Improving the Lifetime of Non-Volatile Cache by Write Restriction

Sukarn Agarwal, *Student Member, IEEE* and Hemangee K. Kapoor, *Senior Member, IEEE*

Abstract—The attractive features such as low static power and high density exhibited by the Non-Volatile Memory (NVM) technologies makes them a promising candidate in the memory hierarchy, including caches. However, the limited write endurance with the write variations governed by the access patterns and the applied replacement policies reduce the chance of NVMs as a successor of SRAM. These write variations are of concern as they not only breakdown the NVM cells but also reduce the effective lifetime. This paper proposes efficient techniques to mitigate the intra-set write variation to improve the lifetime of the NVM cache. Our first two techniques partition the cache into windows of equal size and distribute the writes uniformly across the cache set by employing the window as write-restricted or read-only. The selection of the window in these techniques is by rotation or with the help of counters. In our third technique, different cache ways are employed as a write-restricted over the period of execution to distribute the writes uniformly. Experimental results using full system simulation show the significant reduction in intra-set write variation along with improvement in the cache lifetime.

Index Terms—Cache memory, non-volatile memory, intra-set write variation, lifetime, partition, write restriction, throttling

1 Introduction

In modern Chip-MultiProcessors (CMPs), next-generation workloads are anticipated to have larger working set and highly data intensive. In order to accommodate the huge amount of data and to reduce the off-chip memory accesses, a large-sized Last Level Cache (LLC) is needed. Traditional caches made up of SRAM occupy a larger area and consume a significant portion of leakage energy. To mitigate the issues associated with the SRAM, researchers considered the emerging Non-Volatile Memory (NVM) technology as a choice in the memory hierarchy [1], [2], [3]. These NVM technologies include Spin Transfer Torque Random Access Memory (STT-RAM), Resistive Random Access Memory (ReRAM) and Phase Change Random Access Memory (PCRAM). The benefits of using these NVMs in the cache hierarchy are low static power consumption, good scalability and, high density. However, the major hurdles with the employment of NVMs in the cache hierarchy are costly write operations (such as latency and the energy) and the limited write endurance. The write endurance values for different NVM technologies are 10^{11} write operations for ReRAM [4], 10^{8} write operations for PCRAM [5] and for STT-RAM the predicted value is 10^{15} writes but the actual value tested so far is $4 * 10^{12}$ writes [6], [7]. Compared to these NVM technologies, the write endurance value for traditional charge-based memory technology i.e., SRAM or DRAM is more than 10^{15} writes.

- *The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati, Assam 781039, India. E-mail: {sukarn, hemangee}@iitg.ac.in.*

Manuscript received 28 Feb. 2018; revised 21 Dec. 2018; accepted 27 Dec. 2018. Date of publication 13 Jan. 2019; date of current version 15 Aug. 2019. (Corresponding author: Sukarn Agarwal.)
Recommended for acceptance by Y.-H. Chang, J. Ju, and M. B. Tahoori.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2019.2892424

In the real-time environment, the limited write endurance of NVMs is further affected by the write variations generated by the applications running on CMPs. In cache, the write variations are categorized into two categories: *Inter-set* and *Intra-set* write variation. The inter-set write variation takes place due to uneven write distribution across the cache sets. This indicates that some of the sets inside the bank are heavily written as compared to other sets, which leads to faster failure of the heavily written sets than the others. The intra-set write variation, on the other hand, is due to distinct write counts of the block inside the cache set. In particular, some of the blocks inside the set experience a large number of writes as compared to the other blocks, which results in the breakdown of the heavily written blocks faster than the other blocks. Due to these write variations, not only the lifetime of non-volatile cache reduces but the capacity of cache diminishes over the period. In this paper, we present efficient wear leveling techniques to reduce the intra-set write variation.

Existing literature consists of many state of the art, intra-set wear leveling techniques. Probabilistic Set Line Flush (PoLF) presented by Wang et al. [8] invalidates a cache block after a fixed number of writes based on a Flush Threshold (or FT). The only drawback here is the possible invalidation of MRU blocks, leading to more main memory accesses. EqualChance reported by Mittal et al. [9] transfers/swaps the write-intensive blocks within the cache set with invalid/clean blocks. This transfer/swap is based on a write counter threshold Υ. However, the transfer takes extra cycles and consumes extra energy due to more writes inside a cache bank.

Agarwal et al. [10] proposed a partition-based approach called Static Window Write Restriction (SWWR) that logically divides the cache into m equal sized windows (in such a way that each window contains an equal number of ways). During the execution, one window is used as a write-restricted window over a specific predefined interval. (The detailed

explanation of this approach is given in Section 3.5.) The main drawback of SWWR is the round robin window selection process. In other words, it does not consider the write counts of other windows of the set. Due to this, during the selection process, it may be possible that other heavily written windows (or ways) accumulate inside the cache bank. In this paper, we identify such heavily written windows (and ways) and propose methods to further reduce the intra-set write variation.

We applied our proposed techniques on different cache levels made up of STT-RAM [11] and ReRAM [4] memory technologies. However, the techniques can be easily extended to other non-volatile based caches such as PCRAM. The main contributions of this paper are as follows:

- We investigate SWWR [10] in detail on different configurations and levels of the cache and, with different parameters used in the methodology over multi-threaded and multi-programmed applications.
- We present a Dynamic Window based Write Restriction (DWWR) scheme that considers the write count of the different windows over the execution.
- We also propose a Dynamic Way Aware Write Restriction (DWAWR) scheme that takes into account the heavily written ways of a cache bank during the execution.
- The presented techniques are evaluated against three existing techniques: PoLF [8], WAD [12] and EqualChance [9] and the baseline STT-RAM/ReRAM without wear leveling support.
- Experimental evaluation over different levels of cache with different memory technologies shows significant improvement in the lifetime and reduction in intra-set write variation.

The paper is organized as follows: Related works are discussed in Section 2. Background and motivation are reported in Section 3. Proposed wear leveling techniques are presented in Section 4. Section 5 illustrates the experimental methodology. Results and analysis are presented in Section 6. Section 7 reports the parameter comparison analysis. Finally, we conclude this paper in Section 8.

2 Related Works

In the existing literature, many intra-set cache wear leveling techniques have been proposed. This section briefly illustrates all such policies. In addition, we also discuss the existing inter-set cache wear leveling and some of the write throttling policies.

Wang et al. [8] report a wear leveling technique to reduce both inter-set and intra-set wear leveling called i2wap. The first technique to reduce intra-set write variation is Probabilistic Set Line Flush (PoLF) that we already discussed in Section 1. The second technique to reduce the inter-set write variation is Swap Shift that changes the mapping of two sets after a fixed number of writes called Swap Threshold. Another wear leveling technique to reduce the write variation called Sequoia is reported by Jokar et al. [12]. In the first technique: G-OAS, the inter-set write variation is reduced by dividing the cache into multiple groups of sets and changing the set mapping of heavily written and lightly written set in each group by considering the counter associated with each set. The second technique: WAD reduces the intra-set write variation by displacing the data inside the set after the associated set counter (RSC) saturates. EqualChance proposed by Mittal et al. [9] reduces the intra-set write variation by transfer/swap operation within the cache set. A technique called Start-Gap proposed by Qureshi et al. [13] changes the line location to the neighbor location after a certain number of writes by using the two registers Start and Gap. A ReRAM NUCA that addresses the lifetime problem in a performance conscious manner is reported by Kotra et al. [14]. LastingNVCache proposed by Mittal et al. [15] reduces the intra-set write variation by adding the write counters with each block in the cache. After the counter reaches a specified limit, the write operation is skipped by invalidating the block. Different inter-set wear leveling policies that change the set/cluster/color mapping of the cache are proposed in [16], [17], [18]. A word level scheme reported by Wang et al. [19] reduces the write variation by exploring the narrow width of data. A hybrid cache architecture: Ayush presented by Mittal et al. [20] reduces the write variation by data block movement between the different regions of the cache. EqualWrite reported by Mittal et al. [21] reduces the write variation by moving the block from write-intensive location to cold location within the set. Previously several write throttling schemes at different levels of memory hierarchy have been proposed. A scheme Lady proposed by Lee et al. [22] enhances the lifetime of flash-based solid states drives by throttling the writes dynamically according to the workload characteristics. Mellow writes reported by Zhang et al. [23] improves the endurance of main memory by proposing a wear limiting technique. Here, slow writes are performed with lower power dissipation to improve the endurance. Another throttling technique proposed in [24] improve the reliability and lifetime of MTJ by using dynamic read and write factor.

In this work, we compare our proposed approaches with PoLF, WAD, and EqualChance because their (reasonable) hardware overheads are same as our proposed approaches, unlike that of previous approaches like LastingNVCache and EqualWrite which has counters on every block and are block-level policies. We also present a brief comparative analysis between the existing state of art wear leveling technique: I2WAP and the proposed approaches.

3 Background and Motivation

In this work, we applied our proposed techniques on an STT-RAM [11] and ReRAM [4] based non-volatile cache. This section briefly discusses both these memory technologies followed by the coefficient of write variation and lifetime along with motivation.

3.1 STT-RAM

Fig. 1 shows the representational view of STT-RAM cell. The STT-RAM cell consists of an access transistor and a Magnetic Tunnel Junction (MTJ). The MTJ is made up of tunnel barrier (made up of MgO) and the two ferromagnetic layers. The magnetization direction of one ferromagnetic layer is fixed, called reference layer. Whereas, the magnetization direction of other layer called free layer is variable and can be changed according to spin-polarized current. The tunnel barrier act as

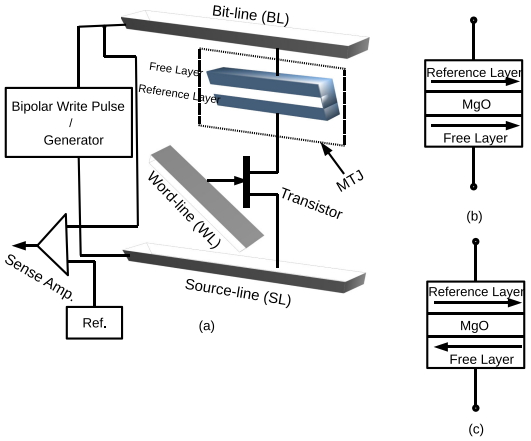

Fig. 1. (a) Representational view of STT-RAM cell (b) Parallel low resistance, representing '0' state (c) Anti-parallel high resistance, representing '1' state.

a thin insulating material between these two layers. The magnetization directions of the ferromagnetic layers are used to represent the data bit stored in the cell. In particular, the anti-parallel magnetization direction resembles high resistance that represents a logical '1'. While the parallel magnetization direction represents logical '0' and low resistance.

The read and write operations in the STT cell is performed by setting up the voltage difference between the source and a bit-line. Writing '0' in the STT cell is completed by applying the large positive voltage between the source and a bit-line. On the other hand, writing '1' in the STT cell is achieved by applying a large negative voltage. The read operation is performed by applying the small voltage between the source and a bit-line which in turn generates the current. This generated current is compared with the reference current in order to detect '0' or '1' state of STT cell.

3.2 ReRAM

A typical ReRAM memory cell is based on memristor technology where the resistance of the cell is based on the duration, polarity, and magnitude of the applied voltage. Fig. 2 shows the representational view of the ReRAM cell. A ReRAM memory cell is made up of two electrodes (e.g., platinum) and the metal oxide layers (e.g., TiO_2 and TiO_{2-x}) of different oxygen concentration. The layer TiO_2 which is having lower oxygen vacancy concentration is electrically non-conductive and, on the other hand, the high oxygen concentration layer TiO_{2-x} is highly conductive. The size of each layer or the resistance/state of the cell is changed by applying the voltage to the electrodes. If a negative voltage is applied, the thickness of TiO_2 layer increases and the generated ion-

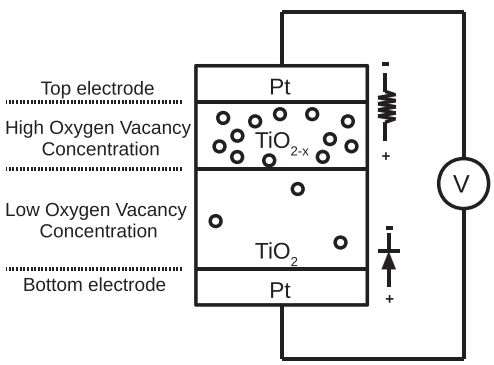

Fig. 2. Schematic view of ReRAM cell.

