. This is done by the module at periodic intervals, checked by the swapping interval timer. All read requests to LLC are serviced normally. However, if a write request is sent to the LLC, it verifies whether the elapsed time since the previous set reorganization exceeds a predefined threshold. If it does, the swapping module is triggered. Since data ways (hot ways) are written extensively, switching them to less written instruction ways (cold ways) lowers the number of writes and minimizes write variation.



(a) (b)

Fig. 1: (a) Block diagram (b) Set Reorganisation [10]

Algorithm 1 Operational Steps in ViSC module [10]

$instStart$ = 0
$dataStart$ = 3
$cacheAsso$ = 8
$numInstWays$ = 3
t = 0 ▷ Time elapsed since last set reorganization, increments every clock cycle
$threshold = 100000$
ListInstWays = List of ways reserved for instructions sequentially from instStart. Size = 3
ListDataWays = List of ways reserved for data; sequentially from dataStart. Size = 5
repeat
  for every L2 cache request R and block B **do**
    if R == read **then**
      Normal read operation
    else
      if t > threshold **then**
        Swap(ListInstWays, ListDataWays)
        instStart = (instStart+numInstWays) % cacheAsso
        dataStart = (dataStart+numInstWays) % cacheAsso
        ListInstWays = {instStart, instStart+1, instStart+2}
        ListDataWays = {dataStart, dataStart+1, dataStart+2, dataStart+3, dataStart+4}
        Normal write operation
      else
        Normal write operation
      end if
    end if
  end for
until end of execution
Swap(ListInstWays, ListDataWays){
for instWayList[i] **do**
  temp = ListInstWays[i]
  ListInstWays[i] = ListDataWays[i]
  ListDataWays[i] = temp
  t = 0
end for}

Figure 1 shows the block diagram and an example of the set reorganisation for a 8-way set associative L2 cache with a virtual splitting of three instruction ways and five data ways. The swapping and copying process in the L2 cache happens in the background and is not in the critical path for instruction execution. The algorithm for the same is shown in 1.

B. Problem in the Prior Approach

The proposed approach in [10] performs the set reorganisation after every 1000000 cycles. This means that the reorganisation happens after a long period of time for a particular set. Now it may happen that in one of these periods, a large number of writes are requested on the last way of the set, which is the data way. But still, the set reorganisation will not occur until the threshold cycle period is over. Now even though the set reorganisation happens, for number of instruction ways to be 3 in an 8-way set associative cache, the swapping for the last data way will happen only after two periods of 1000000 cycles. Hence, in this duration, a lot of writes can happen on this data way.

C. Proposed Approach 1

The initial solution that we looked for was to perform the set reorganisation by swapping the heavily written data ways with the instruction ways after a threshold period of cycles. This can be done by keeping counters for each of the ways

of the set and measuring the number of writes for the set. A 16-bit counter can be maintained. For handling the overflow, whenever swapping is done for the chosen way, the associated counter with this way can be halved. But still maintaining counters for each way of the set is too much of an overhead in terms of space.

Algorithm 2 Modified Write-Aware Swapping for ViSC

```
instStart = 0
dataStart = 3
cacheAsso = 8
numInstWays = 3
threshold = Maximum value for write variation between instruction and data window
ListInstWays = List of ways reserved for instructions sequentially from instStart. Size = 3
ListDataWays = List of ways reserved for data; sequentially from dataStart. Size = 5
Maintain 2 counters for each set
repeat
    for every L2 cache request R and block B do
        if R == read then
            Normal read operation
        else
            instrWindowCount = Number of writes to the instruction window of the current set
            dataWindowCount = Number of writes to the data window of the current set
            WriteVariation = absolute(instrWindowCount - dataWindowCount)
            if WriteVariation > threshold then
                Swap(ListInstWays, ListDataWays)
                instStart = (instStart+numInstWays) % cacheAsso
                dataStart = (dataStart+numInstWays) % cacheAsso
                ListInstWays = {instStart, instStart+1, instStart+2}
                ListDataWays = {dataStart, dataStart+1, dataStart+2, dataStart+3, dataStart+4}
                Normal write operation
            else
                Normal write operation
            end if
        end if
    end for
until end of execution
Swap(ListInstWays, ListDataWays){
for instWayList[i] do
    temp = ListInstWays[i]
    ListInstWays[i] = ListDataWays[i]
    ListDataWays[i] = temp
    t = 0
end for}
```

