AYUSH: A Technique for Extending Lifetime of SRAM-NVM Hybrid Caches

Sparsh Mittal and Jeffrey S. Vetter

Abstract—Recently, researchers have explored way-based hybrid SRAM-NVM (non-volatile memory) last level caches (LLCs) to bring the best of SRAM and NVM together. However, the limited write endurance of NVMs restricts the lifetime of these hybrid caches. We present AYUSH, a technique to enhance the lifetime of hybrid caches, which works by using data-migration to preferentially use SRAM for storing frequently-reused data. Microarchitectural simulations confirm that AYUSH achieves larger improvement in lifetime than a previous technique and also maintains performance and energy efficiency. For single, dual and quad-core workloads, the average increase in cache lifetime with AYUSH is 6.90, 24.06 and 47.62×, respectively.

Index Terms—Non-volatile memory (NVM), hybrid cache, SRAM-NVM cache, device lifetime, write endurance

1 Introduction

With CMOS scaling and increasing system core-count, the size of LLCs in modern processors has grown significantly, for example, Intel's Poulson processor uses 32 MB LLC. Since SRAM has low density and consumes large amount of leakage power, researchers have recently explored non-volatile memories (NVMs) for designing LLCs. NVMs such as resistive RAM (ReRAM) and spin transfer torque (STT-RAM) have several attractive features, such as low leakage power consumption, high density and CMOS compatibility. For example, the size of a typical SRAM and ReRAM cell lies in the range of 125-200F^2 and 4-10F^2, respectively, where F denotes the feature size [1], [8], [12].

A crucial limitation of NVMs, however, is their limited write endurance. For example, while the write endurance of SRAM and DRAM is above 10^{15}, these values for ReRAM and STT-RAM are 10^{11} [5] and 4×10^{12} [2], [16], respectively. Due to process variations, these values may reduce even further. Further, since the existing cache management policies are optimized for performance and not for minimizing write-variation, they may lead to large write-variation, which may lead to failure of cache much before the ideal lifetime assuming uniform write-distribution.

To bring the best of both SRAM and NVM, way-based SRAM-NVM hybrid cache designs [6], [8] have been proposed where a few ways are designed using SRAM and the remaining ways are designed using NVM. The design goal in hybrid caches is to keep the most recently used (MRU) data-items in SRAM blocks to leverage their high write endurance and performance and keep the remaining data-items in NVM blocks to leverage the high density of NVM. However, due to filtering by first level cache, the temporal locality of LLC access stream is reduced [9], [10], [15] 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 have shown that for a 4 MB, 16-way cache with three SRAM and 13 ReRAM ways, the cache lifetime with bzip2 and zeusmp benchmarks is only 28 and 25 days, respectively (more details of experimental setup are provided in Section 4). Clearly, effective techniques are required for fully leveraging the potential of hybrid caches.

In this paper, we present AYUSH (it means *long life* in Sanskrit), a technique for improving the lifetime of SRAM-NVM hybrid caches. AYUSH uses the key idea that since the write-endurance of SRAM is orders of magnitude higher than that of NVM, it can be assumed to be infinite for practical purposes and hence, the writes on NVM can be reduced by redirecting the hot (frequently reused) data into SRAM blocks, which improves the lifetime of the cache. For this, we first estimate the average associativity requirement (say Z) of the application. Then, we assume that an SRAM block with LRU (least recently used)-age greater than Z stores cold data, since it is not likely to be reused. Hence, on a write-hit to an NVM block, if an SRAM is found to store cold data-item, it is swapped with the newly arrived data-item, so that future writes can be redirected to the SRAM block. Microarchitectural simulations have shown that AYUSH achieves larger enhancement in cache lifetime than a previous technique, and incurs negligible loss in performance and energy.

2 Related Work

Cache lifetime enhancement techniques (LETs) can be classified as using either or both of write-minimization (e.g. [3]) and wear-leveling (e.g. [16]). By virtue of redirecting hot data from *MRU among NVM ways* to SRAM ways, AYUSH achieves both NVM write-minimization and NVM wear-leveling. On the basis of their granularity, the wear-leveling techniques can be divided into set-level [6], [16], way-level [16], [17] and memory-cell level [3]. Of these, AYUSH performs way-level wear-leveling.

