Lifetime Enhancement of Non-Volatile Caches by Exploiting Dynamic Associativity Management Techniques

Sukarn Agarwal, Hemangee K. Kapoor

Lifetime Enhancement of Non-Volatile Caches by Exploiting Dynamic Associativity Management Techniques

Sukarn Agarwal and Hemangee K. Kapoor

Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Assam, India-781039
{sukarn,hemangee}@iitg.ac.in

Abstract. By showcasing the attractive features like high density and low static power, the emerging Non-Volatile Memories (NVMs) have recently being accepted act as a prominent choice in the memory hierarchy, including caches. However, the limited write endurance with the write variation introduced by the applications and the existing cache management policies leads to an early breakdown of NVM cells, thus reducing the effective lifetime.
This chapter presents efficient techniques to improve the lifetime by mitigating inter-set write variation. Our first technique: FSSRP partitions the cache into groups of sets called fellow groups. Each set has two logical parts: Normal and Reserve. Sets within the fellow group can use the reserve part of its fellow sets to distribute the uneven writes. The second technique: FSDRP, based on FSSRP, partitions the cache vertically into equal-sized windows and use a different window one at a time as a reserve part during the execution to disperse the writes uniformly. Experimental results using full system simulation show a significant reduction in inter-set write variation over the baseline and existing technique.

Keywords: Cache Memory, Non-Volatile Memory, Inter-set Write Variation, Lifetime, Fellow Sets, Dynamic Associativity Management

1 Introduction

With the large processing and data demands by the next generation applications, many cores and large-sized Last Level Caches (LLCs) are integrated on-chip. Traditional caches made up of SRAM fail to fulfill the application demands in the context of performance, power, and area. Previous studies [17] have shown that the majority of the LLC power consumption is due to leakage (almost 80%) which alarmingly raises circuit reliability issues. Hence, in the recent days, the emergence of Non-volatile Memory (NVM) technologies is motivating the researchers to look beyond for alternative technologies in the memory hierarchy [19–22]. These emerging NVM technologies include Spin Transfer Torque Random Access Memory (STT-RAM), Phase Change Random Access Memory (PCRAM)

and, Resistive Random Access Memory (ReRAM). The benefits of using these NVMs in the memory hierarchy are high density, low static power consumption and, good scalability. However, the major limitations of using NVM technologies in the cache hierarchy are costly write operations (such as latency and energy) and, weak write endurance. The write endurance value of the NVMs are: 10^{11} writes for ReRAM [6], 10^8 writes for PCRAM [5] and, for STT-RAM the predicted write endurance value is around 10^{15} writes, but the value tested so far is $4 * 10^{12}$ write operations [7, 23]. In contrast, the write endurance value for the charge based traditional memory technologies such as SRAM/DRAM is more than 10^{15} writes. This shows that the compared to NVM caches, the lifetime of the conventional caches are at-most 1000 times better.

With the limited write endurance, the lifetime of NVM cache is further affected by the write variations generated by the applications running on multiple cores. Because of the write variations, the running applications create certain hot-spots in a cache to better utilize the temporal locality. In a cache, the write variations are categorized into two types: *Inter-set* and *Intra-set* write variations. Intra-set write variation actually takes place inside a single cache set. Specifically, the intra-set write variation occurs due to distinct write counts of the blocks inside a set. In particular, some of the blocks inside the set experience more number of writes as compared to other blocks, which implies that the heavily written blocks inside the set wear out faster than the other blocks. Inter-set write variation, on the other hand, implies non-uniform write count values across the cache sets. This non-uniform write distribution between the cache sets clearly indicates that some of the cache sets face much more writes compared to other sets inside the bank, which results into the breakdown of the some of the (heavily written) sets faster than the other ones.

This chapter proposes effective techniques to reduce the inter-set write variation with the improvement in the lifetime for the NVM cache. Our first technique: Fellow Set with Static Reserve Part (FSSRP) logically partitions the cache sets into groups of sets called fellow groups. Every group has two logical parts: Normal Part and Reserve Part (NP and RP). The working of NP in the group is same as the conventional cache. While the RP of the group is used to handle the non-uniform write distribution of the heavily written sets in the group. In other words, sets within a fellow group can use the reserve parts from their fellow sets to distribute the writes uniformly. However, the major hurdle with the employment of the FSSRP is the limited lifetime improvement due to large number of writes in the static or fixed RP section of the cache which in turn contributes to the intra-set write variation. To overcome this, our second technique: Fellow Set with Dynamic Reserve Part (FSDRP) based on FSSRP, logically divides the cache into multiple equal-sized windows. During the execution, a different window of the cache is used as RP section for a certain predefined interval in order to distribute the writes. By this way, the FSDRP disperses the redirected writes over the cache.

We implemented our proposed schemes on STT-RAM [4] based non-volatile cache. However, the techniques can be easily extended to other non-volatile

caches such as PCRAM and ReRAM based caches. The main contributions of this chapter are as follows:

- We present the efficient techniques to reduce the inter-set write variation that helps to improve the lifetime of non-volatile caches.
- Our first technique: Fellow Set with Static Reserve Part (FSSRP) [18] partitions the cache sets into groups called fellow groups. Each set in the group has two parts: Normal and Reserve. Sets within the fellow group can use the reserve parts from their fellow sets to distribute the writes uniformly.
- Our second technique: Fellow set with Dynamic Reserve Part (FSDRP) uses fellow groups as same as FSSRP. In addition, it further partitions the cache vertically into multiple windows. During execution, a different window is used as reserve part over a certain interval in order to distribute the writes uniformly.
- We use full system simulator GEM-5 [1] for experimental evaluation. Results are compared with the existing technique: Swap Shift [8] and the baseline STT-RAM based cache with no support of wear leveling. We also provide a detailed analysis with different configurations of the LLC and, by varying the parameters of the techniques.

The rest of the chapter is organized as follows: Background and Motivation are discussed in Section 2. Section 3 reports the related works. Section 4 presents the proposed wear leveling techniques. Experimental Setup is discussed in Section 5. Results and Analysis are presented in Section 6. Finally, Section 7 concludes the chapter.

2 Background and Motivation

2.1 STT-RAM

We applied our schemes on an STT-RAM [4] based non-volatile cache. This subsection gives the brief overview of STT-RAM cell. Figure 1 depicts the schematic view of STT-RAM cell.

The STT-RAM cell contains an Access transistor and a Magnetic Tunnel Junction (MTJ). An MTJ is constructed from two ferromagnetic layers viz. reference and the free layer and the tunnel barrier (made up of MgO). The use of tunnel barrier in the MTJ is to provide an insulation between the ferromagnetic layers. The magnetization direction of the reference layer is fixed whereas, the magnetization direction of the free layer is changed according to the spin-polarized current. These magnetization directions are actually used to represent the data bit stored in the cell. The anti-parallel magnetization direction resembles high resistance that represents a logical '1', while the parallel magnetization direction represents the logical '0' having low resistance.

The read and write operations in the STT-RAM cell are performed with the help of source and a bit-line. The SET operation or writing '1' in the STT cell is completed by applying a large negative voltage between the source and bit-line. On the other hand, the RESET operation or writing '0' in the STT cell

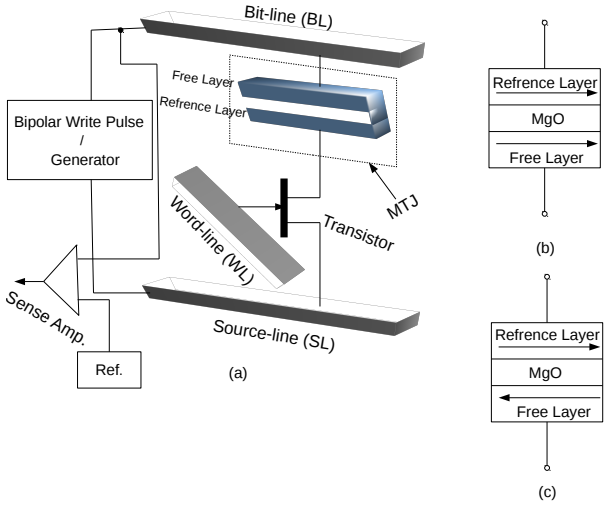

Fig. 1. (a) Representational View of STT-RAM cell (b) Parallel low resistance, representing '0' state (c) Anti-parallel high resistance, representing '1' state.

is achieved by a large positive voltage. In order to perform the read operation or to detect the state of STT cell, a small voltage is applied between the lines which in turn generates the current. The generated current is compared with the reference current to detect the state of a cell.

