AYUSH: Extending Lifetime of SRAM-NVM Way-based Hybrid Caches Using Wear-leveling

Sparsh Mittal*, Jeffrey S. Vetter*§
*Oak Ridge National Lab, §Georgia Tech
Oak Ridge, Tennessee, USA
Email: {mittals,vetter}@ornl.gov

***Abstract*—The features and limitations of both SRAM and NVM (non-volatile memory) technologies have led the researchers to study SRAM-NVM way-based hybrid last level caches (LLCs). Since large leakage power consumption of SRAM allows including only few SRAM ways, the small write-endurance of NVM may still lead to small lifetime of these hybrid caches. We propose AYUSH, a technique for improving lifetime of SRAM-NVM hybrid caches. AYUSH uses data-migration approach to preferentially utilize SRAM for storing write-intensive data. Microarchitectural simulations have shown that AYUSH provides larger improvement in lifetime than three previous techniques. For single, dual and quad-core system configurations, the average increase in cache lifetime with AYUSH is 6.90×, 24.06× and 47.62×, respectively. Also, it does not harm performance or energy efficiency and works well for a range of system and algorithm parameters.**

***Keywords*-Non-volatile memory, write-endurance, write-minimization, intra-set wear-leveling, lifetime enhancement**

I. Introduction

As we enter into the post-petascale era, system core-counts are expected to increase at a much faster rate than the memory bandwidth. To bridge this gap and feed data to the large number of cores, very large last level caches (LLCs) are being used in modern processors, for example, Intel's Ivytown processor has 37.5MB SRAM LLC [1]. Since SRAM offers fast access speed and high write endurance, it has been conventionally used for designing LLCs. However, since SRAM has low density and high leakage power, SRAM caches accordingly occupy a large fraction of chip area and power budget, e.g. leakage power of last level cache accounts for 30% and 20% of the total power in Intel Xeon Tulsa and Intel Core 2 Penryn processors [2], respectively.

To address the limitations of SRAM, researchers have recently explored NVMs such as ReRAM (resistive RAM), STT-RAM (spin transfer torque RAM) and PCM (phase change memory) for designing LLCs [3–6]. Several leading companies are working on developing NVMs and NVM prototypes of large capacity have been demonstrated (e.g. a 32Gb ReRAM prototype [7]). Further, it has been recently announced that the next-generation supercomputer will have 800 GB of NVM [8]. These NVMs have high density and consume near-zero leakage power, for example, the cell size of SRAM and ReRAM are 125–200F^2 and 4–10F^2, respectively, where F denotes the feature size [9–11]. The limited write endurance of NVMs, however, presents a crucial bottleneck in their use. For example, while the write endurance of SRAM is more than 10^{15}, this value for ReRAM is only 10^{11} [11]. Process variations can further reduce these values [12].

To combine the strengths of both SRAM and NVM, SRAM-NVM way-based hybrid caches have been actively studied in recent years [4, 13–17]. These caches use a few SRAM ways along with large number of NVM ways to leverage the high endurance of SRAM along with high capacity (density) and low leakage of NVM. However, the write-variation introduced by conventional cache management policies may increase the writes to a few blocks, which may fail much earlier than the remaining blocks. Also, filtering by first level cache significantly reduces the temporal locality of LLC access stream ([18, 19], also see Section II) and hence, with conventional cache management policies, the write endurance limit of NVM may be reached in a few days even with a few (e.g. 3 out of 16) SRAM ways. Our experiments with a 4MB, 16-way cache (designed using 3 SRAM and 13 ReRAM ways) shows that the cache lifetime with bzip2 and zeusmp benchmarks is only 28 and 25 days, respectively (more details of experimental setup are provided in Section IV). Clearly, effective techniques are required for managing the hybrid caches.

Contributions: In this paper, we present AYUSH (which means *long life* in Sanskrit), a technique for increasing the lifetime of SRAM-NVM hybrid caches. AYUSH utilizes the fact that the write-endurance of SRAM is orders of magnitude higher than that of NVM, and hence, by migrating the write-intensive data to SRAM ways, the writes to NVM ways can be reduced which improves the lifetime of cache (Section II). To implement this, in each interval, we estimate the average associativity requirement (say Z) of the application. Then, assuming LRU (least recently used) replacement policy, the data-item in an SRAM block with LRU-age larger than Z can be considered to be cold, since it is not likely to be reused. Hence, on a write-hit to an NVM block, the oldest SRAM block is checked. If it stores cold data-item, it is swapped with the newly arrived data-item so that future writes can be redirected to the SRAM block.

1526-7539/15 $31.00 © 2015 IEEE
DOI 10.1109/MASCOTS.2015.29

IEEE computer society

AYUSH has several remarkable features. Its implementation overhead is small (Section III) and it does not require per-block counters to record access frequency/intensity (unlike [4, 16, 20, 21]) or large prediction/history table (unlike [17, 20, 22]). Further, unlike a few techniques (e.g. [10, 14]), AYUSH does not require compiler analysis or modifying program binary to guide data placement in hybrid caches. Furthermore, unlike [13], AYUSH does not duplicate the same data in SRAM and NVM and hence, it does not waste cache space. In this paper, we assume ReRAM as the NVM memory technology, although AYUSH can be easily applied for caches designed with other NVMs, such as STT-RAM. We henceforth use the term NVM or ReRAM ways interchangeably for convenience. Also, in this paper, we assume that the LLC is an L2 cache and based on it, AYUSH can be easily applied to an L3 LLC.

We perform microarchitectural simulations using workloads from SPEC2006 suite and HPC field (Section IV). We quantitatively compare AYUSH with three techniques named HDM (hot data migration) [15], RHCA (Region based hybrid cache architecture) [4] and PoLF (probabilistic line flush) [5]. The results show that AYUSH achieves largest enhancement in cache lifetime, and incurs negligible loss in performance and energy (Sections V-A and V-B). For single, dual and quad-core system configurations, the average increase in cache lifetime with AYUSH is 6.90×, 24.06× and 47.62×, respectively and thus, the lifetime improvement of AYUSH scales very well with increasing number of cores. Based on these results, we also qualitatively compare AYUSH with other techniques [16, 23] which bear similarity with RHCA, to further highlight the advantages of AYUSH. Additional experiments show that AYUSH works well for a range of algorithm and system parameters (Section V-C).

II. AYUSH: System Architecture

Notations: We use the following symbols. For the last level cache, S, W, B and G denote the number of sets, associativity, cache line (block) size and tag-size, respectively. In this paper, we assume $B = 64$ byte and $G = 30$ bits. Our baseline is a way-based SRAM-NVM hybrid cache which uses LRU replacement policy. Baseline cache management policies do not differentiate between SRAM or NVM, except that on a miss, the cache controller tries to first place data in SRAM ways. W_S ($0 < W_S < W$) denotes the number of SRAM ways which is fixed at design time. LRU_age refers to the order of a block in the LRU-stack and it ranges from 0 for the MRU way to $W - 1$ for the LRU way. LRU replacement policy naturally tracks LRU_age values, thus they are readily available.