LETs can also be divided as either using data-invalidation [16] or in-cache data movement [4], [6], [17]. Increased misses due to data invalidation lead to contention, energy loss and endurance issues in main memory and may also negate the purpose of using large LLCs. AYUSH uses in-cache data movement, which incurs smaller overhead. Also, in AYUSH, data migration is done within the same set and not across sets (unlike [6]) and hence, set-decoding and block-lookup are not affected. The limitation of data movement is that cache port gets blocked during the movement. For SRAM-NVM hybrid caches, researchers have proposed techniques which place write-intensive blocks in SRAM to mitigate write endurance problem of NVM [4], [7], [13].

3 Methodology

Notations. We use the following symbols. I denotes the interval size. For the LLC, S, W, B and G denote the number of sets, associativity, cache line (block) size and tag-size, respectively. We assume $B = 64$ byte and $G = 30$ bits.

Our baseline is a way-based hybrid SRAM-NVM cache which uses LRU replacement policy. Baseline cache management policies do not differentiate between SRAM or NVM. W_S $(0 < W_S < W)$ denotes the number of SRAM ways which is fixed at design time. We assume that the first W_S physical ways are designed using SRAM and the remaining ways are designed using NVM. For convenience, we use the term NVM or ReRAM ways interchangeably. The LRU_age of MRU position is 0 and that of LRU position is $W - 1$ (and similarly for other ways). LRU replacement policy naturally tracks LRU_age values, thus they are readily available.

Key idea. Due to filtering by first level cache, the locality of LLC access stream is reduced and hence, a single MRU way designed with SRAM cannot capture a large fraction of hits in the LLC. Our experiments with single-core workloads show that for L1-I and L1-D caches, 98 and 92 percent of hits are absorbed in the MRU way (respectively), but for L2 cache, this value is only 57 percent. A naïve approach towards this is to increase the value of W_S and thus, design a large number of ways using SRAM (e.g. 8 out of 16

• *The authors are with Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN. E-mail: {mittals, vetter}@ornl.gov.*

Manuscript received 13 May 2013; revised 7 July 2014; accepted 26 Aug. 2014. Date of publication 4 Sept. 2014; date of current version 1 Jan. 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2014.2355193

in [15]). However, due to large leakage energy consumption of SRAM, increasing W_S leads to increase in energy consumption of LLC, which may defeat the purpose of using hybrid cache. Instead, we propose using only small number of SRAM ways (e.g. $W_S = 2$, 3, or 4 out of 16) and leveraging data migration approach to increase the number of hits in the SRAM ways.

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 (most recently used) ways [9]. Thus, the associativity requirement (Z) of an application can be computed by seeing the number of hits to different LRU-positions of the cache and Z can be used as a threshold to see if an SRAM block stores cold data. For a W-way cache, we use W write-counters, where the ith counter records the number of hits in the ith LRU-position. If the total number of hits in all ways is H, then we choose Z ($\geq W_S$) such that the total hits in most-recent Z positions are equal to or greater than αH, where $\alpha < 100\%$ is a tunable parameter. The smaller the value of α, more aggressively the migrations take place. The value of Z is updated after an interval of every I cycles.

Algorithm 1. Algorithm for Handling a Write-Hit in Set-Index i and Deciding about Possible Swap Operation

```
1: //Here index refers to physical-order, not LRU-order
2: indexOfWritten ← index of write-hit block (found by tag matching)
3: noDataMigration ← FALSE
4: if indexOfWritten < W_S
5:     noDataMigration ← TRUE //It is a 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 very 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
```

Too small value of Z may lead to frequent migrations and harm performance in LLC-sensitive applications. Hence, we impose a minimum limit on Z as follows: $Z \leftarrow \mathrm{MAX}(Z, L)$, where L ($\geq W_S$) is a tunable parameter and Z on the right hand side is as computed above. By increasing the value of L, the number of migrations can be reduced, which reduces the energy loss at the cost of reduced lifetime enhancement.

