EqualWrites: Reducing Intra-set Write Variations for Enhancing Lifetime of Non-Volatile Caches

Sparsh Mittal, *Member, IEEE*, and Jeffrey S. Vetter, *Senior Member, IEEE*

***Abstract*— Driven by the trends of increasing core-count and bandwidth-wall problem, the size of last level caches has greatly increased, and hence the researchers have explored non-volatile memories (NVMs) that provide high density and consume low-leakage power. Since NVMs have low write endurance and the existing cache management policies are write variation (WV) unaware, effective wear-leveling techniques (WLTs) are required for achieving reasonable cache lifetimes using NVMs. We present EqualWrites, a technique for mitigating intra-set WV. Our technique works by recording the number of writes on a block and changing the cache-block location of a hot data item to redirect the future writes to a cold block to achieve wear leveling. Simulation experiments have been performed using an x86–64 simulator and benchmarks from SPEC06 and high-performance computing field. The results show that for single-, dual-, and quad-core system configurations, EqualWrites improves cache lifetime by 6.31×, 8.74×, and 10.54×, respectively. In addition, its implementation overhead is very small and it provides larger improvement in lifetime than three other intra-set WLTs and a cache replacement policy.**

***Index Terms*— Cache memory, device lifetime, intra-set write variation (WV), non-volatile memory (NVM or NVRAM), wear leveling.**

I. Introduction

RECENT years have witnessed remarkable growth in research on the use of non-volatile memory (NVM) devices, such as spin transfer torque RAM (STT-RAM), resistive RAM (ReRAM), and phase change memory (PCM) for designing on-chip caches [1], [2]. Compared with static RAM (SRAM), these NVMs are expected to provide much higher density and consume very low-leakage power. In addition, by virtue of their non-volatile operation, these devices do not require refresh operations for maintaining data integrity, which is a limitation in the use of embedded dynamic RAM (eDRAM) caches [3], [4]. A crucial limitation of NVMs, however, is that their write endurance is significantly smaller than that of DRAM and SRAM. For example, the write endurance of ReRAM and PCM are only 10^{11} and 10^{8}, respectively [4]–[6]. For STT-RAM, although a write-endurance value of greater than 10^{15} has been projected, the actual prototypes have demonstrated a write-endurance value of only up to 4×10^{12} [7]–[10]. Due to process variations, these values may be further reduced by an order of magnitude [11]. In contrast, the write endurance of SRAM and DRAM is above 10^{15} [4], [6].

Further, traditional cache management policies, which have been designed in the context of SRAM, aim to optimize performance and energy efficiency and do not consider the limited write endurance of the device. For example, cache replacement policies [e.g., least recently used (LRU)] aim to exploit temporal locality by increasing the number of hits to the existing cache blocks. This, however, increases the number of writes to a few cache blocks, which may cause those blocks to fail much earlier than the anticipated cache lifetime, which assumes a uniform write distribution. Thus, the limited write endurance of NVMs along with the write variation (WV) introduced by cache management policies necessitates the use of effective wear-leveling techniques (WLTs).

A. Contributions

To address this issue, in this paper, we present EqualWrites, a technique for improving cache lifetime by mitigating intra-set WV. EqualWrites works on the key idea that if the difference between the number of writes to two blocks in a cache set is larger than a threshold (say Ω), it indicates large intra-set WV and to mitigate it, the data item in these blocks can be swapped so that future writes can be redirected from a hot block to a cold block to achieve wear leveling (Section III). This leads to improvement in cache lifetime. Note that in this paper, we refer to a hot or cold block based on the frequency of writes (and not reads) to it. Modern processors use caches of large associativity, for example, both AMD's Opteron processor and Intel's Itanium 9500 series of processors have 32-way last level cache (LLC) [12], [13]. For caches of such large associativity, an intra-set WLT, such as EqualWrites, is highly useful.

EqualWrites does not require offline profiling or modification of program binary (unlike [14]) or including set-index bits as part of the tag (unlike [15]) or any floating-point hardware for computation of coefficient of WV (unlike [16]). Further, it can be easily extended to further improve lifetime by tolerating errors and mitigate security threats to NVM caches (Section VII). In this paper, we assume an ReRAM L2 cache, although our technique can be easily applied to a cache designed with other NVMs. For the sake of convenience, in the remainder of this paper, we use the term NVM and ReRAM interchangeably.

B. Implementation and Evaluation

The storage requirement of EqualWrites is less than 0.8% of the L2 cache storage size and thus its overhead is small (Section IV). We conduct microarchitectural simulations using

Manuscript received August 26, 2014; revised November 23, 2014; accepted December 31, 2014. Date of publication January 29, 2015; date of current version December 24, 2015. This work was supported by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy.

The authors are with the Oak Ridge National Laboratory, Future Technologies Group, Oak Ridge, TN 37831 USA (e-mail: sparsh0mittal@gmail.com; vetter@computer.org).

Digital Object Identifier 10.1109/TVLSI.2015.2389113

a state-of-the-art x86-64 simulator and benchmarks from SPEC CPU2006 suite and high-performance computing (HPC) field (Section V). In addition, we compare EqualWrites with three different intra-set WLTs, viz., probabilistic line flush (PoLF) [6], its variant named probabilistic line swap (PoLSwap) and WriteSmoothing (WrSm) [16] (see Section V-C for more details of these techniques). We also perform simulations by changing cache replacement policy from LRU to random to study whether randomizing replacement decisions alone can achieve wear leveling. Results have shown that EqualWrites provides the largest improvement in cache lifetime at much smaller performance and energy overhead than other approaches (Section VI).

C. Summary of Results

For single-core system, EqualWrites improves cache lifetime by 6.31× with a relative performance and an energy loss of 0.99× and 0.74%, respectively. In comparison, PoLF (which, in turn, provides larger lifetime improvement than PoLSwap, WrSm, and the use of random replacement policy) improves cache lifetime by 5.18×, with a relative performance and an energy loss of 0.97× and 13.01%, respectively. For dual-core system, EqualWrites improves cache lifetime by 8.74× with a relative performance and an energy loss of 0.99× and 0.76%, respectively. By comparison, PoLF improves cache lifetime by 8.12× with a relative performance and an energy loss of 0.97× and 14.62%, respectively. For quad-core system, EqualWrites improves lifetime by 10.54× with a relative performance and an energy loss of 0.99× and 0.94%, respectively. In comparison, PoLF improves lifetime by 10.23× with a relative performance and an energy loss of 0.97× and 23.16%, respectively. Additional experiments show that EqualWrites works well for a wide range of system and algorithm parameters (Section VI-C).

II. Background and Related Work

A. Brief Background on NVMs

Since SRAM provides high performance and write endurance, it has been conventionally used for designing on-chip caches. However, SRAM also has low density and high leakage power consumption, and hence, caches designed with SRAM consume a significant fraction of processor power. For example, in both Niagara and Niagara-2 processors, L2 cache consumes nearly 24% of the total power consumption [17]. With increasing system core count, the size of on-chip caches has also grown and modern multicore processors employ tens of megabytes of LLC. For instance, Intel Itanium 9560 processor uses 32-MB LLCs [18]. Although, it is possible to address the leakage power consumption of SRAM at architecture level [19], the requirements of energy-efficiency imposed on future extreme-scale computing systems demand a fundamental redesign of on-chip caches.

These trends have motivated the researchers to use NVM devices. NVMs rely on change in the physical state of matter to store the information [1], [20]. By virtue of this, NVMs have several features, such as high density, good scalability, and low leakage power consumption; and also limitations, such as high write energy and latency, and low write endurance. Several previous works have shown that the near-zero leakage power consumption of NVMs can compensate for the higher write-energy/latency and thus, NVM caches can achieve higher energy efficiency than the SRAM caches [1], [21]. Thus, limited write endurance of NVMs remains a crucial bottleneck in their use as on-chip caches, and in this paper, we propose a technique to address this.

Due to its very small write endurance and high latency, PCM is considered less suitable for designing on-chip caches. However, some researchers have proposed SRAM-PCM hybrid caches [2], [22], while others have proposed using PCM as an L4 cache [2], just as it is currently used as main memory. Since intra-set WV can be detrimental in these caches also, EqualWrites can be useful for these PCM caches.

