2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

Write Variation aware Cache Partitioning for improved lifetime in Non-Volatile Caches

Arijit Nath and Hemangee K. Kapoor
Department of Computer Science and Engineering, IIT Guwahati, Assam, India-781039
Email: {arijitnath, hemangee}@iitg.ac.in

***Abstract*—With the reduction in transistor size, the transistor density within chips has increased in a dramatic fashion. It results in the introduction of more and more number cores within a Chip Multiprocessor allowing more than one applications to run concurrently. Therefore, applications put more pressure on the memory subsystem, particularly on the shared last level caches. Cache partitioning is a well-known technique to mitigate this issue where the shared last level cache is partitioned among the running processes based on their access demands. At the same time, people are trying to find new alternatives to traditional SRAM based cache memories because of their inherent limitations like low density and high leakage power. Non-volatile memories show great potential for being replacement choices for SRAM. However, NVMs have limitations like weak endurance, high write energy etc. Therefore, Cache partitioning in NVM based shared caches turns out to be more challenging. If not done judiciously, such partitioning may cause severe shortening of the lifetime of NVM caches.**

In this paper, we propose a hardware-based cache partitioning technique to intelligently partition the shared last level cache among concurrently running applications. In this technique, we do periodic re-partitioning of the shared cache by assigning heavily written cache ways to less written partitions so that writes are evenly distributed across the ways within the cache sets. Experimental evaluation shows a significant reduction in intraset write variation and improvement in lifetime.

***Index Terms*—Dynamic Cache Partitioning, Non-Volatile Memories, STT-RAM, Multi-core, Multi-programming, Shared caches, Last Level Cache (LLC).**

I. Introduction

Advancement of semiconductor technology has led to increase in the number of cores in modern chip multiprocessors which results in the execution of multiple applications concurrently. The applications running on various cores exhibit varying demands for memory. As a result, the pressure on the memory system specially on the shared last level caches (LLC) increases dramatically. Applications running on different cores access the shared cache as per the demands of their working set sizes. Certain applications might demand cache space very frequently for blocks having less usage, while certain other applications might have low space demands with longer usage tenure. Thus, an application may evict the blocks of other application(s) and vice-versa, increasing the overall miss rate. Cache Partitioning (CP) is a promising solution to mitigate this issue. It is a technique to provide disjoint shares of cache resources to the executing applications in order to fulfill their demands and provide fairness in the system.

In recent years, people are trying to find alternatives of traditional cache memories made up of SRAM due to their inherent drawbacks such as low density and high leakage energy.

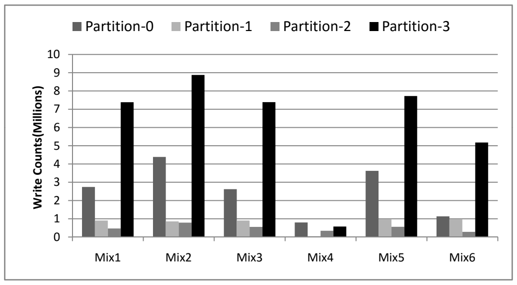

Fig. 1: Variation of write counts across different partitions

With this growing demand for new memory technology, Non-Volatile Memories (NVM) have become a strong candidate for replacement of SRAM in caches and DRAM in main memory. NVMs include Spin Transfer Torque Random Access Memory (STT-RAM), Resistive Random Access Memory (ReRAM), Phase Change Random Access Memory (PCRAM) etc. They provide high density, near-zero leakage power and good scalability. But, very low endurance of NVMs limits the possibility of replacing SRAM in cache by NVMs in practical scenarios. The write endurance values of various NVMs are : 10^{11} writes for ReRAM [1], 10^{8} writes for PCRAM [2] whereas although the prediction for STT-RAM is 10^{15} writes, the write endurance tested so far is $4 * 10^{12}$ [3], [4]. On the other hand, write endurance values for the traditional charge-based memories are more than 10^{15} writes. Cache Partitioning in NVM based caches is more challenging in comparison to partitioning in SRAM based caches. If CP in NVM caches is not done judiciously, then the cache partitions allocated to more write intensive applications may wear out more quickly due to excessive writes performed by the application. As a result, the lifetime of the NVM caches is further reduced because of the heavy write-pressure on the partitions of write-intensive applications. It is necessary to balance the write-pressure across different partitions of the LLC in order to extend its lifetime. Note that in this study we assumed that the partition sizes are sufficient for each application. Hence, we concentrate on the number of writes in the partition towards wear-leveling.