2.2 Coefficients of Write Variation and Lifetime

The chapter gives wear leveling policies to reduce the inter-set write variation present in the cache. The write variations inside the cache are measured with the help of coefficients. Equations (1) and (2) represent these coefficients [8]: (i) $InterV$, that measures the average coefficient of variation across cache sets, and (ii) $IntraV$ measures the average coefficient of variation inside a cache set.

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

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

In Equations (1) and (2), S is the number of cache sets, A implies cache associativity, $W_{k,l}$ is the write count in set k and way l and, $Write_{avg}$ is the average

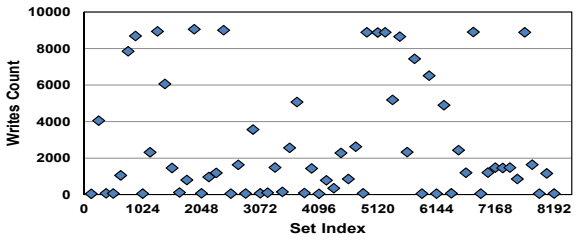

Fig. 2. Uneven write distribution across the cache set.

write count in a cache bank. Following equation represents the average write count:

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

The Lifetime of the cache can be defined either with raw lifetime or error tolerant lifetime. The raw lifetime is determined by the first breakdown of the cache block, whereas, error tolerant lifetime is determined by raw lifetime and the error recovery methods. In this work, we focus on the raw lifetime which is the base of an error-tolerant lifetime. We assume that by integrating other existing error recovery methods with our techniques the lifetime can be further improved. To measure the raw lifetime, it is important to consider three factors: average write count ($Write_{avg}$), the coefficient of Inter-set write variation ($InterV$) and, the coefficient of Intra-set write variation ($IntraV$). To measure the Lifetime Improvement [8] (LI), we use the following equation:

$$LI = \frac{Write_{avg\_base} * (1 + InterV_{base} + IntraV_{base})}{Write_{avg\_pt} * (1 + InterV_{pt} + IntraV_{pt})} - 1 \tag{4}$$

Here $Write_{avg\_base}$, $InterV_{base}$ and $IntraV_{base}$ represent average write count, coefficient of Inter-set write variation and coefficient of Intra-set write variation, respectively of the baseline scheme against which the lifetime is computed. Correspondingly, $Write_{avg\_pt}$, $InterV_{pt}$ and $IntraV_{pt}$ are the average write count, the coefficient of Inter-set write variation and coefficient of Intra-set write variation of the proposed technique, respectively.

2.3 Motivation

Our proposed schemes are dealing with the inter-set write variation present in the cache. To measure the effect of inter-set write variation, we conducted an experiment with 8MB 16-way set associative L2 cache as LLC (details about the experimental setup is discussed in Section 5) on *dedup* workload. Figure 2 shows the write distributions across the cache sets inside a cache bank. The conclusion that can be drawn from Figure 2 is the non-uniform write distribution across the cache sets which in turn indicates the inter-set write variation inside the cache

bank. This write variation is of concern as it not only reduces the lifetime but also shrinks the cache capacity eventually over the period of time.

3 Related Works

In the existing state of the art, previously many inter-set wear leveling techniques have been proposed. This section gives a brief insight about such policies. We also discuss the other existing inter-set wear leveling policies and the Dynamic Associativity Management (DAM) based approaches in the cache.

Wang et al. [8] presented a technique called *iiwap* to reduce both inter-set and intra-set write variation inside the cache. The first technique to reduce the inter-set write variation is Swap Shift where the two cache sets of a bank interchange their mapping by invalidating their data after a certain number of writes represented by the variable $SwapTh$. The second technique to reduce the intra-set write variation is Probabilistic Set Line Flush (PoLF) that invalidates the data block after a fixed number of writes called Flush Threshold (FT). Another inter-set wear leveling technique proposed by Chen et al. [13] that changes the cache set mapping at regular intervals by performing XOR operation between the content of register (called remap register) and the set index of the block. At the end of each interval, the content of remap register is changed. Jokar et al. [9] reports an inter-set wear leveling approach that changes the set mapping between the heavily written set and lightly written set with the help of counters. Partitioning the cache into multiple clusters and dynamically changing their mapping according to their write intensity, history and the number of clean/invalid blocks are proposed by Soltani et al. [15]. Wang et al. [14] presented a word and partition level write variation reduction scheme that explores the narrow width data for the word, and software and hardware partitions for the different running applications. Here, the write variation is balanced by swapping the data in the word and across the partitions. A software controlled cache coloring page mapping approach presented by Mittal et al. [16] changes the mapping of the two colors according to their write traffic. Unlike the previously presented approaches, our technique neither changes the set mapping nor swaps the data, rather it uses the same set mapping by exploiting the Dynamic Associativity Management (DAM) towards balancing the write variation across the set.

Qureshi et al. [10] proposed V-way, a DAM-based approach, where the associativity is managed by decoupling the data and tag array. Another approach based on skew associative cache, called Z-cache was reported in [11]. Here, the associativity is increased by increasing the replacement candidates. A DAM based victim retention approach is reported by Das and Kapoor [12]. Our work is inspired by [12].

4 Proposed Wear Leveling Techniques

In this section, we will illustrate both of our proposed techniques: (i) Fellow Sets with Static Reserve Part and (ii) Fellow Sets with Dynamic Reserve Part. Note

that, former one, Fellow Sets with Static Reserve Part is from our prior work [18] and it is demonstrated and examined extensively in this chapter.

4.1 Fellow Sets with Static Reserve Part (FSSRP)

Architecture: The main idea of FSSRP is to exploit Dynamic Associativity Management (DAM) towards the inter-set wear leveling. The scheme partitions the cache into groups of sets, called fellow groups. To empower DAM for a set, we further partition each set into two parts: Normal Part (NP) and Reserve Part (RP). The NP simply replicates the conventional cache, whereas, RP of all sets within a fellow group are shared and thus can be used by the sets belonging to the same group. This phenomenon enables the heavily written set in the fellow group to use the RP section of a lightly written set in the same group in order to store its blocks. By this way, we can dynamically manage the associativity towards the inter-set wear leveling. Note that, our intention is not to increase the associativity, rather to use the RP section and fellow sets towards the inter-set wear leveling. Particularly, a set having heavy write usage can avoid the additional writes by redirecting the writes to the RP section of a lightly written set in its fellow group. The write redirection is initiated with the help of a counter associated with each set.

One critical issue with the architecture of FSSRP is to search the relocated blocks that are moved to the sets other than their home set. To deal with this, an additional mapping table, called TaG Storage (TGS) is used. Basically, TGS is a two-dimensional table where each entry contains the tag address and a valid-bit for the blocks that belong to the RP section. In particular, each entry of TGS has one to one mapping with every block in the RP.

Let there be a set associative cache with associativity A and S number of cache sets. The total number of ways reserved for the RP section be r with the size of the fellow group m. Note that all the sets in the fellow group are statically grouped. With this information, the fellow group in the cache has the following properties:

- Total number of fellow groups in the cache: S/m.
- Distance between any two sets of the fellow group in the cache: $S/m - 1$.

