EqualChance: Addressing Intra-set Write Variation to Increase Lifetime of Non-volatile Caches

Sparsh Mittal
Oak Ridge National Laboratory
mittals@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory and Georgia Tech
vetter@computer.org

Abstract

To address the limitations of SRAM such as high-leakage and low-density, researchers have explored use of non-volatile memory (NVM) devices, such as ReRAM (resistive RAM) and STT-RAM (spin transfer torque RAM) for designing on-chip caches. A crucial limitation of NVMs, however, is that their write endurance is low and the large intra-set write variation introduced by existing cache management policies may further exacerbate this problem, thereby reducing the cache lifetime significantly. We present EqualChance, a technique to increase cache lifetime by reducing intra-set write variation. EqualChance works by periodically changing the physical cache-block location of a write-intensive data item within a set to achieve wear-leveling. Simulations using workloads from SPEC CPU2006 suite and HPC (high-performance computing) field show that EqualChance improves the cache lifetime by 4.29×. Also, its implementation overhead is small, and it incurs very small performance and energy loss.

Keywords: Non-volatile memory, ReRAM, wear-leveling, intra-set write variation, cache lifetime, write endurance.

1 Introduction

Recent trends of CMOS scaling have led to a large increase in the number of cores on a chip. To feed data to these cores, the size of last level cache (LLC) has also increased, for example, Intel's 22nm 15-core Xeon processor has 37.5MB SRAM LLC [1]. However, since SRAM has high leakage energy and low density, caches designed with SRAM may consume a large fraction of chip area and power budget [2]. To overcome the limitations of SRAM, researchers have recently explored use of NVM devices, such as ReRAM (resistive RAM), STT-RAM (spin transfer torque RAM), and PCM (phase change memory) for designing LLCs [3–6]. NVMs have several attractive features, such as near-zero leakage power consumption, high density and scalability. For example, the size of a typical SRAM cell is in the range of 125–200F^2, while that of a ReRAM cell is in the range of 4–10F^2, where F denotes the smallest lithographic dimension in a given technology node [7–10].

A crucial limitation of NVMs, however, is their small write endurance, which can significantly limit the device lifetime. For ReRAM and PCM, the write endurance values are 10^{11} [11] and 10^{8} [7], respectively. For STT-RAM, although a write endurance value of greater than 10^{15} has been predicted, the best write endurance test so far shows a value of only 4×10^{12} [5, 12]. Process variations may further reduce these values by an order of magnitude [13]. In contrast, the write endurance values of SRAM and DRAM are more than 10^{15} [7]. Further, existing cache management policies are unaware of possible write-variations across the cache. Hence, these variations may significantly increase the writes to a few hotspot cache blocks, causing the devices in those blocks to experience early failures, while other blocks remain reliable. As an example, with LRU (least recently used) replacement policy, most hits are expected to occur at or near MRU (most recently used) position(s). This leads to large intra-set write variation. This is also confirmed by the previous research [5, 14, 15].

To demonstrate the effect of write-variation on cache lifetime and highlight the need of a wear-leveling technique, we take the example of two SPEC CPU2006 benchmarks and compare the L2 cache lifetime when each of these benchmarks is used as a workload in a single-core system. The first benchmark is lbm, which has the highest L2 cache write-intensity among all the SPEC CPU2006 workloads ([16], also confirmed by our experiments) and the second is povray which has the largest coefficient of inter-set and intra-set write variation among all the SPEC2006 workloads (refer Section 3 and Figure 2). We assume an L2 cache (LLC) with LRU replacement policy which does not use any wear-leveling technique. We observe that, *although the total number of writes with lbm is 41 times* ***larger*** *than that of povray, the largest number (i.e. worst-case) of writes on a block for lbm is 20 times* ***smaller*** *than that of povray*. This is due to the fact that with lbm benchmark, the writes are uniformly distributed to different cache blocks, while for povray benchmark, most of the writes are directed to only few cache blocks that would fail much earlier than the rest of the blocks, leading to very short overall cache lifetime.

Thus, contrary to the intuitive expectation, the limited write-endurance of NVMs poses a challenge, for not only the write-intensive workloads but also for the write-unintensive workloads since the *variation* in writes to cache blocks may present more severe challenge than the *magnitude* (i.e. number) of writes. This clearly shows that wear-leveling techniques are crucial for achieving reasonable lifetime with NVM caches and can be very effective in making NVMs the universal memory solution for future extreme-scale computing systems.

1.1 Contributions

In this paper, we present **EqualChance**, a technique for increasing lifetime of NVM caches by mitigating the intra-set write variation. EqualChance records the number of writes to each set and uses this count to periodically shift a write-intensive data item to a block having lower position (i.e. least-recent) in the LRU-stack, since this block is expected to have seen smaller number of writes in the recent execution interval (Section 3). Thus, the future writes will be redirected from a hot (i.e. frequently written) block to a cold block, which helps in achieving wear-leveling. In this paper, we take the example of a ReRAM LLC and based on the explanation, EqualChance can be easily applied to caches designed with other NVMs. In the remainder of the paper, we use the term ReRAM and NVM interchangeably for the sake of convenience.