The write variation in the LLCs can be measured in two ways : *Inter-set* and *Intra-set* write variations. Inter-set write variation accounts for the non- uniform writes across different sets in the LLC. On the other hand, it may happen that different blocks within the same cache set may suffer different write-pressure due to variations in write intensity shown by the writing applications causing intraset write variation. To measure the impact of intra-set write variation of different concurrent applications, we conducted an experiment with an 8-way associative L2 cache in a quad-core system with different workload mixes. (Details about the experimental setup and workload mixes are reported in section IV). The LLC is statically partitioned by assigning an equal number of ways to four applications running concurrently. The applications associated with Partition-0 and Partition-3 are more write intensive as compared to the applications associated with Partition-1 and Partition-2. The results are presented in Figure 1. It is clear from Figure 1 that different partitions suffer different write pressure due to non-uniform write intensity exhibited by the running applications.

This paper proposes an efficient technique to reduce the intra-set write variation present in the LLC in the context of cache partitioning. The LLC is first partitioned way wise and uniformly among the running applications. In order to reduce intra-set variation, we propose an algorithm that assigns ways to the partitions to even out the write counts. The write variation is monitored over an interval and partition reconfiguration is done at the end of each interval. In this paper, we applied our proposed technique to STT-RAM based LLC (L2 cache). However, the proposed technique is easily applicable in other NVM based caches like ReRAM and PCRAM etc.

The main contributions of the paper are as follows :

- We propose a dynamic CP scheme: Write Variation Aware Cache Partitioning (WVarCP) to reduce the intra-set write variation in the LLC.
- The execution is divided into intervals and partitions are re-assigned at the end of each interval to distribute the ways as per the write intensity of the applications.
- The proposed technique is evaluated on GEM5 full system simulator. We also compare our proposed technique with baseline static partitioning and an existing state of the art technique that uses partition swapping [5].

The rest of the paper is organized as follows: Background and Related works are presented in section II. Proposed dynamic CP technique: WVarCP is discussed in section III. Section IV illustrates the experimental methodology and the results and analysis, followed by conclusion in section V.

II. Background and Related Works

A. Background

1) STT-RAM: The representational view of a STT-RAM cell shown in 2(a) . The STT-RAM cell consists of an access transistor and a Magnetic Tunnel Junction (MTJ). The MTJ of the STT-RAM cell is made up of two ferromagnetic layers viz. reference layer and free layer. These two layers are separated by a dielectric insulating material called tunnel barrier made up of Magnesium Oxide (MgO). The magnetic direction of the reference layer is fixed while the magnetic direction of the free layer can be parallel or anti-parallel depending on the spin-polarized current. Parallel magnetic direction represents lower resistance and '0' state in the STT-RAM cell as shown in fig 2(b). On the other hand, anti-parallel magnetic direction

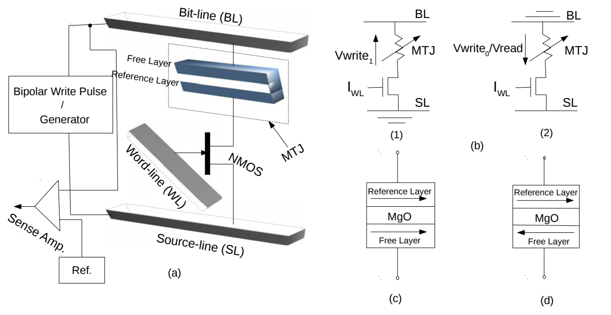

Fig. 2: (a) Representational view of STT-RAM cell (b) Parallel low resistance representing '0' state (c) Anti-parallel high resistance representing '1' state

corresponds to high resistance and '1' state in the STT-RAM cell (Figure 2(c)).

A Read operation in STT-RAM is carried out by applying a small voltage difference between the bit line and the source line. The generated current is sensed across the memory cell and compared with the reference current through the sense amplifier to detect '0' or '1' state stored in a memory cell. On the other hand, in order to perform a write operation, a large positive voltage difference is applied between the source line and the bit line for '0' and a large negative voltage difference for '1' state.