The architecture of TGS has the following characteristics:

- TGS Associativity: $A_{tgs} = r * m$.
- Number of sets in TGS: $S_{tgs} = S/m$.
- Total number of blocks in TGS: $B_{tgs} = r * S$.

As mentioned previously, each entry in the RP section of the cache has one to one mapping with each entry of the TGS. For a given TGS set (S_{tgs_i}) and TGS associativity (A_{tgs_j}), the respective cache set (S_k) and cache associativity (A_l) can be easily mapped with the help of following equations:

$$A_l = (A - r) + (A_{tgs_j} \% r) \tag{5}$$

$$S_k = ((A_{tgs_j}/r) * S_{tgs}) + S_{tgs_i} \tag{6}$$

Similarly, for a given cache associativity (A_v) and cache set (S_u), the respective TGS set (S_{tgs_n}) and TGS way (A_{tgs_m}) can be simply mapped with the help of following equation:

$$A_{tgs_m} = (S_u/S_{tgs}) * r + (A_v - (A - r)), \quad (A - r) \leq A_v \leq A \tag{7}$$

$$S_{tgs_n} = S_u \% S_{tgs} \tag{8}$$

In addition to TGS, we add a write counter with each cache set and a write bit with each block belonging to the NP section of the cache. The use of the write counter is to count the number of writes performed in the set. Detailed use of the write counter and the write bit is explained in the next subsection.

The example of our proposed architecture: FSSRP is shown in the Figure 3. In the example, an L2 cache or LLC with 16-way associativity and 8 cache sets, partitioned into two parts: NP and RP. The number of ways allocated to NP and RP section is 12 and 4 ($r = 4$) respectively. Let the fellow group size to be 2 ($m = 2$). With these given values, the distance between two sets in the fellow group is $S/m - 1 = 3$ and the total number of the groups formed in the cache is $S/m = 4$. As shown in the Figure 3, these four groups are labeled with $G0, G1, G2$ and $G3$ respectively, with the corresponding distance between the two sets $S0$ and $S4$ in the group $G0$ is 3. Similarly, the TGS architecture has the following set of characteristics: $S_{tgs} = S/m = 4$, $A_{tgs} = r * m = 8$ and $B_{tgs} = r * S = 32$. Each set in the TGS resembles one fellow group of the cache. In our example, the set-0 of TGS contains an entry for the fellow group G0 i.e. way-12 to way-15 of cache set 0 and 4.

Operation: We demonstrate the operation of our proposed scheme through algorithm 1. In the algorithm, the variable parameter I is used as a predefined interval (line 1). The variable W_i is used to represent the write counter associated with the cache set i (line 2). Similarly, the variable b_{ij} (set i and way j) resembles the write bit incorporated with each block in the NP section of cache (line 3). The lightly written set of the fellow group is represented by S_l. The decision of the lightly written set in the group is taken with the help of write counter associated with the set of the group. In particular, the set with the least value of the write counter in the group is treated as a lightly written set (S_l) of the group. For the initial I cycles of the process execution, the cache is treated as a normally available cache (line 4). During the interval, if any write happens to any block in the set, the write bit associated with the block b_{ij} is set and the respective write counter W_i is incremented (line 5 and 6).

Once the application crosses I cycles, for each request R coming from L1 cache to L2 cache, the tag lookup operation is performed in the NP section of the cache. Simultaneously, the tag of the requested block is also searched in the RP section of the cache through TGS. Note that in this case, the TGS set location is mapped by using equation 8. In case, the result of lookup operation

Algorithm 1: FSSRP Wear Leveling Algorithm

I : Predefined interval.
W_i : Write counter associated with set i. $0 \leq i \leq S$
b_{ij} : Write bit associated with the block in set i and way j. $0 \leq i \leq S$, $0 \leq j < (A - r)$
Run application for I cycles treating the cache as a normal cache.
During I cycle, the write counter (W_i) is incremented with each write in set i.
Similarly, the write bit(b_{ij}) is set with each write in the block.
repeat
  for each request R coming from L1 cache to block B in L2 cache **do**
    if $R == ReadHit$ **then**
      The Read operation is performed on the block B irrespective of its location.
    else if $R == WriteHit$ **then**
      if Block B is found in NP part of cache **then**
        if the write bit b_{ij} of the block B is set **then**
          if there exist a light written set S_l in the group **then**
            The write request for the block B is redirected to the location L in the RP part of S_l. { **Block B moved to RP on first write back**}
          else
            The write operation is performed on Block B.
            Increment the write counter (W_i) and keep the write bit set.
          end if
        else
          The write operation is performed on Block B.
          Increment the write counter (W_i) and set the write bit b_{ij}.
        end if
      else
        The write operation is performed on block B.
        Increment the write counter of the set in which the write operation is performed. { **Block B in RP part**}
      end if
    else
      Forward the Request R to main memory. Keep the newly arrived block in NP part of cache. {**cache miss**}
    end if
  end for
until the end of the execution

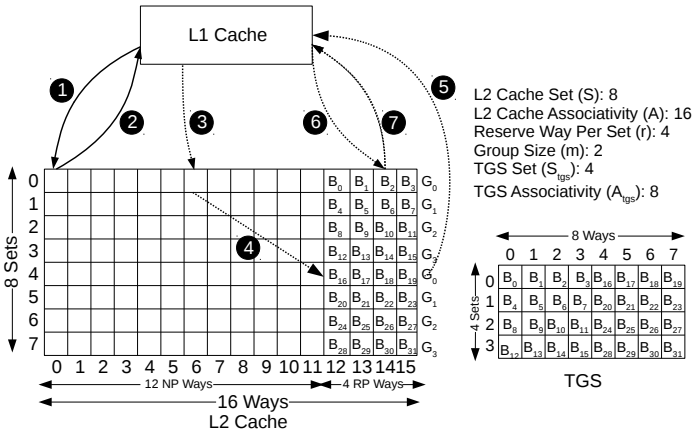

Fig. 3. Working example of FSSRP Wear Leveling Policy

is hit and if the requested block is present in the NP section of the cache, then it is a direct hit. Otherwise, it is an indirect hit. Note that for the block B present in the RP section of the cache, the respective cache set and cache associativity of the block can be easily mapped with the help of Equations (5) and (6).

Depending upon the result of the lookup operation, different operations are performed in the cache which can be explained as follows:

- ***Read Hit:*** The requested block B is served normally to the L1 cache irrespective of its location in the L2 cache (line 9 and 10).
- ***Write Hit (PUTX or write-back) and block B in NP part of the cache:*** If the requested block B is present in the NP section of the cache then we have two cases:
  - If there exist any lightly written set (S_l) other than the current set in the group with write bit b_{ij} of the requested block B set, the write request R from L1 cache is redirected to the RP section of S_l. In this case, if there is an invalid line in the RP section of the cache, the write request R from the L1 cache is simply redirected from the current set to the first invalid location. If there is no invalid line present in the RP section of S_l, the LRU victim line is picked from the RP section and the write-back operation is scheduled according to the status of the dirty bit and the corresponding TGS entry is invalidated. Thereafter, the write request R from the L1 cache is redirected to the generated location of the RP section in S_l. Once the request is served, the subsequent block B is invalidated from its corresponding location in the NP section of the cache. Subsequently, a new entry is created in TGS with the help of Equations (7) and (8) (line 14 to 16).
  - On the other end of the spectrum, if there is no lightly written set (S_l) existing in the group or if the associated write bit (b_{ij}) with the requested block B is not set, the write request R is served from the current set. Subsequently, the write bit b_{ij} of the block is set (line 17 to 21). Note that in this context, the unavailability of lightly written set implies that the current set itself is the lightly written set of the group.