The storage requirement of EqualChance is less than 0.2% of the L2 cache size and thus, its implementation overhead is very small (Section 4). We conduct microarchitectural simulations using an x86-64 simulator and single-core workloads from SPEC2006 suite and HPC (high-performance computing) field (Section 5). We characterize both intra-set and inter-set write-variation for our workloads for different cache configurations and show that EqualChance significantly reduces the intra-set write variation which results in improvement in the cache lifetime (Section 6.1). Also, EqualChance has minimal impact on performance and energy efficiency. Additional results show that EqualChance performs well for different system and algorithm parameters (Section 6.2) and thus, it offers flexibility to the designer to achieve a balance between the improvement in lifetime and performance/energy loss. .

2 Background and Related Work

Techniques for improving NVM cache lifetime: Cache lifetime can be improved by using either or both of write-minimization or wear-leveling techniques. Some researchers propose write-minimization techniques which work at bit-level by avoiding redundant writes [17, 18] and at cache-access level by using buffers or additional level of caches [19,20]. These techniques are orthogonal to EqualChance and hence, can be synergistically integrated with it.

Based on their granularity, the wear-leveling techniques can be further classified as cache-color level [21], set-level [22], way-level [5] and memory-cell level [18]. As we show in Section 6, for our workloads, intra-set write-variation for typical caches can be higher than inter-set write-variation. EqualChance works at way-level (i.e. it addresses intra-set write variation) and can be easily combined with the set-level or memory-cell level wear-leveling techniques for further improving the cache lifetime.

To leverage the high write-endurance and performance of SRAM along with high density and low-leakage of NVMs, researchers have proposed way-based NVM-SRAM hybrid cache designs [9, 23] where a few ways are designed using SRAM and the remaining ways are designed using NVM. EqualChance technique can be easily used in such hybrid caches also to minimize the write-variation in NVM ways and increase the number of writes in SRAM ways.

Qureshi et al. [24] propose a wear-leveling technique for main memory which periodically changes the mapping between logical and physical address to uniformly distribute the writes in the main memory. Since writes to caches show both inter-set and intra-set variations [6], while those to main memory show only inter-set variation, the wear-leveling techniques proposed for main memory cannot be utilized to mitigate intra-set write-variation in caches.

Discussion of a few wear-leveling techniques: Wang et al. [5] propose a technique, named probabilistic line flush (PoLF) to reduce intra-set write variation. After a fixed number of write hits in the entire cache, a write-operation on cache is skipped, instead the data item is directly written-back to memory and the cache-block is invalidated, without updating the LRU-age information. A common limitation of cache wear-leveling techniques based on data invalidation (e.g. [5,22]) is that in an attempt to reduce or uniformly spread the writes to the cache, they may increase the number of writes on main memory, which increases the memory power dissipation and queue contention. Also, in the modern processors, the main memory itself may be designed using NVM and hence, these techniques may exacerbate the write endurance issue in the main memory. In comparison, EqualChance does not use data invalidation, rather it uses in-cache data movement approach and tries to redirect the hot data-item to a cold block in the same set. Also, EqualChance does not require offline profiling or modification of the program binary (unlike [9]) or including set-index bits as part of the tag (unlike [22]). Also, in EqualChance, data movement is done within the same set and not across sets (unlike [23]) and hence, set-decoding and block-lookup mechanisms are not affected.

3 Methodology

Notations: We use the following symbols. For L2 cache, N, M, L and G denote the number of sets, associativity, cache line (block) size and tag-size, respectively. In this paper, we assume $L = 64$B (or 512 bits) and $G = 40$ bits. Also, we use W_{avg} to show the average writes on all cache blocks and $w_{i,j}$ to show the number of writes on a block at way j in set i. Using these, the coefficient of inter-set write variation (InterV) and coefficient of intra-set write variation (IntraV) can be defined [5] as

$$InterV = \frac{100}{W_{avg}} \sqrt{\frac{\sum_{i=1}^{N} \left(\sum_{j=1}^{M} w_{i,j}/M - W_{avg} \right)^2}{N-1}} \quad (1)$$

$$IntraV = \frac{100}{N \cdot W_{avg}} \sum_{i=1}^{N} \sqrt{\frac{\sum_{j=1}^{M} \left(w_{i,j} - \sum_{r=1}^{M} w_{i,r}/M \right)^2}{M-1}} \quad (2)$$

Thus, InterV measures the CoV (coefficient of variation) of the average write count within cache sets, and IntraV measures the average of the CoV of the write counts across a cache set [5]. Note that EqualChance does not require computation of IntraV or InterV. We use them only to characterize the write-variation present in workloads and as a figure of merit for evaluating effectiveness of EqualChance. We now discuss the main idea and algorithm of EqualChance.

3.1 Main Idea

Based on the temporal locality principle, cache management policies such as LRU replacement policy, aim to keep the hot data in the cache as long as possible. However, if a particular data item sees frequent write hits, the number of writes on that block will increase much more than those on the other blocks.