2) Coefficient of Intra-set write variation and lifetime: The paper proposes a technique to partition the STT-RAM based LLC dynamically to reduce the intra-set write variation. The intra-set write variation in a cache is measured in terms of coefficient of intra-set variation $IntraV$ defined as follows [3]:

$$IntraV = \frac{1}{S.Write_{avg}} \sum_{k=1}^{S} \sqrt{\frac{\sum_{l=1}^{A}\left(W_{k,l} - \sum_{m=1}^{A}\frac{W_{k,m}}{A}\right)^2}{A-1}} \tag{1}$$

In the eq, A is the cache associativity, S is the total number of cache sets, $W_{k,l}$ is the write count in way l of set k. $Write_{avg}$ is the average number of writes in a cache bank. The average write count $Write_{avg}$ is defined by the following equation:

$$Write_{avg} = \frac{\sum_{k=1}^{S}\sum_{l=1}^{A} W_{k,l}}{S.A} \tag{2}$$

The lifetime of a cache can be defined in two ways: 1) Raw lifetime or 2) Error-tolerant lifetime. The raw lifetime of a cache is defined by the first breakdown of a cache block. It is the inverse of maximum writes on a block of the cache. It does not consider any effort to error recovery. On the other hand, the raw lifetime of a cache can be extended by using error recovery methods. This is called error-tolerant lifetime [6]. In this paper, we focus on improving the raw lifetime of cache which is the backbone of the error-tolerant lifetime. We believe that by integrating the existing error recovery methods, the lifetime of a cache can be further improved.

B. Related Works

From the literatue survey, many works can be found addressing CP techniques and endurance issues in NVM based cache memories. In this section, we highlight some significant works done in CP and techniques to mitigate endurance issues in NVM based caches.

1) Cache Partitioning: [7] proposes a technique to reduce energy consumed by hybrid caches containing SRAM and STT-RAM. It dynamically partitions the cache based on an existing technique called read-write aware region based hybrid cache architecture. If a store operation causes a cache miss, then the block is assigned to the SRAM cache, whereas if a load operation causes a read miss and leads to a linefill, the block is assigned to the STT-RAM cache. Karthik T et al. [8] reported an energy efficient CP technique that forces data belonging to each core to be way-aligned across all the sets so that a particular way is owned by a single core at a time. It reduces energy in two different ways. Firstly, dynamic energy is reduced on each access because a core only needs to consult its own ways. Secondly, the static energy of the cache can be reduced by shutting down a way unused by any core. In [9], a write-back aware CP scheme is presented to partition the LLC among multiple running applications. It considers the reduction in cache misses as well as write-backs in order to partition the cache. Based on the observation that different applications show different write behaviors during execution, the authors in [5] proposed a CP technique to improve the lifetime of NVM based LLC. They used a swapping technique to balance the write variation across the partitions.

2) Endurance issues in NVM based caches: LastingNVcache presented by Mittal et al. [10] proposed a technique to reduce the intra-set variation in the cache. In this technique, if write counts in a cache block crosses a predefined threshold value, that block is flushed out of the cache. The block is redirected to a cold block on the next access. Such migrations of data items help in reducing intra-set variation. Mittal et al. proposed another technique -"AYUSH" [11] to improve the lifetime of an SRAM-NVM based LLC. On a write-hit to a NVM block, the oldest SRAM block containing cold data item is swapped with the newly arrived data item. Hence, probability will be high that future writes will be directed to SRAM that has high endurance. Authors in [12],[13] aimed to reduce intra-set and inter-set write variation in STT-RAM based LLCs whereas the techniques mentioned in [14], [15] tried to reduce write counts in the STT-RAM region in a SRAM/STT-RAM based hybrid LLC.

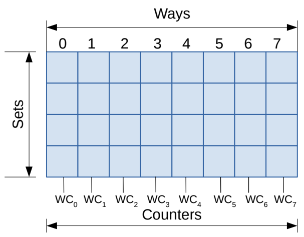

Fig. 3: Example of the proposed LLC architecture

III. Proposed Methodology: WVarCP

A. Architecture

The key idea of the proposed CP technique: WVarCP is to partition the LLC way wise and uniformly among the applications running on the cores. Due to the dissimilar write intensity of the applications, some partitions will suffer more write-pressure than the other partitions. In order to balance the write-pressure in all the partitions, this technique checks the write counts in the ways at regular intervals of application execution and assigns ways with more write counts to the less write intensive process and vice-versa. Each way in the cache is associated with a counter. This counter is incremented on every write operation in its corresponding way.