  Once the write request is served, the write counter (W_i) of the cache set (in which the write operation is performed) is incremented.
- ***Write Hit (PUTX or write-back) and block B in RP part of the cache:*** If the requested block B is present in the RP section of the cache, the write request R from the L1 cache is served normally from the respective location of the block B in the L2 cache. Once the write request R is served, the write counter (W_i) of the cache set in which the write operation is performed is incremented (line 22 to 24).
- ***Cache Miss:*** In case, if requested block B is not present in the cache, the request R from the L1 cache will be forwarded to the next level of memory (i.e. main memory in our case). In this case, the incoming block from main memory is placed in NP section of the cache. Simultaneously, the write bit b_{ij} of the cache location in which the incoming block is placed will be reset (line 25 to 27).

Table 1. Percentage Increase in Coefficient of Intra set write variation

Workloads	Swap	Dedup	Body	Fluid	Freq	Stream	X264	Mean
IntraV %	1.3%	45.8%	47.1%	-2.5%	7.1%	8.7%	1.2%	15.5%

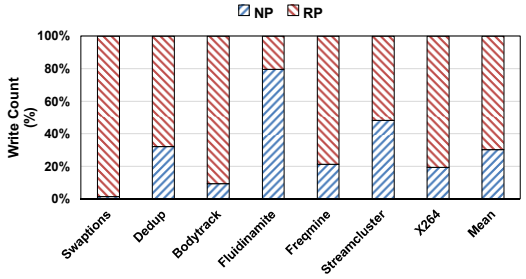

Fig. 4. Write Count percentages in the different section of FSSRP

The working methodology of an algorithm is presented in Figure 3. Note that, the architecture of the cache and TGS have already been elaborated in the previous subsection.

Example: To explain the methodology, three cases are considered with respect to set-0. In the first case, a read request R from L1 cache to the way-0 (shown by arrow 1) of L2 cache is served normally (shown by arrow-2) irrespective of the location (NP or RP) of requested block. In the second case, a write request R (shown by arrow 3) is coming from the L1 cache to the block in way-6 of L2 cache that implies the NP section of the cache. In this case, the write request is redirected to the RP section of the lightly written set (shown by arrow 4) of the fellow group, let say set-4 in our example. Once the request is scheduled in the RP section, the respective TGS entry is updated in the set-0 with the corresponding data attributes of the redirected blocks. Afterwards, the write-back acknowledgment is sent back to L1 cache (shown by arrow 5). In the third case, the write request R (arrow 6) from the L1 cache to the way-14 of L2 cache (RP section) is served normally with the write-back acknowledgment (arrow 7).

Limitation: The limitation of FSSRP is the extensive write accesses and redirections in the limited sized static RP section of the cache. Figure 4 presents the write count percentages in the NP and RP sections of the cache for different benchmark applications (Details about the experimental setup over the selected value is given in Section 5). The conclusion that can be derived from Figure 4 is that, on an average 69.6% of the write access is handled by the limited size static RP section which in turn generates the intra-set write variation as shown in the Table 1. This increment in Coefficient of intra-set write variation of 15.5% over the baseline limits the lifetime improvement (given in Equation (4)) despite the reduction in inter-set write variation by the proposed approach: FSSRP. This

motivates us to make RP section dynamic in the cache in an attempt to control intra-set write variation.

4.2 Fellow Sets with Dynamic Reserve Part (FSDRP)

Architecture The architecture framework of FSDRP is based on FSSRP in that respect it also creates fellow groups. The main idea of FSDRP is to further logically partition the cache vertically into multiple uniformly sized windows (such that each window contains an equal number of ways) and over a certain period of execution a single window is used as an RP, exclusively. In other words, instead of partitioning the cache into two static parts: NP and RP, FSDRP partitions the cache into multiple equal-sized windows of size r ways and dynamically uses a different partition as RP for a certain period of time over the execution. However, during that time, the remaining ways behave as the NP section. By using different windows as RP over the execution, the writes are not concentrated on one part of the cache but get dispersed over the different ways of the sets in the fellow group. Note that our intention is not to reduce the intra-set write variation of the cache but to distributes the redirected writes of the RP section in the cache uniformly over the set.

The one to one static mapping between the TGS entry and relocatable blocks in the other sets of the fellow group has evolved as a major bottleneck towards the implementation of FSDRP. Practically, the dispersion of relocatable blocks in the fellow sets by the dynamic RP window destroys the static mapping setup by FSSRP. In order to deal with this, we add an additional field called win_num with each entry of TGS and a relocate bit (r_{ij}) is added to each entry of the cache along with the write bit (b_{ij}). The use of the win_num field is to store the partition or window number where the relocatable block resides in the cache. The use of relocate bit (r_{ij}) is to identify the normal block from the relocated block in the cache set since the RP window keeps moving.

The partition or window has the following characteristics:

- Size of the window or partition in the cache: r.
- Total number of window or partition in the cache (P): A/r.
- Partition or window number for a given way number (W) in the cache: $\lfloor W/r \rfloor$

Similar to FSSRP, for a given TGS set (S_{tgs_i}) and TGS associativity (A_{tgs_j}), the corresponding cache set (S_k) can be easily mapped with Equation (6). However, the corresponding cache way (A_l) is identified by the following search operation in the cache:

$$A_l = Search(TGS[S_{tgs_i}][A_{tgs_j}].win\_num, TGS[S_{tgs_i}][A_{tgs_j}].tag) \quad (9)$$

The $Search()$ used in Equation (9) takes two arguments: win_num and tag address (tag) from the respective location of TGS and searches the corresponding block in the cache. Similarly, the respective TGS set (S_{tgs_n}) for the cache set (S_u) and cache way (A_v) is mapped from Equation (8). On the other hand, the

TGS associativity (A_{tgs_m}) is derived by the search operation:

$$A_{tgs_m} = Search(Cache[S_u][A_v].tag, A_{tgs_{st}}, A_{tgs_{st}} + r), \quad A_{tgs_{st}} = (S_u/S_{tgs}) * r \tag{10}$$

Here, the $Search()$ used in Equation (10) takes three arguments: tag address of the cache block ($Cache[S_u][A_v].tag$) and the range of the TGS way location as second and third argument ($A_{tgs_{st}}$ to $A_{tgs_{st}} + r$) where the searching takes place for the respective cache block.

Example: The example of FSDRP architecture is depicted in Figure 5. In the example, a 16-way ($A = 16$) associative L2 cache having 8 sets is considered with the values of $m = 2$ and $r = 4$. With these given parameters, the cache is partitioned into four ($A/r = 4$) equal-sized windows of size four ways each($r = 4$). As shown in the example, these four windows are labeled with Win_0, Win_1, Win_2 and Win_3 and the relocatable blocks from the different cache sets of the fellow groups are dispersed in these different windows. The lookup of these relocatable blocks is performed with the help of TGS through Equations (6) and (9). As shown in the example, the block B_{12} placed in window W_0 of the cache has an entry in TGS and it is searched with the help of win_num field (value 0) stored within the TGS entry. In order to distinguish these relocatable blocks, the corresponding relocate bit is set to one. For example, the write bit and the relocate bit for the relocatable block B_{31} are set to 0 and 1.

Operation: The operation of the FSDRP is elaborated through Algorithm 2. In this algorithm, the use of parameters I, W_i and b_{ij} is same as the FSSRP (line 1 to 3). In addition of these parameters, the parameter P acts as the total number of logical partitions or windows in the cache (line 4). The relocate bit associated with each block of the cache is represented by variable r_{ij} (line 5).

For the initial I cycles of the process execution, the cache is available as a normal cache. Meanwhile, during the interval (I), the write operation to any block in the set increments the write counter (W_i) of that cache set, and the respective write bit (b_{ij}) of the block is set (line 6 to 8).

Once the application crosses first I cycles, one window of the cache is selected and treated as RP section of the cache and the rest of the windows act as an NP section. Afterwards, periodically for every interval I, a new window is treated as RP by rotation (line 9 to 13). The process continues until the execution is over.