EqualChance works on the *key idea* that periodically, if the physical cache-block location of a write-intensive data item is changed, future writes will occur to another cache-block, and thus, the writes can be more uniformly distributed in a cache set. EqualChance records the number of writes on each set individually and uses this count to trigger a write-redirection operation for that set. By virtue of recording per-set counters, it performs more aggressive wear-leveling for the heavily written sets, which is especially advantageous for applications which have high inter-set write variation. Unlike previous techniques (e.g., [3,9]) we do not use compiler analysis or extra storage to detect write-intensive blocks. Instead, we assume that, probabilistically, the data-item being shifted on a write-hit is hot, since the most frequently accessed data-item is most likely to be chosen for redirection when the shifting is performed. This assumption has been confirmed by our experiments. For a 16-way cache, assuming that the age of the MRU-way is 0 and that of the LRU-way is 15 (and similarly for other ways), we record the average age of the data-item selected for shifting in all shifting-operations and then average it over all the workloads. This value is observed to be 0.29, which is very close to 0, the age of the MRU-way, which confirms that the MRU-way is most frequently chosen for shifting.

In a set-associative cache, any block in a set can be a candidate for write-redirection. A candidate block may store either an invalid, clean or dirty data item. We term I-shifting (resp. C-shifting) as the case when a write is redirected to an invalid (resp. clean) block. We do not redirect to another dirty block since the dirty data item itself may have write-intensive nature. I-shifting is preferred over C-shifting, since it incurs less overhead. Also, less recent (i.e. “older”) blocks are expected to have seen smaller number of cache accesses than the more recent blocks in the near-past execution window. Hence, for achieving wear-leveling, EqualChance searches for the least-recent invalid or clean block.

3.2 Working of EqualChance

For each cache set, we use a *numWrite* counter and a *FlagBit*. The *numWrite* counter is incremented when a write-access takes place in that set. When the counter reaches a pre-determined *shifting interval* (Υ), the *FlagBit* is turned ON and the counter is reset. When a write-access happens and the *FlagBit* is ON, Algorithm 1 is triggered which finally turns OFF the *FlagBit*. If no candidate for write-redirection is found (line 16-18) or *FlagBit* is OFF (line 21-23), a normal write is performed and LRU-age (i.e. cache replacement policy) information is also updated (line 24-26). In Algorithm 1 if there is no invalid (resp. clean) block in a set, a NULL value is returned in line 5 (resp. line 11). Further, on lines 9, 14 and 15, it is assumed that when data are shifted, corresponding tags are also shifted accordingly

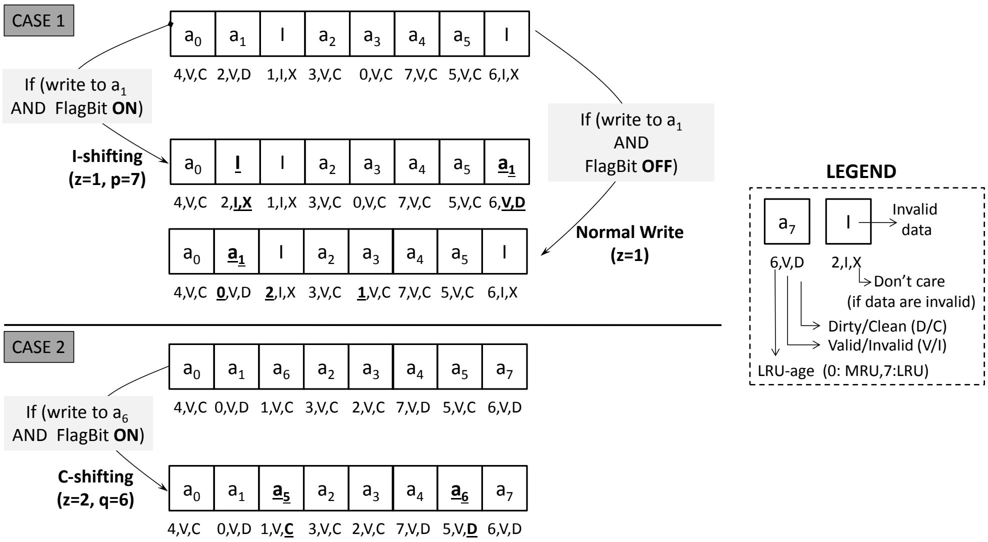
CASE 1
a0 a1 I a2 a3 a4 a5 I
4,V,C 2,V,D 1,I,X 3,V,C 0,V,C 7,V,C 5,V,C 6,I,X
If (write to a1 AND FlagBit ON)
I-shifting (z=1, p=7)
a0 I I a2 a3 a4 a5 a1
4,V,C 2,I,X 1,I,X 3,V,C 0,V,C 7,V,C 5,V,C 6,V,D
If (write to a1 AND FlagBit OFF)
Normal Write (z=1)
a0 a1 I a2 a3 a4 a5 I
4,V,C 0,V,D 2,I,X 3,V,C 1,V,C 7,V,C 5,V,C 6,I,X
CASE 2
a0 a1 a6 a2 a3 a4 a5 a7
4,V,C 0,V,D 1,V,C 3,V,C 2,V,C 7,V,D 5,V,C 6,V,D
If (write to a6 AND FlagBit ON)
C-shifting (z=2, q=6)
a0 a1 a5 a2 a3 a4 a6 a7
4,V,C 0,V,D 1,V,C 3,V,C 2,V,C 7,V,D 5,V,D 6,V,D
LEGEND
a7
I
Invalid data
6,V,D
2,I,X
Don't care (if data are invalid)
Dirty/Clean (D/C)
Valid/Invalid (V/I)
LRU-age (0: MRU,7:LRU)

Figure 1: Illustration of working of EqualChance algorithm. Updated items are highlighted as: **item**.