Consider a set associative last level L2 cache with S number of cache sets and associativity A. Let, total number of cores in the system be C, each core runs one application on it. Therefore, the total number of running application in the system is C. The cache is partitioned way wise and uniformly among the applications. Therefore, the partitioned cache has the following attributes:

- Total number of counters for the cache : A.
- Total number of ways per partition : A/C.

An example of the LLC architecture is shown in Figure 3. As shown in the figure, the LLC is 8-way associative and has 4 sets in total. A counter is associated with each way to count the number of write accesses in that way. For example WC_i is the counter associated with ith way, where $i = 0, ...7$.

B. Operation

The operation of the proposed Cache Partitioning algorithm is illustrated in Algorithm 1. In the algorithm, I acts as the tunable parameter for the reconfiguration interval (line 2). The applications are run for I_{normal} cycles treating the cache as normally available cache. At the beginning of each interval I, the array WI containing way ids are sorted in the non-increasing order of their write counts (line 17). Similarly, the array PI containing partition ids are sorted in non-decreasing order of their write counts (line 18). Total write counts in a partition can be defined as the sum of write counts of the ways assigned to that partition. At the beginning of the reconfiguration interval I, the algorithm re-partitions the LLC. It does so by assigning m (m is the number of ways per partition as defined in line 5) most heavily written ways to the least heavy partition p_0 (line 21), assigning the next m heavy lines to the next least heavy partition p_1 and so on. Ultimately, it assigns the m least written ways to the most heavily written partition p_{N-1} (line 23). Then, all the counters associated with the ways and partitions are reset for the next reconfiguration interval (line 23). The algorithm is repeated until the end of execution.

Figure 4 shows a working example of the proposed technique. Initially, an 8-way associative LLC (L2 cache) is partitioned way wise and uniformly among 4 running applications where each partition gets 2 ($m = 2$) ways. The initial partition as shown in the figure is: $P_0 = \{0, 1\}$, $P_1 = \{2, 3\}$, $P_2 = \{4, 5\}$, $P_3 = \{6, 7\}$ i.e., ways 0 and 1 are assigned to partition P_0, ways 2,3 are assigned to partition P_1, ways 4,5

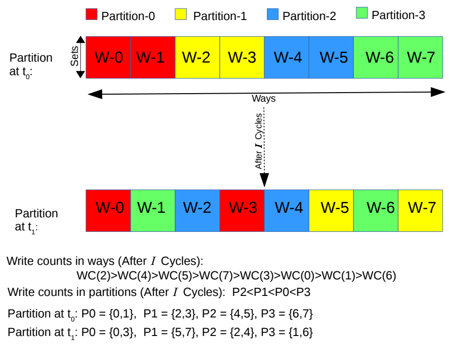
Partition-0
Partition-1
Partition-2
Partition-3
Partition at t0:
Sets
W-0 W-1 W-2 W-3 W-4 W-5 W-6 W-7
Ways
After I Cycles
Partition at t1:
W-0 W-1 W-2 W-3 W-4 W-5 W-6 W-7
Write counts in ways (After I Cycles):
WC(2)>WC(4)>WC(5)>WC(7)>WC(3)>WC(0)>WC(1)>WC(6)
Write counts in partitions (After I Cycles): P2<P1<P0<P3
Partition at t0: P0 = {0,1}, P1 = {2,3}, P2 = {4,5}, P3 = {6,7}
Partition at t1: P0 = {0,3}, P1 = {5,7}, P2 = {2,4}, P3 = {1,6}