A. Hybrid cache design tradeoffs

Effective management of hybrid caches requires accounting for properties of both LLC itself and the memory technologies (i.e. SRAM and NVM). The LLC access stream shows much lower amount of locality than that in first level cache, due to which a single MRU way designed with SRAM cannot capture a large fraction of hits in the LLC. To quantify this, we perform the following experiment. We take all 35 single-core workloads evaluated in the paper (refer Table III) and run them for 100M instructions, remaining experimental setup is same as shown in Section IV. We record the number of hits to different LRU-positions and show their value for 4-way L1 (both instruction and data) and 16-way L2 caches. In Table I, we show the percentage of total hits that are captured by just 1 most-recently-used (MRU) way, the first two MRU ways and first three MRU ways, respectively. These results confirm that the locality in L2 cache is much lower than that in L1-I and L1-D caches.

Table I
Percentage of cumulative hits in different MRU ways

	L1-I	L1-D	L2
% Hits in the MRU way	97.81	92.38	57.11
% Hits in first two MRU ways	99.64	98.01	72.21
% Hits in first three MRU ways	99.90	99.44	78.66

A naïve solution to increase the SRAM hits consists of increasing the value of W_S, i.e., designing a large number of ways using SRAM (e.g. 8 out of 16 in [19]). However, the leakage power of SRAM is much higher than that of NVM (e.g. up to an order of magnitude, refer Table II). Hence, an increase in W_S increases the leakage energy consumption and area of the cache which negates the purpose of using hybrid cache itself. Instead, we propose designing only small number of ways using SRAM (e.g. W_S = 2, 3, or 4 out of 16), and leveraging data migration approach to increase the number of writes that are served from the SRAM ways.

B. Data migration and lifetime enhancement algorithm

To trigger a migration operation, we need to decide if an SRAM block stores cold data. We use the fact that due to temporal locality of access stream, on using LRU replacement policy, most hits are concentrated *near* MRU ways [19]. We use a tunable parameter α ($\alpha < 100\%$) to quantify the "associativity requirement" (Z) of an application as follows. For a W-way cache, we use W counters, where the kth counter records the number of hits in the kth LRU-position. Assuming that the total number of hits in all the ways is H, we choose Z ($\geq W_S$) such that the total hits in most-recent Z positions are equal to or greater than αH.

To illustrate this, we take the following example. Assume that in an 8-way cache, the number of hits in an interval at positions ranging from MRU to LRU are 11200, 8800, 4900, 4600, 4300, 4100, 3100, 600, respectively, thus H = 41600. Then for α = 60%, αH = 24960 and we get Z = 4, thus 60% of hits can be captured by 4 ways. If, instead, α were 50%, then αH = 20800 and we get Z = 3, thus only 3 ways are sufficient for capturing 50% of hits.

Since Z shows the associativity requirement, an SRAM block with age larger (i.e. older) than Z can be considered

to be cold and such data-item is a candidate for migration to NVM which helps in using SRAM primarily for storing hot data. This approach provides two important advantages. First, a data-item is *not* removed from SRAM as soon as leaves MRU position, instead it is removed from SRAM only when it becomes sufficiently old and hence, is not likely to be reused. This avoids unnecessary migrations. Second, we update the value of Z after every I cycles, and thus, AYUSH can easily account for variation in cache requirements of different applications in different intervals.

As Z decreases, aggressiveness of migrations is increased. Hence, too small value of Z may lead to frequent migrations and harm performance in LLC-sensitive applications. Hence, we impose a lower limit on Z as follows: $Z \leftarrow$ MAX(Z, L), where Z on the right hand side is as computed above and L ($\geq W_S$) is a tunable parameter. Increasing L reduces the number of migrations, which lowers the energy loss at the cost of reduced lifetime enhancement.

Algorithm 1 presents the algorithm for handling a write-hit. No migration operation is required on a write to an SRAM block (lines 4-6). On a write to an NVM block, the algorithm checks for a possible migration. First, the LRU among SRAM blocks is selected (lines 9-10) and its LRU_age is compared with that of NVM block being written and the associativity requirement of the application (lines 11-12). Note that '1' is added on line 11 since the indexing begins with 0. If the SRAM block stores older data, the data are migrated to NVM block and the newly arrived data are stored in the SRAM block (lines 12-14). Otherwise, a migration operation is not performed (line 16).

Algorithm 1: Algorithm for handling a write-hit in set-index i and deciding about possible migration operation.

```
1  //Here index corresponds to physical-order and
   age corresponds to LRU-order
2  indexOfWritten ← index of write-hit block (found by tag matching)
3  noDataMigration ← FALSE
4  if indexOfWritten < W_S
5  |  noDataMigration ← TRUE //Write in SRAM block
6  else
7  |  //Check for possibility of migration
8  |  ageOfWritten ← LRU_Age[indexOfWritten]
9  |  indexOfSRAMLru ← index of LRU among SRAM ways
10 |  ageOfSRAMLru ← LRU_Age[indexOfSRAMLru]
11 |  if ageOfSRAMLru > ageOfWritten AND (ageOfSRAMLru +1)≥
   |  Z
12 |  |  //Found an SRAM-way which would store old
   |  |  data if not migrated
13 |  |  Migrate cacheData[i][indexOfSRAMLru] to
   |  |  cacheData[i][indexOfWritten]
14 |  |  Write new data to cacheData[i][indexOfSRAMLru]
15 |  else
16 |  |  noDataMigration ← TRUE
17 |  end
18 end
19 if noDataMigration
20 |  Write new data to cacheData[i][indexOfWritten] and update
   |  LRU-stack
21 end
```

C. Salient features of AYUSH

AYUSH migrates data only on write-hits and not on read-hits. Thus, by not moving read-intensive blocks into SRAM, AYUSH makes best use of SRAM ways for improving cache lifetime. By proactively migrating write-intensive data to SRAM ways, AYUSH adapts to changing working set much faster than LRU policy. NVM cache lifetime enhancement techniques use either in-cache data-movement [4, 13, 15, 16, 23, 24] or data-invalidation [5, 25]. AYUSH uses in-cache data movement, which does not increase writes to main memory and thus, it does not harm performance or energy efficiency. The limitation of data movement is that during the movement, the cache port gets blocked.

The wear-leveling techniques can be divided based on their granularity, as either way-level [4, 5], set-level [5, 24–26] and memory-cell level [6]. Among these, AYUSH performs way-level (also called intra-set) wear-leveling. As we show in Section V, several workloads show large intra-set write-variation which increases further with increasing associativity. Hence, way-level wear-leveling is very effective in improving the lifetime of cache. By migrating data within the same set and not across sets, AYUSH does not affect the cache set-decoding (unlike [24]). Also, it does not require including set-index bits as part of the tag (unlike [26]).

Compiler-based techniques [10, 14] work by detecting access patterns of certain algorithms/data-structures and inserting special instructions to guide data placement/migration. However, compiler hints may be misleading due to input variation and lack of runtime information. Further, with rising core count, number of program combinations increase rapidly and thus, use of offline profiling becomes infeasible. By contrast, AYUSH uses only runtime information and hence, it works well regardless of the number of cores.

III. Implementation and Overhead Assessment

Implementation of SRAM-NVM hybrid cache: The physical design of hybrid caches can be done using 3D stacking [4, 10, 15, 16, 23, 27] where the SRAM and NVM layers are stacked using through silicon vias (TSVs). This facilitates denser form factor, smaller footprint and cost-efficient integration of heterogeneous technologies.