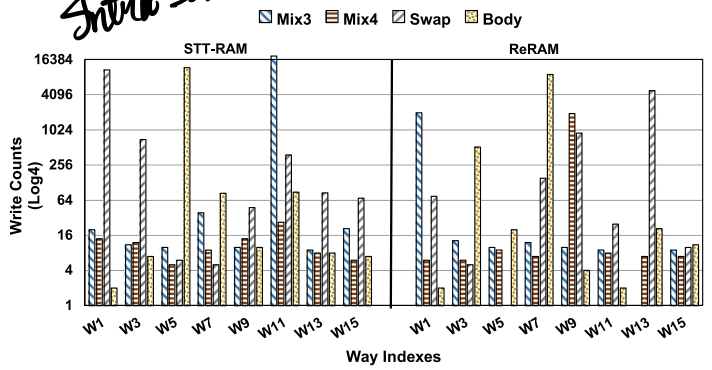

Fig. 3. Write counts inside the cache set for the four different workloads in the baseline STT-RAM/ReRAM.

path is highly non-conductive, and the cell resistance value is high (OFF state). On the other hand, the opposite effect is seen in the case of positive voltage (ON state).

3.3 Coefficient of Write Variation and Lifetime

This paper attempts to reduce the write variation present inside the cache. In order to quantify the write variation, we are taking the help of coefficients. Equations (1) and (2) represent these coefficients [8]: (i) $IntraV$: coefficient measuring the average variation within the cache set, and (ii) $InterV$: coefficient measuring the average variation across cache sets

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

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

In the above equations, S is the number of cache set, A implies the cache associativity, $W_{k,l}$ is the write count in the set k and way l. $Write_{avg}$ is the average write count in a cache bank and is defined by the following equation:

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

To quantify the lifetime we used the relative lifetime which is the inverse of maximum write count on the block of the cache.

3.4 Motivation

To measure the impact of write variation present inside the cache, we conducted experiments with 16 MB 16-way set-associative STT-RAM L2 cache with no wear leveling support in a quad-core system (Details about the experimental setup are reported in Section 5) and the results are presented in the Figs. 3 and 4. Fig. 3 shows the write counts of different blocks inside the cache set for four different benchmark applications and Fig. 4 represent the write counts across the cache set for the *bodytrack* workload. Table 1 shows the lifetime (in years) comparison analysis for the ideal case (by assuming writes are uniformly distributed), and in the baseline design along with the coefficient values of write variation for the different workload with no wear

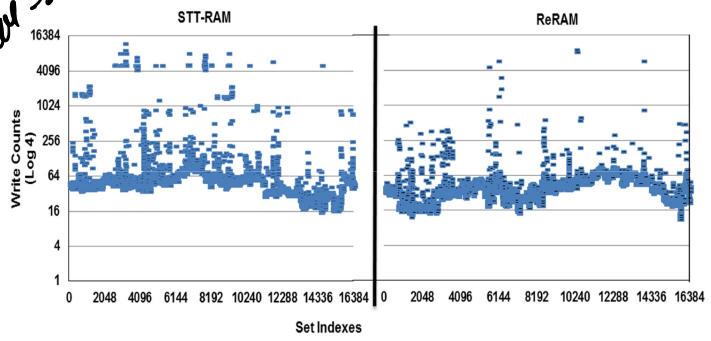

Fig. 4. Write counts across the cache sets in the baseline STT-RAM/ ReRAM.

leveling support. The conclusion that can be derived from the figures and table is the non-uniformity in the write counts within the cache set (Fig. 3 and row-2 and row-6 of Table 1) and across the cache sets (Fig. 4 and row-3 and row-7 of Table 1) reduces the effective lifetime (shown in row-4 and row-8 of the Table 1) of the cache and eventually shrinks the cache capacity. The write endurance value for STT-RAM is $4 * 10^{12}$ writes at 180 nm technology node [25] and that of Re-RAM is 10^{11} writes with reference to [6]. Table 1 presents the analysis using these values for endurance. Recently fabricated ICs at 22 and 28 nm technology nodes[1] for STT-RAM have reported the endurance to be 10^6 writes cycles [27], [28]. Other NVM technologies also suffer from lifetime issues. Thus the existing state-of-the-art highlights the endurance problems of NVMs at different technology nodes, motivating one to propose methods to handle the same. Our proposed techniques are general enough to be applicable to any device technology. To conclude, this paper attempts to reduce the write variation inside the cache set by proposing three effective policies.

3.5 Static Window Write Restriction (SWWR)

Our new proposed approaches Dynamic Window and Way Aware Write Restriction (DWWR/DWAWR) are based on SWWR. This section briefly illustrates SWWR which is our prior work [10] and, is given here for completeness.

3.5.1 Main Idea

The main idea of the SWWR is to partition the cache logically into m equal-sized windows and use a different window sequentially during the execution for a specific predefined interval (I). In each interval, one window of the cache is selected and, is treated as write restricted (i.e., read-only) window. In particular, during that interval, all the writes coming from L1/L2 cache i.e., Upper-Level Cache (called ULC) to the write restricted window of an L2/L3 cache (i.e., LLC) are redirected to other windows of the same cache set. At the end of a predefined interval, the next window of the cache is selected (as a write-restricted), and the process continues until the end of execution.

3.5.2 Algorithm

The working approach of the SWWR is elaborated through Algorithm 1. In our case, L2/L3 or LLC is the non-volatile STT-RAM/ReRAM based cache. In the algorithm, the tunable parameter I is used as a predefined interval (line 3). The total number of logical partitions or windows in the LLC cache are denoted by m (line 4). Note that each partition or window in the algorithm is represented by the variable W_i (line 5), where the range of i is from 0 to $m-1$.

When the application execution begins then for the initial I cycles, the cache is treated as an ordinarily available cache (line 6). Once the application executes I cycles, one of the windows in the cache is treated as a write-restricted window for next interval I (line 7) and, periodically for each interval, a new window is selected by rotation (lines 9 to 11). The process continues until the end of execution (line 26).

When the request R comes from ULC to LLC, the tag lookup operation is performed. Depending upon the result of the lookup operation for requested *block* B, the actions are taken as given below:

- *Read Hit:* The requested block B in the LLC is served normally to higher level cache irrespective of its location in the cache (lines 13 and 14).
- *Write Hit (PUTX or write-back) and requested Block B in W_i:* If B belongs to the selected window W_i with the invalid line(s) in the other windows of the same cache set, the request R from ULC is redirected to the first invalid line and the block B is invalidated from the respective location in write restricted window. In the other case, when there is no invalid line in the other windows of the same cache set, the LRU victim line v is picked from the Location L and the write-back operation is performed according to the dirty bit. Note that the location L is the location other than the write restricted window location. Once the victim v is evicted from the LLC, the write request from a ULC is redirected to the generated location L and the block B is invalidated from its respective location (lines 15 to 17).
- *Write Hit (PUTX or write-back) and B not in W_i:* The requested block B in the LLC is written normally by the ULC (lines 18 to 20).
- *LLC miss:* When the requested block is not present in the LLC, the request R from the ULC is forwarded to the next level of the memory hierarchy (main memory in our case). In this case, the newly arrived block is placed in the location other than the write restricted window W_i location (lines 21 to 23).

TABLE 1
Lifetime (in Years) Comparison for the Different Workloads in the Ideal STT-RAM/ReRAM and the Actual STT-RAM/ReRAM Based Caches

	PARSEC v2.1					SPEC CPU 2006			
Workload	Body	Cann	Dedup	Swap	X264	Mix1	Mix2	Mix3	Mix4
STT-RAM									
Ideal Lifetime	41.8K	3.8K	4.8K	14.3K	8.7K	3.21K	7.44K	14.9K	25.5K
IntraV	328.7%	57.4%	136.4%	361.4%	246.3%	21.6%	95.4%	223.6%	191.9%
InterV	450.7%	32.9%	66.4%	137.4%	325.3%	15.7%	213.9%	152.2%	96.5%
Baseline Lifetime	38	19.2	30.9	41.3	8.65	47.3	6.33	25.8	45.7
ReRAM									
Ideal Lifetime	2.12K	99.5	795.4	6.37K	462.2	185.4	175.2	565.5	666.2
IntraV	191.4%	49.37%	75.56%	396.6%	97.47%	11.25%	26.4%	80.6%	40%
InterV	332.8%	20.4%	161.9%	961.6%	50.9%	17.57%	17.1%	34.54%	17.45%
Baseline Lifetime	0.61	0.51	0.53	1.17	1	1.16	2.22	3.03	3.02

1. Advances in device technology at sub-10 nm can affect the lifetime [26] in future generation designs.

Algorithm 1. Static Window Write Restriction (SWWR)

ULC : Upper Level Cache i.e., L1/L2.
LLC : Last Level Cache i.e., L2/L3.
I : Predefined interval.
m : Number of logical partitions or windows.
W_i : ith logical partition or window that treated as read only (or write restricted) in the current interval. $0 \leq i < m$
Run application for I cycles treating the whole cache as normally available cache.
After I cycles treat one window at a time as write restricted and rotate turns in round robin fashion.
repeat
  for every interval I **do**
    $i = WinSelect(i, m)$
    Window W_i is selected as Write Restricted Window (WRW) for the current interval I.
    for each request R from ULC to the block B in LLC during I cycles **do**
      if $R == ReadHit$ **then**
        $NormOpr(R, B)$
      else if $R == WriteHit$ **then**
        if $B \in W_i$ **then**
          $WriteRedirect(R, B, \text{WRW})$
        else
          $NormOpr(R, B)$
        end if
      else
        $ProcessCacheMiss(R, \text{WRW})$ ▷ **cache miss**
      end if
    end for
  end for
until the end of the execution

Write Restricted Window Selection for SWWR

function WinSelect(i, m)
  $i = (i + 1)\%m$
  return i
end function

Functions used by the Algorithms

function NormOpr(R, B)
  Request R is served normally from block B as the conventional cache.
end function
function WriteRedirect(R, B, WRW)
  Write the block B to other location L in the same cache set. Note that the location L does not belong to currently selected WRW.
  return L
end function
function processCacheMiss(R, WRW)
  Forward the Request R to main memory to fetch the block. Keep the newly arrived block in a location other than WRW location.
end function

3.5.3 Working Example

Fig. 5 depicts the working methodology of the SWWR. In the figure, 16-way set-associative LLC is partitioned into 4 ($m = 4$) equal sized windows (W_0, W_1, W_2 and W_3). Each window of the LLC contains four ways and, the write restricted window is in W_0, i.e., way-0 to way-3. In order to explain the methodology, we used the arrows to show the request and response from the ULC and LLC. For the read request (shown by the arrow 1) to the way-0 of LLC, the response is served normally by the LLC (as indicated by arrow 2). On a write

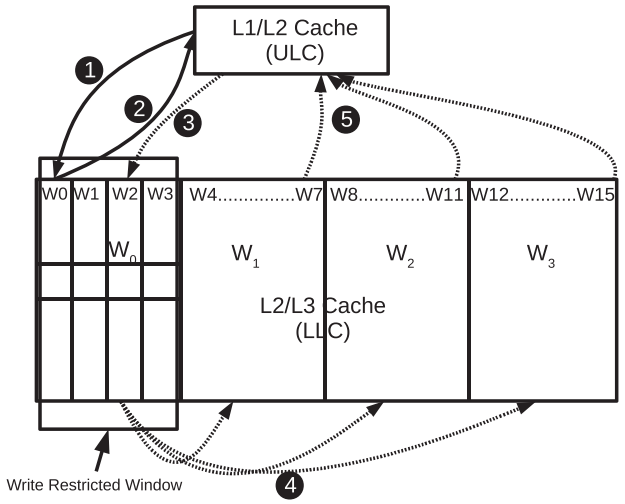

Fig. 5. Working example of proposed SWWR wear leveling policy.

request from ULC cache to the block belongs to W_0 of the LLC (arrow 3), the request is redirected to the one of the available way(s) of W_1, W_2 and W_3 (arrow 4). If all the ways are occupied, the LRU victim among these ways are selected, and the write-back operation is scheduled with the redirection of a write request. Once the write operation is performed, the write-back acknowledgment is sent to the ULC (arrow 5), and the requested block is invalidated from the W_0.