B. Addressing Write Endurance Issue in NVMs

Write-endurance issue can be addressed by either or both of the write-minimization techniques (WMTs) or the WLTs. We first briefly discuss the techniques proposed for main memory, and then discuss the techniques proposed for caches in detail.

1) Techniques for NVM Main Memory: Cho and Lee [23] propose a WMT named Flip-N-Write for non-volatile main memory. Their technique provisions writing a bit only if it is different from the one originally written. Further, if storing the flipped value of data requires less number of bit-write operations, their technique stores the data in flipped form and signals (remembers) this using an additional bit. Zhou *et al*. [20] propose a WLT for main memories that works by periodically swapping memory segments of high and low write accesses. Writes to caches show both inter-set and intra-set variation, while those to main memory show only inter-set variation and hence, the WLTs proposed for main memory cannot be applied to address intra-set WV in caches.

2) Techniques for NVM Caches: Several researchers have proposed WMTs for caches that use write coalescing buffers [24], additional levels of caches [25] or read-before-write scheme to avoid redundant writes [26]–[28]. These approaches are orthogonal and complementary to our technique and, hence, can be synergistically integrated with it.

a) Classification of cache WLTs: Based on their granularity, the WLTs can be classified as inter-color [29], inter-set [6], [15], intra-set [6], [16], [30]–[32], and memory cell level [27], [28], [33]. We propose an intra-set WLT and compare our technique to other intra-set WLTs (see Sections V and VI for more details). In addition, our technique can be combined with inter-set/inter-color WLTs for further improving the lifetime of caches.

The WLTs can also be divided as whether they use data-invalidation (also called flushing) [6], [15], [29], [30] or in-cache data movement (also called data migration or shifting) (see [16] and PoLSwap shown in Section V). Data invalidation increases off-chip accesses, leading to contention and endurance issues in main memory. Our technique uses in-cache data movement and thus incurs smaller overhead. The limitation of in-cache data movement is that it may require additional hardware such as swap buffer.

b) Discussion of a few cache WLTs: We discuss a few intra-set WLTs in Section V-C. Here, we discuss some of the inter-set WLTs. The inter-set WLT proposed in [15] remaps all set indices after a certain length of time. For this, a register, called remap register is used which is XORed with the set-index bit of the cache address. By periodically changing this register, randomization is introduced, which helps in uniformly distributing writes to different sets. The inter-set WLT proposed in [6] aims to distribute the write traffic evenly to different cache sets by shifting the mapping of cache physical sets. Since shifting all cache sets at once would incur a large overhead, their technique swaps only two sets at a time and after multiple swaps, all the cache sets are automatically shifted.

III. System Architecture

A. Notations

Let N denote the number of cores. Let S, A, B, and T denote the number of cache sets, associativity, block size, and tag size, respectively. In this paper, we assume, $B = 64$ B and $T = 40$ bits. In addition, let $w_{i,j}$ denotes the number of writes on any block at set i and way index j. Further, let W_{avg} denotes the average number of writes on all the blocks. Then, the coefficient of intra-set WV (IntraV) [6] is defined as follows:

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

Note that EqualWrites does not require computing the value of IntraV. We use it only as a figure of merit for evaluating the effectiveness of EqualWrites. In what follows, we use the terms shifting and redirection synonymously.

B. Key Idea

EqualWrites works on the key idea that if the difference between the number of writes to two blocks in a cache set is larger than a threshold (say Ω), it indicates the presence of large intra-set WV. In a set-associative cache, a data item mapped to a set can be placed in any of its ways. Using this fact, we can reduce the intra-set WV by swapping the data item in those blocks. Thus, future writes can be redirected from a write-intensive block to a cold block. This leads to uniform distribution of writes, which improves the cache lifetime.

C. Algorithm Description

We use counters for recording the number of writes on each block in the current generation, where the pth generation begins immediately after the pth miss to that cache block, when a new data item is brought into that block [34]. We do not record global write history, since this would require very large number of bits in each counter and, hence, would incur high overhead. As shown next, updating of the counters is done in an intelligent manner to record the number of writes to a block *relative to other blocks in the same set*, while minimizing the storage requirement of the counters.

Algorithm 1 Algorithm for Handling a Write Hit in Set i

```
Let s be the way-index of the write-hit block (found by tag matching)
if Counter[i][s] ≠ Ω − 1 then
    Write new data to Data[i][s], mark dirty, update LRU-stack
    Counter[i][s] = Counter[i][s] +1
else
    Let target = NULL                          ▷ Target for write-redirection
    for all way-locations q (≠ s) do
        if Counter[i][q]==0 then
            target = q
            Break from for loop
        end if
    end for
    if target ≠ NULL then          ▷ Candidate for write-redirection exists
        if Data[i][target] is invalid then                          ▷ Case-I
            Write new-data to Data[i][target] and mark valid and dirty
            Mark Data[i][s] as invalid
        else                ▷ Case-V: Target is valid (either clean or dirty)
            Write Data[i][target] to Data[i][s] and mark clean or dirty
appropriately
            Write new-data to Data[i][target] and mark dirty
        end if
        Counter[i][s] = Counter[i][target] = Ω/2
    else
        for all way-locations q (≠ s) do ▷ Note that overflow cannot occur
            Counter[i][q]= Counter[i][q]-1
        end for
        Write new data to Data[i][s], mark dirty, update LRU-stack
    end if
end if
```

Let Ω denote the maximum number of writes that can differ between two blocks in a set, after which a write-redirection operation is triggered. To avoid dealing with negative numbers and also provide more intuitive explanation of the algorithm, we provision that the counters are initialized with $\Omega/2$. Thus, the value of counters varies from 0 to $\Omega - 1$, and $\lceil \log_2 \Omega \rceil$ bits are sufficient to store each counter. A counter is initialized at the beginning of a new generation to $\Omega/2$.

Algorithm 1 shows the logic for handling a write hit and updating the counters. The algorithm works as follows. On a write hit, the value of write counter of a block is compared with the threshold. If the threshold is not reached, the counter of the block is simply incremented. If the threshold is reached, a target for write redirection is searched from the other ways in the same set. For such a target, write-counter value is equal to zero, indicating a difference of Ω from the block which has seen a write hit. Depending on the state of the candidate, either a single write or write with a data transfer is performed. When the target for write redirection is an invalid block, only a single write is required. This case is called Case-I. On the other hand, when the target is a valid block (either clean or dirty), the data of the block need to be swapped with that of the source block. This is referred to as Case-V.

On a write redirection, the value of both the source and destination blocks are set to $\Omega/2$. If no candidate for write redirection exists, it implies that the difference in writes among different blocks is less than the threshold. In such a case, the write-counter value of all blocks (except one seeing write hit) is decreased by one. Fig. 1 shows the updating of counter values assuming an example cache access sequence.

Fig. 2 shows the cache structure with EqualWrites. The tunable parameter of EqualWrites is Ω, which is stored in

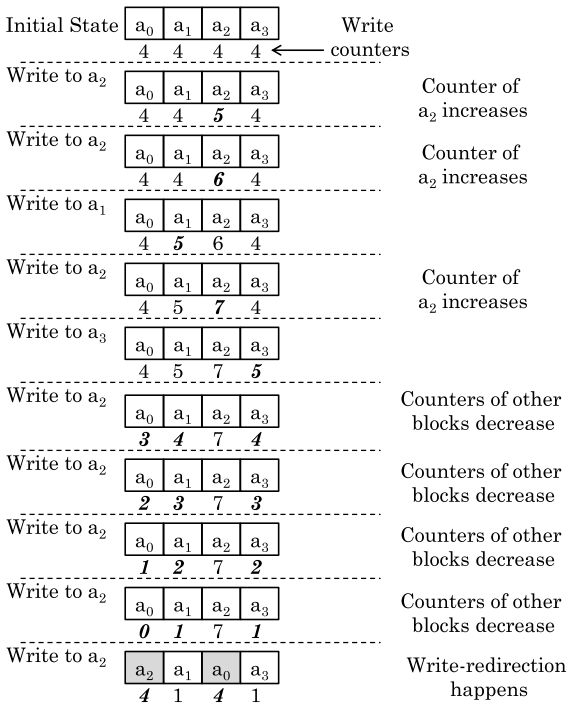

Fig. 1. Example of the update of counters in EqualWrites technique for $\Omega = 8$ for an arbitrary write sequence. The counters that are changed are highlighted as ***X***.

a register. The comparison operation can be implemented using XOR and redirection is performed using data-movement instructions and swap buffer.