Fig. 4: An working example of the proposed methodology

Algorithm 1: NVM Cache Partitioning

```
1  $I_{normal}$ : Warm up interval treating the cache as normally available cache
2  $I$ : Predefined reconfiguration interval
3  $N$ : Total number of running applications = Number of partitions
4  $A$ : Associativity of the LLC
5  $m$ : Number of ways per partition
6  $w_i$ : Id of $i$th way in the LLC, $0 \leq i < A$
7  $WI$ : Array containing way ids
8  $PI$ : Array containing partition ids
9  $p_i$ : Set of ways belonging to ith partition
10 $WC(i)$ : Write count of $i$th way in the LLC
11 $wcP_i$ : Write count of partition $p_i$ i.e., $wcP_i = \sum_{j=0}^{m-1} WC(j) \mid j \in p_i$
12 Initialize $WI$ to contain all the way ids i.e., $WI = \langle w_0, w_1, .., w_{A-1} \rangle$
13 Initialize $PI$ to contain all the Partition ids i.e., $PI = \langle p_0, p_1, .., p_{N-1} \rangle$
14 Run the application for $I_{normal}$ cycles treating the whole cache as normally
     available cache
15 repeat
16     for Every reconfiguration interval $I$ do
17         Update $WI$ to have ids of ways in non-increasing order of their write
             counts
18         Update $PI$ to have partition ids in non-descending order of their write
             counts
           /* By using expression in line 10                    */
19
20         Assign heavily written ways to lightly used partition by using arrays
             $WI$ and $PI$ i.e.,
21         $p_0 = \cup w_j \mid 0 \leq j < m$
22         $p_1 = \cup w_j \mid m \leq j < 2m$
           ⋮
23         $p_{N-1} = \cup w_j \mid m(N-1) \leq j < A$
24         $WC(j) = 0 \mid 0 \leq j < A$
25         $wcP_i = 0 \mid 0 \leq j < N$ /* Reset all the counters   */
26     end
27 until End of execution
```

are assigned to partition P_2 and finally ways 6,7 are assigned to partition P_3. At the end of the time interval, the write counts of the ways follow the descending order as shown in the figure. Also, write counts in the four partitions in ascending order is shown in the figure. The algorithm assigns the two most heavily written ways: way-2 and way-4 to the least heavy partition P_2. Then, it assigns the next two heavy ways: way-5 and way-7 to next least heavy partition P_1 and so on. Finally the two least heavy ways: way-1 and way-6 are assigned to most heavy partition P_3. So, the new cache partition is: $P_0 = \{0, 3\}$, $P_1 = \{5, 7\}$, $P_2 = \{2, 4\}$, $P_3 = \{1, 6\}$.

IV. EXPERIMENTAL EVALUATION

A. Experimental Setup

We implemented our proposed technique on a full system simulator: GEM5 [16]. The memory system is simulated using

TABLE I: System Parameters

Components	Parameters
Processor	2Ghz, Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 2-way set associative cache, 64B block, 1-cycle latency, LRU, write-back policy
L2 Cache	2MB, 8-way, Shared, 64B block, LRU, write-back policy
Protocol	MESI CMP Directory

TABLE II: Timing and Energy Parameters for STT-RAM L2 cache

L2 Configuration	Leakage Power (mW)	Hit Energy (nJ)	Miss Energy (nJ)	Write Energy (nJ)	Hit Latency (ns)	Miss Latency (ns)	Write Latency (ns)
2MB, 8way	6.504	0.158	0.047	0.314	14.556	5.235	12.520

Ruby memory module inside GEM5 along with MESI CMP based cache controller. The system parameters used in the experiments are shown in Table I. The energy parameters were obtained from NVSIM [17] at 32nm technology. These parameters are reported in Table II.

We evaluated our results using PARSEC benchmark suite [18] which consists of many multi-threaded applications like data mining, animations, multimedia etc. We constitute 6 Mixes with 4 benchmarks per mix as shown in Table III. The mixes are composed by combining the write-intensive applications with the less write intensive ones. Results are compared against an existing state of the art technique (Hardware based dynamic partition swapping [5]) and a baseline architecture :

- Baseline (BL): In BL, a statically partitioned STT-RAM based LLC is considered. The LLC is partitioned way wise and uniformly among 4 applications. If the cache is 8-way associative, then each application will have an equal share of two ways. The ways assigned to an application do not change during the entire execution.
- Partition Swapping [5]: In this existing technique, the LLC is partitioned way wise and uniformly among concurrently running applications. Each application will have an equal number of ways. A counter is associated with each partition to count the number of total writes in that partition. After a certain predefined time interval, the counters of each partition are checked and the partitions are swapped based on the counter values. The partition with the highest write counts is swapped with the partition having lowest write counts; the partition with the second-highest write counts is swapped with the partition having second-lowest write counts and so on. Note that the allocation is at the level of partition granularity, whereas we propose a way level wear-leveling before assigning the partitions.