Storage overhead: Similar to [4, 23], we assume that the migrated data-item is temporarily stored in a buffer. The buffer uses E entries, each having a capacity equal to cache line size. For single-core system E = 256 and for dual-core E = 512 and so on. This is expected to be sufficient since due to inter-set access-variation and other constraints in Algorithm 1, not all sets are expected to perform migrations simultaneously. The Z value is stored in a 5-bit counter. For a W-way set-associative cache, we use W counters, each of which is assumed to use 40 bits, which is sufficient since these counters are reset after every I cycles (we use $I \leq$ 1M). Thus, the overhead of AYUSH, as a percentage of L2 cache size can be computed as

$$Overhead = \frac{W \times 40 + E \times B + 5}{S \times W \times (B + G)} \times 100 \qquad (1)$$

As an example, for a 4MB, 16-way cache with single-core system, the $Overhead$ is only 0.37% of the L2 cache, which is very small. The counters are not stored in NVM and hence, they do not face endurance issue.

Latency and energy overhead: We ignore the energy overhead of counters and buffer, since it is orders of magnitude smaller than that of L2 + main memory (which we model in Section IV-C). On a migration operation, writes to SRAM and NVM take place sequentially. We assume that in addition to the latency of writing on SRAM and NVM blocks (shown in Table II), each migration takes 6 cycles (4 cycles for transfer of 64B data over 32B bus to and from the buffer and 2 cycles for updating LRU-age). Similarly, in addition to energy consumed in writing SRAM and NVM blocks, each migration consumes 0.5 nJ energy.

Note that due to the large value of I, periodically updating Z does not affect the performance. AYUSH performs migrations only on write operations which are not performance-critical and due to relatively large write-latency of NVMs, the latency of updating LRU (etc.) can be overlapped with that of writing on NVM block. Since the number of SRAM ways is small (e.g. typically 2, 3, 4 etc.), the LRU among SRAM ways can be found with low latency. Furthermore, a small increase in the latency of LLC is easily hidden using techniques such as out-of-order execution, write-buffers etc.

Use of LRU replacement policy: AYUSH assumes use of LRU (also called true LRU [28]) replacement policy, as also assumed by previous works [4, 5, 5, 17, 19]. For a W-way cache set, LRU uses $W \times \log_2(W)$ bits storage, compared to W bits in (MRU-based) pseudo-LRU replacement policy [29]. For a 16-way cache, this leads to a difference of 3 bits per block. Accounting for this, above mentioned $Overhead$ changes from 0.37% to 0.92%, which is still small. Despite its slightly higher overhead, LRU policy has some key advantages. Unlike pseudo-LRU and FIFO replacement policies, LRU policy does not suffer from domino effect [30] and has better predictability [30], which is very important for mission-critical and real-time systems. It has also been implemented in caches in commercial processors, such as the 32MB 16-way L2 cache in IBM's Blue Gene/Q system [28]. These facts support our choice of LRU policy.

IV. Experimental Methodology

A. Simulation platform and workloads

We use the interval-core model in Sniper x86-64 simulator with a processor frequency of 2GHz. Both L1 instruction/data caches have 32KB size with 4-way associativity and 2 cycle latency. L2 cache configuration is shown in Section V. The L1 caches are private to core and L2 cache is shared among cores and is inclusive of L1 caches. All caches use LRU, write-back, write-allocate policy. Main memory latency is 220 cycles. For single, dual and quad-core systems, the peak memory bandwidth is 10, 15 and 25GB/s, respectively and queue contention is also modeled.

L2 cache parameters for 16-way SRAM-only and ReRAM-only caches are obtained using NVSim [9], assuming sequential cache access, 32nm process and write energy-delay-product optimized cache design. These parameters are shown in Table II. For the hybrid cache, it is assumed that the miss latency and miss energy are same as that in a ReRAM cache. The hit and write energy/latency are incurred depending on whether the read or write operation is served from an SRAM or a ReRAM region and the leakage power scales linearly with the number of ways designed using SRAM and ReRAM. We use all 29 benchmarks from SPEC CPU2006 suite with *ref* inputs and 6 benchmarks from HPC field (shown as italics in Table III) as single-core workloads. Using these, we randomly create 18 dual-core and 9 quad-core multiprogrammed workloads, such that each benchmark is used exactly once, except for completing the left-over group. Table III shows the workloads.

Table II
L2 cache parameters for SRAM-only and ReRAM-only caches (Lat = latency, En = Energy, Lk = leakage)

	SRAM			ReRAM		
	4 MB	8 MB	16 MB	4 MB	8 MB	16 MB
Hit Lat(ns)	2.13	3.38	6.57	5.12	5.90	5.97
Miss Lat(ns)	0.44	0.52	0.76	1.65	1.68	1.74
Write Lat(ns)	1.08	1.67	3.49	22.18	22.67	22.53
Hit En(nJ)	0.366	0.487	0.61	0.537	0.602	0.662
Miss En(nJ)	0.009	0.013	0.016	0.187	0.188	0.190
Write En(nJ)	0.341	0.457	0.58	0.827	0.882	0.957
Lk Power(W)	0.389	0.742	1.389	0.037	0.083	0.123

Table III
Single, dual and quad-core workloads used

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), Xb(xsbench)*
GmDl, AsPo, GcGa, BzXa, LsLb, GkCo, OmGr, NdCd, CaTo SpBw, LqPo, SjWr, PeZe, HmH2, SoMi, McLu, NeAm, MkXb
AsGaXaLu, GcBzGrTo, CaWrMkMi, LqCoMcBw, LsSoSjH2 PeZeHmDl, GkPoGmNd, LbOmCdSp, AmXbNeGa

B. Comparison with other techniques

We compare AYUSH with the following three techniques.

HDM: Sun et al. [15] present a technique for SRAM-NVM hybrid caches, which we refer to as HDM (hot-data-migration). In HDM technique, a data-item in NVM is migrated to SRAM when it is accessed by two successive write operations. Sun et al. [15] show results with a single SRAM way in a 32-way cache and hence, do not detail the procedure for finding an SRAM target for migration when there are multiple SRAM ways. We experimented with two policies for finding a suitable SRAM target, viz. LRU (the

LRU SRAM is chosen as target) and round-robin (different SRAM ways are chosen in round-robin) and selected LRU policy since it provides higher improvement in lifetime.

RHCA: RHCA [4] technique uses a sticky bit for each SRAM line and a saturating counter for each NVM line. A hit to SRAM line sets its sticky bit. On a hit to NVM line, its counter is incremented by one and if it reaches the threshold value, it is swapped with the LRU SRAM line if the sticky bit of the LRU SRAM is 0. Effectively, sticky bit protects an SRAM line once and thus, avoids unnecessary migrations. More details can be found from the original paper [4]. Following Wu et al. [4], we experimented with the threshold values of 1, 2 and 3 and chose the value of 3 since it gives the largest improvement in lifetime.