IV. Implementation and Overhead Assessment

In this section, we discuss the hardware implementation and overhead of EqualWrites.

A. Storage and Area Overhead

We assume that swapping is achieved using a centralized swap buffer [2] or a similar scheme. The swap buffer is designed using SRAM. Due to the presence of inter-set variation and possibility of also using Case-I, all the sets are not expected to perform Case-V simultaneously and hence, only few swap-buffer entries are sufficient. In this paper, we assume 64 swap-buffer entries, each of which is 64 B in size. Thus, the storage overhead of EqualWrites includes the overhead of write counters and swap buffer. Using this, we compute storage overhead of EqualWrites relative to L2 cache (Θ) using an approach similar to previous works [1], [16]. We obtain

$$\Theta = \frac{(\log_2(\Omega) \times S \times A) + 64 \times 64 \times 8}{S \times A(B+T)} \times 100. \quad (2)$$

As an example, for $\Omega = 16$, we obtain $\Theta = 0.7\%$ and thus, the overhead of EqualWrites is very small. Access to write counter does not lie on critical path and hence, they can be optimized for low area and low leakage. Assuming that they are implemented using a technology $1/2\times$ to $1/3\times$ dense than ReRAM, we get the area overhead of EqualWrites as nearly 2% of the L2 cache, which is small. The counters themselves are not stored in NVM and hence, they do not have write-endurance issue.

TABLE I
Parameters for a 16-Way ReRAM L2 Cache

	2MB	4MB	8MB	16MB	32MB
Hit latency (ns)	4.33	4.13	4.74	6.21	5.36
Miss latency (ns)	1.47	1.44	1.55	1.81	1.93
Write latency (ns)	21.72	21.55	21.87	23.09	21.91
Hit Energy (nJ)	0.524	0.547	0.646	0.679	0.826
Miss Energy (nJ)	0.204	0.188	0.194	0.200	0.207
Write Energy (nJ)	0.834	0.851	0.925	0.967	1.114
Leakage Power (W)	0.204	0.325	0.785	1.118	2.377

B. Latency Overhead

We estimate the latency overhead of our technique following a procedure similar to that of [2] and [16]. We use L_w to show the cache write latency and its value is shown in Table I. Then, we assume that checking the counter takes one cycle, incrementing the counter takes another cycle, and when the remaining counters are changed, additional 3 cycles are taken. In addition, Case-I takes $L_w + 2$ cycles (L_w cycles for writing the data and 2 cycle for setting appropriate valid/dirty bits and counters) and Case-V takes $4 + 2L_w + 2$ cycles (4 cycles for transferring a 64-B block on a 32-B bus to and from the swap buffer, $2L_w$ cycles for writing both blocks, and 2 cycles for setting appropriate bits and counters). Case-V shifting incurs an additional write compared with the normal case, which is included in total number of L2 writes. We have incorporated these latency overheads in our performance simulator in Section V. The results presented in Section VI show that EqualWrites provides nearly $0.99\times$ performance compared with the baseline. This confirms that the latency overhead of EqualWrites is small.

C. Energy Overhead

We first discuss the dynamic energy overhead of counters. From [35], we note that a 5-bit counter consumes 0.96 pJ in each access. To cross check, from [34], we note that a 2-bit counter consumes 0.1 pJ in each access, thus a 5-bit counter would consume 0.25 pJ, which is in the same ballpark. We take the case of single-core and assume each write counter is 5-bits (since $\Omega < 32$). Since a write counter is incremented on a write, we compare the energy of writing a single cache block to that of incrementing a write counter. For this, we assume an ReRAM L2 cache and obtain its parameters using NVSim [36]. We assume sequential cache access, 32-nm CMOS process, 16-way set-associativity, and write energy delay product optimized cache design. In addition, we assume H-tree routing, internal sensing, 350 K temperature, and area-optimized buffer design. The output parameters are shown in Table I.

From Table I, we note that for a 4-MB cache, write energy is 0.85 nJ. In comparison with it, 0.96 pJ incurred in accessing a write counter is three orders of magnitude smaller. To minimize the dynamic energy of counters even further, gray coding can be used so that only one bit changes when a counter is incremented.

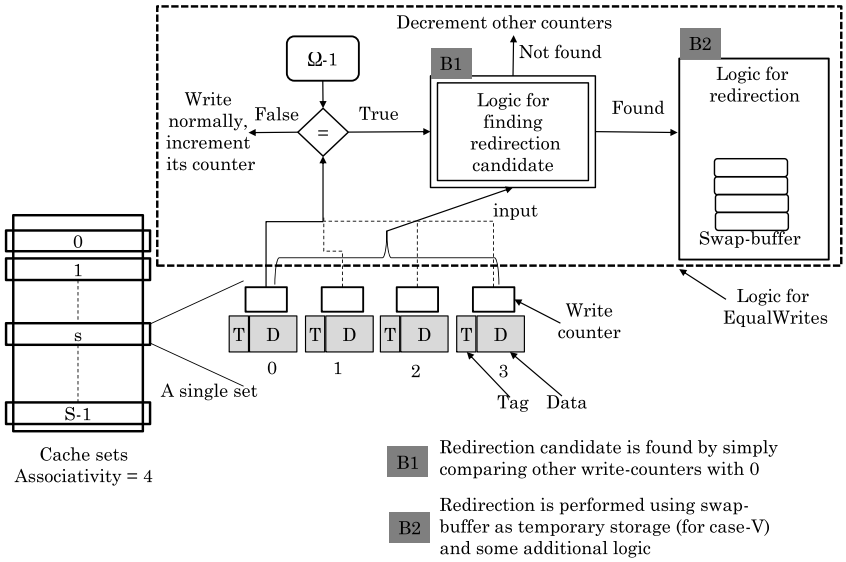

Fig. 2. Cache diagram with extensions for EqualWrites (assuming that the write hit happens to way 0).

Further, redirections happen infrequently and swap buffer is accessed only in Case-V and thus, it is accessed even less frequently than the counters. Thus, the dynamic energy of swap buffer is small. Using a similar analysis, the leakage energy of counters and swap buffer can be shown to be negligible. On including the energy of main memory and read energy of cache, the fractional contribution of counters/swap buffer in total energy becomes even smaller. Thus, the energy consumed by counters and swap buffer is orders of magnitude smaller than that of (L2 cache + main memory) and hence, it is ignored. To account for the energy consumed in data movement, we assume 0.5 nJ energy overhead for each Case-V shifting [32].

D. Discussion

It is noteworthy that wear leveling provided by EqualWrites has the additional benefit of providing thermal density minimization [37], which may reduce the temperature of the chip and reduce the energy consumption. On using EqualWrites, read operations are not affected, Case-I is almost same as regular write and Case-V adds only one extra write. A small increase in latency of LLC is easily hidden by techniques such as out-of-order execution and write buffers, and thus, EqualWrites has only minimal effect on application performance, as confirmed by the results. In addition, by choosing a suitable value of Ω, the shifting overhead can be amortized over a large instruction window. In Section VI-C, we also evaluate our technique assuming $2\times$ and $4\times$ overhead compared with the above mentioned latency and energy overhead and observe that the performance and energy loss of EqualWrites still remains small. From this, we conclude that the complexity of our technique is small.

V. Experimental Methodology

A. Simulation Infrastructure

We use interval-core model in Sniper x86-64 multicore simulator. The frequency of processor is 2 GHz. L1 I/D caches are 32-kB four-way LRU caches and are private to each core. L2 cache is shared among cores and its parameters are shown in Table I. Main memory latency is 220 cycles and peak memory bandwidth is 10 GB/s for single-core system, 15 GB/s for dual-core system, and 25 GB/s for quad-core system. Queue contention is also modeled.