(but LRU-age information is not updated).

Figure 1 presents an illustration of the working of Algorithm 1. In case 1, invalid blocks exist in the cache (p = 2 and 7), and if the $FlagBit$ is ON, the least-recent one (p = 7 having LRU-age as 6) is selected and I-shifting is performed. If the $FlagBit$ is OFF, a normal write is performed where the block is written and LRU-age values are updated. In case 2, no invalid block exists and hence, the algorithm searches for the clean blocks. Of the four clean blocks at indices 0, 3, 4, 6, the one with index 6 is the least-recent in LRU-position and hence, it is selected for C-shifting (q = 6). If the set had no clean block, a normal write would be performed.

Algorithm 1: EqualChance: algorithm for handling a write-access in set-index i

```
/* Way-index refers to the physical-order,
   not the LRU-order                                  */
$z \leftarrow$ way-index of the write-hit block (found by tag matching)
isNormalWrite $\leftarrow$ FALSE
if $FlagBit[i]$ is ON then
    $p \leftarrow$ way-index of the least-recent invalid block in set $i$
    if $p \neq NULL$ then
        /* Perform I-shifting                         */
        Mark $Data[i][z]$ as invalid
        Write new data to $Data[i][p]$ and mark dirty (and valid)
    else
        $q \leftarrow$ way-index ($\neq z$) of the least-recent clean block in
        set $i$
        if $q \neq NULL$ then
            /* Perform C-shifting                     */
            Write $Data[i][q]$ to $Data[i][z]$ and mark clean
            Write new data to $Data[i][q]$ and mark dirty
        else
            isNormalWrite $\leftarrow$ TRUE
        end
    end
    Turn $FlagBit[i]$ OFF
else
    isNormalWrite $\leftarrow$ TRUE
end
if isNormalWrite then
    Write new data to $Data[i][z]$, mark dirty and update LRU-age
    information
end
```

4 Implementation and Overhead Assessment

Storage Overhead: We use Υ values less than 16 and thus, 4-bits are sufficient for the *numWrite* counter. We assume, C-shifting is performed using a centralized swap buffer [3] or a similar scheme. Due to the inter-set write variation and the preference for I-shifting, not all sets are expected to perform C-shifting simultaneously and hence, few swap-buffer entries are sufficient. Assuming 64 swap-buffer entries (64B each) and 5-bit storage for each set (*numWrite* counter + $FlagBit$), the percentage overhead of EqualChance (*Overhead*), compared to the L2 cache is

$$Overhead = \frac{(N \times 5) + (64 \times L)}{N \times M \times (L + G)} \times 100 \qquad (3)$$

As an example, for an 8MB, 16-way L2 cache, we find *Overhead* of EqualChance as merely 0.15% of L2 cache, which is very small.

Effect on Latency and Energy: We assume that in the normal case, checking for $FlagBit$ can be folded into the address decode tree of the cache and hence, the latency of normal case is not affected. Let L_w denote the cache write latency in cycles and its value is shown in Table 1. Then, we assume, I-shifting takes $2 + L_w + 1$ cycles (2 cycles for searching an invalid cache block, L_w cycles for writing the data and 1 cycle for invalidating the existing block and setting appropriate valid/dirty bits). Also, C-shifting takes $2+2+4+2L_w+1$ cycles (2 cycles each for searching an invalid block first and then a clean block, 4 cycles for transferring a 64B block on a 32B bus to and from the swap-buffer, L_w cycles for writing the clean data, L_w cycles for writing the newly-arrived data and 1 cycle for setting appropriate valid/dirty bits). Compared to the baseline case, C-shifting incurs an extra write, which we include in total number of L2 writes. In addition, we assume 0.5 nJ energy overhead of each C-shifting operation.

As confirmed by the results, due to the instruction-level parallelism present in modern processors, a small increase in the latency of LLC is easily hidden and by choosing a large Υ value, the overhead of shifting can be amortized over a large execution window (refer Section 6.2). In Section 6.2, we also evaluate EqualChance assuming higher shifting overhead (i.e. using ‘pessimistic’ estimates of overhead) and find that the overhead of EqualChance still remains small. For further optimization, the search for invalid/clean blocks can be overlapped with the tag-matching.

5 Experimental Methodology

Simulation Platform: We conduct simulation using interval core model in Sniper x86-64 simulator [25], which has been verified against real hardware. Processor frequency is 2GHz and both L1 I/D are 4-way 32KB caches with 2 cycle latency. L2 cache is a 16-way 4MB cache. L2 cache is inclusive of L1 and all caches use LRU, write-back, write-allocate policy. We find the parameters for ReRAM L2 cache using NVSim [10], assuming sequential cache access, 32nm CMOS process, 16-way set-associativity and write EDP (energy delay product) optimized cache design. These parameters are shown in Table 1. Main memory latency is 220 cycles. Peak memory bandwidth is 10 GB/s and queue contention is also modeled.