D. Proposed Approach 2

The main aim is to reduce the waiting time of a way for swapping. This can be achieved by observing the write variation between the instruction window and the data window. For this, we can maintain two counters per set, that will keep the count of the writes for the instruction window and the data window. Thus, whenever a write request comes to a block, the write variation can be calculated for the respective set and if it is greater than some threshold value then set reorganisation takes place. In this way, rapid movement of the instruction window can be achieved if there are a large number of writes to a particular set, to distribute the writes uniformly in the set. Also, whenever the swapping occurs, the counters are flushed to avoid overflow and repetitive swapping. The key part of this approach is to choose the appropriate threshold value for comparing the write variation. The algorithm for the same is shown in 2.

IV. Experimental Setup

We have used Champsim for carrying out our simulations, which is a trace-based simulator. We have evaluated the performance of our approach using the SPEC CPU 2006 benchmark suite [11]. For the cache replacement policy, we have used the LRU policy. We have carried out the simulations for the proposed ViSC [10], unoptimised baseline and our proposed approach 2. The simulations are carried out for uni-core systems.

Regarding the system configurations, in [10], the authors have used a two-level cache system with the L2 cache as their LLC, but because of simulator dependencies, we have used the general configuration of the 3-level cache system. We have used a 2 GHz Uni-core CPU, a private 8-way set associative SRAM-based L1 cache of size 32 KB with 64 B block size, and a shared 8-way set associative SRAM-based L2 cache of size 64 KB with 64 B block size. The NVM-based LLC is a shared 8-way set associative cache with 512 KB size and 64 B block size. The baseline used is a normal NVM-based L3 cache that does not have any write-balancing approach. The lifetime of the NVM is analysed by comparing the results of ViSC, the unoptimised baseline and our proposed approach.

The benchmarks used for comparisons are - namd, soplex, astar, milc, libquantum, lbm, bzip2, leslie3d, mcf, hmmer and bwaves. We have extensively analysed the results for the traces which are having large number of writes such as - libquantum, lbm, mcf and hmmer. As discussed in [10], these traces have very high write variation, based on the WPKI values (Writes Per Kilo Instructions).

To quantify the write variation, the two parameters used are coefficient of intra-set variation (IntraV) and coefficient of inter-set variation (InterV) defined as follows,

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

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

Here, N is number of set in cache, M is the number of ways in a set, $W_{k,l}$ is the write count in set k and way l.

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.

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

Further we have also estimated the overall improvement (or degradation) of the cache lifetime. Lifetime Improvement (LI) is one such fairness metric discussed in [3]. Since the raw lifetime of non-volatile caches is determined by the first memory cell that wears out, it is necessary to consider three factors: the average write count, the inter-set write count variation, and the intra-set write count variation.

$$LI = \frac{W_{aver\_base}(1 + InterV_{base} + IntraV_{base})}{W_{aver\_imp}(1 + InterV_{imp} + IntraV_{imp})} - 1$$

Here, W_{aver_base} is the average write count for baseline, W_{aver_imp} is the average write count for the proposed implementation, $InterV_{base}$ & $IntraV_{base}$ are the coefficients for baseline while $InterV_{imp}$ & $InterV_{imp}$ are the coefficients for the proposed implementation.

V. Results and Analysis

As per the implementation of our base paper [10], we are able to swap the instruction and data ways by following the constraints given in the paper. But because of the use of a 3-level cache, we are far from improving compared to the baseline because the baseline has negligible intra-way write variation in our case. These abnormalities are also due to the simulator differences. As we can observe in Figure 2 the writes across ways in different sets is almost constant. But for the ViSC implementation, for the same sets and the same trace the number of writes are varying, shown in Figure 3. For the ViSC implementations, it is also observed that if the threshold cycles are increased then the intraset write variation is increasing which can also be observed from Figure 4. Further, a decrease in threshold results in an increase in swapping which may impact the overall performance. A key point to note here is that the intraset write variation for baseline is already less than the ViSC which can be observed from Figure 6.