TABLE II
Workloads Used in This Paper

Single-core workloads and their acronyms
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),
Po(povray), Pe(perlbench), Sj(sjeng), So(soplex), Sp(sphinx), To(tonto)
Wr(wrf), Xa(xalancbmk), Ze(zeusmp), *Am(amg2013), Co(CoMD)*
Lu(LULESH), Mk(MCCK), Ne(nekbone), Xb(XSBench)
Dual-core workloads (Using acronyms shown above)
AsDl, GcGa, BzXa, LsLb, GkBz, OmGr, NdCd, CaTo, LqPo, SjWr
GmMk, PeZe, SpMi, HmH2, SoBw, NeXb, McLu, CoAm
Quad-core workloads (Using acronyms shown above)
LbWrSpMi,CaOmCdMk, LsPoGmH2, GkSoSjNd
AsGaXaLu, GcBzGrTo, LqCoMcBw, PeZeHmDl, PoAmXbNe

B. Workloads

All 29 SPEC CPU2006 benchmarks with reference inputs and six benchmarks from HPC field (shown as italics in Table II) are taken as single-core workloads. Using these, 18 dual-core and 9 quad-core multiprogrammed workloads are randomly created such that each benchmark is used exactly once (except for fulfilling the left-over group). These workloads are shown in Table II.

C. Comparison With Other Techniques

We compare EqualWrites with three techniques for improving NVM lifetime.

1) PoLF: In PoLF [6], after a fixed number of write hits [called flush threshold (FT)] in the entire cache, a write operation is skipped; instead, the data item is directly written back to memory and the cache block is invalidated without updating the LRU age information. Probabilistically, the block selected is expected to be hot and hence, based on LRU replacement policy, the hot data item will be loaded in another cold block leading to intra-set wear leveling. The latency of incrementing the write counter and comparison with the threshold are taken as 1 and 2 cycles, respectively.

2) PoLSwap: PoLSwap is the same as PoLF, except that it works by migrating the hot data to the LRU block within same set, instead of flushing it to the main memory. PoLSwap handles hot data in the same manner as EqualWrites does (i.e., using in-cache data movement), while both PoLF and PoLSwap identify hot data differently from EqualWrites. Thus, our motivation in designing and testing this variant of PoLF is to separate the effect of identification and handling of hot data as we compare PoLF/PoLSwap with EqualWrites. We test PoLSwap with same threshold as PoLF and hence use the same term FT, although it is noteworthy that PoLSwap swaps hot data item and does not flush it.

3) WriteSmoothing: WrSm [16] logically divides the cache-sets into multiple modules, for example, in a cache with 4096 sets and 32 modules, each module contains 128 sets. For each module, the coefficient of intra-set WV is computed, and when it is larger than a threshold, the most frequently written way in a module is made unavailable to shift the write pressure to other ways in the sets of the module. This helps in achieving wear leveling.

Further, to evaluate the impact of cache replacement policy on cache lifetime, we also study the case where L2 replacement policy is changed from LRU to random.

D. Evaluation Metrics

Our baseline is a cache that uses LRU replacement policy, but does not use any scheme for improving cache lifetime. Using 'technique' to refer to EqualWrites, PoLF, and so on, we show results on the following metrics.

1) Relative cache lifetime ($\text{Lifetime}_{\text{technique}}/\text{Lifetime}_{\text{baseline}}$) [lifetime with a technique (or baseline) is defined as the inverse of the maximum number of writes on any cache block on using that technique (or baseline)].
2) IntraV (discussed in Section III).
3) Weighted speedup (called relative performance) [38].
4) Percentage energy loss (energy model shown below).
5) Absolute increase in miss per kilo instructions (MPKI).

We have used raw cache lifetime, since it provides useful insights and also forms the basis of error-tolerant lifetime. Note that we present the results on both maximum number of writes on any block and IntraV. The former considers the worst case writes on any block, while the latter considers the average writes and accounts for writes on all the blocks of the cache. Taken together, they evaluate a technique in comprehensive manner. These metrics have been used by other research works also [6], [15], [16], [27], [30], [32].

TABLE III
RESULTS WITH WrSm

Number of cores	Relative Lifetime	% IntraV Baseline	% IntraV WrSm	% Energy Loss	MPKI increase
1	2.17×	133.2	45.1	2.32	0.05
2	2.48×	125.4	49.8	2.96	0.06
4	3.21×	127.4	49.8	3.53	0.06

We model the energy of algorithm execution (shown in Section IV), L2 cache, and main memory. Parameters for L2 cache are shown in Table I. Dynamic energy and leakage power for main memory are 70 nJ/access and 0.18 W, respectively [38], [39]. For multicore system, we have also computed fair speedup [38] on using our technique and have found fair speedup to be almost the same as weighted speedup. Thus, our technique does not cause unfairness. Speedup values are averaged using geometric mean and the remaining metrics are averaged using arithmetic mean [38]. Simulations are performed till each core runs 500M instructions. In multicore system, the program that finishes earlier is allowed to run, but its instruction per cycle is only recorded for the first 500M instructions. Remaining metrics are computed for the entire simulation (following well-established simulation methodology [38], [40]), since they are system-wide metrics. To provide additional insights, we also present the data on the number of shifting operations in EqualWrites and the number of flushes in PoLF.

VI. EXPERIMENTAL RESULTS

A. Main Results

Figs. 3–5 show the results for single-, dual-, and quad-core systems, respectively. These experiments have been conducted using the following parameters: 16-way set-associativity, 4-MB L2 with $\Omega = 10$ for single-core system, 8-MB L2 with $\Omega = 16$ for dual-core system, and 16-MB L2 with $\Omega = 24$ for quad-core system. PoLF and PoLSwap are simulated with FT = 10, following the original work [6]. Discussion on the choice of Ω is presented in Section VI-C. Per-workload figures for the increase in MPKI are omitted for brevity and their average value are discussed in the following sections. Results on number of migrations/invalidations are shown in Table IV.

1) Results With WriteSmoothing: We simulate WrSm with default parameters, following the original work [16]. Lifetime improvement with WrSm is significantly lower compared with other techniques and hence, to reduce the clutter of figures, we omit the per-workload results with WrSm and only present the average results in Table III. For all core configurations, the relative performance is 0.99× and hence, it is omitted from the table.

WrSm performs intra-set wear leveling, but it does so at a coarse-level of modules and not individual sets as done by other techniques. Since the most frequently written ways in different sets of a module may not have the same index, the effectiveness of wear leveling becomes limited. WrSm also requires additional hardware to periodically compute the coefficient of WV. Other techniques use simpler approaches to estimate the amount of WV.

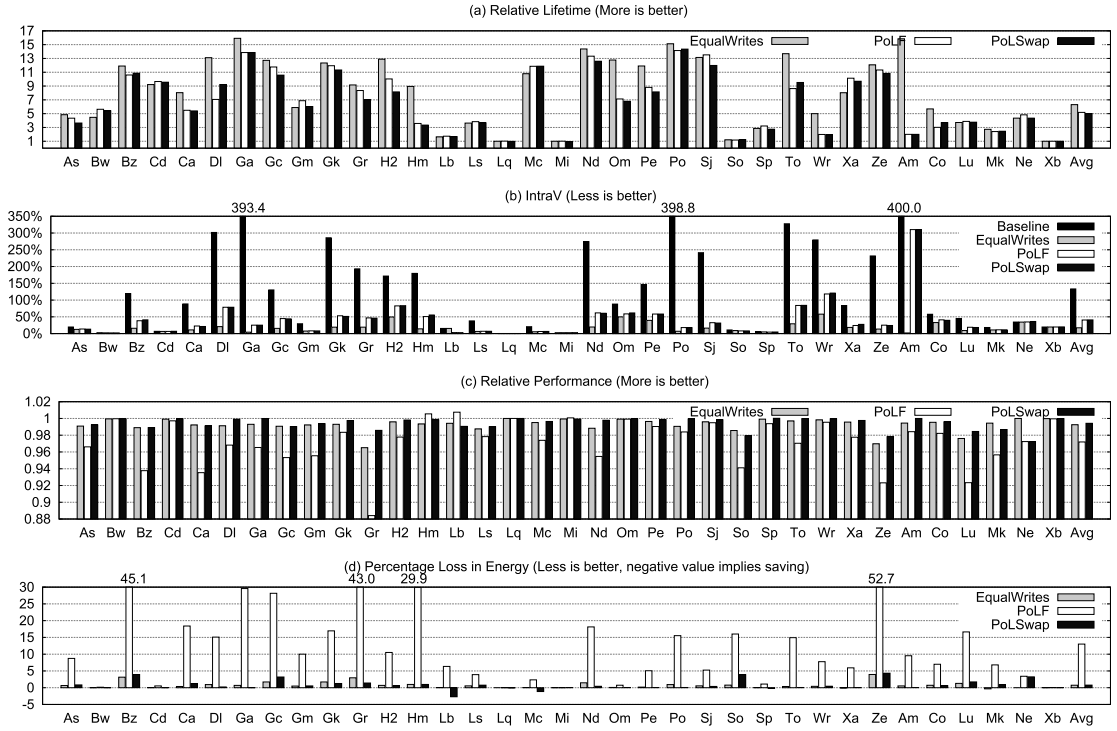

Fig. 3. Results for single-core system. Results with WrSm and random cache replacement policy are separately presented in Table III and discussed in Section VI-A to reduce the clutter of graphs and also because their effectiveness is significantly lower. (a) Relative lifetime (more is better). (b) IntraV (less is better). (c) Relative performance (more is better). (d) Percentage loss in energy (less is better, negative value implies saving).