Workloads: All 29 benchmarks from SPEC CPU2006 suite with *ref* inputs and 5 benchmarks from HPC field (shown in italics in Table 2) are used as workloads. These workloads, along with their acronyms are shown in Table 2.

Table 1: Parameters for 16-way ReRAM L2 cache

	2MB	4MB	8MB
Hit Latency (ns)	5.06	5.12	5.90
Miss Latency (ns)	1.73	1.65	1.68
Write Latency (ns)	22.11	22.18	22.67
Hit energy (nJ)	0.542	0.537	0.602
Miss energy (nJ)	0.232	0.187	0.188
Write energy (nJ)	0.876	0.827	0.882
Leakage Power (W)	0.019	0.037	0.083

Table 2: Workloads used in the paper

As(astar), Bw(bwaves), Bz(bzip2), Cd(cactusADM)
Ca(calculix), Dl(dealII), Ga(gamess), Gc(gcc)
Gm(gemsFDTD), Gk(gobmk), Gr(gromacs)
H2(h264ref), Hm(hmmer), Lb(lbm), Ls(leslie3d)
Lq(libquantum), Mc(mcf), Mi(milc), Nd(namd)
Om(omnetpp), Pe(perlbench), Po(povray), Sj(sjeng)
So(soplex), Sp(sphinx), To(tonto), Wr(wrf)
Xa(xalancbmk), Ze(zeusmp), *Am(amg2013)*
Co(CoMD), *Lu(LULESH)*, *Mk(MCCK)*, *Ne(Nekbone)*

Evaluation Metrics: Our baseline is a ReRAM L2 cache which uses LRU replacement policy but does not use any mechanism for lifetime enhancement. We show the results on **a)** coefficient of inter-set write-variation (InterV, see Section 3), **b)** coefficient of intra-set write-variation (InterV), **c)** relative cache lifetime where the lifetime is defined as the inverse of maximum writes on any cache block. Further, we show **d)** relative performance [2], which is the ratio of IPC with EqualChance to IPC with baseline, **e)** percentage energy loss, **f)** total number of I-shifting and C-shifting operations which took place in EqualChance. We model energy of L2, main memory and algorithm execution. The values of leakage power and dynamic energy of main memory are taken as 0.18W and 70nJ/access, respectively [15] and the energy parameters for L2 are shown in Table 1. The energy overhead of algorithm execution includes 0.5 nJ for each C-shifting operation. We ignore the overhead of counters and buffers, since it is several orders of magnitude smaller compared to the memory subsystem (L2 + main memory). As an example, the dynamic energy of writing a 5-bit counter is 0.96pJ [26], which is 3 orders of magnitude smaller than the energy consumed in writing an L2 cache block (refer Table 1).

We fast-forward the benchmarks for 10B instructions and simulate each workload for 300M instructions. Across the workloads, relative lifetime and relative performance values are averaged using geometric