In the meantime, between the intervals, for each request R coming from L1 cache to L2 cache, the tag lookup operation is performed in the L2 cache simultaneously in both NP and RP (through TGS by Equation (8)). Note that, in the case of an indirect hit, the respective cache set and the cache way is mapped through Equations (6) and (9). Depending upon the result of the lookup and the cache request, different operations are performed in the L2 cache:

- ***Read Hit:***The read operation is same as the FSSRP, as given in Section 4.1 (line 16 and 17).
- ***Write Hit (PUTX or write-back) and block B in NP section of the cache:*** In case of write request R, if the requested block B belongs to the NP section of the cache then we have two cases:

Algorithm 2: FSDRP Wear Leveling Algorithm

I : Predefined interval.
W_i : Write counter associated with set i. $0 \leq i \leq S$
b_{ij} : Write bit associated with each cache block in set i and way j. $0 \leq i < S$, $0 \leq j < A$
P : Number of logical partition or windows.
r_{ij} : Relocate bit associated with each cache block in set i and way j. $0 \leq i < S$, $0 \leq j < A$
Run application for I cycles treating the whole cache as a normal available cache.
During I cycle, the write counter (W_i) is incremented with each write in set i.
Similarly, the write bit(b_{ij}) is set with each write in the block.
After I cycle, treat one window of the cache as a RP window and rotate window number in a round robin fashion.
repeat
  for every interval I **do**
    $i = (i + 1)\%P$
    Window Win_i is selected as a RP section for the current interval I.
    Windows other than the W_i is treated as NP section of the cache.
    for each request R coming from L1 cache to block B in L2 cache **do**
      if $R == ReadHit$ **then**
        The Read operation is performed on the block B irrespective of its location.
      else if $R == WriteHit$ **then**
        if Block B is found in NP part of cache **then**
          if the write bit b_{ij} of the block B is set and the relocate bit r_{ij} is not set **then**
            if there is any light written set S_l exist in the group **then**
              The write request for the block B is redirected to the location L in the RP part of S_l. { **Block** B **move to RP after first write back**}
              The relocate bit (r_{lm}) for the redirected block is set.
            else
              The write operation is performed on Block B.
              Increment the write counter (W_i) and keep the write bit set.
            end if
          else
            The write operation is performed on Block B.
            Increment the write counter (W_i) and set the write bit in set i and way j.
          end if
        else
          The write operation is performed on block B.
          Increment the write counter of the set in which the write operation is performed. { **Block** B **in RP part**}
          if r_{ij} is not set **then**
            Set the write bit (b_{ij}) for the block B.
          end if
        end if
      else
        Forward the Request R to main memory. Keep the newly arrived block in NP part of cache (location other than Win_i). {**cache miss**}
      end if
    end for
  end for
until the end of the execution

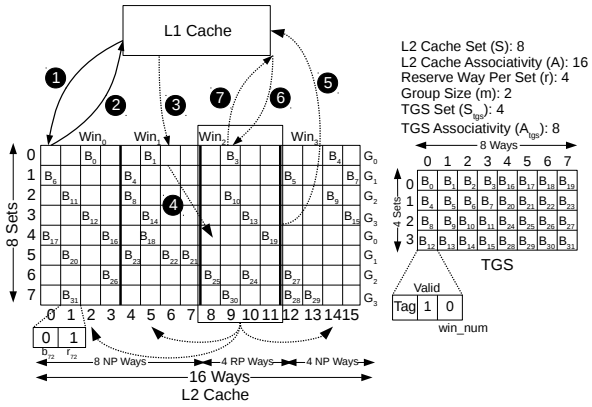

Fig. 5. Working example of FSDRP Wear Leveling Policy

- In the first case, for the requested block B if the write bit (b_{ij}) is set and the relocate bit (r_{ij}) is zero. And, at the same time, if the lightly written set (other than the current set) (S_l) is present in the fellow group, the write request R is redirected to the RP window (Win_i) of the S_l. In this case, if the invalid entries exist in the TGS (within the range $A_{tgs_{st}}$ to $A_{tgs_{st}}+r$) and the RP window, the write request (R) is simply redirected to the invalid location of the RP section by updating the TGS entry. On the other hand, if there is no invalid entry present in the RP window of the cache or if there is no vacant entry present in the TGS, the respective LRU victim entry is picked from the RP window or from the TGS and the write-back operation is performed for either or both the entries of the L2 cache. Note that, the cache location of the LRU TGS entry is mapped by the Equations (6) and (9). Afterwards, the write request R from an L1 cache is redirected to the newly generated entry in the RP window. Simultaneously, the newly generated TGS entry is also updated with the redirected data entry attributes. Once the write request R is redirected, the subsequent block B is invalidated from the NP section and the relocate bit (r_{lm}) is set for the redirected block in the RP window section for future identification (line 21 to 23).
- In the second case, if the write bit (b_{ij}) is not set or if the relocate bit (r_{ij}) is set or if there is no lightly written set other than the current set exist in the group, the write operation is performed in the current location of the block B by setting the write bit b_{ij} (line 24 to 28). Note that the setting of the relocate bit r_{ij} implies that the current block belongs to the other set of the fellow group.

Once the write operation is performed, the respective write counter (W_i) of the cache set is incremented.

– ***Write Hit (PUTX or write-back) and block B in RP section of the cache:*** In this case, the write request R is performed normally from the

Table 2. System Parameters

Components	Parameters
Processor	2Ghz, 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
L2 Cache	Shared, 64B block, LRU, write-back policy
Protocol	MESI CMP Directory

current location of the block B. If the relocate bit (r_{ij}) of the block B is not set then the corresponding write bit (b_{ij}) is set. This implies that the block B is not yet redirected and currently belongs to the home set (line 30 to 33).

– ***Cache Miss:*** In case of cache miss, the request R from L1 cache is forwarded to the next level of memory (main memory in our case). In this case, the incoming block from the main memory is placed into the window other than the RP window (Win_i). In particular, the block is placed in one of that window which is currently treated as NP section of cache. The write bit (b_{ij}) and the relocate bit (r_{ij}) of the cache location in which the block is placed are reset (line 36 to 38).

Example: The working example of FSDRP is explained through Figure 5. In this example, the window Win_2 is treated as an RP section and the rest of the windows (Win_0, Win_1, Win_3) act as an NP section of the cache. As same as FSSRP, three cases are considered to demonstrate the method with respect to set-0. In the first case, a read request (arrow 1) is served normally irrespective of the location (NP or RP) (arrow 2). In the second case, a write request (arrow 3) to the block (way-6) that belong to the NP section is redirected (arrow 4) to the RP section window (Win_2) of the lightly written set (set-4 in our case) of the fellow group. At the same time, the respective TGS entry of set-0 is updated with the respective data attributes. Once the write operation is performed, the write-back acknowledgment is sent back to L1 cache (arrow 5). In the last case, the write request (arrow 6) to the block B_3 in the RP section of the cache is served normally by the L2 cache with the write-back acknowledgment (arrow 7).

5 Experimental Setup

We implemented our proposed schemes on a full system simulator GEM-5 [1]. Table 2 shows the system parameters used in the simulations. In the simulator, the memory module is simulated with the help of Ruby memory module along with the MESI CMP based cache controller. We perform our experiments on a quad-core system with the different configuration of L2 or LLC. Table 3 reports the timing and energy parameters for these configurations. The timing and the energy parameters are obtained by using NVSIM [2] at 32 nm technology node.