2) Results With Random Replacement Policy: On changing the replacement policy from LRU to random, for single-, dual-, and quad-core systems, we observe a relative lifetime of 1.31×, 1.62×, and 2.33×, respectively, and relative performance of 0.99×, 0.97×, and 0.98×, respectively. It is clear that use of random replacement policy provides negligible improvement in cache lifetime and thus, it cannot address the issue of WV. The reason behind this is that a replacement policy is invoked only on a miss. When the cache hit rate is reasonably high, hardly few replacements take place and, a single or few cache block(s) may still get repeatedly written, which leads to high WV and small cache lifetime. Compared with using random cache replacement policy, EqualWrites provides equal or better performance and much larger improvement in lifetime. This clearly highlights the need of using an intelligent technique, such as EqualWrites for improving cache lifetime. In the remainder of this paper, we do not discuss WrSm technique or results with random replacement policy due to their limited effectiveness. We now analyze the results in detail.

3) Results on Lifetime and IntraV: On average, EqualWrites provides better results than PoLF on all the metrics for all system configurations. Using EqualWrites (resp., PoLF and PoLSwap), the average improvement in lifetime for single-, dual-, and quad-core systems are 6.31× (resp., 5.18× and 5.01×), 8.74× (resp., 8.12× and 7.94×), and 10.54× (resp., 10.23× and 9.62×), respectively. For $N = 1$, EqualWrites reduces IntraV from 133.2% to 17.2%, while PoLF reduces it to only 40.7%. For $N = 2$, EqualWrites reduces IntraV from 125.4% to 15.7%, while PoLF reduces it to only 32.7%. For $N = 4$, EqualWrites reduces IntraV from 127.4% to 19.2%, while PoLF reduces it to only 26.5%. The value of IntraV with PoLSwap is almost similar to that with PoLF. Clearly, EqualWrites is more effective in mitigating intra-set WV. For several workloads, EqualWrites improves cache lifetime by more than 10×, e.g., Po, Ga, AsDl, HmH2, LsPoGmH2, and PoAmXbNe.

On using a WLT, the maximum possible increase in the cache lifetime depends on the variation present in the workload. Thus, the largest improvement in lifetime is obtained for workloads that show high WV in the baseline case, for example, Ga, Po, Sj, Nd, Am, SjWr, and PoAmXbNe. Conversely, for workloads such as Lq, Mi, and So, the intra-set WV in the baseline execution is close to zero, and hence, the scope of improvement in lifetime using wear leveling is also negligible. Cache lifetime improvement for these workloads can be achieved using other techniques such as the use of write buffers and filter cache.

4) Results on Performance, Energy, and Miss Rate: On average, for all systems, the relative performance on using EqualWrites and PoLSwap is 0.99× and for PoLF, it is 0.97×. Further, the average loss in energy on using EqualWrites (resp., PoLF and PoLSwap) for single-, dual-, and quad-core systems are 0.74% (resp., 12.92% and 0.77%), 0.76% (resp., 14.61% and 1.27%), and 0.94% (resp., 23.20% and 2.82%), respectively.

Clearly, EqualWrites incurs negligible loss in performance and energy, while PoLF incurs large loss in these parameters. Compared with EqualWrites, PoLSwap incurs larger loss in energy. EqualWrites does not use data invalidation, instead

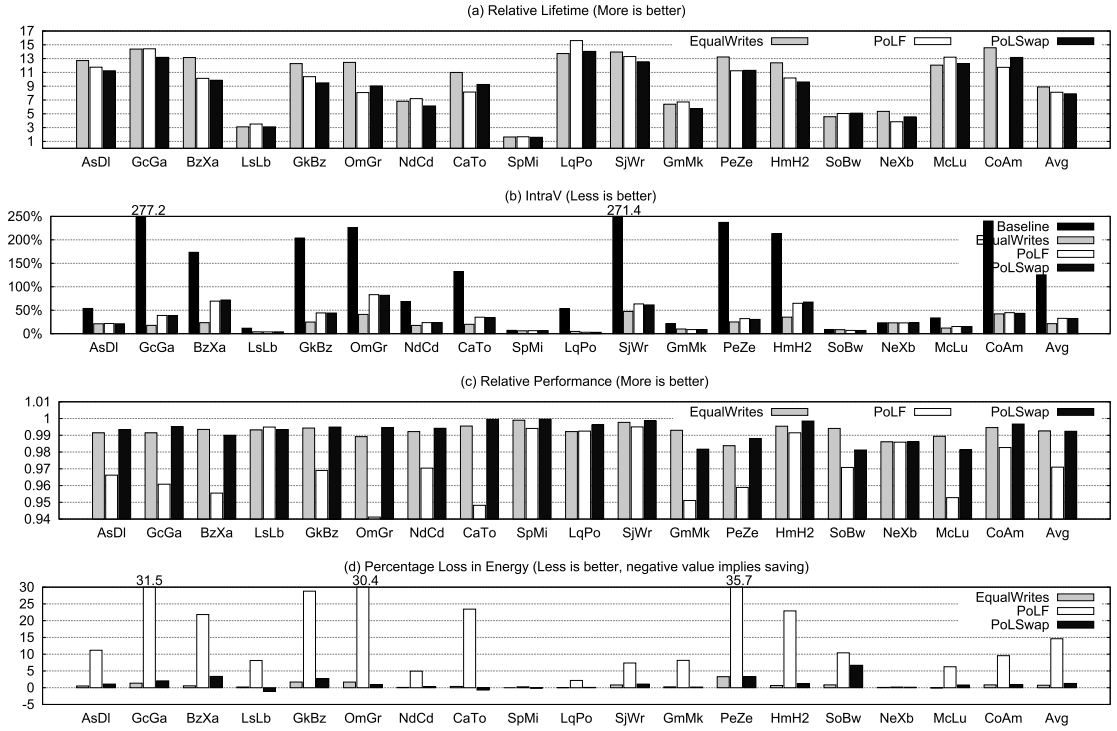

Fig. 4. Results for dual-core system. (a) Relative lifetime (more is better). (b) IntraV (less is better). (c) Relative performance (more is better). (d) Percentage loss in energy (less is better, negative value implies saving).

it performs write redirection within the same set and hence, avoid costly off-chip accesses. On using EqualWrites and PoLSwap, the increase in MPKI is negligible (<0.03), and on using PoLF, the increase in MPKI for single, dual and quad-core systems is 0.56, 0.57 and 0.55, respectively. Clearly, EqualWrites does not increase memory traffic or cause queue contention.

5) Results on Shifting/Flushing Operations: Table IV shows the number of shifting (write-redirection) operations for EqualWrites and the number of flush operations for PoLF. It is clear that the shifting operations are much smaller in number in EqualWrites than the number of flush operations in PoLF. This is because, EqualWrites accurately identifies a hot block by virtue of using write-frequency counters. Hence, a shifting operation in EqualWrites is only triggered when the difference between writes to different blocks is high, or in other words, the write *variation is high*. For PoLF and PoLSwap, higher number of flush operations indicate that the write *intensity is high*, and this may happen even if the WV itself is small. For example, for Lb (lbm), IntraV is low [Fig. 3(b)], and thus, the scope of lifetime improvement is low. Hence, EqualWrites only performs 418 redirections, while PoLF flushes nearly 2 219 000 blocks (Table IV), which leads to high performance loss without achieving appreciable improvement in lifetime. This clearly shows that EqualWrites performs wear leveling in much more intelligent manner. In addition, its operational overhead is very small.