Lifetime enhancement algorithm. Algorithm 1 presents the pseudo-code for handling a write-hit. A write to an SRAM block does not trigger migration (lines 4-6). On a write to an NVM block, we first find the LRU among SRAM blocks and see if it stores a data-item which is older than currently written block and the associativity requirement of the application (lines 8-12). If so, the SRAM block is assumed to store old data and hence, its data-item is migrated to the NVM block and the newly arrived data-item is stored in the SRAM block (lines 13-15).

TABLE 1
L2 Cache Parameters (Lk. = leakage)

	SRAM			ReRAM		
	4 MB	8 MB	16 MB	4 MB	8 MB	16 MB
Hit latency (ns)	2.13	3.38	6.57	5.12	5.90	5.97
Miss latency (ns)	0.44	0.52	0.76	1.65	1.68	1.74
Wrt latency (ns)	1.08	1.67	3.49	22.18	22.67	22.53
Hit Energy (nJ)	0.366	0.487	0.61	0.537	0.602	0.662
Miss Energy (nJ)	0.009	0.013	0.016	0.187	0.188	0.190
Write Energy (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

Salient features. By proactively using SRAM ways for storing hot data, AYUSH adapts to changing working set much faster than the LRU policy. Fast access speed of SRAM also improves the performance which offsets the latency overhead of data migration. A migration operation is only performed on a write-hit and not on a read-hit. Thus, migrations happen in proportion to the write-intensity of applications and the read-intensive applications are minimally affected. Also, unlike previous techniques, AYUSH does not use prediction/history table (as in [11], [14]) or compiler analysis (as in [7]) or per-block counters to record accesses (as in [17]). Further, unlike [4], AYUSH does not waste cache space by storing same data in SRAM and NVM.

Overhead assessment. We assume that the migrated data-item is temporarily stored in a buffer [17]. The buffer uses 256 entries, each 64 B wide. For a W-way cache, we use W counters, each of which is assumed to use 40 bits and Z value is stored in a five-bit counter. Thus, the overhead of AYUSH, as a percentage of L2 cache size is

$$Overhead = \frac{W \times 40 + 256 \times 64 \times 8 + 5}{S \times W \times (B + G)} \times 100. \quad (1)$$

For example, for a 4 MB, 16-way cache, the $Overhead$ is only 0.36 percent of the L2 cache, which is very small. In addition to latency of writing SRAM and NVM blocks, each migration takes four cycles for transfer of 64 B data over 32 B bus to and from the buffer and two cycles for updating LRU-age. Each migration operation consumes 0.5 nJ energy.

4 Experimental Methodology

Simulation Platform. We use interval-core model in Sniper simulator. Processor frequency is 2 GHz. Both L1 I/D are four-way 32 KB caches with two cycle latency. L2 cache configuration is shown in Section 5. L2 cache is inclusive of L1 and is shared among cores, while L1 caches are private. All caches use LRU, write-back, write-allocate policy. Main memory latency is 220 cycles. Peak memory bandwidth for single, dual and quad-core systems is 10, 15 and 25 GB/s, respectively and queue contention is also modeled.

L2 cache parameters are shown in Table 1, which are obtained using NVsim [1], assuming sequential cache access, 32 nm process and write energy-delay-product optimized cache design. For the hybrid cache, we assume 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 is from an SRAM or a ReRAM region. The leakage power is assumed to scale linearly with the number of ways designed using SRAM and ReRAM.

Workloads. All 29 benchmarks from SPEC CPU2006 suite with *ref* inputs and six benchmarks from HPC field (shown as italics in Table 2) are used as single-core workloads. Using these, we create

TABLE 2
List of Single, Dual and Quad-Core Workloads

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

18 dual-core and nine quad-core multiprogrammed workloads, such that except for completing the left-over group, each benchmark is used exactly once.

Comparison with prior work. We compare AYUSH with a technique for SRAM-NVM hybrid caches [13], which we refer to as HDM (hot-data-migration) for convenience. In HDM, a data-item in NVM is migrated to SRAM when it is accessed by two successive write operations [13].