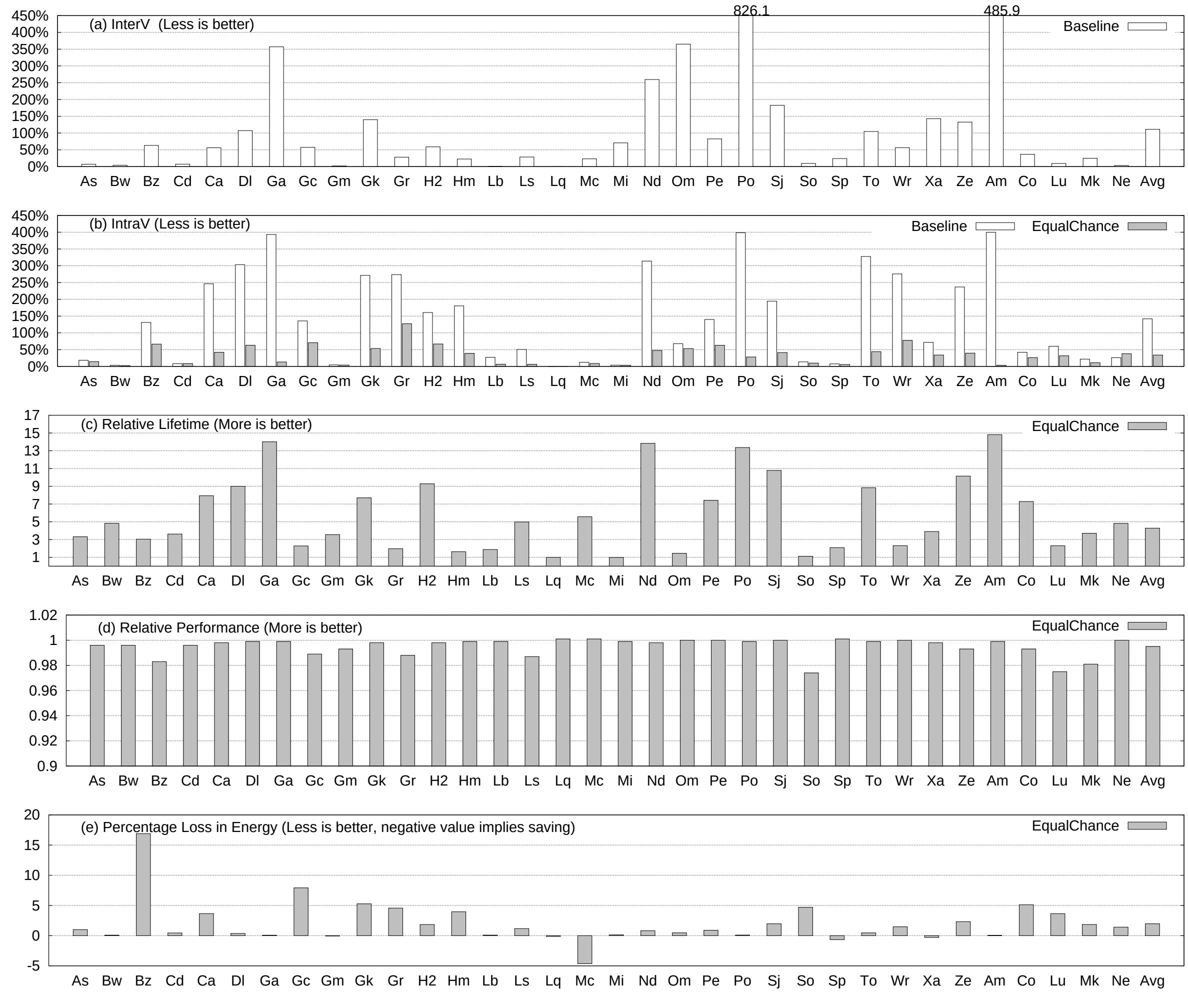

Figure 2: Results with EqualChance technique

mean and the remaining metrics are averaged using arithmetic mean, since they can be zero or negative. We have also computed absolute increase in MPKI (miss-per-kilo-instructions) and found its average value across workloads to be less than 0.04 for all configurations and hence, EqualChance does not increase off-chip accesses. For sake of brevity, we omit the results on MPKI.

6 Results and Analysis

6.1 Results with Default Parameters

Figure 2 shows the experimental results. We now analyze the results.

Results on InterV and IntraV: Firstly, from Figure 2(a) and 2(b), we note that for a *16-way* 4MB cache, average IntraV is larger than InterV. Note that for a fixed cache capacity, on increasing the associativity, InterV reduces and IntraV increases and vice versa (refer Table 3 and Section 6.2). With increasing number of cores, the associativity will also increase, since to avoid conflict misses, the cache associativity should be equal to at least the number of cores. This highlights the importance of an intra-set wear-leveling technique such as EqualChance in improving the lifetime of NVM caches. EqualChance reduces the average intra-set write variation from 141.8% to 33.8%. For several workloads, the IntraV in baseline cache is nearly 400%, for example, Ga (gamess), Po (povray) and Am (amg2013). Also, for several other workloads, the IntraV in baseline cache is more than 200%. This also underscores the necessity of using an intra-set wear-leveling technique.

Results on Lifetime Improvement: On average, EqualChance increases the cache lifetime by 4.29×. For several workloads, the lifetime is improved by more than 10×, for example, Ga, Nd (namd), Po, Sj (sjeng), Ze (zeusmp) and Am. Similarly, for several other workloads, the improvement in lifetime is more than 5×. The maximum increase possible in the cache lifetime depends on the variation present in the workload. Thus, the largest improvement in lifetime is obtained for workloads which show high write-variation, e.g., Ga, Po, Sj, Nd etc. Conversely, for workloads such as Lq (libquantum) and Mi (milc), the intra-set write variation in baseline is close to zero, and hence, the scope of improvement in lifetime is negligible. Cache lifetime improvement for these workloads can be achieved using other techniques such as cache bypassing, use of write-buffer, etc.

Results on Performance: The relative performance is 1.00× and thus, EqualChance has negligible effect on performance. This is due to the fact that EqualChance uses in-cache data-movement and unlike previous data-invalidation based techniques (e.g. [5,22]), EqualChance does not flush the data to achieve wear-leveling. Thus, EqualChance does not increase DRAM traffic significantly or cause queue-contention, or write-endurance issues at main memory. The largest loss in performance is seen for workloads such as So (soplex) where relative performance is 0.97×. This is because, soplex uses L2 cache intensely.

Results on Energy: The average loss in energy on using EqualChance is 1.97%. Thus, the effect of EqualChance on energy efficiency is small. Since, in general, the higher density and lower leakage power of NVMs help in achieving high performance and energy efficiency, a small loss in these parameters may be acceptable for addressing the crucial limitation of NVMs, viz. their limited write endurance. As we show in Section 6.2, by changing the shifting-interval (Υ), a designer can exercise trade-off between performance/energy loss and improvement in lifetime. Further, power density minimization comes as a side-benefit of wear-leveling [27], which may reduce the chip-temperature, leakage power and the cooling requirement, thus offsetting a small energy loss incurred by EqualChance.

For some workloads, such as Mc (mcf), Sp (sphinx), Xa (xalan) etc., EqualChance leads to small saving in energy. This can be attributed to the fact that EqualChance alters the cache replacement policy and periodically shifts some blocks from the more recent position to the less recent position in the LRU stack. Since a large fraction of blocks in mcf etc. are used only once (i.e. dead-on-arrival) [28], demoting them in LRU stack enables their faster eviction, which also improves the hit-rate.