3.6 Limitation of SWWR

The limitation of the SWWR is the lack of consideration of write intensity of other windows in the window selection process. Because of write variation generated by the applications, the write intensity of the windows changes over the period. In other words, during execution, the lightly written window becomes heavily written, and vice-versa. Fig. 6 presents the write counts of different blocks inside the cache sets after applying SWWR. By considering these heavily written windows as a write-restricted during the execution, we can further improve the relative lifetime and reduce the coefficient of intra-set write variation (observed in Fig. 6 for SWWR). Table 2 shows the percentage availability of heavily written windows (H-win) during the window selection process in SWWR. From the table, on an average 13.51 percent times, a heavily written window other than the selected write restricted window is available in the cache. This motivates us to identify such windows and further improve the lifetime.

4 Proposed Wear Leveling Techniques

This section demonstrates our two new contributions: dynamic window and dynamic way aware write restriction

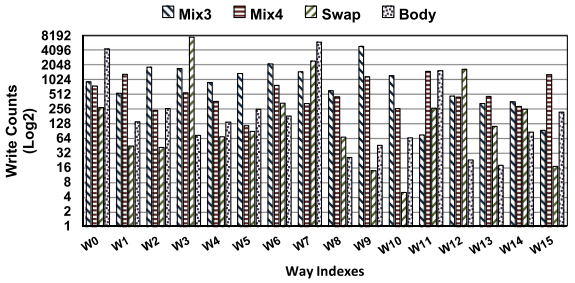

Fig. 6. Write counts for four different workloads in the SWWR.

that is based on SWWR. At the end of this section, we give a summary flow chart for all in Fig. 10.

Algorithm 2. Dynamic Window Write Restriction (DWWR)

1: ULC : Upper Level Cache i.e., L1/L2.
2: LLC : Last Level Cache i.e., L2/L3.
3: I : Predefined interval.
4: m : Number of logical partitions or windows.
5: W_i : ith logical partition or window that is treated as read-only (or write restricted) in the current interval. $0 \leq i < m$
6: C_i : Counter corresponding to jth window that records the number of write accesses from ULC to that window. $0 \leq < m$.
7: Run application for I cycles treating the whole cache as normally available cache.
8: After I cycles treat one window at a time as read-only or write restricted.
9: **repeat**
10: **for** every interval I **do**
11: $i = WinSelect(C_i, m)$
12: Let W_i is the selected Write Restricted Window (WRW) for current interval I.
13: **for** each request R from ULC to the block B in LLC during I cycles **do**
14: **if** $R == ReadHit$ **then**
15: $NormOpr(R, B)$
16: **else if** $R == WriteHit$ **then**
17: **if** $B \in W_i$ **then**
18: $L = WriteRedirect(R, B, \text{WRW})$
19: The corresponding counter of the window that contains the location L is incremented.
20: **else**
21: $NormOpr(R, B)$
22: Increment the counter C_j of the window where the block B is present.
23: **end if**
24: **else**
25: $ProcessCacheMiss(R, \text{WRW})$ ▷ **cache miss**
26: **end if**
27: **end for**
28: **end for**
29: **until** the end of the execution

Write Restricted Window Selection for DWWR

30: **function** WinSelect(C_i, m)
31: $i = max(C_i), \quad 0 \leq i < m$
32: Reset the counter C_i to zero
33: **return** i
34: **end function**

4.1 Dynamic Window Write Restriction (DWWR)

4.1.1 Main Idea

The main idea of DWWR is to partition the cache into m equal sized windows and use different window during the execution for a predefined interval. In SWWR, the selection of write restricted window was in a round robin fashion. Whereas, in DWWR, the selection of window is based on a counter associated with each window. The counter is used to track the number of writes during the past interval to the window (i.e., from ULC to LLC). In each interval, the window with maximum writes is chosen and is treated as write restricted (or read-only) window. At the end of the interval, the next window of the cache is selected based upon the counter values, and the process continues until the end of execution. Note that, in order to remove the possibility of selecting the same window in the consecutive intervals, we reset the counter for the selected write restricted window.

TABLE 2
Percentage Times Heavily Written Window Available in Cache

Workloads		H-Win		
Type	Bench	1	2	Avg.
PARSEC	Body	14.4%	3.01%	17.44%
	Cann	2.54%	-	2.54%
	Ded	6.03%	1.41%	7.44%
	Swap	17.2%	7.27%	24.5%
	X264	10.04%	-	10.04%
SPEC	Mix1	11.17%	3.71%	14.88%
	Mix2	9.34%	4.11%	13.45%
	Mix3	9.88%	4.26%	14.14%
	Mix4	12.4%	4.74%	17.2%
MEAN		**10.35%**	**3.17%**	**13.51%**

4.1.2 Algorithm

The working approach of DWWR is elaborated through Algorithm 2. In the algorithm, the partition or window of the cache is represented by the variable W_i. Similarly, the counter associated with each window is represented by the variable C_i (lines 5 and 6). Note that the range of i is from 0 to $m-1$.

For the initial I cycles of the application execution, the cache is employed as an ordinarily available cache (line 7). Once the application crosses the I cycles, one of the windows of the cache is treated as read-only or write restricted window. The selection of the window is based upon the counter associated with each window of the cache (lines 10 and 11). Thus, the window with most write accesses in the previous intervals is selected as a write-restricted in the next interval (line 31). This helps to restrict the heavily written window to get the further writes in the next interval. Once the window is selected, the corresponding counter is reset to zero (line 32). At the end of the interval, the next window with maximum writes is selected by the method. The process continues until the end of execution (line 29).

Same as in SWWR, the tag lookup operation is performed on the last level cache, LLC for each request coming from upper-level cache, ULC to LLC (line 13). Depending upon the outcome of the lookup for requested Block B, different operations are performed:

- *Read Hit:* The read operation is performed in the same way as given in Section 3.5.2 (lines 14 and 15 of Algorithm 2).
- *Write Hit (Write-back or PUTX) and block B in W_i:*The write request for the block B in W_i is redirected to the invalid way(s) of the same cache set in the other windows. If there is no invalid line available in the same cache set, the victim v is evicted from the location L. Note that location L is different from the write restricted window location. Afterward, the write request R from a ULC is redirected to the location L (let say in window W_j). The counter C_j corresponding to the window W_j is incremented and the block B is invalidated from W_i (lines 17 to 19).
- *Write Hit (Write-back or PUTX) and block B not in W_i:* The write request R is performed normally on the block B. The corresponding counter C_j of window

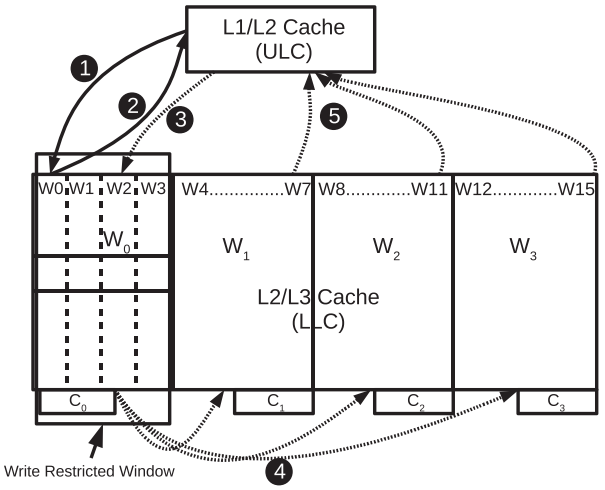

Fig. 7. Working example of proposed DWWR wear leveling policy.

W_j (in which the write operation is performed) is incremented (lines 20 to 22).

- *Cache Miss:* In the case of LLC miss, the request R from a ULC is forwarded to the next level of memory (main memory in our case). When the requested block has arrived at the LLC, it will be placed in the location apart from the location belonging to W_i (lines 24 to 26).

4.1.3 Working Example

Fig. 7 presents the working example of DWWR. As shown in the figure, the counters C_0 to C_3 are associated with each window W_0 to W_3. Let W_0 be the write restricted window having maximum write count i.e., $C_0 = max(C_i),\ \forall i$. In the first case, a read request from the ULC to LLC (arrow 1) is served normally (arrow 2). In the second case, a write request from the ULC to the way-2 of LLC (arrow 3) is redirected to another way in the same cache set (arrow 4). In case, if there is no invalid line present in the way, one of the blocks is invalidated from other windows (W_1, W_2 and W_3 in our example) in the same cache set. Once the write operation is performed, the block (present in the way-2) is invalidated from the write restricted window W_0 and the corresponding counter of the window in which the write redirection happens is incremented.

4.1.4 Limitation of DWWR

The only limitation of DWWR is the lack of consideration for heavily written ways available in the other lightly written windows. In other words, a particular window may have a lower write count compared to others but can contain some ways that are written more number of times compared to those in windows with overall higher write count. Fig. 8 shows the write counts of blocks within the cache set after incorporating DWWR. Table 3 presents the availability percentage of heavily written ways (H-ways) in the windows apart from the selected window. From Table 3, we can conclude that on an average 25.2 percent times H-ways are present in windows other than the selected window of the cache. This motivates us to identify such H-ways in the cache and further reduce the intra-set write variation and improve the lifetime over SWWR and DWWR.

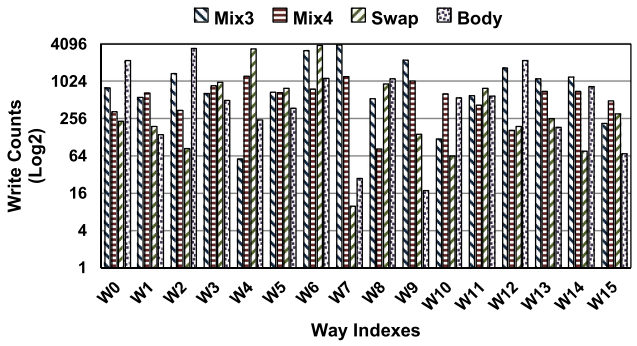

Fig. 8. Write counts for different workloads in DWWR.