6) Analysis of Features/Limitations of WLTs: PoLF and PoLSwap do not use write-frequency counters, instead they identify a hot block in probabilistic manner. However, this approach may not select a hot block in all circumstances; in fact, PoLF can flush a newly installed block also. For PoLF, this is especially harmful since it flushes data blindly and hence, in the cases where the write intensity is high but intra-set WV is small, PoLF would still invalidate large number of cache blocks, leading to large performance and energy loss. This is especially evident with some workloads such as So and Lb. In addition, using PoLF, the increase in MPKI is greater than 0.55; thus, data invalidation in PoLF increases memory traffic and power consumption, which may be unacceptable for the given bandwidth-wall problem. Further, in the case where the main memory is itself designed using NVM (such as PCM), this technique may worsen the write-endurance issue in main memory. PCM also has higher write latency than DRAM and hence, the overhead of extra misses introduced due to PoLF will be higher with PCM memory.

PoLSwap partially addresses the limitation of PoLF by migrating data, instead of flushing it and hence, it incurs smaller loss. PoLSwap achieves a smaller lifetime improvement than PoLF. The reason for this is that swapping of data within a set in PoLSwap leads to an extra write and hence, PoLSwap increases the writes to the cache. The benefit of PoLF and PoLSwap is that they only require a single global counter. In addition, PoLF does not use in-cache data movement or swap buffer.

Note that EqualWrites provides larger lifetime improvement than both PoLF and PoLSwap. Thus, it is clear that the superiority of EqualWrites over PoLF is not merely because of using data migration (versus data invalidation), since EqualWrites performs better than PoLSwap, which also uses data migration. EqualWrites uses a better method of detecting hot data items

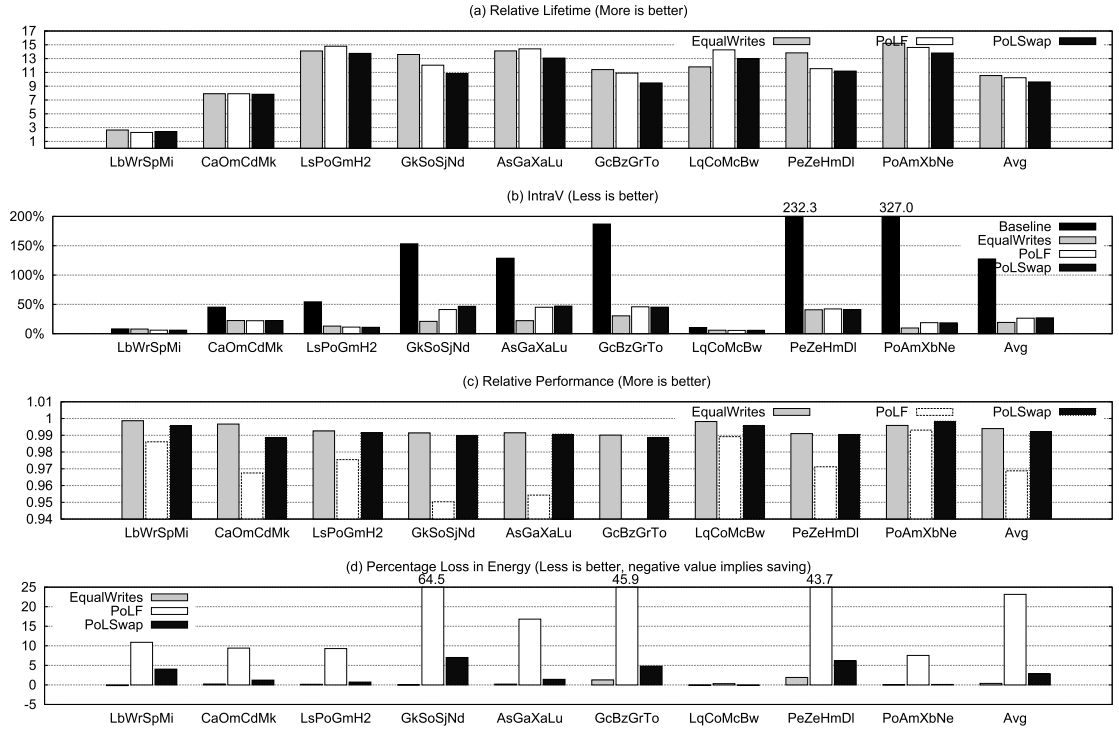

Fig. 5. Results for quad-core system. (a) Relative lifetime (more is better). (b) IntraV (less is better). (c) Relative performance (more is better). (d) Percentage loss in energy (less is better, negative value implies saving).

TABLE IV

DATA ON THE NUMBER OF CASE-I AND CASE-V OPERATIONS IN EQUALWRITES AND FLUSHES IN POLF (THE NUMBER OF MIGRATIONS IN POLSWAP ARE ALMOST SIMILAR TO THE FLUSHES IN POLF AND HENCE ARE OMITTED)

Single-core system								Dual-core system				Quad-core system			
	EqualWrites		PoLF		EqualWrites		PoLF		EqualWrites		PoLF		EqualWrites		PoLF
	Case-I	Case-V	Flush		Case-I	Case-V	Flush		Case-I	Case-V	Flush		Case-I	Case-V	Flush
As	39	161K	318K	Nd	85K	30K	65K	AsDl	5345	123K	449K	LbWrSpMi	3	7927	3294K
Bw	47	4323	50K	Om	593	3988	7745	GcGa	107K	220K	356K	CaOmCdMk	2722	111K	676K
Bz	24K	224K	232K	Pe	3515	32K	29K	BzXa	2984	265K	351K	LsPoGmH2	5594	446K	2201K
Cd	22	610	73K	Po	173K	15K	101K	LsLb	1610	185K	3071K	GkSoSjNd	33K	655K	1485K
Ca	152K	79K	309K	Sj	28K	51K	57K	GkBz	71K	241K	425K	AsGaXaLu	14K	530K	1413K
Dl	80K	33K	67K	So	379	576K	1134K	OmGr	15K	206K	236K	GcBzGrTo	35K	374K	826K
Ga	235K	6219	130K	Sp	9	5457	49K	NdCd	3946	85K	160K	LqCoMcBw	1136	96K	1719K
Gc	561	188K	173K	To	94K	15K	64K	CaTo	150K	88K	419K	PeZeHmDl	31K	352K	803K
Gm	0	982K	1194K	Wr	38K	5191	30K	SpMi	0	19K	430K	PoAmXbNe	83K	163K	336K
Gk	83K	119K	133K	Xa	978	150K	107K	LqPo	3355	359K	597K	**Avg**	**23K**	**304K**	**1417K**
Gr	49K	308K	227K	Ze	42K	718K	507K	SjWr	56K	53K	126K				
H2	28K	15K	43K	Am	126K	240	68K	GmMk	196	325K	1562K				
Hm	181K	12K	149K	Co	14K	28K	59K	PeZe	30K	418K	536K				
Lb	0	418	2219K	Lu	637	299K	323K	HmH2	124K	27K	207K				
Ls	6488	341K	548K	Mk	2078	153K	230K	SoBw	20	28K	1192K				
Lq	0	0	277K	Ne	13	1	10K	NeXb	0	8	19K				
Mc	2054	105K	405K	Xb	0	0	4463	McLu	1548	188K	729K				
Mi	0	0	358K	**Avg**	**41K**	**133K**	**279K**	CoAm	39K	59K	132K				
								Avg	**34K**	**161K**	**611K**				

than PoLF, PoLSwap, and WrSm and further, EqualWrites handles the hot data item in a better manner than PoLF.

B. Sensitivity of PoLF and PoLSwap Toward Their Respective Thresholds

We henceforth omit the detailed figures and only present the average results. Table V shows the results for PoLF and PoLSwap at different values of thresholds. For brevity, results on only lifetime and energy are presented.