Results on Shifting Operations: Table 3 shows the number of I and C-shifting operations on using EqualChance. On average, 199K I-shifting and 96K C-shifting operations take place. The number of these operations depends on the interaction of several factors such as cache-usage intensity, write-intensity and write-variation present in cache access behavior of a workload. The higher the write-intensity, the more frequently EqualChance algorithm is executed, but if most blocks store dirty data-item, then a candidate for shifting may not be found. Also, the higher the cache-usage intensity, the more will be the number of valid blocks in the cache and hence, the more will be the number of C-shifting operations. Similarly, if the write-variation is high, most writes are directed to only few blocks and the remaining blocks remain invalid or store clean data-item and hence, candidates for I or C-shifting can be easily found.

Table 3: Number of I-and C-shifting operations

	I-shifting	C-shifting		I-shifting	C-shifting
As	120K	143K	Mi	298K	50K
Bw	1K	61K	Nd	76K	0
Bz	271K	0	Om	4K	2K
Cd	344	86K	Pe	11K	571
Ca	261K	0	Po	107K	0
Dl	63K	0	Sj	45K	0
Ga	133K	0	So	737K	577K
Gc	97K	35K	Sp	361	58K
Gm	217K	517K	To	120K	0
Gk	131K	0	Wr	33K	0
Gr	156K	6K	Xa	34K	66K
H2	47K	0	Ze	625K	0
Hm	176K	0	Am	82K	0
Lb	2580K	1K	Co	71K	33
Ls	51K	595K	Lu	134K	239K
Lq	0	201K	Mk	21K	251K
Mc	38K	361K	Ne	8K	0

6.2 Parameter Sensitivity Results

We now study the sensitivity of EqualChance for different parameters. Except the parameter mentioned, all other parameters have default values as shown in Section 5. The energy/latency value for other cache configurations were computed as shown in Section 5 and are omitted for brevity. The results are summarized in Table 4.

Changing Shifting Interval (Υ): Shifting interval decides the number of writes to a cache-set after which an I- or C-shifting is performed. On increasing the shifting interval, EqualChance algorithm is executed less frequently, which is reflected in the reduction in number of I- and C-shifting operations. Thus, increasing Υ reduces the loss in performance and energy, although it also reduces the enhancement in lifetime, since the wear-leveling is performed less frequently. Note that even with $\Upsilon = 15$, the improvement in lifetime is significant. Thus, the value of Υ can be chosen to exercise a trade-off between desired lifetime enhancement and acceptable loss in performance and energy. Also notice that on increasing Υ from 5 (default) to 10, the total number of shifting

Table 4: Parameter Sensitivity Results for EqualChance. Default values: Υ = 5, overhead values shown in Section 4, 16-way, 4MB L2 cache (Perf. = performance).

	% InterV	% IntraV		Relative	Relative	% Loss in	I-shifting	C-shifting
	Baseline	Baseline	EqualChance	Lifetime	Perf.	Energy	operations	operations
Default	111.1	141.8	33.8	4.29	1.00	1.97	199K	96K
Υ=10	111.1	141.8	41.5	3.79	1.00	1.11	78K	69K
Υ=15	111.1	141.8	44.9	3.60	1.00	0.75	48K	48K
Higher overhead	111.1	141.8	33.8	4.29	0.99	2.01	199K	96K
8-way L2	154.2	111.7	28.8	3.09	0.99	1.55	148K	144K
32-way L2	74.0	171.6	36.3	5.02	1.00	3.29	251K	44K
2MB L2	71.9	108.6	25.9	3.38	0.99	2.96	157K	139K
8MB L2	152.3	181.5	45.1	6.20	0.99	2.74	239K	53K

operations are reduced by nearly half, which is expected.

Higher Shifting Overhead: To evaluate the effect of shifting overhead, we assume 3 and 9 cycles extra latency overhead of each I-shifting and C-shifting operation (respectively) over and above that mentioned in Section 4 and 0.5nJ additional energy overhead of C-shifting. The results in Table 4 show that the energy efficiency and performance are minimally affected and hence, EqualChance is not very sensitive to the shifting overhead. This can be easily understood by noting that shifting happens relatively infrequently and a small increase in LLC latency is easily hidden by the instruction-level parallelism.

Changing Associativity (*M*): For a fixed cache capacity, reducing the associativity increases the inter-set write variation but reduces the intra-set write variation which is confirmed from the InterV and the IntraV values for baseline caches of different associativity values in Table 4. This can be easily understood by considering the two extreme cases, viz. a fully-associative cache and a direct-mapped cache. A fully-associative cache has only one set and hence, its InterV will be zero and similarly, a direct-mapped cache has only a single way and hence its IntraV will be zero. From the table, we conclude that EqualChance works well for caches of all associativity values and provides lifetime improvement in proportion to the amount of write-variation present in the original workload. Specifically, it provides large improvement for the 32-way set associative caches. Comparing with the default case, the total number of shifting operations remain nearly the same, which is expected. Further, for large associativity, invalid blocks in a set can be more easily found and since I-shifting is preferred over C-shifting, the number of I-shifting operations increase with increasing associativity and vice versa.