PoLF: PoLF [5], a technique for mitigating intra-set write-variation, has been proposed for NVM-only caches. For the hybrid cache, we apply PoLF algorithm in NVM ways only, thus PoLF will achieve wear-leveling in NVM ways and aim to displace the hot data from NVM ways only and not from SRAM ways. Experiments with PoLF confirm that the techniques proposed for NVM-only caches may not sufficiently enhance the lifetime of a hybrid cache. In PoLF, after FT (flush threshold) number of write hits in the NVM portion of cache, a write-operation is skipped; instead, the data-item is written-back to memory and the cache-block is invalidated, without updating the LRU-age information. Since the flushed data-item is expected to be hot, the next time it will be loaded in an LRU block (which is expected to be cold) leading to intra-set wear-leveling. The choice of FT values are discussed in Sections V-A and V-B.

Qualitative Comparison: PoLF uses data-invalidation, while other techniques use data-migration. Before migration, RHCA checks the sticky bit of SRAM way and AYUSH checks its LRU_age, while HDM does not perform such a check. PoLF is proposed for NVM-only caches, while others are proposed for hybrid caches. PoLF requires a single global counter, AYUSH requires per-way counters, HDM requires per-set counters and RHCA requires per-block counters, thus they have different implementation overheads. RHCA algorithm is triggered on both read and write accesses, while other algorithms are triggered on only write accesses. Different techniques utilize different properties of cache and application, such as access frequency and recency. Thus, by choosing these three techniques, we provide a comparison with well-known techniques which cover a broad spectrum of cache management approaches. Based on these results, we additionally compare AYUSH with two other techniques [16, 23], refer Section V-A.

C. Evaluation Metrics

We show the results relative to the baseline (mentioned in Section II) on following metrics: **a)** relative cache lifetime where the lifetime is defined as the inverse of maximum writes on any NVM (and not SRAM) cache blocks **b)** Coefficient of intra-set write-variation (IntraV) [5] for NVM blocks **c)** decrease in WPKI (writes-per-kilo instructions) for NVM blocks **d)** weighted speedup [18], hereafter called relative performance **e)** percentage energy loss and **f)** number of migration operations (C_{migr}) for AYUSH, HDM and RHCA and flush operations (C_{flush}) for PoLF.

We model the energy of main memory, L2 cache (refer Table II) and data migrations (refer Section III). The dynamic energy and leakage power of main memory are 0.18W and 70nJ/access, respectively [18]. The benchmarks are fast-forwarded for 10B instructions and each workload is simulated till the slowest application executes 350M instructions. An early completing benchmark is allowed to run, but its IPC is recorded only for the first 350M instructions, following well-established simulation methodology [18, 31]. If the maximum writes on any NVM block is zero in the baseline, we assume lifetime improvement as 1×. In our experiments, this only occurs with gamess and amg2013 for a few configurations. Relative lifetime and relative performance are averaged using geometric mean, while remaining metrics are averaged using arithmetic mean.

The NVM cache lifetime can be improved by using wear-leveling (e.g. [5]) and/or write-minimization (e.g. [32]). IntraV and decrease in WPKI show the efficacy (or contribution) of wear-leveling and write-minimization, respectively. The metrics **(a)**, **(b)** and **(c)** account for the worst-case, average-case and total writes on NVM, respectively and thus, together, they evaluate a technique in a comprehensive manner. These metrics have also been used by other researchers [5, 13, 15, 26]. Further, fair speedup [18] values were found to be close to weighted speedup, thus, AYUSH does not introduce unfairness. For brevity, we omit these results.

V. Results and Analysis

A. Results with default parameters

Figures 1, 2 and 3 and Table IV show the results. Here W=16, W_S=3 and the size of cache for single, dual and quad-core systems are 4MB, 8MB and 16MB, respectively. This ratio of core-to-LLC size is in line with commercial processors, such as Intel Itanium and Xeon models [33]. The results on AYUSH are obtained using the following parameters: α =60%, $L = W_S$=3, I =500K for single-core, I =250K for dual-core and I =125K for quad-core system.

Table IV
C_{migr} OR C_{flush} FOR DIFFERENT TECHNIQUES

	AYUSH	HDM	RHCA	PoLF
N =1	624K	154K	1224K	223K
N =2	1753K	324K	2891K	375K
N =4	4253K	943K	7166K	575K

AYUSH: Firstly, AYUSH provides the largest improvement in lifetime for all system-configurations and scales very well with increasing number of cores. For single, dual and quad-core systems, AYUSH improves cache lifetime by 6.90×, 24.06× and 47.62×, respectively. With HDM, these

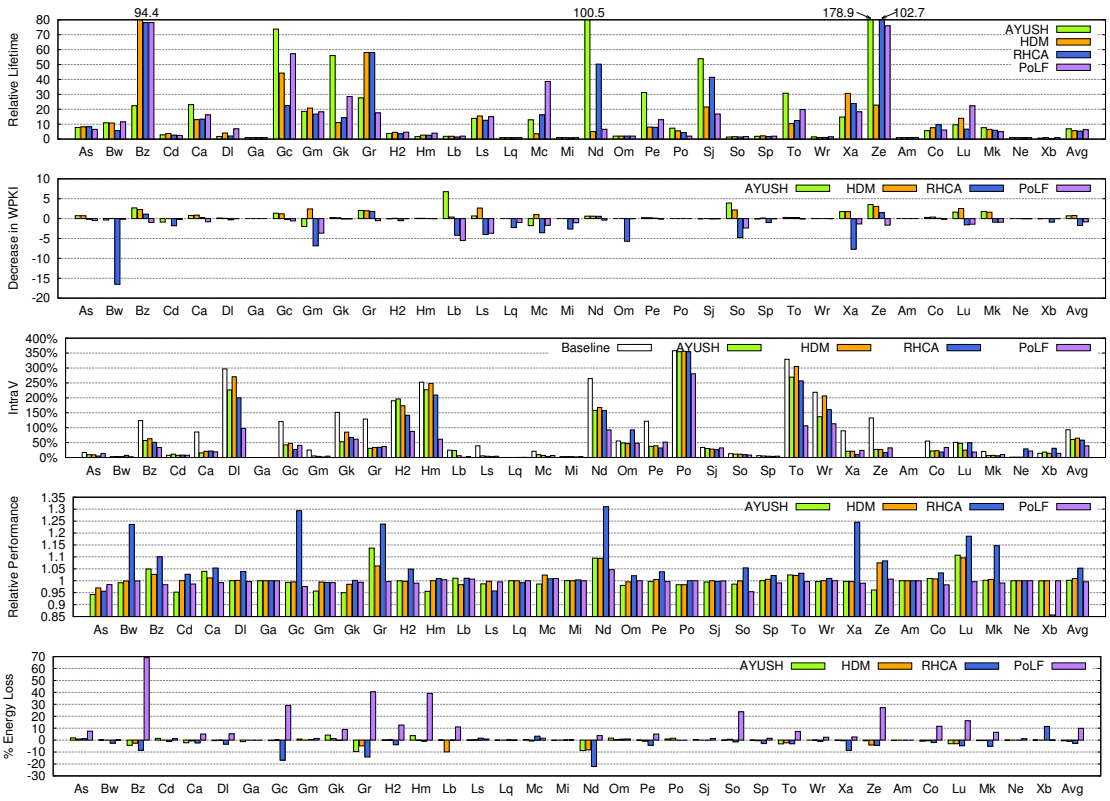

Figure 1. Results on single-core system (4MB L2). For lifetime, decrease in WPKI and performance, more is better. For IntraV and energy loss, less is better. Per-workload results for C_{migr}(or C_{flush}) are omitted for brevity, they are summarized in Table IV.

values are 5.66×, 17.30× and 20.14×, respectively; with RHCA, they are 5.37×, 14.57× and 29.98×, respectively and with PoLF, they are 6.33×, 17.11× and 26.23×, respectively. For several workloads, AYUSH improves lifetime by one or even two orders of magnitude. For several workloads, IntraV in baseline is very high (up to 350%) which shows the need of a wear-leveling technique such as AYUSH.