TABLE 3
Percentage Times Heavily Written Ways (Apart from Write Restricted Ways) Present in the Cache

Workloads		H-Ways								
Type	Bench	1	2	3	4	5	6	7	8	Avg.
PARSEC	Body	24.9%	7.4%	4.9%	4.1%	0.76%	1.16%	1.22%	0.24%	44.7%
	Cann	11.8%	8.14%	4.9%	2.35%	0.78%	0.33%	0.11%	-	28.5%
	Ded	4.85%	3.41%	2.35%	1.49%	-	-	-	-	12.1%
	Swap	23.2%	17.4%	11.6%	7.34%	2.14%	1.4%	0.61%	0.54%	64.3%
	X264	6.07%	2.82%	1.83%	1.03%	-	-	-	-	12.2%
SPEC	Mix1	2.78%	2.79%	1.82%	1.10%	0.73%	0.34%	0.13%	-	9.7%
	Mix2	4.14%	3.69%	3.03%	2.47%	1.46%	0.96%	0.49%	0.29%	16.57%
	Mix3	5.74%	4.07%	2.87%	1.88%	0.47%	0.38%	0.19%	0.12%	15.7%
	Mix4	7.26%	5.91%	4.42%	3.13%	1.16%	0.74%	0.39%	0.18%	23.2%
MEAN		**10.1%**	**6.18%**	**4.21%**	**2.76%**	**1.07%**	**0.76%**	**0.45%**	**0.23%**	**25.2%**

4.2 Dynamic Way Aware Write Restriction (DWAWR)

4.2.1 Main Idea

The main idea of DWAWR is to select the n heavily written ways and designate them as write-restricted for a specific predefined interval I. The selection of ways in DWAWR is based upon examining the way counters (Z) associated with each cache way. The way counter is used to track the number of write accesses in the past intervals from the ULC to that way in LLC. In each interval, the n ways with maximum writes are selected and treated as read-only (or write restricted). At the end of the interval, the next n ways are chosen for the next interval, and the process continues until the end of execution. Note that to remove the possibility of choosing the same way (s) in the successive intervals, the counters (Z) associated with the selected ways of the previous interval are reset to zero.

4.2.2 Algorithm

Algorithm 3 presents the working approach of DWAWR. In the algorithm, the use of predefined interval is the same as in the SWWR and DWWR (line 3). The parameter n acts here as the number of ways that are treated as write-restricted (line 4). $waysList$ is the list of integers of size n. It contains the list of ways in the cache that is treated as write-restricted in the current interval (line 5). The way counter of the cache is represented by Z_j (line 6). Note that the range of j is from 0 to $A-1$ (where A is the cache associativity).

Same as in the previous approaches, for the initial I cycles of application execution, the cache is employed as an ordinarily available cache (line 7). Once the application crosses the I cycles, n ways are selected and treated as write-restricted (or read-only) for the next interval I (line 8). In particular, the way counters having maximum value in the previous interval is selected in the next interval (lines 11, 29 to 31). Once the ways are selected for the write

restriction, the counter corresponding to the ways is reset to zero (line 32). This restricts the chances of heavily written ways in the previous intervals to get further more writes in the next interval. At the end of the interval, the next n ways are selected, and the process continues until the end of execution (line 27).

Algorithm 3. Dynamic Way Aware Write Restriction (DWAWR)

1: ULC : Upper Level Cache i.e., L1/L2.
2: LLC : Last Level Cache i.e., L2/L3.
3: I : Predefined interval.
4: n : Number of ways that is treated as read only (or write restricted.
5: List $< integer > waysList$: List of Write Restricted Ways (WRW) in the current interval. Size of list is n.
6: Z_j : Way counter with respect to jth way that records the number of write accesses from L1/L2 cache to the way. $0 \leq j < cache\_assoc$.
7: Run application for I cycles treating the whole cache as normally available cache.
8: After I cycles treat n ways as read-only or write restricted.
9: **repeat**
10: **for** every interval I **do**
11: $WinSelect(Z_j, n, waysList)$
12: **for** each request R from L1/L2 cache to the block B in L2/L3 cache during I cycles **do**
13: **if** $R == ReadHit$ **then**
14: $NormOpr(R, B)$
15: **else if** $R == WriteHit$ **then**
16: **if** $B \in waysList$ **then**
17: $L = WriteRedirect(R, B, \text{WRW})$
18: The corresponding counter Z_L of the location L is incremented.
19: **else**
20: $NormOpr(R, B)$
21: **end if**
22: **else**
23: $ProcessCacheMiss(R, \text{WRW})$ ▷ **cache miss**
24: **end if**
25: **end for**
26: **end for**
27: **until** the end of the execution

Write Restricted Way Selection for DWAWR

28: **function** WinSelect(Z_j, n, $waysList$)
29: **for** $k \leftarrow 0$ to n **do**
30: Let Z_j be the maximum counter in the cache. $0 \leq j < cache\_assoc$
31: $waysList.add(j), \quad 0 \leq j < cache\_assoc$
32: $Z_j = 0$
33: **end for**
34: **end function**

For each request coming from ULC to LLC, the tag lookup operation is performed on the cache. Depending upon the result of the lookup operation, the read (lines 13 and 14), write (lines 15 to 21) and forward to main memory operation (lines 22 to 24) is performed in the LLC. The handling of these operations is already described in Section 4.1.2. The only difference is that in case of write operation, the corresponding counter of the way in which the write is redirected or in which the write operation is performed is incremented.

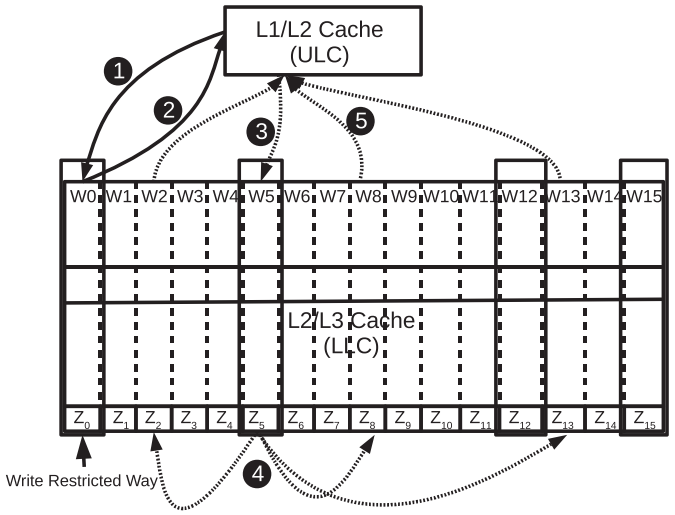

Fig. 9. Working example of proposed DWAWR wear leveling policy.

4.2.3 Working Example

Fig. 9 depicts the working example of the proposed DWAWR approach. In the example, the way counters Z_0 to Z_{15} are associated with each way W_0 to W_{15} in the cache. Let way-0, way-5, way-12, and way-15 are selected as write-restricted ways in the current interval. In the first case, the read request from the ULC to LLC (arrow 1) is served normally (arrow 2) by the LLC. In the second case, the write request from ULC coming to way-5 of LLC (arrow 3) is redirected (arrow 4) to one of the inferior ways in the cache (let say way-2, way-8, and way-13) depending upon the availability of invalid and victim location in the cache. Once the write operation is performed, the write-back acknowledgment is sent back to the ULC (arrow-5). The working flow diagram of the presented approaches: SWWR, DWWR and DWAWR is summarized in Fig. 10.

5 Experimental Methodology

5.1 Simulator Setup

We evaluate our proposed approaches on a full system simulator GEM-5 [29]. Table 4 shows the system parameters used in the simulation. The memory system is simulated with the help of a memory module inside GEM-5 called Ruby. The coherence is managed with the help of MESI CMP based cache controller. We conducted our experiments on a dual and quad-core system with different levels of cache made up of different memory technologies and with different configurations of cache and parameters. Table 5.1 shows the timing and energy parameter for these configurations obtained by using NVSIM [30] at 32 nm technology node. We employed STT-RAM as an LLC in two level cache hierarchy and ReRAM as an LLC in a three level cache hierarchy. The reasons behind to use the STT-RAM as the LLC in the 2-level cache hierarchy is because the LLC at two-level cache hierarchy experience more number of writes as compared to LLC at 3-level cache hierarchy and with large write endurance, the STT-RAM is best suited for such situations compared to ReRAM.

We compared our proposed approaches against three existing approaches: Probabilistic Set Line Flush (PoLF) [8], Write-Back Aware Intraset Displacement (WAD) [12] and EqualChance [9] and, the baseline STT-RAM/ReRAM that

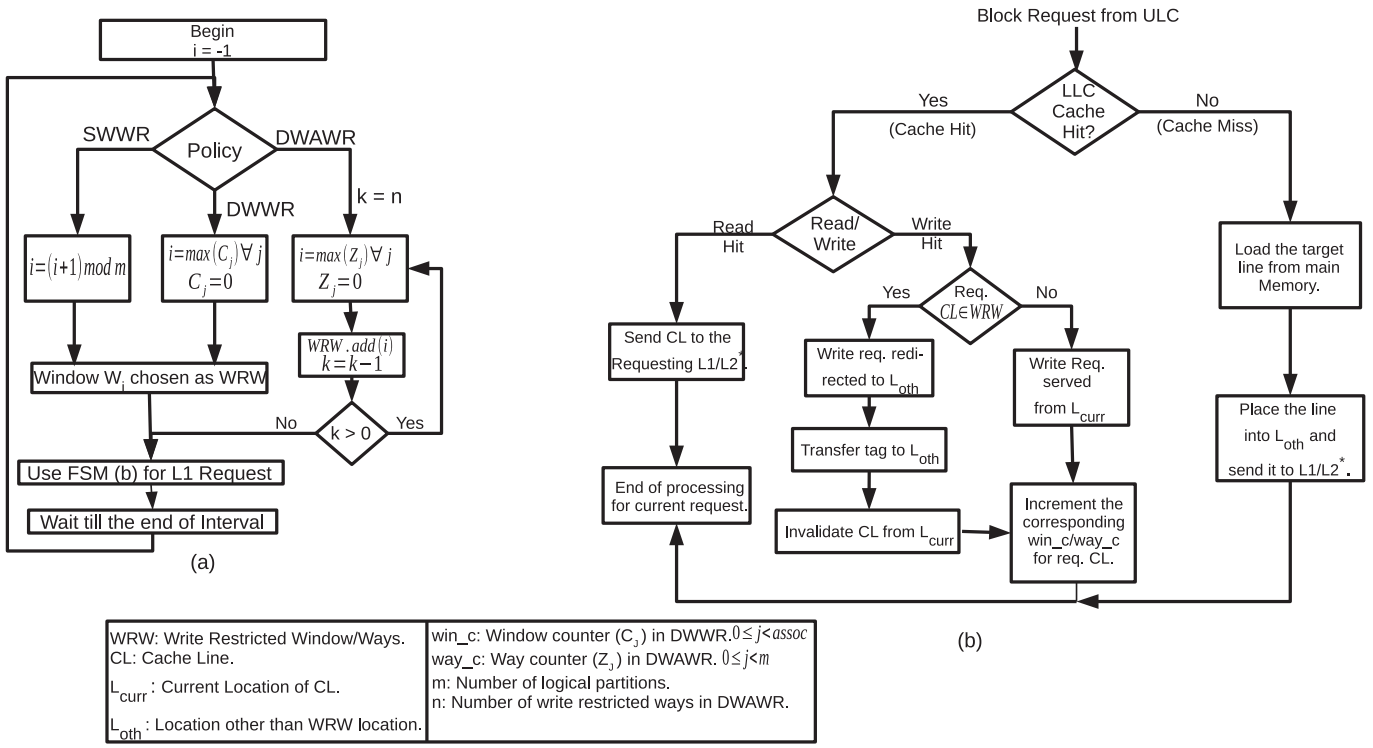

Fig. 10. (a) WRW construction flow chart (b)The working flow chart summarizing the proposed schemes: SWWR, DWWR and DWAWR.

uses LRU as a replacement policy with no wear leveling policy associated. In PoLF, the value of Flush Threshold or FT for skipping the write operation is set to 16. With WAD, we set the value of intraset saturation counter (RSC) to 7 and use the clean-LRU block intraset displacement approach. On the other hand, in EqualChance, a 4-bit write counter along with a flag bit are associated with each set. The value of Υ to trigger the transfer/swap operation within the cache set is set to 16. In addition to the write counter, a 64B swap buffer is used for the swap operation of the block within the cache set. Note that we model only single swap buffer in our setup as the bank contention model is used in our approaches. The C-shifting and I-shifting in Equalchance take extra cycles for transfer/swap operation. We model these extra cycle in our simulator setup as same as in [9].