From the results in Table V, we note that except for $FT = 6$ for quad-core system, the lifetime improvement of PoLF remains smaller than EqualWrites for all cases. For $FT = 6$ for quad-core system, although PoLF improves lifetime by 10.68×, which is higher than 10.54× for EqualWrites, PoLF also incurs an unacceptably high energy loss of 38%, compared with 0.39% for EqualWrites. The lifetime improvement of PoLSwap remains smaller than that of EqualWrites for all cases. In addition, note that with increasing

TABLE V
RESULTS WITH PoLF AND PoLSWAP FOR DIFFERENT VALUES OF THRESHOLDS

	Single-core system			
	Relative Lifetime		% Energy Loss	
FT	PoLF	PoLSwap	PoLF	PoLSwap
6	5.62×	5.13×	21.2	1.25
10	5.18×	5.01×	13.01	0.77
16	4.73×	4.50×	8.24	0.48
20	4.57×	4.34×	6.62	0.40
24	4.15×	4.08×	5.54	0.33
32	4.10×	3.86×	4.19	0.27
100	2.84×	2.83×	1.40	0.14
	Dual-core system			
6	8.82×	8.02×	23.94	2.17
10	8.12×	7.90×	14.62	1.27
16	7.76×	7.06×	9.26	0.77
20	7.18×	6.89×	7.46	0.61
24	6.77×	6.61×	6.26	0.52
32	6.62×	6.85×	4.75	0.39
100	4.74×	4.84×	1.57	0.13
	Quad-core system			
6	10.68×	9.76×	37.98	4.54
10	10.23×	9.62×	23.16	2.82
16	10.08×	9.79×	14.54	1.84
20	9.97×	9.47×	11.60	1.48
24	9.11×	9.26×	9.77	1.23
32	8.98×	8.93×	7.66	0.90
100	6.89×	7.24×	2.44	0.30

threshold value, the lifetime improvement reduces, although the energy loss also reduces.

C. Parameter Sensitivity Results

We now focus exclusively on EqualWrites and evaluate its sensitivity for different parameters. Each time, we change just one parameter from the default configuration and summarize the results in Table VI. The increase in MPKI is always less than 0.05 and hence, its value is omitted.

1) Change in Ω*:* We first discuss the reasoning behind choice of Ω (write redirection threshold). First, Ω is chosen to be even, although it is trivial to extend the algorithm for odd values of Ω. The value of Ω is chosen to be large enough such that redirections do not happen frequently, rather they happen only when the WV exceeds a lower limit (redirections themselves involve writes and hence, need to be controlled). For this reason, $\Omega < 6$ values are avoided. At the same time, Ω is chosen to be small enough such that the overhead of counters remains small and aggressiveness of wear leveling is not low. Further, with increasing number of cores, the write-intensity increases and hence, the best (or optimal) value of Ω is also increased, since at lower value of Ω, redirections happen more frequently, and lead to lower improvement in the cache lifetime.

From Table VI, we note that smaller values of Ω lead to more fine-grain wear leveling, as seen from the value of IntraV. However, reducing the value of Ω below a certain point has adverse effect on the cache lifetime, as evident from the value of lifetime improvement. The reason for this is that for small values of Ω, extra writes due to write redirection are significantly increased, which lead to reduced improvement in lifetime. In addition, for small values of Ω, energy loss and MPKI are slightly increased.

TABLE VI
PARAMETER SENSITIVITY RESULTS. DEFAULT PARAMETERS ARE SHOWN IN SECTION VI-A. BASE = BASELINE, EqWr = EQUALWRITES, REL. PERF. = RELATIVE PERFORMANCE, nSHIFTING = TOTAL SHIFTING OPERATIONS (CASE-I + CASE-V). $\Gamma = 2\times$ SHOWS 2× OVERHEAD OF SHIFTING OPERATIONS COMPARED WITH THAT MENTIONED IN SECTION IV

	Relative Lifetime	IntraV Base %	IntraV EqWr %	Rel. Perf.	Energy Loss %	nShifting
Single-core System						
Default	6.31×	133.2	17.2	0.99×	0.74	175K
Ω = 6	6.20×	133.2	13.6	0.99×	1.36	342K
Ω =8	6.30×	133.2	15.5	0.99×	1.16	240K
Ω =14	6.22×	133.2	21.2	0.99×	0.65	107K
Ω = 18	6.13×	133.2	24.5	0.99×	0.52	67K
$\Gamma = 2\times$	6.31×	133.2	17.2	0.99×	1.04	175K
$\Gamma = 4\times$	6.33×	133.2	17.2	0.98×	1.74	175K
A=8	3.99×	104.6	15.3	0.99×	0.68	158K
A=32	9.10×	163.5	18.3	0.99×	0.87	182K
2MB	4.39×	101.5	10.5	0.99×	0.83	145K
8MB	8.22×	171.0	26.9	0.99×	0.65	193K
Dual-core System						
Default	8.89×	125.4	21.4	0.99×	0.74	194K
Ω = 8	8.48×	125.4	13.6	0.99×	1.55	541K
Ω = 12	8.73×	125.4	17.6	0.99×	0.90	307K
Ω = 20	8.63×	125.4	24.1	0.99×	0.54	143K
Ω = 24	8.61×	125.4	26.9	0.99×	0.48	115K
$\Gamma = 2\times$	8.88×	125.4	21.4	0.99×	1.03	194K
$\Gamma = 4\times$	8.82×	125.4	21.4	0.99×	1.64	194K
A=8	5.08×	99.7	19.7	0.99×	0.64	178K
A=32	15.24×	154.9	22.9	0.99×	0.83	215K
4MB	6.90×	89.2	14.2	0.99×	0.58	171K
16MB	10.03×	158.2	31.2	0.99×	0.76	220K
Quad-core System						
Default	10.54×	127.4	19.2	0.99×	0.39	327K
Ω = 16	10.47×	127.4	15.4	0.99×	0.94	544K
Ω =20	10.43×	127.4	17.4	0.99×	0.50	412K
Ω =28	10.46×	127.4	20.8	0.99×	0.31	272K
Ω = 32	10.49×	127.4	22.3	1.00×	0.27	233K
$\Gamma = 2\times$	10.57×	127.4	19.3	0.99×	0.56	325K
$\Gamma = 4\times$	10.79×	127.4	19.2	0.98×	1.08	322K
A=8	5.36×	99.6	17.8	0.99×	0.41	296K
A=32	17.26×	159.6	20.5	0.99×	0.63	339K
8MB	7.85×	91.4	14.3	1.00×	0.23	242K
32MB	12.17×	158.7	28.7	0.99×	0.57	387K

2) Higher Redirection Overhead: We experiment with values of latency and energy overhead of shifting operations, which are 2× and 4× of those mentioned in Section IV. From the results in Table VI (referring to the rows $\Gamma = 2\times$ and $\Gamma = 4\times$), we observe that performance and energy are minimally affected. This confirms that shifting overhead incurred in EqualWrites is minimal.

3) Change in Associativity (A): Caches with smaller associativity show smaller value of intra-set WV and vice versa. This is evident from the value of IntraV for baseline cache in Table VI and is also confirmed by previous works [16], and can be explained as follows. For the fixed cache size and fixed working set size of the application, the hit-rate of application increases with increasing cache associativity due to reduced conflict misses. Due to increased hit-rate, same blocks get repeatedly accessed and evictions due to miss reduce. Due to temporal locality, only few most recently used ways absorb most of the accesses [39]. With higher-associativity, this is

even more true and this leads to higher WV. For smaller associativity value, the number of replacement candidates is smaller and miss rate is higher. Hence, the number of evictions is higher and repeated writes to a few cache blocks are less likely to occur.

This can be further understood by considering two extreme examples, viz., direct-mapped cache and fully-associative cache. For a direct-mapped (i.e., single-way) cache, all the writes happen to the same way and the WV is zero. On the other hand, for a fully-associative cache (i.e., single-set), the associativity is equal to the number of blocks in the cache, of which only few ways will be written. Thus, the WV will be highest.

The results in Table VI show that EqualWrites provides lifetime improvement in proportion to the amount of variation present in the baseline. Moreover, for 32-way associative cache, the lifetime improvement in single-, dual-, and quad-core systems are 9.10×, 14.67×, and 17.26×, respectively. This clearly highlights that a WLT is extremely important for caches with large associativity value.

4) Change in Cache Capacity: For a fixed working set size, the increase in cache capacity improves the hit-rate and increases repeated accesses to a few cache blocks. This increases the intra-set WV, as evident from the value of IntraV for caches with different capacities. From Table VI, we also conclude that depending on the WV originally present in the application, EqualWrites provides corresponding improvement in cache lifetime.

The results presented in this section confirm that EqualWrites works well for a wide range of system and algorithm parameters and incurs negligible loss in performance and energy.

VII. Future Work: Extension of EqualWrites for Special Cases

In this section, we show the manner in which EqualWrites can be extended for handling special cases.