We compared our proposed techniques with baseline STT-RAM cache that uses LRU as a replacement policy with no wear leveling strategy associated

Table 3. Timing and Energy Parameters for STT-RAM L2 cache

	Leakage Power (mW)	**Hit Energy (nJ)**	**Miss Energy (nJ)**	**Write Energy (nJ)**	**Hit Latency (ns)**	**Miss Latency (ns)**	**Write Latency (ns)**
16MB, 16way	15.674	0.367	0.096	4.322	78.453	11.854	71.035
8MB, 32way	8.116	0.366	0.185	6.454	74.792	8.259	70.981
8MB, 16way	8.030	0.273	0.093	4.387	78.497	11.964	70.981
8MB, 8way	7.983	0.227	0.047	3.221	74.454	7.921	70.981
4MB, 16way	7.960	0.217	0.093	4.228	23.876	5.575	26.585

and, the existing technique: Swap Shift. Swap Shift invalidates the data of two sets after the certain number of writes called swap threshold ($SwapTh$). In our experiment, the value of $SwapTh$ is set to 511. The rationale behind the large value of the $SwapTh$ is to restrict the frequent invalidation process of data sets in the cache. The frequent invalidation process in the Swap shift increases the accesses in the main memory that results in the extra performance and energy overhead.

In the proposed schemes, the searching of the block in the home set and the other sets of the fellow group (through TGS) will take place in parallel, so it does not affect the system performance. However, the write redirection of the blocks from the home set to the other set takes 3 extra cycles and an additional swap buffer to the transfer the tag. These 3 cycles are divided as follows: 1 cycle for tag transfer into the swap buffer, 1 cycle for writing the tag in swap buffer and, 1 cycle for transferring the tag to TGS and RP part of the cache. In addition to these cycles, an extra cycle is required in FSDRP for the searching of the block into the respective window of the other sets of the fellow group. We have considered all these overheads in our simulations. We also take into account the energy consumption due to accesses in the TGS. The energy overhead of TGS (made up of SRAM) is modeled by using NVSIM. There is some logic overhead associated with the algorithms 1 and 2 which will consume extra area overhead.

We verified our proposed techniques with the help of PARSEC benchmarks suite [3]. Seven applications (Swaptions (Swap), Dedup, Bodytrack (Body), Fluidinamite (Fluid), Freqmine (Freq), Streamcluster (Stream), X264) with large input set are used in the simulation. These benchmarks are the multi-threaded real time applications such as animation, data mining, multimedia etc. We run each application for five billion instructions in the parallel region (i.e. Region Of Interest (ROI)).

6 Results and Analysis

We evaluate our proposed approaches on a quad-core system. Out of the different configurations of L2 cache, we conducted our experiment on 8MB, 16-way associative L2 cache. In the proposed schemes, we set the value for m, r and I to 4, 4 and 5 million cycles. Later in the section, we analyze the effects by changing these values. We present our results on the following metrics: reduction in coefficient of Inter-set write variation ($InterV$) calculated with the help

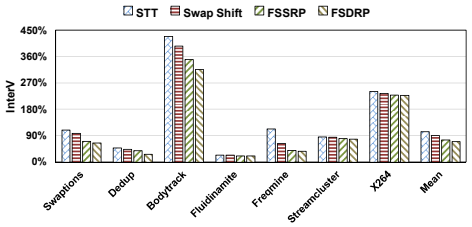

Fig. 6. Inter-set write variation of proposed schemes: FSSRP and FSDRP and, Swap Shift against baseline STT-RAM. (Lesser is better)

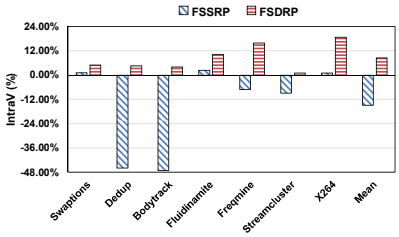

Fig. 7. Percentage reduction in intra-set write variation by FSSRP and FSDRP over baseline STT-RAM. (More is better)

of Equation (1), percentage reduction in coefficient of Intra-set write variation over the previous proposed scheme: FSSRP and the baseline STT-RAM, lifetime improvement percentage calculated with the help of Equation (4), speedup, energy overhead and the number of invalidation/flushes.

6.1 Inter-set Write Variation

Figure 6 shows the inter-set write variation. Our proposed schemes reduce the inter-set write variation from 103.9% (STT), 91.4% (Swap Shift), 76.3% (FSSRP) to 69.9% (FSDRP). The reduction in the coefficient of inter-set write variation is due to uniform write distribution across the cache sets by redirecting the writes from the heavily written sets to the lightly written sets of the fellow groups. However, further improvement in the inter-set write variation for FSDRP over FSSRP is due to different RP window over the period of execution that restricts the write redirection of the blocks from their home sets. Note that, FSSRP redirects every block on the second write without considering its write intensity. While, this is not the case with FSDRP as it partially restricts the redirection of the block that belongs to the current RP window.

6.2 Reduction in Intra-set Write Variation

Figure 7 shows the percentage reduction in intra-set write variation by FSDRP. Compared to baseline and FSSRP, FSDRP reduces the intra-set write variation by 8.55% and 17.7%, respectively. The reduction in the intra-set write variation is mainly due to the dispersion of the relocatable blocks over the cache set by dynamic RP windows. This reduces the possibility of write concentration in the specific section/region of the cache.

6.3 Lifetime Improvement

Figure 8 presents the lifetime improvement percentage by the proposed schemes. Compared to STT-RAM, the improvement in the lifetime by FSSRP is 14.77%

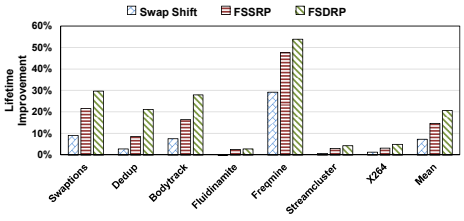

Fig. 8. Lifetime improvement by FSSRP, FSDRP and, Swap Shift with respect to baseline STT-RAM. (More is better)

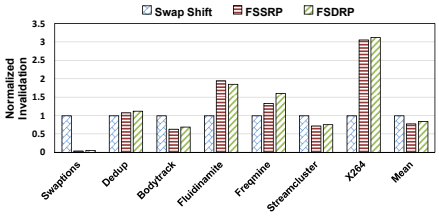

Fig. 9. Normalized invalidations by the proposed techniques: FSSRP and FSDRP over Swap Shift. (Lesser is better)

and by FSDRP is 20.77%. However, the respective values for the lifetime improvement over Swap Shift is 6.58% by FSSRP and 12.11% by FSDRP. These improvements by the proposed schemes are mainly due to the reduction in the coefficient of inter-set write variation. The improvement in the lifetime by FSDRP with respect to FSSRP is 3.03%. The reason for the further lifetime improvement by FSDRP is due to reduction in intra-set write variation in lieu of the dynamic RP window.

6.4 Invalidation/flushes

The invalidation/flushes by the proposed schemes with respect to Swap Shift is depicted in Figure 9. Flushes take place due to the write redirection from heavily written sets to lightly written sets of the fellow group. Compared to swap shift, the proposed schemes reduce the invalidation by 21.59% (FSSRP) and 15.58% (FSDRP). However, as can be seen, FSDRP increases the invalidation against the FSSRP by 7.66%. This is because, in FSDRP, a block gets evicted: (i) from current RP to accommodate the redirected block and (ii) in order to make an entry in the TGS one would need to evict the relocatable block from the current NP or RP section. Note that in some of the cases (dedup, fluid, freqmine and x264) the number of invalidation by our techniques is large than the swap shift. This is because we set a larger value of threshold ($SwapTh$) in swap shift, in order to maintain the performance and energy consumption.

6.5 Energy Overhead

The energy overheads of the proposed schemes are shown in Figure 10. Note that the negative values in the figure implies the energy savings. The energy overhead percentage over the baseline and the Swap Shift by the proposed schemes are 0.78% and 0.99% by FSSRP and, 0.84% and 1.05% by FSDRP. This marginal increment in the energy is basically due to the transfer of tag, invalidation/flushes, accesses in TGS and, the extra window search operation for the relocatable block in the cache by FSDRP.