The lifetime improvement is decided by many factors, e.g. characteristics of write-stream, IntraV present in baseline etc. For low write-intensity workloads, most writes are already absorbed in SRAM, and hence, the decrease in NVM WPKI is negligible, e.g. povray(Po), wrf(Wr) etc. Opposite is true for lbm(Lb), soplex(So), bzip2-xalancbmk(BzXa) etc. For workloads with high IntraV, large enhancement in lifetime is seen, e.g. namd(Nd), tonto(To), gcc-gamess(GcGa), amg2013-xsbench-nekbone-gamess(AmXbNeGa) etc. Converse is true for libquantum(Lq), milc(Mi), leslie-lbm(LsLb), libquantum-comd-mcf-bwaves(LqCoMcBw) etc. Thus, AYUSH performs both write-minimization and wear-leveling since it redirects the hot data from *MRU among NVM ways* to SRAM ways.

For all parameter configurations evaluated in this paper (including those in Sections V-B and V-C), AYUSH provides an average weighted speedup in range [0.99×, 1.01×]. Clearly, on average, AYUSH maintains performance, since it does not increase the miss-rate and the increased writes to fast SRAM offset the latency overhead of data migration. The limitation of AYUSH is that it causes larger number of migrations than HDM, which leads to overhead. For some workloads, such as gemsFDTD(Gm), Mc, LqPo, LqCoMcBw etc. AYUSH increases the WPKI. This is due to the large number of migrations which harm the performance.

HDM: HDM migrates a data-item if it is accessed by *two successive writes*. However, due to reduced locality in LLC, successive writes to a block are not common, which is also confirmed by small C_{migr} value of HDM (refer Table IV). Further, in multicore systems with shared LLC, interleaving of access streams from multiple applications reduces the locality of the final LLC access stream [18]. Hence, successive writes to the same block become infrequent with increasing number of cores. These factors limit the effectiveness of HDM. By comparison, AYUSH uses the LRU-age of the data and the associativity requirement of workload to decide a migration and hence, it is expected to be very effective for computing systems with tens of cores. The benefit of HDM is that it does not increase WPKI for any workload.

RHCA: RHCA migrates both read and write intensive data to SRAM ways, while other techniques migrate only write intensive data. Due to this, largest number of migrations happen in RHCA and RHCA achieves largest improvement in performance. The disadvantage of this is that a migration happening on a read-access introduces an extra write,

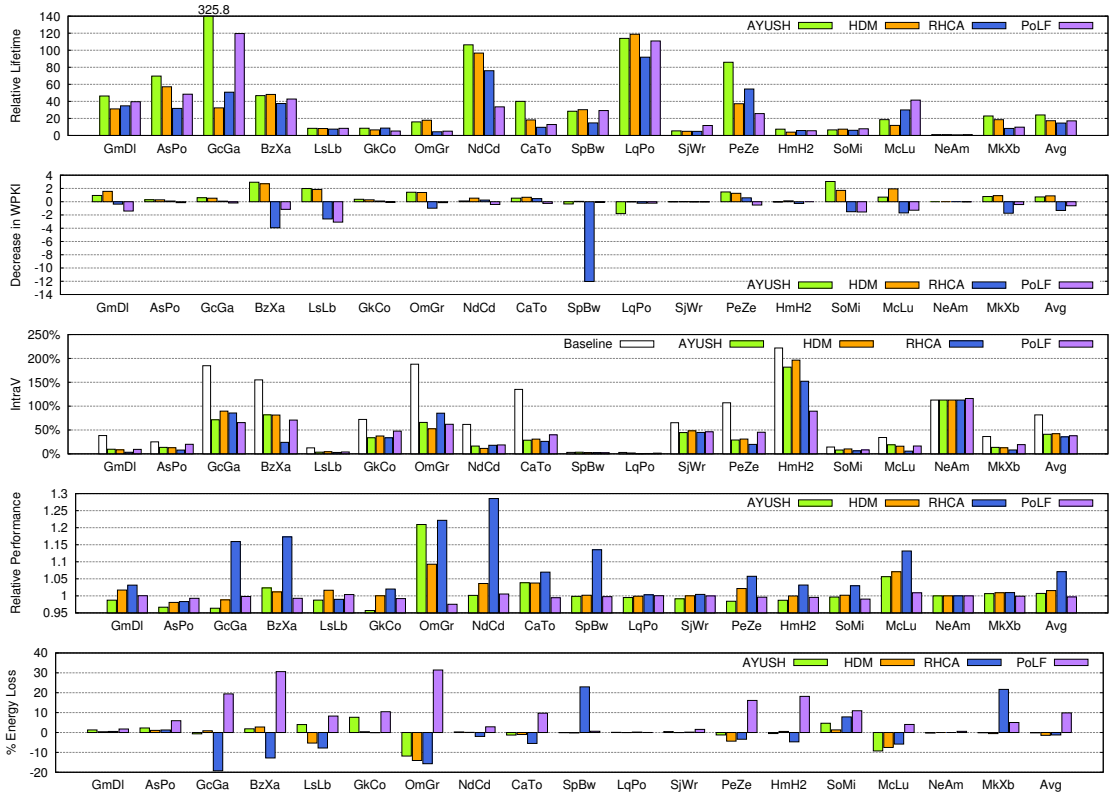

Figure 2. Results on dual-core system (8MB L2)

which worsens the write-endurance issue. Thus, migrations happen even for write-unintensive workloads, and read-intensive data compete with write-intensive data for SRAM space. For several workloads, read-intensive data even migrate write-intensive data from SRAM ways back to NVM ways. Hence, RHCA *increases the writes to NVM ways*, as shown from the negative value of 'decrease in WPKI', especially for workloads such as bwaves(Bw), xalancbmk(Xa), sphinx-bwaves(SpBw) and LqCoMcBw. This reduces the effectiveness of RHCA in improving lifetime.

Further, since reads are performance-critical, overhead of migration in read-accesses may not be easily hidden. Clearly, careful cache management is required to strike the right balance between lifetime improvement and performance. With N = 1, 2 and 4, RHCA achieves an average relative performance of 1.05×, 1.07× and 1.06×, respectively. Thus, while providing a performance improvement of less than 10%, migrations on read accesses may reduce the lifetime improvement of a technique. The net energy loss is decided by multiple factors, e.g. performance improvement and increased writes to NVM etc. With N = 1 and 2, the improved performance compensates for extra energy due to increased NVM writes. However, with increasing core-count, the write-intensity also increases and since NVM writes are costly, a net energy loss is seen for N = 4.