B. Results and Analysis

The experiments were done on a quad-core system considering the following configurations of the last level L2 cache: 2 MB, 8-way associative L2 cache. We consider 4 applications running concurrently (each application on one processor) out of which two are more write-intensive and two are less write intensive. The reconfiguration interval I is chosen to be 500k cycles. A smaller interval gives small intraset variation in the cache but, at the cost of more swapping overhead. On the

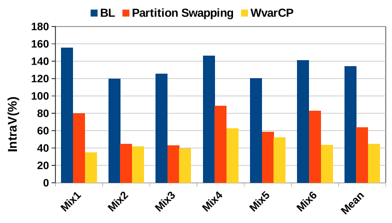

Fig. 5: Intraset write variation (Lesser is better)

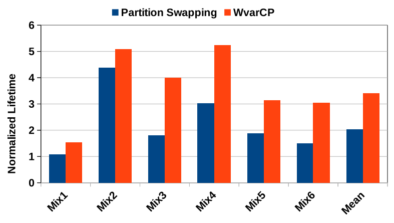

Fig. 6: Normalized lifetime with respect to BL (More is better)

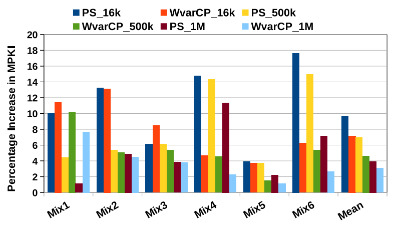

Fig. 7: Percentage increase (wrt. BL) in MPKI

TABLE III: Workloads

Mix1	Blackscholes,Bodytrack,Freqmine,Dedup
Mix2	x264,Bodytrack,Freqmine, Fluidanimate
Mix3	Canneal,Freqmine,Fluidanimate, Dedup
Mix4	Blackscholes, Bodytrack, Freqmine, Dedup
Mix5	Blackscholes, Canneal, Swaptions, Fluidanimate
Mix6	x264, Bodytrack, Freqmine, Dedup

other hand, a longer interval gives more intraset variation but, overhead related to swapping is less. Considering this existing trade-off, the interval of 500k cycles turns out to be the optimal interval that balances both intraset variations and swapping overhead. The figures show a comparative analysis between the proposed technique: WVarCP with the baseline (BL) and the existing technique: Partition swapping. However, results related to interval 16k and 1M cycles are reported in tableIV for better comprehension.

Effect on Intraset Variation: Figure 5 shows intraset variation across different workload mixes for BL, Partition swapping and WVarCP. On an average, WVarCP reduces intraset variation by 66.49% and 29.45% from BL and Partition Swapping respectively. Since BL uses static partitioning, its high intraset variation is quite straightforward. Due to the non-uniform write intensity of the running applications, the heavy applications will write much more than the less heavy applications. Hence, the write-pressure in the partitions belonging to the heavy applications will be more than the partitions belonging to the less heavy applications. The consequence is the obvious increase in intraset variation. However, the reduction in the intraset variation shown by WVarCP from Partition Swapping is because Partition Swapping swaps the heaviest partition with the lightest partition, the partition with second most write count with second least write count and so on. But, not all the cache ways inside a partition may be heavy. In other words, due to one or few heavy cache ways the entire partition gets heavy. So, assigning a less heavy way (inside a heavy partition) to a less heavy partition is not a good choice. In our technique: WVarCP, on the other hand, we make partitions on the basis of write counts of the ways. We assign most heavily written m (m is the number of ways per partition) ways to the least heavy partition, the second most heavy m ways to the second least heavy partition and so on. Thereby, we avoid assigning a less heavy way to a less heavy partition as in the case of Partition Swapping. Therefore, WVarCP shows a significant reduction in intraset variation as compared to Partition Swapping.

Effect on Lifetime: WVarCP shows a mean lifetime improvement of 3.41 times and 1.7 times that of BL and Partition Swapping respectively. Figure 6 shows normalized lifetime (wrt. BL) of Partition Swapping and WVarCP for different workload mixes. WVarCP distributes the writes to all the ways in the cache in a manner that avoids excessive writing in a particular cache block in a set. Since lifetime is inversely proportional to the maximum writes in a block, therefore the lifetime will definitely be increased by WVarCP.