Fig. 10. Energy overhead by the proposed techniques: FSSRP and FSDRP against baseline STT-RAM. (Lesser is better)

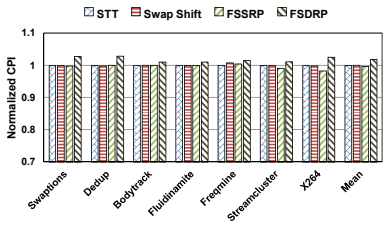

Fig. 11. Normalized performance by the proposed techniques: FSSRP and FSDRP and Swap Shift with respect to baseline STT-RAM.

6.6 Performance

The proposed scheme: FSSRP maintains the same performance as in the baseline STT-RAM and Swap Shift as shown in Figure 11. Performance is not affected in FSSRP because searching for the block in RP section (with the help of TGS) takes place in parallel with the lookup operation in the NP section, and the writing in TGS will be in parallel with the writing in the RP. However, a small degradation of 1% is observed in the CPI for FSDRP due to time taken by extra search operations in the dynamic RP windows.

6.7 Storage Overhead

In our proposed schemes, we use 12-bit write counter (W_i) to measure the write counts in the set. In addition, we add a relocate bit (r_{ij}) and a write bit (b_{ij}) with each block in the cache. Further, each entry of TGS is made up of 42-bit tag address (t), one valid bit (v) and $\log_2 A/r$ bits for window number (win_num) (used in case of FSDRP). We also add a 42-bit swap buffer to transfer the tag to the new location. Thus, the overhead of FSDRP and FSSRP are computed by using the following couple of equations:

$$FSDRP\_Overhead = \frac{S*W_i+S*A*(b_{ij}+r_{ij})+S*r*(t+v+win\_num)+42}{S*A*(B+t)} * 100 \quad (11)$$

$$FSSRP\_Overhead = \frac{S*W_i+S*(A-r)*b_{ij}+S*r*(t+v)+42}{S*A*(B+t)} * 100 \quad (12)$$

In the above Equations (11) and (12), B represents the block size. As an example, in our selected configuration: 8MB 16-way associative L2 cache with the following set of attributes: $m = 4$, $r = 4$ and $I = 5$ million cycles, the percentages of storage overhead in FSSRP and FSDRP are merely 2.21% and 2.52%.

6.8 Parameter Comparison Analysis

In addition to the results presented in the previous subsections, we also performed experiments with different configurations of the cache and the parameters

Table 4. Comparison Analysis for different interval values (I) (LI = Lifetime Improvement, Base = Baseline STT-RAM, EDP = Energy Delay Product) Ref.= 8MB, 16-way, $m = 4$, $r = 4$ and $I = 5M$

Param.	LI (%)	InterV Base	InterV FSDRP	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref.($I = 5M$)	20.7%	103.9	69.9	8.5%	1.02	173k
$I = 2M$	23.2%	103.9	68.2	9%	1.03	188k
$I = 10M$	19.3%	103.9	73.3	3.5%	1.01	155k

Table 5. Comparison Analysis for different fellow group size (m)

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. (m=4)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
m=8	FSSRP	15.5%	103.9	71.3	-17.6%	1.02	172k
	FSDRP	22.1%	103.9	68.9	2.2%	1.02	186k
m=2	FSSRP	12.8%	103.9	78.5	-13.6%	1.00	119k
	FSDRP	19.8%	103.9	73.5	9.3%	1.01	157k

(m, r and I) for the algorithms. Here, we show the results for the reduction in inter-set write variation, the percentage reduction in the intra-set write variation over the baseline, lifetime improvement, EDP overhead and number of invalidations. This analysis is very useful to pick the optimal values (of the parameters) for the proposed approaches in different cache configurations.

Change in Interval (I): Table 4 reports the values over distinct intervals against the reference interval with $I = 5$ Million cycles in the case of FSDRP. Change in the interval-span affects the RP window rotation process. Smaller intervals increase the number of rotations of the RP window over the cache and number of rotations is reduced for the case of larger intervals. Small interval value reduces the inter-set write variation more compared to the large interval. This is because, the large interval value increases the residency of the block that belongs to the home set in the RP window. Such blocks residing in the RP window do not get redirected and thus increase the write count of the set and become eventually dead over the interval. In particular, the blocks belonging to the home set and the RP window incurs several writes before the RP window gets rotated to another location. Simultaneously, with large interval, the write concentration in the RP window region of the cache increases that in turn impacts (and increases) the intra-set write variation. However, in case of small interval values, due to increased invalidations and write redirections, the system performance is affected with more energy consumption which is further resulting into the increment in EDP.

Change in Group size (m): Table 5 presents the results for the different group size (m) in the proposed schemes. Note that, the negative values in the table implies the increment in the intra-set write variation. Change in group size affects the availability of the lightly written sets in the fellow group. In particu-

Table 6. Comparison Analysis for different window size or RP size (r)

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. ($r = 4$)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
$r = 6$	FSSRP	12.7%	103.9	78.2	-18%	1.00	139k
$r = 8$	FSDRP	16.7%	103.9	73.8	2.1%	1.01	169k
$r = 2$	FSSRP	14.3%	103.9	75.8	-12.4%	1.02	201k
	FSDRP	19.3%	103.9	70.3	8.4%	1.03	187k

Table 7. Comparison Analysis for different cache capacity (C)

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. ($C = 8$MB)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
$C = 4$MB	FSSRP	4.2%	73.6	66	-18.4%	1.01	138k
	FSDRP	13.7%	73.6	53.6	4%	1.02	145k
$C = 16$MB	FSSRP	11.2%	169.5	141.2	-2.20%	1.01	153k
	FSDRP	13%	169.5	133.2	2%	1.03	198k

lar, the large group size increases the chance of finding the lightly written set(s) in the group compared to small group size. With the large availability of lightly written set, the chances of write redirection increases and this improves the lifetime and further reduce the inter-set write variation. However, due to increased invalidation and the write redirections, the system performance is marginally affected along with the energy consumption that in turn increases the EDP. Also, the increased number of redirections populates more data in the RP window over the interval, which further impacts the reduction in intra-set write variation as can be seen by the reduced reduction percentage in the Table 5 for both FSSRP and FSDRP.

Change in Window or RP size (r): Table 6 lists the result metrics for changing the window size (i.e. RP size (r)). Change in the window size affects the residency of the relocatable block in the different sets of the fellow group. An increment in window size increases the residency of the relocatable blocks. This, in turn, increases the write count of the lightly written set and it becomes heavily written set over the period of the time. This reduces the possibility of finding the lightly written set in the group and leads to more inter and intra-set write variation compared to small RP/window size. However, in these cases, the inter-set write variation is marginally affected compared to the reference case.

Change in Capacity (C): Table 7 lists the change in metrics values for different cache capacities. The capacity of the cache impacts the number of sets in the cache. Larger caches suffer from large inter-set variation as compared to the smaller sized cache. This is because, in case of larger caches with fixed associativity, the number of cache sets are very large. With the large number of sets, there is a good possibility for the non-uniform write distribution across the cache set which in turn generates the inter-set write variation. However, in both

Table 8. Comparison Analysis for different cache associativity (A)

Param.	Policy	LI (%)	InterV Base	InterV FS	IntraV Red. (%)	Norm. EDP	Invalid-ation(k)
Ref. (A=16way)	FSSRP	14.7%	103.9	76.3	-14.8%	1.01	161k
	FSDRP	20.7%	103.9	69.9	8.5%	1.02	173k
A=8way	FSSRP	8.8%	149.8	130.3	-11.4%	1.00	132k
	FSDRP	16.5%	149.8	110.8	12.9%	1.01	151k
A=32way	FSSRP	5.4%	74.8	66.2	-4.9%	1.02	202k
	FSDRP	10.6%	74.8	58.3	1.6%	1.03	243k

Table 9. Recommended values of m, r and I

	FSSRP		FSDRP		
Cache Configuration	m	r	m	r	I
Small Size, Small Assoc.	↑	↓	↑	↓	= /↑
Small Size, Large Assoc.	↓	↑	↓	↑	↓
Large Size, Small Assoc.	↑	↓	↑	↓	= /↑
Large Size, Large Assoc.	↑	↑	↑	↑	↓

the cases (for smaller and larger caches) our proposed schemes show considerable improvement in the coefficient of inter-set write variation and thus improves the lifetime.