Changing Cache size: We experiment with both double-sized and half-sized caches. The results in Table 4 show that on increasing the cache size, the average improvement in lifetime is also increased and vice-versa. This can be explained by noting that since applications have a fixed working set size (i.e. unique number of cache lines accessed in a period of time), on increasing the cache size, the hit-rate is also increased. Hence, only few blocks are repeatedly accessed which increases the intra-set write variation and hence, the scope for improving the cache lifetime is also increased. Opposite is seen on decreasing the cache size. From the results, it is clear that for both the cache sizes, EqualChance provides significant reduction in the intra-set write variation and improvement in cache lifetime. For large cache size, larger number of invalid blocks can be found and hence, the number of I-shifting operations are also higher in the 8MB cache and opposite is seen in 2MB cache.

For all parameters, the relative performance is equal to or greater than 0.99× and the loss in energy is less than 3.30%. This confirms that EqualChance works well for a wide-range of system and algorithm parameters.

7 Conclusion

The limited write endurance of NVMs is a key bottleneck preventing their use in designing on-chip caches. In this paper, we presented EqualChance, a technique for mitigating intra-set write variation to improve lifetime of NVM caches. Experimental results have shown that EqualChance is effective in increasing the cache lifetime for NVM caches and works well for a wide range of parameters. Our future efforts will focus on synergistically integrating EqualChance with techniques for write minimization and inter-set write variation mitigation to improve the cache lifetime even further. We also plan to compare our technique with other techniques for addressing intra-set write-variation (e.g. [5, 6]) to fully evaluate and highlight the effectiveness of our technique. Finally, we are currently evaluating our technique with multicore system configurations to study how it scales with increasing number of cores.

Acknowledgment

Support for this work was provided by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy.

References

[1] S. Rusu, H. Muljono, D. Ayers, S. Tam, W. Chen, A. Martin, S. Li, S. Vora, R. Varada, and E. Wang, "Ivytown: A 22nm 15-core enterprise Xeon® processor family," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 102–103.

[2] S. Mittal, Y. Cao, and Z. Zhang, "MASTER: A Multicore Cache Energy Saving Technique using Dynamic Cache Reconfiguration," *IEEE Transactions on VLSI Systems*, vol. 22, pp. 1653 – 1665, 2014.

[3] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, 2009, pp. 34–45.

[4] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," in *49th Annual Design Automation Conference*, 2012, pp. 243–252.

[5] 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 *International Symposium on High-Performance Computer Architecture (HPCA)*, 2013, pp. 234–245.

[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 Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[7] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.

[8] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin *et al.*, "A 4Mb embedded SLC Resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011, pp. 200–202.

[9] Y. Li, Y. Chen, and A. K. Jones, "A software approach for combating asymmetries of non-volatile memories," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012, pp. 191–196.

[10] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[11] 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 *et al.*, "Bi-layered RRAM with unlimited endurance and extremely uniform switching," in *Symposium on VLSI Technology (VLSIT)*. IEEE, 2011, pp. 52–53.

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

[13] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *International Symposium on Microarchitecture (MICRO)*, 2009, pp. 2–13.

[14] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. Prete, "Way adaptable D-NUCA caches," *International Journal of High Performance Systems Architecture*, vol. 2, no. 3, pp. 215–228, 2010.

[15] S. Mittal, Z. Zhang, and J. Vetter, "FlexiWay: A Cache Energy Saving Technique Using Fine-grained Cache Reconfiguration," in *31st IEEE International Conference on Computer Design (ICCD)*. Asheville, NC, USA: IEEE, 2013.

[16] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das, "Architecting on-chip

interconnects for stacked 3D STT-RAM caches in CMPs," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, 2011, pp. 69–80.

[17] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *IEEE/ACM International Conference o Computer-Aided Design-Digest of Technical Papers (ICCAD)*, 2009, pp. 264–268.

[18] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy-and endurance-aware design of phase change memory caches," in *Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 136–141.

[19] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *IEEE 15th International Symposium on High Performance Computer Architecture (HPCA)*, 2009, pp. 239–249.

[20] J. Ahn and K. Choi, "Lower-bits cache for low power STT-RAM caches," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 480–483.

[21] S. Mittal, "Using cache-coloring to mitigate interset write variation in non-volatile caches," Iowa State University, Tech. Rep., 2013.

[22] Y. Chen, W.-F. Wong, H. Li, C.-K. Koh, Y. Zhang, and W. Wen, "On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 16:1–16:22, May 2013.

[23] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2011, pp. 19–28.

[24] 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 *International Symposium on Microarchitecture (MICRO)*, 2009, pp. 14–23.

[25] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011.

[26] J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir, "Exploiting program hotspots and code sequentiality for instruction cache leakage management," in *International Symposium on Low-Power Electronics and Design (ISLPED)*, 2003, pp. 402–407.

[27] J. C. Ku, S. Ozdemir, G. Memik, and Y. Ismail, "Thermal management of on-chip caches through power density minimization," in *International Symposium on Microarchitecture (MICRO)*, 2005, pp. 283–293.

[28] M. Chaudhuri, "Pseudo-LIFO: the foundation of a new family of replacement policies for last-level caches," in *International Symposium on Microarchitecture (MICRO)*, 2009, pp. 401–412.