In SWWR, DWWR, and DWAWR, during the write redirection, the tag needs to be transferred from the current write restricted window/way to the new location within the cache set. The transfer of tag requires an additional 42-bit buffer and additional latency of 3 cycles (one cycle to transfer the tag in tag buffer, one cycle to writing the tag in tag buffer and, one cycle to transfer the tag to the new location within the cache set). In DWWR, the size of the counter C_i associated with each window is set to 13 bit. Whereas, in DWAWR, the size of the way counter Z_j is set to 11 bit. Later, we analyze the appropriate sizes of the counter values in the Section 7.5 with different cache configuration and parameters. Note that the selection of window/way in DWWR and DWAWR is not on the critical path. This is because, in the proposed approaches, the windows/way selection happens at the end of the interval before the arrival of a new request.

5.2 Workloads

We verified our proposed approaches on both multi-threaded: PARSEC [31] and multi-programmed: SPEC CPU 2006 [32] benchmark suites. Five benchmarks with *medium* input set are used from the PARSEC. Twenty benchmarks with *ref* input are used from SPEC CPU 2006. Table 6 lists the name of the benchmarks used for the evaluation and the mix of application for multi-programmed workload. We run each multi-programmed workload for 1 billion instructions after warming up for 250 million instructions.

TABLE 4
System Parameters

Components	Parameters
Processor	2Ghz, Dual Core and Quad Core, X86
L1 Cache	Private, 32 KB SRAM Split I/D caches, 4-way set-associative cache, 64B block, 1-cycle latency, LRU, write-back policy
Two-Level Cache Memory Parameters	
L2 Cache	Shared, STT-RAM, 64B block, LRU, write-back policy
Three-Level Cache Memory Parameters	
L2 Cache	Private, 512 KB SRAM, 8-way set-associative cache, 64B block, 5-cycle latency, LRU, write-back policy
L3 Cache	Shared, ReRAM, 16-way set-associative cache, 64B block, LRU, write-back policy
Main Memory	2GB, 160 cycle Latency
Protocol	MESI CMP Directory

TABLE 5
Timing and Energy Parameters for STT-RAM/ReRAM LLC

LLC Configuration	Leakage Power (mW)	Hit Energy (nJ)	Miss Energy (nJ)	Write Energy (nJ)	Hit Latency (ns)	Miss Latency (ns)	Write Latency (ns)
Two Level STT-RAM L2 Cache System							
32MB, 16way	17.187	0.544	0.093	4.261	259.536	16.8	747.147
16MB, 32way	15.668	0.457	0.186	6.548	84.074	17.475	271.035
16MB, 16way	15.674	0.367	0.096	4.322	78.453	11.854	271.035
16MB, 8way	15.659	0.317	0.047	3.244	78.283	11.684	271.035
8MB, 32way	8.116	0.366	0.185	6.454	74.792	8.259	270.981
8MB, 16way	8.030	0.273	0.093	4.387	78.497	11.964	270.981
8MB, 8way	7.983	0.227	0.047	3.221	74.454	7.921	270.981
4MB, 16way	7.960	0.217	0.093	4.228	23.876	5.575	126.585
Three Level ReRAM L3 Cache System							
8MB, 16way	60.196	0.65	0.093	1.62	54.71	48.66	67.71
16MB, 16way	132.32	1.128	0.122	2.078	54.92	48.702	67.736

TABLE 6
Benchmarks Used for Evaluation

Benchmark Suite	Benchmarks			
PARSEC v2.1	Bodytrack (Body), Canneal (Cann), Dedup, Swaptions (Swap). X264			
SPEC CPU2006	Dual Core Workloads		Quad Core Workloads	
	Mix1	bwaves, gamess	Mix1	zeusmp, bwaves, leslie3d, cactusADM
	Mix2	lbm, zeusmp		
	Mix3	perlbench, bzip2	Mix2	perlbench, omnetpp, gcc, libquantum
	Mix4	gcc, bzip2		
	Mix5	omnetpp, milc	Mix3	gobmk, tonto, sjeng, namd
	Mix6	dealII, namd		
	Mix7	h264ref, gobmk	Mix4	calculix, astar, dealII, h264ref
	Mix8	sjeng, calculix		

6 Results and Analysis

6.1 Two Level Cache Analysis: L2-STT-RAM

We evaluate our proposed approaches: SWWR, DWWR, and DWAWR on the dual and quad-core system. For a dual-core system, an 8 MB 16-way set-associative STT-RAM L2 cache is used and, for the quad-core system, 16 MB 16-way set-associative STT-RAM L2 cache is used. In our evaluation, we set the values of I (predefined interval) to 2 million cycles in case of dual-core and 1 million cycles in case of quad-core and, m (SWWR/DWWR)/n (DWAWR) to 4. The rationale behind the different interval values is that the number of accesses is doubled in the case of a quad-core system compared to the dual-core. Later in the Section 7, we analyze the effects of changing these values. Due to space limitations, graphs are only presented for quad-core.

We present our results on the following metrics: coefficient of Intra-set write variation ($IntraV$) calculated with the help of Equation (1), coefficient of Inter-set write variation ($InterV$) calculated by using Equation (2), relative lifetime improvement, energy overhead, speedup and number of invalidations/flushes.

6.1.1 Coefficient of Intra Set Write Variation

Fig. 11 presents the coefficient of intra-set write variation for quad-core system. Table 7 lists the percentage reduction in coefficient of intra-set write variation by the proposed approaches against the existing techniques and, the baseline STT-RAM for both multi-threaded and multi-programmed applications. Note that the negative values (row 14) in the table implies the increase of write variation.

- *STT:* Compared to baseline STT, the reduction in intra-set write variation (row 1-3 and 16-18) is mainly due to the more effective uniform write distribution by the proposed approaches.
- *PoLF:* The improvement in the write variation (row 4-6 and 19-21) over PoLF is because the PoLF invalidates the data block randomly or probabilistically without concerning its write behavior. On the other hand, our proposed technique SWWR redirects the heavily written block repeatedly in the other window of the same cache set over the period. Whereas, the DWWR and DWAWR redirect the writes in the same cache set in a pseudo-random fashion by taking into account the window/way write counts from the previous interval.

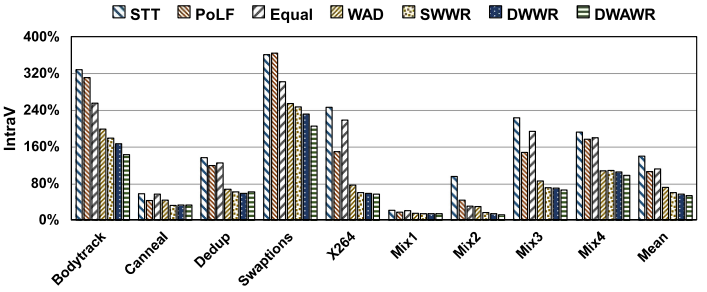

Fig. 11. Intra set write variation for Quad core (lesser is better).

- *EqualChance:* The improvement by the proposed approaches (row 7-9 and 19-21) concerning EqualChance is because in the existing technique the transfer/swap operation takes place only when the clean/invalid data entry is present in the cache set (which is very limited). On the other hand, in our techniques, the write redirection takes place to the Least Recently used entry (always available) in the same cache set.
- *WAD:* With respect to WAD, the improvement (row 10-12 and 25-27) is mainly due to the migration of the block in a pseudo probabilistic fashion by the existing technique. Note that the WAD probabilistically assumes that upon the write saturation of counter, the next block to be written in the set is hot.

In our analysis, we also observed that some of the rows (row 14 and 29) of the table indicate the limited reduction in intra-set write variation in SPEC compared to PARSEC. This is because, in SPEC, a limited amount of data is shared between the multiple cores [31], [33] compared to multi-threaded PARSEC applications. Also with the limited number of cores, the percentage of

TABLE 7
Percentage Reduction in Coefficient of Intra Set Write Variation (More Is Better)

Core	Reference Policy	Suites	SWWR	DWWR	DWAWR	Row
Dual	STT	PARSEC	94.2%	95.4%	101.6%	1
		SPEC	41%	40.8%	42%	2
		Total	56.7%	56.9%	59.1%	3
	PoLF	PARSEC	43.3%	44.6%	50.8%	4
		SPEC	19%	18.8%	20%	5
		Total	26.2%	26.4%	28.6%	6
	Equal	PARSEC	38.7%	40%	46.2%	7
		SPEC	31.2%	31%	32.2%	8
		Total	35.4%	35.6%	37.9%	9
	WAD	PARSEC	11.4%	12.7%	18.9%	10
		SPEC	12.1%	11.9%	13%	11
		Total	13.1%	13.3%	15.5%	12
	SWWR	PARSEC	-	1.3%	7.5%	13
		SPEC	-	-0.20%	1%	14
		Total	-	0.2%	2.4%	15
Quad	STT	PARSEC	99.2%	102.7%	106.7%	16
		SPEC	60.2%	62.4%	65.2%	17
		Total	80%	82.9%	86.5%	18
	PoLF	PARSEC	65.8%	69.3%	73.3%	19
		SPEC	29.4%	31.6%	34.4%	20
		Total	46%	48.9%	52.5%	21
	Equal	PARSEC	76.2%	79.7%	83.7%	22
		SPEC	31.9%	34.1%	36.9%	23
		Total	51.7%	54.6%	58.2%	24
	WAD	PARSEC	14.6%	18.2%	22.2%	25
		SPEC	8.4%	10.6%	13.4%	26
		Total	11.4%	14.3%	17.9%	27
	SWWR	PARSEC	-	3.6%	7.6%	28
		SPEC	-	2.2%	5%	29
		Total	-	2.9%	6.5%	30

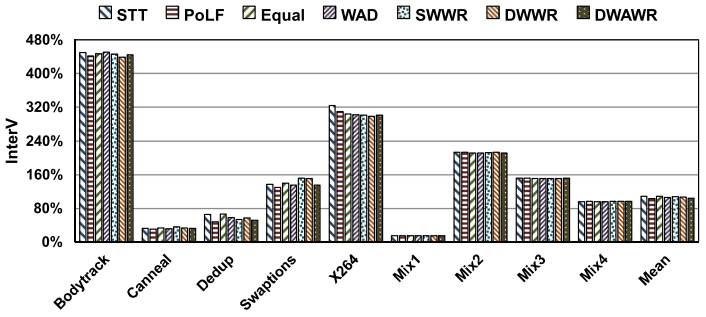

Fig. 12. Inter set write variation for Quad core (lesser is better).

TABLE 8
Percentage Reduction in Coefficient of Inter Set Write Variation (More Is Better)

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	STT	1.12%	3.7%	5.3%	1
	PoLF	0.3%	2.85%	4.45%	2
	Equal	-0.3%	2.3%	3.9%	3
	WAD	-3.1%	-0.5%	1.1%	4
Quad	STT	0.75%	1.3%	4.2%	5
	PoLF	-4.8%	-4.3%	-1.4%	6
	Equal	0.5%	1.1%	4%	7
	WAD	-1.9%	-1.4%	1.5%	8

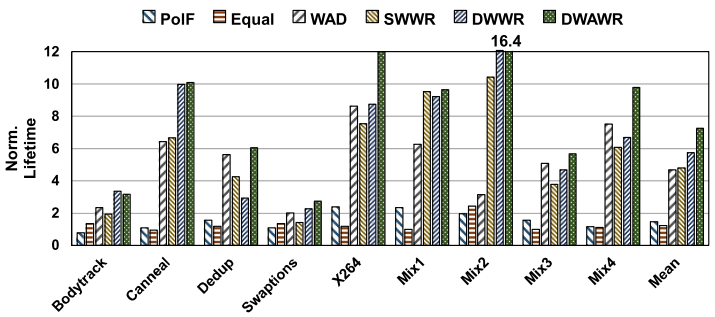

Fig. 13. Normalized lifetime with respect to baseline STT-RAM for Quad core (more is better).

shared data is less (row 15 and 30). All these factors limit the write-backs from the other cores in the write-restricted window/ways.