Change in Associativity (A): Table 8 shows the behavior of the proposed schemes with different associativity of the cache. Caches with lower associativity suffers from large inter-set write variation as compared to cache with larger associativity. This is because the cache with the same size and lower associativity have a large number of sets compared to the cache with higher associativity. As same as previous case, the large number of cache sets introduces the uneven write distribution across the set. Although, in both the cases: cache with lower associativity and higher associativity, our proposed schemes efficiently reduce the inter-set write variation and improves the lifetime, accordingly.

Recommended Values: Based on the above analyses, we recommend the values to be used for the m, r (in case of FSSRP) and I (in case of FSDRP) with the different configurations of caches. The Table 9 lists these recommended values for the proposed schemes: FSSRP and FSDRP. Note that these recommended values are with respect to the reference values i.e. $m = 4$, $r = 4$ and $I = 5$ million cycles. The rationale behind the recommended values are presented below:

- **Interval (I):** To control the block residency, higher associativity needs small interval value. Whereas, the cache with lower associativity is not directly affected by I. Here, the value of I may be increased or kept same according to the requirement.
- **Reserve ways (r):** In case of larger associativity, to handle the large write redirections from the NP the value of r need to be increased. The value of r has to be decreased for lower associativity.
- **Fellow Group Size (m):** Large sized cache suffers from large variation. To control this, the value of m needs to be increased. However, in the case of

smaller cache, the value of m is decided according to the associativity and the window size.

7 Conclusion

Write variation inside the cache are affected by the applied replacement policy and the access patterns of the next generation applications running on many cores. In NVM based cache, this large write variation not only curtails the life but also diminishes the capacity of the cache over the period of time. This chapter presented efficient techniques to reduce the write variation across the cache sets called the inter-set write variation. Our first technique: FSSRP partitions the cache into groups of sets called fellow groups. Further, each group is logically divided into two parts: Normal and Reserve. In order to distribute the writes uniformly across the set, the normal part of the set can use the reserve part of the other sets in a fellow group. However, the major concern with the architecture of FSSRP is the increment in write concentration in the reserve part of the cache which in turn increases the intra-set write variation thus limiting the lifetime improvement. To overcome this shortcoming, our second technique: FSDRP, based on FSSRP, partitions the cache vertically into multiple equal-sized windows. During the execution, a different window is selected exclusively as a reserve part for a certain predefined interval. This helps to disperses the redirected writes over the cache. We examine the efficacy of the proposed approaches against the baseline STT-RAM and an existing technique: Swap Shift. Experimental evaluation with a full system simulator shows significant reduction in the coefficient of inter-set write variation along with the lifetime improvement of 14.77% (FSSRP) and, 20.77% (FSDRP) respectively. Simultaneously, we also observe some reduction in intra-set write variation by FSDRP due to dispersion of redirected writes. Thus, minimizing write variation inside the limited endurance non-volatile caches can make the system even more reliable and efficient.

References

1. Binkert, N., et al.: The gem5 simulator. SIGARCH Comput. Archit. News 39, 2, 1-7. (2011)
2. Dong, X., Xu, C., Xie, Y., Jouppi, N. P.: NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. In: IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, pp. 994-1007. (2012)
3. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The PARSEC benchmark suite: Characterization and architectural implications. In: International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 72-81. (2008)
4. Apalkov, D., et al.: Spin-transfer torque magnetic random access memory. J. Emerg. Technol. Comput. Syst. 9, 2, Article 13. (2013)
5. Qureshi, M. K., Gurumurthi, S., Rajendran, B.: Phase Change Memory: From Devices to Systems (1st ed.). Morgan & Claypool Publishers. (2011)
6. Kim, Y. B., et al.: Bi-layered RRAM with unlimited endurance and extremely uniform switching. In: Symposium on VLSI Technology - Digest of Technical Papers. pp. 52-53. (2011)

7. Mittal, S., Vetter, J.S., Li, D.: A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-Volatile On-Chip Caches. In: IEEE Trans. on Parallel and Distributed Systems, vol. 26, no. 6, pp. 1524-1537. (2015)
8. Wang, J., Dong, X., Xie, Y., Jouppi, N.P.: i2WAP: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations. In: IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), pp. 234-245. (2013)
9. Jokar, M.R., Arjomand, M., Sarbazi-Azad, H.: Sequoia: A High-Endurance NVM-Based Cache Architecture. In: IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 24, no. 3, pp. 954-967. (2016)
10. Qureshi, M.K., Thompson, D., Patt, Y.N.: The V-Way cache: demand-based associativity via global replacement. In: 32nd International Symposium on Computer Architecture (ISCA'05), pp. 544-555. (2005)
11. Sanchez, D., Kozyrakis, C.: The ZCache: Decoupling Ways and Associativity. In: 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 187-198. (2010)
12. Das, S., Kapoor, H.K.: Dynamic Associativity Management in Tiled CMPs by Runtime Adaptation of Fellow Sets. In: IEEE Trans. on Parallel and Distributed Systems, vol. 28, no. 8, pp. 2229-2243. (2017)
13. Chen, Y., et al.: On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations. J. Emerg. Technol. Comput. Syst. 9, 2, Article 16. (2013)
14. Wang, S., Duan, G., Li, Y., Dong, Q.: Word- and Partition-Level Write Variation Reduction for Improving Non-Volatile Cache Lifetime. ACM Trans. Des. Autom. Electron. Syst. 23, 1, Article 4. (2017)
15. Soltani, M., Ebrahimi, M., Navabi, Z.: Prolonging lifetime of non-volatile last level caches with cluster mapping. In: International Great Lakes Symposium on VLSI (GLSVLSI), pp. 329-334. (2016)
16. Mittal, S., Vetter, J.S.: Addressing Inter-set Write-Variation for Improving Lifetime of Non-Volatile Caches. In: 5th Annual Non-Volatile Memories Workshop University of California. (2014)
17. Kim, C.H., Kim, J.J., Mukhopadhyay, S., Roy, K.: A forward body-biased-low-leakage SRAM cache: device and architecture considerations. In: Proceedings of the International Symposium on Low Power Electronics and Design, pp. 6-9. (2003)
18. Agarwal, S., Kapoor, H.K.: Targeting inter set write variation to improve the lifetime of non-volatile cache using fellow sets. In: IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 1-6. (2017)
19. Wu, X., Li, J., Zhang, L., Speight, E., Xie, Y.: Power and performance of read-write aware Hybrid Caches with non-volatile memories. In: Design, Automation & Test in Europe Conference & Exhibition, pp. 737-742. (2009)
20. Agarwal, S., Kapoor, H.K.: Restricting writes for energy-efficient hybrid cache in multi-core architectures. In: IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 1-6. (2016)
21. Park, S.P., Gupta, S., Mojumder, N., Raghunathan, A., Roy, K.: Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture. In: DAC Design Automation Conference, pp. 492-497. (2012)
22. Mishra, A.K., Dong, X., Sun, G., Xie, Y., Vijaykrishnan, N., Das, C.R.: Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs. In: International Symposium on Computer Architecture (ISCA), pp. 69-80. (2011)
23. Huai, Y.: Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. AAPPS Bulletin, Vol. 18, No. 6, pp. 33-40. (2008)