Effect on MPKI with the change in Reconfiguration Interval (I): Figure 7 demonstrates the percentage increase (from BL) in MPKI (Misses per Kilo instructions) for Partition Swapping and WVarCP with change in the reconfiguration interval I for different workload mixes (In the figure, PS_x: Partition Swapping with x cycles interval length and WVarCP_x: WVarCP with x cycles interval length). Since BL uses static partitioning during the entire execution, therefore its MPKI will be less compared to Partition Swapping and WVarCP. Therefore, we take MPKI of BL as a benchmark of optimality and show the percentage increase in MPKI from BL for both Partition Swapping and WVarCP. It is evident from the figure that MPKI decreases for both Partition Swapping and WVarCP as the interval length I gets larger. When I is small, the reconfiguration of the partitions takes place more frequently compared to when I is large. Due to frequent re-partitioning, blocks allocated to a partition by an application may be evicted by another application if that partition is assigned to the second application after reconfiguration. It results in the increase in misses in the cache.

Table IV gives a summary of Normalized lifetime (wrt. BL), percentage reduction in intraset variation over BL and percentage increase in MPKI of Partition Swapping and WVarCP over BL for different reconfiguration interval I. It is clearly understood from the figure that for both Partition Swapping and WVarCP, intraset variation and lifetime decreases as I gets larger. On the other hand, MPKI decreases for larger Is due infrequent partition reconfiguration. It is also evident from the figure that WVarCP outperforms Partition Swapping by providing lesser write intraset variation, more lifetime and lesser MPKI for all the intervals.

Due to increase in the number of misses as I gets smaller, CPI in WVarCP shows an increase of 3.3%, 2.6% and 0.55% (as compared to 4.3%, 3.4% and 1.15% for Partition Sweapping) for interval size 16k, 500k and 1M cycles respectively over BL.

TABLE IV: Comparison Analysis for different Reconfiguration Interval values (I) Norm. Lft.=Normalized Lifetime wrt. BL, % Intra. Red= % Reduction in Intraset write variation from BL, MPKI. Red. =% Reduction in Misses per Kilo Instructions from BL

Policy	Param	Norm. Lft.	Intra .Red.	MPKI. Red.
Partition Swapping	Ref(I=500k)	2.04	52.51%	7.44%
	I=16k	4.82	61.92%	11.21%
	I=1M	2.05	51.06%	3.31%
WVarCP	Ref(I=500k)	3.41	66.50%	5.56%
	I=16k	8.71	71.16%	13.53%
	I=1M	3.77	60.79%	2.17%

Effect on Energy-Delay Product (EDP) with the change in Reconfiguration Interval (I): Figure 8 shows variation of normalized EDP(wrt BL) for Partition Swapping and WVarCP with interval length I. EDP increases when reconfiguration interval I gets smaller for both Partition Swapping and WVarCP (17%,11%,5% for Partition Swapping and 12%,10.6% and 4% for WVarCP for 16k, 500k, 1M reconfiguration intervals respectively.). As I gets smaller, frequent re-partitioning leads to more misses in the cache. It increases linefills or allocated writes in the cache which in turn increases dynamic energy consumption in the LLC due to high write energy associated with write operations in STT-RAM based caches. It results into high EDP for smaller interval values. Conversely, as I increases EDP improves (i.e. degrades). Thus, there exists a trade-off between interval duration I and EDP.

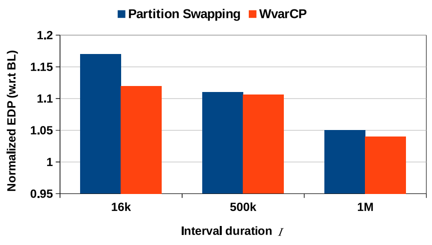

Fig. 8: Normalized EDP wrt BL (Lesser is better)

Storage Overhead: To measure the write count in each way, we use a 32-bit counter per way. In the 8-way associative LLC that we consider, total numbers of counters will be 8. Total number of bits used for all the 8 counters is $32 \times 8 = 256$ bits. The LLC is of size 2MB having $4096(= 2^{12})$ sets with 42 bits tag and 64B (512 bits) block size. Therefore, the total storage overhead associated with WVarCP over BL is: $\frac{256}{2^{12}\times 2^{3}\times(42+64\times 8)} \times 100\% = 0.00141\%$.