6.1.2 Coefficient of Inter-Set Write Variation

Fig. 12 and Table 8 shows the coefficient of inter-set write variation for quad-core and the percentage reduction in the coefficient values against the existing techniques. From the results, we observe that the inter-set write variation does not change as our proposed techniques do not redirect or move the data from one set to another set.

6.1.3 Relative Lifetime

Fig. 13 shows the improvement in the lifetime by the proposed approaches against baseline STT-RAM for quad-core. Table 9 lists these improvement values against the existing techniques and the baseline. The reason behind the large lifetime improvement (row 1 to 12 and 16 to 27) is due to great reduction in coefficient of intra-set write variation (as presented in Table 7). However, we observe that with the higher core count, the improvement is more over the previous work: SWWR by the proposed approaches: DWWR and DWAWR (row 13 to 15 and 28 to 30). This is because, with the higher core count, data sharing between the multiple

TABLE 9
Relative Lifetime Improvement (in Times) (More Is Better)

Core	Reference Policy	Suites	SWWR	DWWR	DWAWR	Row
Dual	STT	PARSEC	5.1	6.5	7.44	1
		SPEC	5.11	5.29	6.38	2
		Total	5.1	5.72	6.77	3
	PoLF	PARSEC	2.31	2.94	3.38	4
		SPEC	2.44	2.52	3.04	5
		Total	2.4	2.68	3.17	6
	Equal	PARSEC	2.62	3.33	3.83	7
		SPEC	4.50	4.66	5.62	8
		Total	3.65	4.1	4.85	9
	WAD	PARSEC	0.87	1.11	1.28	10
		SPEC	1.34	1.39	1.68	11
		Total	1.14	1.28	1.51	12
	SWWR	PARSEC	-	1.27	1.46	13
		SPEC	-	1.03	1.24	14
		Total	-	1.12	1.32	15
Quad	STT	PARSEC	3.6	4.56	5.78	16
		SPEC	6.92	7.7	9.7	17
		Total	4.8	5.75	7.27	18
	PoLF	PARSEC	2.77	3.52	4.47	19
		SPEC	4.04	4.5	5.65	20
		Total	3.3	3.92	4.96	21
	Equal	PARSEC	2.97	3.78	4.80	22
		SPEC	5.4	6	7.53	23
		Total	3.87	4.63	5.86	24
	WAD	PARSEC	0.83	1.05	1.34	25
		SPEC	1.31	1.46	1.84	26
		Total	1.02	1.22	1.54	27
	SWWR	PARSEC	-	1.27	1.61	28
		SPEC	-	1.11	1.4	29
		Total	-	1.2	1.51	30

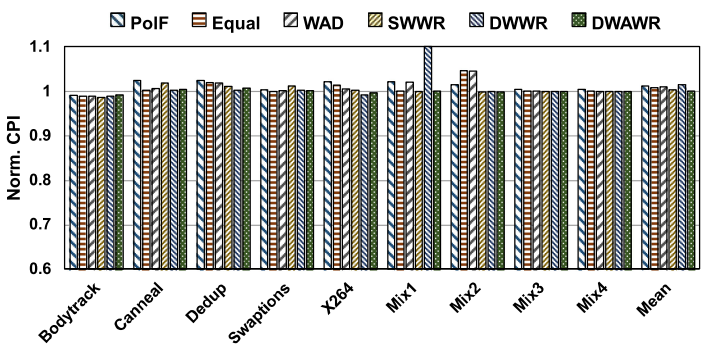

Fig. 14. Normalized speedup with respect to baseline STT-RAM for Quad core (lesser is better).

cores increases and in consequence the increase number of write-backs to be controlled giving more opportunity of applying our dynamic policies.

6.1.4 Performance

Fig. 14 shows the normalized CPI (with respect to STT) for quad-core. Our proposals maintain the same performance with PoLF, EqualChance, and WAD. The reason behind having the same performance despite lesser invalidations can be accounted for the extra cycles taken by the swap operations during the write redirection process.

6.1.5 Overheads

1) *Energy Overhead:* As the proposed techniques redirect the writes by invalidating the data block, they need slightly more energy (row 1 and 5) compared to the baseline. Fig. 15 presents the energy overhead by the proposed techniques against the existing techniques and the baseline. Table 10 lists these values against

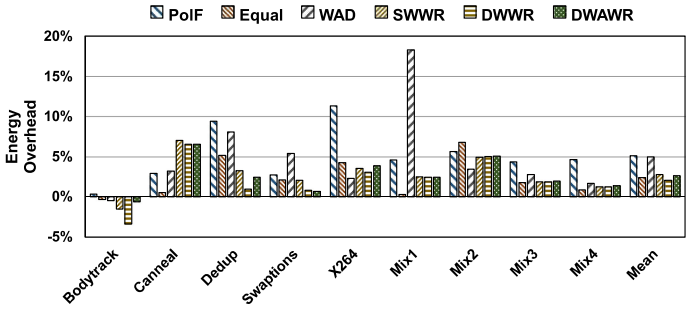

Fig. 15. Energy overhead with respect to baseline STT-RAM for Quad core (lesser is better).

TABLE 10
Energy Overhead (in Percentage) (Lesser Is Better)

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	STT	2.36%	2.05%	1.82%	1
	PoLF	-2.46%	-2.61%	-2.85%	2
	Equal	-1.61%	-1.91%	-2.13%	3
	WAD	-0.61%	-0.91%	-1.13%	4
Quad	STT	2.78%	2.1%	2.65%	5
	PoLF	-2.16%	-2.83%	-2.28%	6
	Equal	-1.02%	-1.6%	-1.03%	7
	WAD	-1.88%	-2.55%	-1.99%	8

the baseline and the existing techniques. Note that the negative value in the table signifies the energy savings. The reasons behind these energy improvements against the existing techniques are presented below:

- *PoLF:* The energy improvement by the proposed approaches (row 2 and 6) with respect to PoLF is due to two reasons: First, the invalidation of the MRU block by the PoLF increases the allocations in the cache. Second, the number of invalidation by the proposed approaches is less as compared to PoLF.
- *EqualChance:* In case of EqualChance, the energy improvement (row 3 and 7) is due to transfer/swap operation performed by the existing technique within the cache set. These kinds of operations incur extra writes in the cache which in turn increases the energy consumption. However, in quad-core, the energy improvement is limited because of the less availability of clean entries (due to increase in the residency of a dirty block) which in turn reduces the write operations.
- *WAD:* The energy improvement (row 1 and 8) by the proposed approaches over WAD is due to migration of the block in either clean LRU or LRU position by WAD which incurs extra write operations in the cache that results into extra energy.

Note that we have also considered the energy overhead to transfer the tag from the original location to the redirected location in our calculations.

2) *Invalidation/Flushes:* Fig. 16 present the normalized invalidations by the WAD and proposed approaches against PoLF. Table 11 lists the percentage reduction in invalidation by the proposed techniques: DWWR and DWAWR against the existing techniques: PoLF, WAD, and SWWR. Note that the negative value in the table implies the increase in invalidation (row 2, 3, 5 and 6). The improvement in the invalidation with

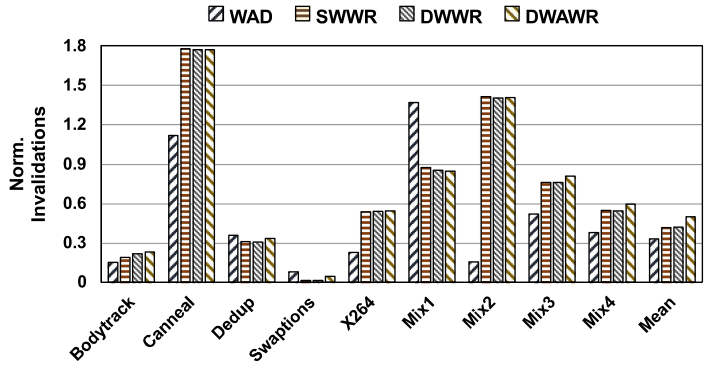

Fig. 16. Normalized Invalidation against PoLF for Quad core (lesser is better).

TABLE 11
Percentage Reduction in Invalidations (More Is Better)

Core	Reference Policy	SWWR	DWWR	DWAWR	Row
Dual	PoLF	57.4%	57.8%	52.8%	1
	WAD	0.8%	1.74%	-9.97%	2
	SWWR	-	-0.96%	-11.93%	3
Quad	PoLF	57.7%	58.5%	50%	4
	WAD	-28.4%	-25.9%	-51.8%	5
	SWWR	-	-2%	-20.6%	6

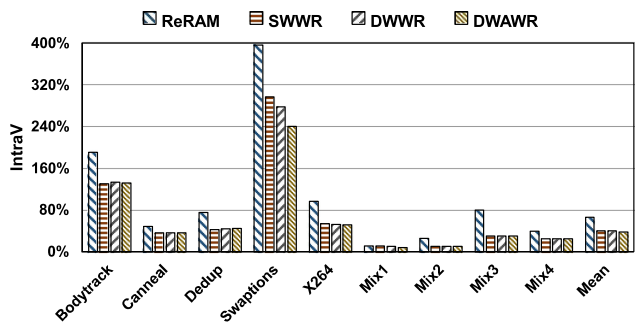

Fig. 17. Intra set write variation for Quad core ReRAM L3 cache. (Lesser is better).

respect to PoLF is due to selective invalidation done by the proposed approaches. However, compared to SWWR, the invalidation is increased due to the increase in write-back operations in write restricted window/ways from L1 cache to L2 cache in the proposed approaches. On the other hand, with respect to WAD, the invalidation by the proposed approaches is increased (row 2 and 5) because, in WAD, the block is migrated to either invalid location or the clean LRU block position only when the RSC counter saturates.

6.2 Three Level Cache Analysis: L2-SRAM, L3-ReRAM

To evaluate our proposed techniques: SWWR, DWWR, and DWAWR in the three-level cache, we used 8 MB 16-way set-associative ReRAM based L3 cache for dual-core and 16 MB 16-way set-associative ReRAM based L3 cache for quad-core. Same as in two level, we set the parameters values to 2 million (1 million) cycles for I and m/n to 4 in case of dual(quad) core system (Due to space limitation we have not shown comparative analysis for three-level cache). Figs. 17 and 18 plot the intra-set write variation and lifetime for a quad-core system. Table 12 shows the brief results with respect to following metrics: IntraV, InterV and normalized lifetime (with

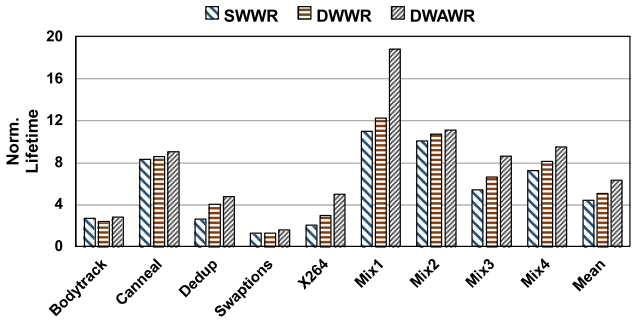

Fig. 18. Normalized lifetime with respect to baseline ReRAM for Quad core ReRAM L3 cache (more is better).