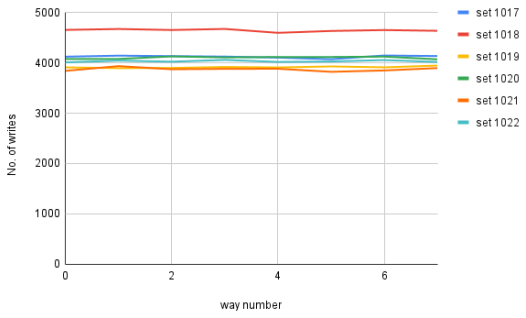


Fig. 2: Number of writes in different ways across different sets for baseline

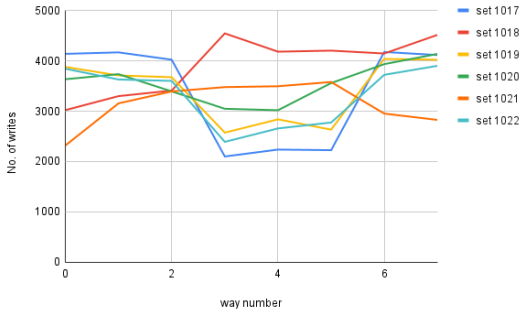


Fig. 3: Number of writes in different ways across different sets for ViSC

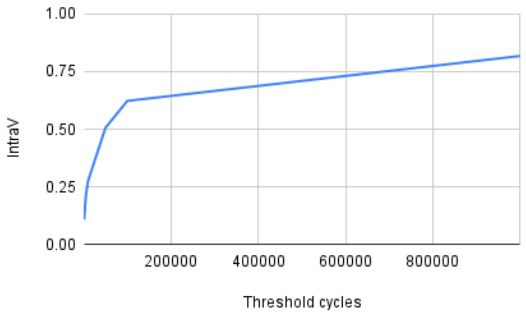


Fig. 4: Cycles vs IntraV for ViSC

Figure 5 shows the comparison of the average number of writes for baseline and ViSC. It is observed that the average writes are increased in the ViSC implementation. The reason for this can be the additional swapping of instruction and data ways which are indeed write operations.

The reason for the degradation in the lifetime can be the use of a 3-level cache or there is some automatic write balancing approach that is used by default in the Champsim simulator because if we compare the results which are proposed in the [10], the write variation results of our implementation of ViSC are matching with the write variation mentioned in [10]. Further, they have mentioned in [10] that they are using a

normal NVM-based L2 cache that does not have any write balancing approach as the baseline but Champsim may not have this type of configuration.

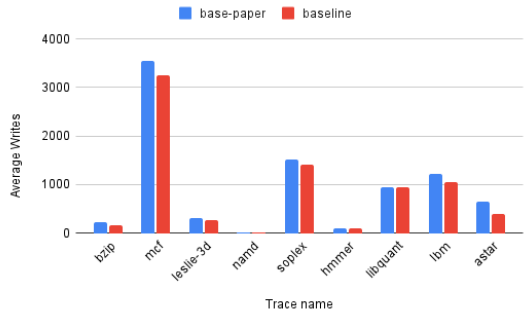


Fig. 5: Comparison of Average writes in baseline and ViSC

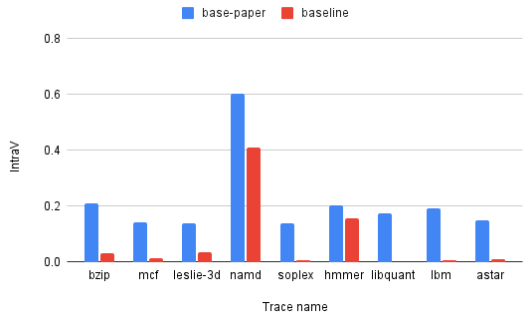


Fig. 6: Comparison of IntraV for baseline and basepaper for different traces

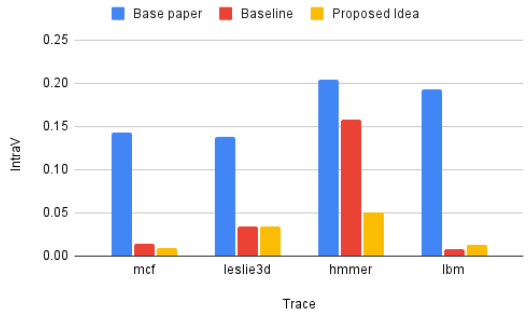


Fig. 7: Comparison of IntraV for baseline, Basepaper (ViSC) and proposed idea for traces with high write variation

Next, we have carried out the simulations on the traces which are having high write variations and compared the IntraV of baseline, ViSC with our proposed strategy, shown in Figure 7 It is observed that our strategy is performing better than the others in the majority of the traces. The threshold value of the write variation between the instruction and the data window taken for this analysis was 120 by observing the previous write variations. The reason for this improvement is the rapid movement of the windows so that the writes are distributed uniformly in the set. But this again has the overhead of swapping, as rapid movement means more swapping and ultimately results in more number of writes. So it is important to choose a better value for the threshold.