PoLF: Due to the use of data-invalidation, PoLF improves lifetime at the cost of energy and performance loss. To achieve a reasonable balance between these, we choose its FT value as follows. For each N and W_S, we choose an FT value which leads to largest lifetime improvement while incurring an average energy loss of no more than 10% since allowing larger energy loss leads to unacceptable worst-case energy losses (e.g. 80%). With W_S=3, for N = 1, 2 and 4, we obtained FT = 5, 7 and 11, respectively.

As seen from the results, even with 10% average energy loss, lifetime improvement with PoLF remains smaller than that of AYUSH. Write-intensive workloads show a large energy loss, e.g., bzip2(Bz) incurs an energy loss of 69% and gromacs(Gr), hmmer(Hm), bzip2-xalancbmk(BzXa) and gcc-bzip2-gromacs-tonto(GcBzGrTo) incur more than 30% energy loss. Future computing systems may have NVM (e.g. PCM) main memory which have higher latency and lower bandwidth and write-endurance compared to DRAM. Hence, in an attempt to increase "cache" lifetime, PoLF will degrade "main memory" lifetime. Further, with increasing core count, write-pressure increases and hence, AYUSH's approach of using SRAM to absorb writes greatly helps in reducing NVM writes, while PoLF's approach of using wear-leveling alone becomes insufficient. Clearly, with increasing core count, the utility of an NVM-only technique for improving the lifetime of a hybrid cache will decrease.

The advantage of PoLF is that it only uses a single global counter. This, however, also leads to its most crucial limitation, that it flushes the cache blindly. For workloads

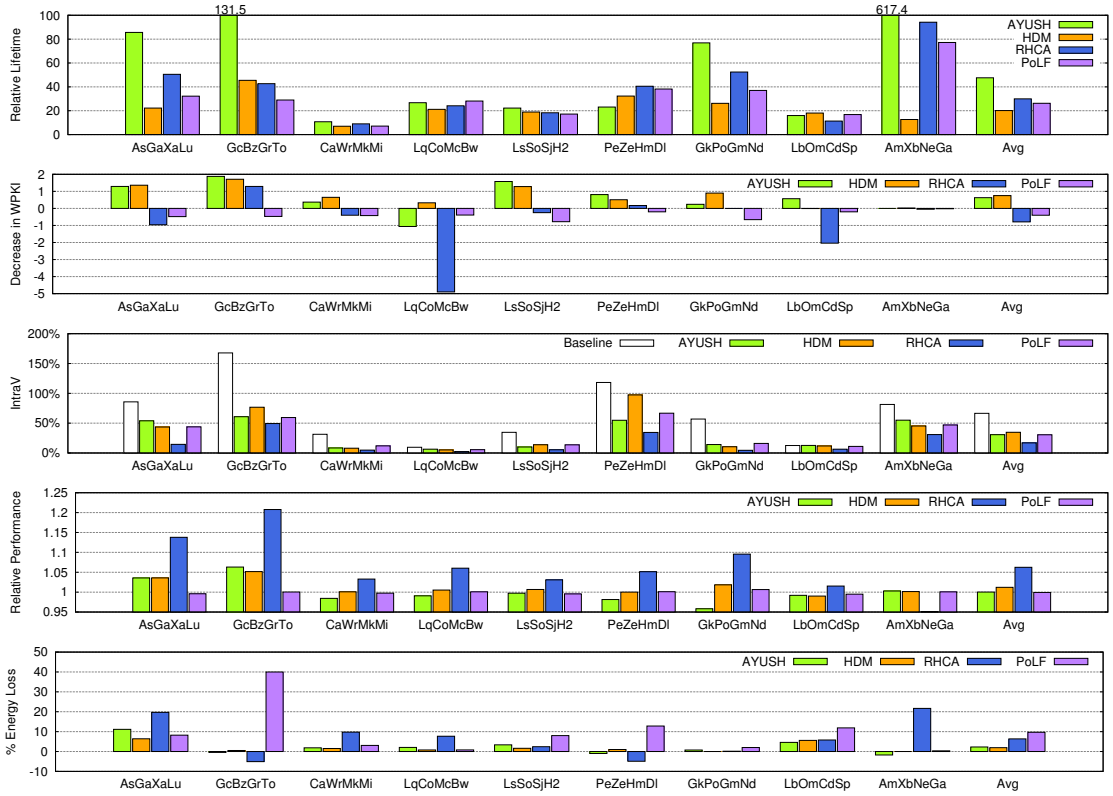

Figure 3. Results on quad-core system (16MB L2)

where write-intensity is high, but write-variation is small, PoLF flushes very large number of blocks, without achieving corresponding improvement in lifetime, e.g. for lbm(Lb) and soplex(So), the IntraV in baseline is 24.3% and 12.7%, respectively which are very small. For these workloads, PoLF flushes 2449K and 1173K blocks while improving lifetime by only 2.00× and 1.73×, respectively. Also, with PoLF, even a newly installed block may be flushed, thus, it may not necessarily flush a hot data-item. Clearly, the primary limitation of PoLF is its ineffectiveness in identifying hot-data and *not* the use of data-invalidation.

Qualitative comparison with similar techniques: Sharifi et al. [16] use the data migration scheme of RHCA with a threshold value of 2. In the scheme of Lee et al. [23], when a hit occurs in NVM, it migrates the data to SRAM. Thus, it is also similar to RHCA with a threshold value of 1 and no sticky bit for SRAM region. Since MRU way captures only a small fraction of hits in LLC, this technique will introduce large number of migrations. Thus, based on the results and discussion presented above and in Section II-A, we expect AYUSH to perform better than these schemes also.

B. Results with different number of SRAM ways

We now experiment with W_S (number of SRAM ways) value of 2 and 4. For PoLF, with W_S=2, for N = 1, 2 and 4, we obtained FT = 7, 9 and 13, respectively and for W_S=4, FT values obtained were 4, 6 and 9, respectively. The results are summarized in Table V. Clearly, for all cases, AYUSH provides largest improvement in lifetime. On increasing W_S, with AYUSH, for single-core system, most writes are absorbed in SRAM ways themselves which reduces the scope of improving NVM lifetime. However, for dual and quad-core systems, the write-pressure is higher and hence, increasing W_S presents higher opportunity of migrating hot data into SRAM blocks, leading to higher enhancement in lifetime. For HDM, with increasing W_S, NVM blocks store progressively older (i.e. less recent) blocks only and successive writes to them become progressively infrequent, hence, its lifetime improvement reduces with increasing W_S.

For RHCA, with increasing W_S, more SRAM ways become available for storing hot data and thus, the competition for SRAM ways reduces. This improves the lifetime improvement and vice versa. For experiments with PoLF, we have fixed the energy loss bound, hence, with increasing W_S, NVM ways and NVM writes are reduced. Due to this, smaller FT value can be used and there is a marginal increase in the lifetime enhancement. If, instead, the FT was kept fixed, then on increasing W_S, smaller energy loss and smaller lifetime enhancement will be observed.