TABLE 12
Brief Results and Analysis for Three Level ReRAM Cache System

Core	Metric	ReRAM	SWWR	DWWR	DWAWR	Row
Dual	IntraV	53.9%	27.7%	26.8%	26.5%	1
	InterV	45.2%	44.1%	42.3%	42%	2
	Lifetime	1	4.96	6.76	7.39	3
Quad	IntraV	66.7%	40.8%	40.3%	38.4%	4
	InterV	59.8%	59.8%	56%	57.9%	5
	Lifetime	1	4.41	5.04	6.31	6

respect to ReRAM (presented in times)) for dual and quad-core ReRAM based L3 cache system. The limited improvement (row 1 and 4) between the proposed techniques in the intra-set write variation is due to the less write access in the L3 cache compared to an L2 cache. However, the simulation results confirm that the inferences drawn out from the two-level STT-RAM cache system are still applicable to L3 ReRAM cache system.

6.3 i2wap versus Write Restriction

Table 13 present the comparison analysis of InterV (percentage reduction), IntraV (percentage reduction), normalized lifetime (in times) and normalized EDP with respect to i2wap for dual and quad-core STT-RAM L2 cache system. Note that negative values in the table imply the increase in inter-set write variation. The respective improvements in the coefficient of intra-set write variation represent the effectiveness of the proposed approaches: SWWR, DWWR, and DWAWR. However, by integrating some of the inter-set wear leveling techniques with the proposed approaches, the lifetime and the inter-set write variation can further be improved with respect to i2wap. The gain in EDP by the proposed approaches compared to i2wap is mainly due to large random invalidations being done by the existing approach which in turn affect the system performance and increase the energy consumption.

7 Comparative Analysis for Parameters

In addition to the results presented in the previous sections, we conducted experiments with the different STT-RAM based L2 cache configurations (Due to space limitation we have not shown analysis for ReRAM based L3 cache) and, with different values for the parameters of the algorithms. Here, we show the effect on various metrics in comparison to the reference parameters. This analysis is helpful in choosing the optimal values in the proposed algorithms for different configurations of the cache.

TABLE 13
Results Comparison Analysis Between i2wap and Write Restriction

Core	Metric	SWWR	DWWR	DWAWR
Dual	IntraV	22.73%	22.85%	25.12%
	InterV	-6.11%	-3.54%	-1.94%
	Lifetime	1.96	2.20	2.60
	EDP	3.32%	3.6%	3.95%
Quad	IntraV	36.71%	39.65%	43.2%
	InterV	-9.43%	-8.92%	-6%
	Lifetime	2.60	3.12	3.94
	EDP	6.16%	5.72%	6.58%

TABLE 14
Comparative Analysis for Different Interval Values (I) LFT.= Lifetime, BASE = Baseline STT-RAM, WrRes = Write Restricted, EDP = Energy-Delay Product

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. (%)	Norm. EDP	Invalidation (k)	Row
Dual	SWWR	Ref.(I=2M)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		I=1M	5.36	90.94%	33.5%	3.7%	1.02	483k	2
		I=5M	4.38	90.94%	35.6%	0.06%	1.01	364k	3
	DWWR	Ref.(I=2M)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		I=1M	5.68	90.94%	33.9%	2.4%	1.03	483k	5
		I=5M	4.96	90.94%	34.8%	0.2%	1.02	382k	6
	DWAWR	Ref.(I=2M)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		I=1M	6.18	90.94%	31.8%	4%	1.03	555k	8
		I=5M	5.57	90.94%	33.1%	-1.2%	1.02	420k	9
Quad	SWWR	Ref.(I=1M)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		I=0.5M	5.3	139.7%	59.8%	2.74%	1.03	558k	11
		I=2M	4.4	139.7%	61.6%	2%	1.01	430k	12
	DWWR	Ref.(I=1M)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		I=0.5M	6.1	139.7%	57.7%	1.95%	1.03	568k	14
		I=2M	4.87	139.7%	59.7%	4.3%	1.01	438k	15
	DWAWR	Ref.(I=1M)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		I=0.5M	5.93	139.7%	54.1%	4.4%	1.03	674k	17
		I=2M	6.45	139.7%	56.8%	2.2%	1.01	525k	18

7.1 Change in Interval (I)

Table 14 reports the different metrics for distinct interval values in the proposed schemes. Change in the interval affects the frequency of the selection process of write restricted window/ways in the cache. With large interval value (row 3, 6, 9, 12, 15 and 18), the frequency of window/ways selection process is low compared to the reference case. This, in turn, lessens the premature invalidation from the L2 cache because the block evicted from other windows/ways are mostly LRU blocks compared to the reference case. The long interval value leads to the bigger coefficient of intra-set write variation with lesser improvement in the lifetime. On the other hand, smaller interval value (row 2, 5, 8, 11, 14 and 17) increases the write restricted window/ways selection frequency which in turn increases the invalidations in the cache. Also, the smaller interval value does not capture all the write-backs from the L1 cache to the write restricted window/ways. Thus, the coefficient of intra-set variation and relative lifetime is not improved as much as compared to the reference case. Also, with the smaller interval, due to premature evictions of the block, the system performance is severely affected with increased EDP (shown in column 8 of row 2,5,8,11,14 and 17 of Table 14).

7.2 Change in Write Restricted Window/Ways (m/n)

Table 15 lists the result metrics for different write restricted window/ways sizes (m/n). By altering the size of write restricted windows/ways, the write redirection process of the proposed approaches is influenced. On

TABLE 15
Comparative Analysis for Different Write Restricted Window/ Ways Size (m/n) LFT.= Lifetime, BASE = Baseline STT-RAM, WrRes = Write Restricted, EDP = Energy-Delay Product

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(m=4)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		m=2	5.2	90.94%	33.7%	1.55%	1.03	410k	2
		m=8	4.7	90.94%	37.8%	-0.22%	1.00	309k	3
	DWWR	Ref.(m=4)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		m=2	5.96	90.94%	33.3%	1.68%	1.03	585k	5
		m=8	4.73	90.94%	37.4%	0.85%	1.00	313k	6
	DWAWR	Ref.(n=4)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		n=2	4.9	90.94%	36.6%	0.72%	1.01	325k	8
		n=8	6.83	90.94%	31.4%	3%	1.04	590k	9
Quad	SWWR	Ref.(m=4)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		m=2	5.92	139.7%	58.6%	0.74%	1.04	600k	11
		m=8	4.14	139.7%	64.8%	-1.97%	1.02	368k	12
	DWWR	Ref.(m=4)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		m=2	6.14	139.7%	56.5%	5.1%	1.04	602k	14
		m=8	4.24	139.7%	65.6%	-1.15%	1.03	385k	15
	DWAWR	Ref.(n=4)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		n=2	5.1	139.7%	63.1%	-1.71%	1.02	143k	17
		n=8	7.77	139.7%	51.7%	3.1%	1.04	399k	18

TABLE 16
Comparative Analysis for Different Capacity

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(8MB)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		4MB	5.2	76.2%	25.1%	5.92%	1.02	432k	2
		16MB	3.55	111.6%	51.5%	0.66%	1.01	334k	3
	DWWR	Ref.(8MB)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		4MB	5.65	76.2%	24.7%	6.37%	1.03	439k	5
		16MB	4.79	111.6%	49.2%	0.05%	1.02	339k	6
	DWAWR	Ref.(8MB)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		4MB	6.25	76.2%	23.6%	6.78%	1.03	447k	8
		16MB	4.81	111.6%	47.4%	0.53%	1.02	385k	9
Quad	SWWR	Ref.(16MB)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		8MB	5.1	112.5%	43.1%	4.47%	1.01	612k	11
		32MB	5.3	163.1%	78.1%	14.5%	1.02	414k	12
	DWWR	Ref.(16MB)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		8MB	5.5	112.5%	41.5%	2.4%	1.02	617k	14
		32MB	6	163.1%	76.9%	17.2%	1.02	423k	15
	DWAWR	Ref.(16MB)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		8MB	5.8	112.5%	38.3%	5%	1.02	703k	17
		32MB	6.23	163.1%	72.8%	15.5%	1.02	517k	18

increasing the size (row 2, 5, 9, 11, 14 and 18), the number of write redirection increase (as more ways are selected) which in turn improves the lifetime and reduces the coefficient on intra-set write variation more than the reference case. However, at the same time, it increases the invalidations (due to less availability of ways in the other window/ways) and also increases the EDP. Besides, on reducing the size (row 3, 6, 8, 12, 15 and 17), fewer ways are selected for the write redirection process and this, in turn, reduces the lifetime improvement and increases the coefficient of intra-set write variation.

7.3 Change in Capacity

Table 16 shows the behavior of the proposed schemes with different cache capacities. Larger cache (row 3, 6, 9, 12, 15 and 18) suffers from large intra-set write variation as compared to smaller cache (row 2, 5, 8, 11, 14 and 17). This is because, in case of larger cache, the cache block faces less capacity miss as compared to smaller cache. This, in turn, increases the residency of the block and increases the intra-set write variation more than the reference case. In these cases, for small and large size cache, the proposed schemes effectively reduces the coefficient of intra-set write variation and improves the relative lifetime very efficiently.

TABLE 17
Comparative Analysis for Different Associativity (A)

Core	Policy	Param.	Norm. Lft.	IntraV Base	IntraV WrRes	InterV Red. %	Norm. EDP	Invalid-ation (k)	Row
Dual	SWWR	Ref.(A=16)	5.1	90.94%	34.2%	1.12%	1.02	433k	1
		A=8 way	2.78	69.94%	32%	1.94%	1.02	407k	2
		A=32 way	6.73	109.6%	37.5%	3.25%	1.04	462k	3
	DWWR	Ref.(A=16)	5.72	90.94%	34.1%	3.7%	1.02	437k	4
		A=8 way	2.91	69.94%	30.3%	1.92%	1.02	412k	5
		A=32 way	8.53	109.6%	36.1%	2.1%	1.04	464k	6
	DWAWR	Ref.(A=16)	6.77	90.94%	31.8%	5.3%	1.02	485k	7
		A=8 way	3.40	69.94%	29.2%	2.3%	1.02	425k	8
		A=32 way	8.78	109.6%	34.5%	2.3%	1.04	508k	9
Quad	SWWR	Ref.(A=16)	4.8	139.7%	59.7%	0.75%	1.02	490k	10
		A=8 way	3.16	109.8%	50.6%	2.8%	1.02	491k	11
		A= 32 way	6.40	168.8%	66.4%	2.3%	1.03	522k	12
	DWWR	Ref.(A=16)	5.75	139.7%	56.7%	1.26%	1.03	500k	13
		A=8 way	3.62	109.8%	48.8%	6.3%	1.02	494k	14
		A=32 way	7.53	168.8%	64.7%	3.8%	1.04	568k	15
	DWAWR	Ref.(A=16)	7.27	139.7%	53.2%	4.2%	1.03	591k	16
		A=8 way	3.84	109.8%	45.5%	7.55%	1.02	545k	17
		A=32 way	7.92	168.8%	62.4%	-0.34%	1.04	606k	18

TABLE 18
Lifetime Comparison Analysis (in Years) by the Proposed Schemes: SWWR, DWWR and DWAWR

	PARSEC v2.1					SPEC CPU 2006			
Workload	Body	Cann	Dedup	Swap	X264	Mix1	Mix2	Mix3	Mix4
STT-RAM									
Ideal	41.8K	3.8K	4.8K	14.3K	8.7K	3.21K	7.44K	14.9K	25.5K
Baseline	38	19.2	30.9	41.3	8.65	47.3	6.33	25.8	45.7
SWWR	72.9	128.5	134.6	60.9	65.5	451.1	66	98.1	278.5
DWWR	125.5	191.6	91.4	94.1	75.1	436.5	76.5	121	306.9
DWAWR	119.5	194.1	189.4	114.2	104.6	456.4	103.9	146.6	447.8
ReRAM									
Ideal	2.12K	99.5	795.4	6.37K	462.2	185.4	175.2	565.5	666.2
Baseline	0.61	0.51	0.53	1.17	1	1.16	2.22	3.03	3.02
SWWR	1.63	4.26	1.48	1.48	2.05	13.3	22.51	16.34	21.86
DWWR	1.44	4.43	2.10	1.45	2.96	14.87	22.1	20.13	24.6
DWAWR	1.71	4.63	2.58	1.82	5.10	22.85	24.8	26.2	28.77

7.4 Change in Associativity (A)

Table 17 shows the different behaviors by the proposed schemes under different associativity of L2 cache. Cache with higher associativity (row 3, 6, 9, 12, 15 and 18) suffers from large intra-set write variation as compared to cache with lower associativity (row 2, 5, 8, 11, 14 and 17). This is because cache associativity impacts the conflict misses which in turn affects the residency of the block. In particular, the cache with large associativity faces fewer conflict misses as compared to cache with lower associativity. This increases the residency of the block in case of cache with higher associativity. In both the cases (cache with lower and higher associativity), our proposed schemes are able to reduce the coefficient of intra-set write variation and improve the lifetime.