V. Conclusion

With the reduction in transistor density as a consequence of the reduction in transistor sizes, the number of cores being placed inside a single chip has been increased. As a result, concurrent execution of multiple applications on these cores has become feasible. However, due to increased multiprogramming, the memory system, particularly the LLC suffers from large access pressure by these running applications. Cache Partitioning proves to be an excellent solution to cope up with this issue where the LLC is partitioned among the applications based on how often these applications access the cache. On the other hand, people are trying to replace the traditional cache memories made up SRAM with emerging Non Volatile Memory technologies. Although NVM based caches provide high density and low leakage energy, they have their own downsides that are high write energy and low endurance. Therefore, imprudent cache partitioning in these caches may lower their lifetime in a severe manner.

In this paper, we proposed an efficient cache partitioning technique: WVarCP for STT-RAM based last level L2 cache to reduce intraset write variation and thereby enhancing its lifetime. In WVarCP, we re-partition the cache way wise and uniformly among running applications so that the relatively less heavy ways are assigned to the heavy partitions and vice-versa. Experimental results show that WVarCP reduces intraset variation by 66.49% and 29.45% respectively over BL and an existing technique- Partition swapping. Also, it improves the lifetime of the cache by 3.41 times and 1.7 times from the BL and Partition Swapping. Thus judicious assignment of cache ways to applications can make the NVM cache provide a longer service lifetime.

References

[1] Y. B. Kim et al., “Bi-layered rram with unlimited endurance and extremely uniform switching,” in *2011 Symposium on VLSI Technology - Digest of Technical Papers*, June 2011, pp. 52–53.
[2] M. K. Qureshi et al., “Phase change memory: From devices to systems,” *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.
[3] J. Wang et al., “i^{2}wap: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations,” in *HPCA*, Feb 2013, pp. 234–245.
[4] Y. Huai, “Spin-transfer torque mram (stt-mram): Challenges and prospects,” *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.
[5] S. Wang et al., “Word- and partition-level write variation reduction for improving non-volatile cache lifetime,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 1, pp. 4:1–4:18, Aug. 2017.
[6] S. Schechter et al., “Use Ecp, not Ecc, for hard failures in resistive memories,” in *ISCA*. New York, NY, USA: ACM, 2010, pp. 141–152.
[7] D. Lee and K. Choi, “Energy-efficient partitioning of hybrid caches in multi-core architecture,” in *VLSI-SoC*, Oct 2014, pp. 1–6.
[8] K. T. Sundararajan et al, “Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps,” in *HPCA*, Feb 2012, pp. 1–12.
[9] M. Zhou et.al., “Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems,” *ACM TACO*, vol. 8, no. 4, pp. 53:1–53:21, Jan. 2012.
[10] S. Mittal et al., “Lastingnvcache: A technique for improving the lifetime of non-volatile caches,” in *ISVLSI*. IEEE, 2014, pp. 534–540.
[11] S. Mittal and J. S. Vetter, “Ayush: Extending lifetime of sram-nvm way-based hybrid caches using wear-leveling,” in *MASCOTS*, Oct 2015, pp. 112–121.
[12] S. Agarwal and H. K. Kapoor, “Towards a better lifetime for non-volatile caches in chip multiprocessors,” in *VLSID*, Jan 2017, pp. 29–34.
[13] ——, “Targeting inter set write variation to improve the lifetime of non-volatile cache using fellow sets,” in *VLSI-SoC*, Oct 2017, pp. 1–6.
[14] ——, “Restricting writes for energy-efficient hybrid cache in multi-core architectures,” in *VLSI-SoC*, Sept 2016, pp. 1–6.
[15] ——, “Reuse-distance-aware write-intensity prediction of dataless entries for energy-efficient hybrid caches,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1881–1894, Oct 2018.
[16] N. Binkert et al., “The gem5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.
[17] X. Dong et al., “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, July 2012.
[18] C. Bienia et al., “The parsec benchmark suite: Characterization and architectural implications,” in *PACT*. New York, NY, USA: ACM, 2008, pp. 72–81.