C. Parameter Sensitivity Results

We now focus exclusively on AYUSH and present results for different parameters. Each time, only one parameter is changed from the default parameters. Results are sum-

Table V
PARAMETER SENSITIVITY RESULTS WITH DIFFERENT NUMBER OF SRAM WAYS (REL. PERF. = RELATIVE PERFORMANCE). INTRAV FOR BASELINE IS PRESENTED IN ROW-HEADER AND THAT FOR EACH TECHNIQUE IS PRESENTED IN COLUMN 4.

	Rel. Lft.	⇓ WPKI	IntraV	Energy Loss %	Rel. Perf.	C_{migr} or C_{flush}
Single-core System, W_S =2, IntraV (baseline) = 118.3						
AYUSH	7.73	1.02	85.5	0.04	1.00	643K
HDM	6.24	0.84	86.7	-0.70	1.01	160K
RHCA	3.73	-1.50	82.9	-3.25	1.06	1344K
PoLF	6.07	-0.70	54.6	9.95	0.99	180K
Single-core System, W_S =4, IntraV (baseline) = 88.8						
AYUSH	6.70	0.39	59.0	-0.75	1.00	602K
HDM	4.96	0.69	63.0	-1.09	1.01	142K
RHCA	5.48	-1.90	57.8	-2.31	1.05	1133K
PoLF	6.28	-0.90	40.7	8.86	1.00	246K
Dual-core System, W_S =2, IntraV (baseline) = 87.6						
AYUSH	24.21	0.98	41.3	0.12	1.00	1779K
HDM	17.67	0.94	45.1	-1.12	1.01	354K
RHCA	7.47	-1.35	38.5	-1.66	1.08	3240K
PoLF	15.60	-0.51	40.4	9.93	1.00	326K
Dual-core System, W_S =4, IntraV (baseline) = 76.6						
AYUSH	24.51	0.45	42.1	0.09	1.01	1734K
HDM	14.74	0.78	41.4	-1.28	1.01	301K
RHCA	17.76	-1.40	35.3	-0.95	1.06	2614K
PoLF	17.27	-0.66	38.0	8.72	1.00	390K
Quad-core System, W_S =2, IntraV (baseline) = 73.4						
AYUSH	40.53	0.81	32.3	1.91	0.99	4312K
HDM	21.86	0.83	36.6	1.61	1.01	1009K
RHCA	17.52	-0.87	21.8	5.63	1.07	8160K
PoLF	18.63	-0.37	32.8	10.00	1.00	541K
Quad-core System, W_S =4, IntraV (baseline) = 62.5						
AYUSH	52.27	0.42	30.4	2.45	1.00	4175K
HDM	19.86	0.70	31.4	1.51	1.01	855K
RHCA	40.44	-0.80	14.8	6.36	1.06	6423K
PoLF	28.11	-0.46	28.7	9.45	1.00	629K

marized in Table VI. Relative performance is omitted for brevity, since AYUSH maintains performance.

Algorithm parameters: AYUSH algorithm has three tunable parameters, viz. α, I and L. The α value affects value of Z. On increasing α, computed values of Z in different intervals are increased which reduces the number of migrations. Although the migration operations move hot data to SRAM, they also harm performance and thus introduce extra writes. Also note that in addition to α, W_S and L also affect Z such that $Z \geq W_S$ and $Z \geq L$. Due to this, depending on the characteristics of different workloads, change in α has different effect on lifetime improvement. Thus, although with increasing α, number of migrations decrease monotonically, the change in average lifetime improvement is not monotonic (Table VI). The same is also true for I, the interval after which Z value is recomputed.

L shows the lower bound on Z and if Z is already sufficiently large or if L is small, L does not play any role in deciding value of Z. The full effect of L is seen on increasing it to 5, since it increases Z which leads to a sharp reduction in the number of migrations. This reduces energy loss, although it also reduces the improvement in lifetime. Thus, by controlling L, a tradeoff between the aggressiveness and overhead of the algorithm can be achieved.

These results show that AYUSH is not very sensitive to

Table VI
PARAMETER SENSITIVITY STUDY (REL. LFT. = RELATIVE LIFETIME).

	Rel. LfT.	⇓ WPKI	IntraV		Energy Loss %	C_{migr} (K)
			Base	AYUSH		
Single-core System						
Default	6.90	0.70	93.0	60.4	-0.58	624
α =40%	7.55	0.68	93.0	60.0	-0.46	657
α =50%	7.08	0.69	93.0	60.2	-0.51	644
α =70%	6.75	0.71	93.0	60.3	-0.51	602
α =80%	6.77	0.72	93.0	60.7	-0.48	562
I=250K	7.61	0.72	93.0	60.1	-0.47	610
I=1M	6.60	0.66	93.0	60.6	-0.55	641
L = 4	6.90	0.70	93.0	60.4	-0.58	624
L = 5	3.49	0.58	93.0	71.5	-0.68	580
8-way	6.61	-0.06	64.3	48.7	-0.53	561
32-way	8.97	1.14	133.2	87.6	-0.90	637
2MB	6.61	0.50	78.6	53.2	-0.16	662
8MB	9.33	0.47	128.2	95.9	-0.68	408
Dual-core System						
Default	24.06	0.72	81.7	41.1	-0.15	1753
α =40%	24.03	0.74	81.7	40.9	-0.08	1854
α =50%	23.76	0.73	81.7	40.8	0.02	1813
α =70%	22.32	0.71	81.7	41.3	-0.48	1682
α =80%	20.76	0.68	81.7	42.3	-0.58	1569
I=125K	26.14	0.74	81.7	41.1	-0.06	1743
I=500K	23.34	0.73	81.7	41.1	0.01	1784
L = 4	24.13	0.72	81.7	41.1	-0.14	1756
L = 5	9.59	0.65	81.7	48.2	-0.63	1576
8-way	24.76	-0.13	62.7	41.5	0.21	1615
32-way	22.74	1.16	106.0	42.3	-0.28	1747
4MB	14.52	0.44	56.6	22.2	0.04	1847
16MB	35.25	0.94	117.8	77.1	0.17	1359
Quad-core System						
Default	47.62	0.63	66.5	30.7	2.29	4253
α =40%	48.96	0.65	66.5	30.6	2.37	4466
α =50%	50.40	0.64	66.5	30.7	2.34	4387
α =70%	48.23	0.59	66.5	30.9	2.15	3851
α =80%	46.22	0.58	66.5	31.0	2.07	3491
I=62.5K	46.12	0.63	66.5	30.7	2.28	4228
I=250K	49.68	0.63	66.5	30.7	2.30	4295
L = 4	48.64	0.63	66.5	30.7	2.30	4259
L = 5	16.15	0.61	66.5	35.4	2.06	4080
8-way	36.52	0.37	48.7	30.0	2.82	3136
32-way	46.36	0.94	89.5	34.4	1.46	4265
8MB	30.47	0.46	53.2	21.7	2.21	4516
32MB	65.99	0.81	91.4	47.2	-1.46	3266

choice of algorithm parameters in their reasonable range of values. Regarding setting the parameters, L can be set to be equal to W_S, the number of SRAM ways. I scales (reduces) uniformly with increasing core count, since on doubling the number of running cores, nearly the same amount of write accesses come to the cache in half the number of cycles. Value of α can be set to be anywhere between 50% to 70% or close to them. Thus, AYUSH does not require offline profiling or per-workload tuning of its parameters.