7.5 Storage Overhead

Our techniques consume the limited amount of overhead over the baseline STT-RAM. The amount of overheads by SWWR is only 42 bits. However, the window counter (C_i) and way counter (Z_k) sizes in case of DWWR and DWAWR depend upon the cache configuration and the parameters of the algorithms. Table 19 reports the counter size and storage overhead for the different configurations and parameters of the algorithms with respect to the chosen configuration:

TABLE 19
Counter Sizes and Storage Overhead (in Bits) of DWWR and DWAWR

Param.	Core	DWWR C_i (bits)	DWWR Overhead (bits)	DWAWR Z_k (bits)	DWAWR Overhead (bits)
Reference	Dual	13	94	11	218
	Quad	13	94	11	218
I=5M/2M	Dual	15	102	12	234
	Quad	14	98	12	234
I=2M/0.5M	Dual	12	90	10	202
	Quad	11	86	10	202
m=2/n=8	Dual	14	98	11	218
	Quad	16	106	11	218
m=8/n=2	Dual	12	90	11	218
	Quad	11	86	11	218
4MB/8MB	Dual	12	90	10	202
	Quad	11	86	9	186
16MB/32MB	Dual	16	106	14	266
	Quad	15	102	12	234
8way	Dual	10	82	8	170
	Quad	9	78	7	154
32way	Dual	18	114	16	298
	Quad	17	110	15	282

8 MB/16 MB 16-way set-associative L2 cache, $I = 2M/1M$ and $m/n = 4$ with the dual/quad core system. The depicted overheads are mainly due to the window and way counter size and the 42-bit tag buffer.

7.6 Recommended Values

Based on the above-detailed analyses, we recommend the values to be used for m, n and I for the different configurations of caches. Table 20 lists these recommended values for the proposed schemes: SWWR, DWWR, and DWAWR. Note that these recommended values are with respect to $m = n = 4$ and $I = 2M$ (for Dual), $1M$ (for quad) cycles. The rationale behind the recommendations is presented below:

- *Interval (I):* The value of I depends upon the associativity. For the cache with smaller associativity, in order to capture the maximum write-backs the value of I can be kept same or increased (according to requirement). On the other hand, in the case of cache with larger associativity, to control the block residency and to reduce the intra-set variation, the value of I needs to be reduced.
- *Write Restricted Window/Ways (m/n):*The value of m, n is decided based upon the following configuration of cache:
  - *Small Associativity:* Smaller associative cache needs small number of write restricted ways. Hence, the value of n should reduce in case of DWAWR. In the case of SWWR and DWWR, the value of m can be the same however the window size (regarding ways) will reduce as associativity is small.
  - *Large Associativity, Small Size:* In this case, to control the block residency, the value of n (Write restricted ways) needs to be increased. On the other hand, in the case of SWWR and DWWR, the partition size is increased accordingly with the number of partitions (m). Hence, the value of m is kept the same.

TABLE 20
Recommended Values of m and I with Respect to Reference Values $m = n = 4$ and $I = 1M$ (Dual) $= 2M$ (Quad) Cycles

Cache Configuration	SWWR/DWWR m	SWWR/DWWR I	DWAWR n	DWAWR I
Small Assoc., Small Size	$=$	$=/\uparrow$	$\downarrow$	$=/\uparrow$
Small Assoc., Large Size	$=$	$=/\uparrow$	$\downarrow$	$=/\uparrow$
Large Assoc., Small Size	$=$	$\downarrow$	$\uparrow$	$\downarrow$
Large Assoc., Large Size	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$

  - *Large Associativity, Large Size:* In order to control the large intra-set write variation, in this case, the partition (SWWR/DWWR)/way (DWAWR) (n) (DWAWR) size need to be increased, or the number of partitions (m) has to be reduced.

7.7 Lifetime Comparison Analysis

Table 18 presents the lifetime comparison analysis (in years) over the ideal STT-RAM/ReRAM, (where the writes are uniformly distributed) baseline STT-RAM/ReRAM, (with no wear leveling policy) and our proposed schemes: SWWR, DWWR, and DWAWR. Note that the values presented in the table are calculated from the reference value presented in the Section 6 (For this analysis, the write endurance values are taken as $4 * 10^{12}$ and 10^{11} writes for STT-RAM and ReRAM respectively as given in [6], [25]). The conclusion that can be derived from the table is that our technique: DWAWR works better than the SWWR and DWWR for both types of NVMs.

8 Conclusion

Due to the differing working set sizes and run-time access patterns of applications, the non-volatile caches suffer from write variations. These write variations not only reduce the lifetime but also shrink the capacity of cache over the period. Write variations inside the cache are governed by the access pattern as well as replacement policies. In this paper, we presented three techniques to mitigate the intra-set write variation. Our first two techniques partition the cache into multiple windows and use a different window during the execution as write-restricted or read-only. In our first technique, the window is selected in a round robin fashion. On the other hand, in our second technique, the window is selected by considering its write intensity over a period of execution. In the third technique, instead of partitioning the cache into windows, a set of ways is selected based on their write-intensity. The selected ways are treated as write restricted or read-only over specific predefined intervals.

The efficacy of the proposed schemes is examined with the help of three existing techniques: PoLF, WAD and EqualChance and, the baseline STT-RAM/ReRAM with no wear leveling policy support. Experimental results show the significant reduction in the coefficient of intra-set write variation along with the improvement in the relative lifetime for dual and quad-core systems. Thus, if we reduce the non-uniform write distribution inside the cache set, we can effectively utilize the emerging non-volatile memories in hardware systems.

REFERENCES

[1] J. Wang, Y. Tim, W. F. Wong, Z. L. Ong, Z. Sun, and H. H. Li, "A coherent hybrid SRAM and STT-RAM L1 cache architecture for shared memory multicores," in *Proc. 19th Asia South Pacific Des. Autom. Conf.*, Jan. 2014, pp. 610–615.

[2] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 34–45.

[3] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 24–33.

[4] Y. B. Kim, S. R. Lee, D. Lee, C. B. Lee, M. Chang, J. H. Hur, M. J. Lee, G. S. Park, C. J. Kim, U. I. Chung, I. K. Yoo, and K. Kim, "Bi-layered RRAM with unlimited endurance and extremely uniform switching," in *Proc. Symp. VLSI Technol. Dig. Tech. Papers*, Jun. 2011, pp. 52–53.

[5] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, *Phase Change Memory: From Devices to Systems*, 1st ed. San Rafael, CA, USA: Morgan & Claypool, 2011.

[6] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 2015.

[7] Y. Huai, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS Bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[8] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i2WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2013, pp. 234–245.

[9] 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.

[10] S. Agarwal and H. K. Kapoor, "Towards a better lifetime for non-volatile caches in chip multiprocessors," in *Proc. 30th Int. Conf. VLSI Des. 16th Int. Conf. Embedded Syst.*, Jan. 2017, pp. 29–34.

[11] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *J. Emerging Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.

[12] 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.

[13] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2009, pp. 14–23.

[14] J. B. Kotra, M. Arjomand, D. Guttman, M. T. Kandemir, and C. R. Das, "Re-NUCA: A practical NUCA architecture for ReRAM based last-level caches," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2016, pp. 576–585.

[15] S. Mittal, J. S. Vetter, and D. Li, "LastingNVcache: A technique for improving the lifetime of non-volatile caches," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2014, pp. 534–540.

[16] Y. Chen, et al., "On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations," *J. Emerging Technol. Comput. Syst.*, vol. 9, no. 2, pp. 16:1–16:22, May 2013.

[17] M. Soltani, M. Ebrahimi, and Z. Navabi, "Prolonging lifetime of non-volatile last level caches with cluster mapping," in *Proc. Int. Great Lakes Symp. VLSI*, May 2016, pp. 329–334.

[18] S. Mittal and J. S. Vetter, "Addressing inter-set write-variation for improving lifetime of non-volatile caches," *5th Annual Non-Volatile Memories Workshop*, 2014.

[19] S. Wang, G. Duan, Y. Li, and Q. Dong, "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.

[20] S. Mittal and J. S. Vetter, "AYUSH: Extending lifetime of SRAM-NVM way-based hybrid caches using wear-leveling," in *Proc. IEEE 23rd Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Oct. 2015, pp. 112–121.

[21] 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.

[22] S. Lee and J. Kim, "Effective lifetime-aware dynamic throttling for NAND flash-based SSDs," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1075–1089, Apr. 2016.

[23] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *Proc. 43rd Int. Symp. Comput. Archit.*, 2016, pp. 519–531.

[24] A. S. Iyengar, S. Ghosh, and N. Rathi, "Magnetic tunnel junction reliability assessment under process variations and activity factors and mitigation techniques," *J. Low Power Electron.*, vol. 14, no. 2, pp. 217–226, 2018.

[25] M. Hosomi, et al., "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *Proc. IEEE Int. Electron Devices Meet.*, Dec. 2005, pp. 459–462.

[26] J. Kan, C. Park, C. Ching, J. Ahn, L. Xue, R. Wang, A. Kontos, S. Liang, M. Bangar, H. Chen, et al., "Systematic validation of 2x nm diameter perpendicular MTJ arrays and MgO barrier for sub-10 nm embedded STT-MRAM with practically unlimited endurance," in *Proc. IEEE Int. Electron Devices Meet.*, 2016, pp. 27–4.

[27] O. Golonzka, et al., "MRAM as embedded non-volatile memory solution for 22FFL FinFET technology," in *Proc. IEEE Int. Electron Devices Meet.*, 2018, pp. 18–1.

[28] Y. J. Song, et al., "Demonstration of highly manufacturable STT-MRAM embedded in 28nm logic," in *Proc. IEEE Int. Electron Devices Meet.*, 2018, pp. 18–2.

[29] 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.

[30] X. Dong, C. Xu, 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, Jul. 2012.

[31] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 72–81.

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

[33] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation—A pin-based memory characterization of the SPEC CPU2000 and SPEC CPU2006 benchmark suites," VSSAD, 2010, http://www. glue. umd. edu/ajaleel/workload

Sukarn Agarwal received the BEng degree in information technology from the Madhav Institute of Technology and Science, Gwalior, India, in 2013. He is currently working toward the PhD degree in computer science and engineering at the Indian Institute of Technology Guwahati, India. His research interests include emerging memory technologies, memory system design, and network-on-chip design. He is a student member of the IEEE.

Hemangee K. Kapoor received the BEng degree in computer engineering from the College of Engineering, Pune, India, in 1998, the MTech degree in computer science and engineering from the Indian Institute of Technology Bombay, Mumbai, India, in 2000, and the PhD degree in computer science from London South Bank University, London, United Kingdom, in 2004. Presently, she is a professor with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. Her current research interests include multiprocessor computer architecture, networks-on-chip design, power aware computing, high performance computing. She is a senior member of the IEEE and ACM.