At last, we have calculated the lifetime improvement of ViSC over baseline and the lifetime improvement of the proposed idea over baseline. It can clearly be observed that our proposed strategy gives better results as compared to the ViSC for the traces which are having high write variations. Though overall it is a degradation as compared to the baseline.

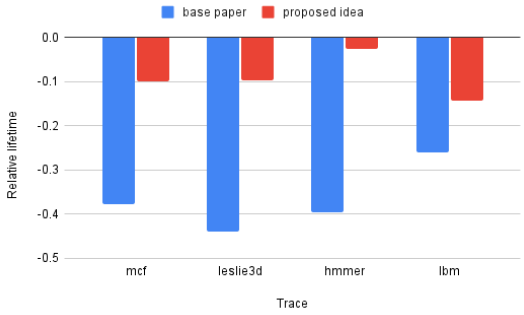


Fig. 8: Comparison of Relative lifetime improvement for Basepaper (ViSC) and proposed idea

VI. Conclusion

Our research indicates that the lifespan of NVM caches is significantly reduced due to variations at both the inter-set and intra-set levels. To address this issue, [10] proposed a technique called ViSC that effectively reduces the intra-set variation within the LLC sets. The primary contribution of the paper is the virtual division of the LLC into separate data and instruction ways. Since the data cache is written heavily and the instruction cache is written sparsely, the ideology suggested periodically changing the logical mapping of the instruction and data ways to evenly distribute the writes.

But the swapping period may be long for some hot data ways. To address this, we proposed a method called Write Aware Swapping that will allow the rapid swapping of the instruction and the data ways by observing the write variations between both windows. We analysed our results for different traces and compared the results with the baseline and ViSC. Further, we majorly focused on the traces which were having high write variations and concluded that our technique performed better as compared to the other two. We also found that the lifetime improvement was better for our modified approach as compared to ViSC, though overall resulting in degradation as compared to the baseline. The major drawback observed is the increase in the number of writes due to more swapping.

References

[1] Mittal, Sparsh, and Vetter, Jeffrey S. A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems. United States: N. p., 2015. Web. doi:10.1109/TPDS.2015.2442980.

[2] Mittal, S.; Vetter, J. A Technique for Improving Lifetime of Non-Volatile Caches Using Write-Minimization. J. Low Power Electron. Appl. 2016, 6, 1. https://doi.org/10.3390/jlpea6010001

[3] 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), Shenzhen, China, 2013, pp. 234-245, doi: 10.1109/HPCA.2013.6522322.

[4] Agarwal, Sukarn & Kapoor, Hemangee. (2017). Towards a Better Lifetime for Non-volatile Caches in Chip Multiprocessors. 29-34. 10.1109/VLSID.2017.4.

[5] 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.

[6] S. Mittal, J. S. Vetter and D. Li, "LastingNVCache: A Technique for Improving the Lifetime of Non-volatile Caches," 2014 IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, 2014, pp. 534-540, doi: 10.1109/ISVLSI.2014.69.

[7] Mittal, Sparsh & Vetter, Jeffrey. (2014). EqualChance: Addressing Intra-set Write Variation to Increase Lifetime of Non-volatile Caches.

[8] Mittal, Sparsh & Vetter, Jeffrey & Li, Dong. (2014). WriteSmoothing: Improving Lifetime of Non-volatile Caches Using Intra-set Wear-leveling. Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI. 10.1145/2591513.2591525.

[9] Mittal, Sparsh & Vetter, Jeffrey. (2015). AYUSH: Extending Lifetime of SRAM-NVM Way-based Hybrid Caches Using Wear-leveling. 10.1109/MASCOTS.2015.29.

[10] S. Sivakumar, T.M. Abdul Khader, and John Jose. 2021. Improving Lifetime of Non-Volatile Memory Caches by Logical Partitioning. In Proceedings of the 2021 on Great Lakes Symposium on VLSI (GLSVLSI '21). Association for Computing Machinery, New York, NY, USA, 123–128. https://doi.org/10.1145/3453688.3461488

[11] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News 34, 4 (September 2006), 1–17. https://doi.org/10.1145/1186736.1186737