System parameters: For a fixed cache capacity, on increasing the cache associativity, conflict misses reduce and only few ways absorb most of the writes. This increases the write-variation as shown from the IntraV values of baseline (Table VI). This can also be understood by considering two extreme cases: a direct-mapped cache which shows zero IntraV and a fully-associative cache which shows largest IntraV. Thus, with increasing associativity, AYUSH provides

larger improvement in lifetime and vice versa.

Since the applications have a fixed working set size, on increasing the cache size, capacity misses reduce and the hit rate increases. Thus, repeated writes to a few block increase which increases the write-variation (IntraV). Thus, an increase in cache size provides higher opportunity of improving lifetime, e.g. for 4-core system with 32MB cache, AYUSH improves lifetime by 66×. Clearly, AYUSH will be very effective for future systems with large sized caches.

VI. Related Work

PCM is generally considered less suitable for designing on-chip caches due to its high latency and low write endurance ($10^8 - 10^9$ writes). However, some researchers study SRAM-PCM or STTRAM-PCM hybrid caches [4, 13], which others study PCM-only caches [6] and propose using PCM as an L4 cache [4]. These proposals, along with the trends of very large LLCs designed with SRAM-alternatives, e.g. 128MB embedded DRAM LLC in Intel's Haswell processor, indicate that PCM may be used in on-chip caches in near future due to its high density. Thus, AYUSH is expected to be useful for such SRAM-PCM hybrid caches.

Although STT-RAM write endurance has been predicted to be greater than 10^{15}, the best endurance value in demonstrated prototypes is only 4×10^{12} [5]. Since process variations may further reduce this value by even 50× [12] and malicious programs can repeatedly write a block to make the system fail, some researchers believe that endurance may be an issue for STT-RAM also [5, 13, 26]. In such a case, AYUSH may be useful for SRAM-STTRAM hybrid caches.

Some techniques minimize NVM writes by using additional data-storage components such as write-buffer [15], by using cache bypassing for blocks which are unlikely to be reused [17] or by avoiding the write-operation for redundant bits [6]. AYUSH can be easily integrated with these techniques and/or inter-set write-variation mitigation techniques to improve the cache lifetime even further.

Similar to several previous works [5, 13, 26], in this paper, we have evaluated 'raw lifetime', which is determined by the first failure of a cache block. To extend this lifetime, AYUSH can be integrated with error-correction and error-tolerance techniques. As an example, spare blocks can be used to store data of failed blocks. The spare block has to receive large number of writes before reaching its limit, and hence, its data-item need not be migrated to SRAM. Instead, remaining NVM blocks in the set can be migrated to SRAM.

VII. Conclusion

In this paper, we presented AYUSH, a technique for improving the lifetime of SRAM-NVM hybrid caches. A comprehensive evaluation using single, dual and quad-core system configurations have shown that AYUSH provides higher cache lifetime than three other techniques and also scales well with increasing number of cores. In future, we plan to integrate AYUSH with error-correction and error-tolerance schemes to further increase the cache lifetime.

References

[1] S. Rusu *et al.*, "Ivytown: A 22nm 15-core enterprise Xeon® processor family," in *ISSCC*, 2014, pp. 102–103.

[2] S. Li *et al.*, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," *ICCAD*, pp. 694–701, 2011.

[3] J. S. Vetter *et al.*, "Opportunities for nonvolatile memory systems in extreme-scale high performance computing," *CiSE*, 2015.

[4] X. Wu *et al.*, "Hybrid cache architecture with disparate memory technologies," in *ISCA*, 2009, pp. 34–45.

[5] J. Wang *et al.*, "i2WAP: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations," *HPCA*, 2013.

[6] Y. Joo *et al.*, "Energy-and endurance-aware design of phase change memory caches," *DATE*, pp. 136–141, 2010.

[7] T.-Y. Liu *et al.*, "A 130.7mm^2 2-layer 32Gb ReRAM memory device in 24nm technology," in *ISSCC*, 2013, pp. 210–211.

[8] http://www.anandtech.com/show/8727/nvidia-ibm-supercomputers.

[9] X. Dong *et al.*, "NVsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, 2012.

[10] Y. Li *et al.*, "A software approach for combating asymmetries of non-volatile memories," in *ISLPED*, 2012, pp. 191–196.

[11] S.-S. Sheu *et al.*, "A 4Mb embedded SLC Resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability," in *ISSCC*, 2011, pp. 200–202.

[12] W. Zhang *et al.*, "Characterizing and mitigating the impact of process variations on phase change based memory systems," *MICRO*, 2009.

[13] Y. Joo *et al.*, "A Hybrid PRAM and STT-RAM Cache Architecture for Extending the Lifetime of PRAM Caches," *CAL*, 2013.

[14] Q. Li *et al.*, "Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache," *LCTES*, 2012.

[15] G. Sun *et al.*, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *HPCA*, 2009, pp. 239–249.

[16] A. Sharifi *et al.*, "Automatic Feedback Control of Shared Hybrid Caches in 3D Chip Multiprocessors," in *PDP*, 2011, pp. 393–400.

[17] Z. Wang *et al.*, "Adaptive placement and migration policy for an STT-RAM-based hybrid cache," in *HPCA*, 2014, pp. 13–24.

[18] S. Mittal *et al.*, "MASTER: A Multicore Cache Energy Saving Technique using Dynamic Cache Reconfiguration," *TVLSI*, 2014.

[19] A. Valero *et al.*, "Analyzing the optimal ratio of SRAM banks in hybrid caches," in *ICCD*, 2012, pp. 297–302.

[20] B. Quan *et al.*, "Prediction table based management policy for STT-RAM and SRAM hybrid cache," in *ICCCT*, 2012, pp. 1092–1097.

[21] J. Ahn *et al.*, "Write intensity prediction for energy-efficient non-volatile caches," in *ISLPED*, 2013, pp. 223–228.

[22] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *MICRO*, 2011, pp. 329–338.

[23] S. Lee *et al.*, "Runtime Thermal Management for Three-Dimensional Chip-Multiprocessors with Hybrid SRAM/MRAM L2 Cache," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 2014.

[24] J. Li *et al.*, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *ESTIMedia*, 2011, pp. 19–28.

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

[26] Y. Chen *et al.*, "On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations," *ACM JETC*, 2013.

[27] M. Poremba *et al.*, "DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches," in *DATE*, 2015.

[28] R. A. Haring *et al.*, "The IBM Blue Gene/Q compute chip," *Micro, IEEE*, vol. 32, no. 2, pp. 48–60, 2012.

[29] H. Al-Zoubi *et al.*, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in *ACMSE*, 2004.

[30] R. Wilhelm *et al.*, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *TCAD*, 2009.

[31] S. Mittal *et al.*, "AYUSH: A Technique for Extending Lifetime of SRAM-NVM Hybrid Caches," *Computer Architecture Letters*, 2015.

[32] P. Zhou *et al.*, "Energy reduction for STT-RAM using early write termination," in *ICCAD*, 2009, pp. 264–268.

[33] http://goo.gl/h0hiZG.