A. Improving Lifetime by Tolerating Errors

In this paper, we have evaluated *raw cache lifetime*, which is defined by the first failure of any cache block. This lifetime can be easily extended if the system can tolerate the failure of a few cache blocks, which decides the *error-tolerant lifetime*. Use of error-correction codes (ECCs) allows for correcting and thus tolerating a given number of failures. The number of errors which can be corrected depends on the strength of ECC. The cache lifetime is then determined by the time it takes for the number of failed bits in a memory reference to become larger than the number of errors that can be corrected. Several researchers have proposed techniques that achieve this by adding data redundancy [41], [42] and these techniques can be synergistically integrated with EqualWrites, since they are orthogonal to our work.

B. Mitigating Security Threats in NVMs

The limited write endurance of NVMs presents a crucial security threat, since using a simple attack, a malicious attacker can write a cache block repeatedly, leading to the failure of the system. A greedy user may also run such codes to get a new system in the warranty period. Conventional endurance-unaware cache management policies do not provide a mechanism for preventing such attacks and thus leave a serious security vulnerability. EqualWrites can be useful for mitigating such attacks. In EqualWrites, the attacked data item is most likely to be migrated and by randomizing the value of Ω within a predetermined range, the location of the hot data can be changed in a manner that makes it difficult for the attacker to predict the new location. Using this, the raw lifetime of the system can be significantly extended and within this time, any unusual access sequence can be easily detected.

VIII. Conclusion

Addressing the limitations posed by low write endurance of NVMs is essential for making them a universal memory solution. In this paper, we presented EqualWrites as a technique for improving the lifetime of non-volatile caches by minimizing intra-set WV. EqualWrites achieves wear leveling by redirecting a hot data item to a cold block using in-cache data movement. The experimental results have shown that EqualWrites is effective in improving the cache lifetime and works well for a wide range of system and algorithm parameters. In addition, it provides better results than three other WLTs and the use of random replacement policy. Our future work will focus on integrating EqualWrites with bit-level WMTs and error correction/detection techniques to further improve the improvement in cache lifetime.

Acknowledgment

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] A. Jog *et al.*, "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in CMPs," in *Proc. 49th Annu. Design Autom. Conf.*, 2012, pp. 243–252.

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

[3] S. Iyer *et al.*, "Embedded DRAM: Technology platform for the Blue Gene/L chip," *IBM J. Res. Develop.*, vol. 49, nos. 2–3, pp. 333–350, 2005.

[4] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, 2015, doi: 10.1109/TPDS.2014.2324563.

[5] Y.-B. Kim *et al.*, "Bi-layered RRAM with unlimited endurance and extremely uniform switching," in *Proc. IEEE Symp. VLSI Technol. (VLSIT)*, Jun. 2011, pp. 52–53.

[6] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i^{2}WAP: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013, pp. 234–245.

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

[8] Grandis. (2010). *Latest Advances and Future Prospects of STTRAM*. [Online]. Available: http://nvmw.ucsd.edu/2010/documents/Driskill-Smith_Alexander.pdf

[9] Y. Joo and S. Park, "A hybrid PRAM and STT-RAM cache architecture for extending the lifetime of PRAM caches," *IEEE Comput. Archit. Lett.*, vol. 12, no. 2, pp. 55–58, Jul./Dec. 2013.

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

[11] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2009, pp. 2–13.

[12] J. Dorsey *et al.*, "An integrated quad-core Opteron processor," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2007, pp. 102–103.

[13] *Intel Itanium Processor 9500 Series: Reference Manual*, Intel Corp., Santa Clara, CA, USA, 2014.

[14] Y. Li, Y. Chen, and A. K. Jones, "A software approach for combating asymmetries of non-volatile memories," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2012, pp. 191–196.

[15] 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.

[16] S. Mittal, J. S. Vetter, and D. Li, "WriteSmoothing: Improving lifetime of non-volatile caches using intra-set wear-leveling," in *Proc. 24th ACM Great Lakes Symp. VLSI (GLSVLSI)*, 2014, pp. 139–144.

[17] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[18] Intel. (2013). *Product Brief: Intel Itanium Processor 9500 Series*. [Online]. Available: http://download.intel.com/newsroom/archive/Intel-Itanium-processor-9500_ProductBrief.pdf

[19] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustain. Comput., Informat. Syst.*, vol. 4, no. 1, pp. 33–43, 2014.

[20] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 14–23.

[21] K. Swaminathan, E. Kultursay, V. Saripalli, V. Narayanan, and M. Kandemir, "Design space exploration of workload-specific last-level caches," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2012, pp. 243–248.

[22] S. Guo, Z. Liu, D. Wang, H. Wang, and G. Li, "Wear-resistant hybrid cache architecture with phase change memory," in *Proc. IEEE 7th Int. Conf. Netw., Archit. Storage (NAS)*, Jun. 2012, pp. 268–272.

[23] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2009, pp. 347–357.

[24] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2009, pp. 239–249.

[25] J. Ahn and K. Choi, "Lower-bits cache for low power STT-RAM caches," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 480–483.

[26] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *IEEE/ACM Int. Conf. Comput.-Aided Design-Dig. Tech. Papers (ICCAD)*, Nov. 2009, pp. 264–268.

[27] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proc. Conf. Design, Autom. Test Eur.*, 2010, pp. 136–141.

[28] G. Duan and S. Wang, "Exploiting narrow-width values for improving non-volatile cache lifetime," in *Proc. Conf. Design, Autom. Test Eur.*, 2014, Art. ID 52.

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

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

[31] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based energy-efficiency hybrid cache for CMPs," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI Syst.-on-Chip (VLSI-SoC)*, Oct. 2011, pp. 31–36.

[32] S. Mittal and J. S. Vetter, "AYUSH: A technique for extending lifetime of SRAM-NVM hybrid caches," *IEEE Comput. Archit. Lett.*, 2015, doi: 10.1109/LCA.2014.2355193.

[33] S. Yazdanshenas, M. R. Pirbasti, M. Fazeli, and A. Patooghy, "Coding last level STT-RAM cache for high endurance and low power," *IEEE Comput. Archit. Lett.*, vol. 13, no. 2, pp. 73–76, Jul./Dec. 2014.

[34] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. 28th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2001, pp. 240–251.

[35] 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 *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2003, pp. 402–407.

[36] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[37] J. C. Ku, S. Ozdemir, G. Memik, and Y. Ismail, "Thermal management of on-chip caches through power density minimization," in *Proc. 38th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Nov. 2005, pp. 283–293.

[38] S. Mittal, Y. Cao, and Z. Zhang, "MASTER: A multicore cache energy-saving technique using dynamic cache reconfiguration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 8, pp. 1653–1665, Aug. 2014.

[39] S. Mittal, Z. Zhang, and J. S. Vetter, "FlexiWay: A cache energy saving technique using fine-grained cache reconfiguration," in *Proc. 31st IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 100–107.

[40] M. Chaudhuri, "Pseudo-LIFO: The foundation of a new family of replacement policies for last-level caches," in *Proc. 42nd Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 401–412.

[41] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2011, pp. 466–477.

[42] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 83–93, 2010.

Sparsh Mittal (S'12–M'14) received the B.Tech. degree in electronics and communications engineering from IIT Roorkee, Roorkee, India, and the Ph.D. degree in computer engineering from Iowa State University, Ames, IA, USA.

He is currently a Post-Doctoral Research Associate with the Oak Ridge National Laboratory, Oak Ridge, TN, USA. His current research interests include nonvolatile memory, memory system power efficiency, cache architecture, and CPU-GPU (graphics processing unit) heterogeneous computing.

Jeffrey S. Vetter (SM'11) received the Ph.D. degree from the Georgia Institute of Technology (GT), Atlanta, GA, USA.

He holds a joint appointment between the Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, USA, and GT. At ORNL, he is currently a Distinguished Research and Development Staff Member, where he is the Founding Group Leader of the Future Technologies Group. At GT, he is a Joint Professor with the Computational Science and Engineering School, the Project Director of the NSF Track 2-D Experimental Computing Facility for large-scale heterogeneous computing using graphics processors, and the Director of the NVIDIA CUDA Center of Excellence. His current research interests include massively multithreaded processors, nonvolatile memory, and heterogeneous multicore processors.