Evaluation metrics. We show the results on *a)* relative cache lifetime where the lifetime is defined as the inverse of maximum writes on any NVM (and not SRAM) cache block, *b)* decrease in WPKI, defined as writes-per-kilo instructions on all NVM (and not SRAM) blocks, *c)* Coefficient of intra-set write-variation (IntraV) [16] for NVM (and not SRAM) blocks, *d)* percentage energy loss, *e)* weighted speedup [9] and *f)* total number of migration operations (nMigration).

We model the energy of L2, main memory and data migrations. Leakage power and dynamic energy of main memory are 0.18 W and 70 nJ/access, respectively [9]. We ignore the energy overhead of counters and buffer, since it is orders of magnitude smaller than that of L2 + main memory. We fast-forward the benchmarks for 10 B instructions. Each workload is simulated till the slowest application executes 350 M instructions. An early completing benchmark in a multi-core workload is allowed to run, but its IPC is recorded only for the first 350 M instructions [9]. If the maximum writes on any NVM block is zero in the baseline, we assume lifetime improvement as $1\times$. In our experiments, this only occurs with gamess and amg2013 for a few configurations.

5 Results and Analysis

Results with default parameters. Fig. 1 shows the results on lifetime improvement, obtained using following parameters: $\alpha = 60\%$, $W = 16$, $L = W_S = 3$, 4 MB L2 with $I = 500$ K for single-core, 8MB L2 with $I = 250$ K for dual-core and 16 MB L2 with $I = 125$ K for quad-core system. Remaining metrics are summarized in Table 3, under the row 'Default' for AYUSH and *Default* for HDM.

Firstly, AYUSH provides larger improvement in lifetime than HDM. AYUSH performs migrations much more effectively than HDM (see nMigration), and its lifetime improvement also scales much better with increasing number of cores. HDM migrates a data-item only if it is accessed by *two successive writes*. In multicore systems with shared LLC, independent access streams from multiple applications are interleaved and thus, the final LLC access stream exhibits reduced locality [9], [10], hence two successive writes to the same block become progressively infrequent with increasing number of cores. Instead of requiring successive writes, AYUSH uses the LRU-age (i.e. recency) of the data-items to regulate migrations. Clearly, for future computing systems with tens of cores, the effectiveness of AYUSH will increase.

AYUSH improves lifetime by more than $10\times$ and even $100\times$ for several workloads. The improvement in lifetime achieved depends on several factors, such as IntraV present in baseline, distribution of writes to different LRU positions etc. For large IntraV, a large improvement in lifetime is observed, e.g. Nd, Bz, GcGa, AmXbNeGa etc. Opposite is true for workloads such as Lq, Mi, LsLb, LqCoMcBw etc. Similarly, decrease in WPKI also contributes to increase in lifetime, for example Ze, BzXa, GcBzGrTo etc. If most of the writes are already absorbed in SRAM ways, decrease in WPKI is negligible e.g. Po, Wr etc. From the decrease in WPKI and IntraV, we conclude that AYUSH performs both NVM write-minimization and NVM wear-leveling.

For all parameters evaluated in this paper, the average value of weighted speedup for AYUSH lies in $[0.99\times, 1.01\times]$ and hence, it is omitted from Table 3. AYUSH achieves small energy saving in single and dual-core systems and small energy loss in quad-core system. Clearly, AYUSH does not harm the performance and energy efficiency.

Parameter sensitivity results. We now present results for a wide range of algorithm and system parameters to show the effectiveness of AYUSH. Each time, only one parameter is changed from the default parameters. Results are summarized in Table 3.

On changing I (*interval size*), the average improvement in lifetime does not change monotonically and depends on the characteristics of different workloads. Similar is also true for a change in α. On increasing L to 4, lifetime increases due to a small increase in migration operations, but on increasing it to 5, migration operations decrease significantly, which leads to a sharp reduction in lifetime enhancement, although it also reduces the energy loss. On increasing W_S (*number of SRAM ways*), with AYUSH, for single-

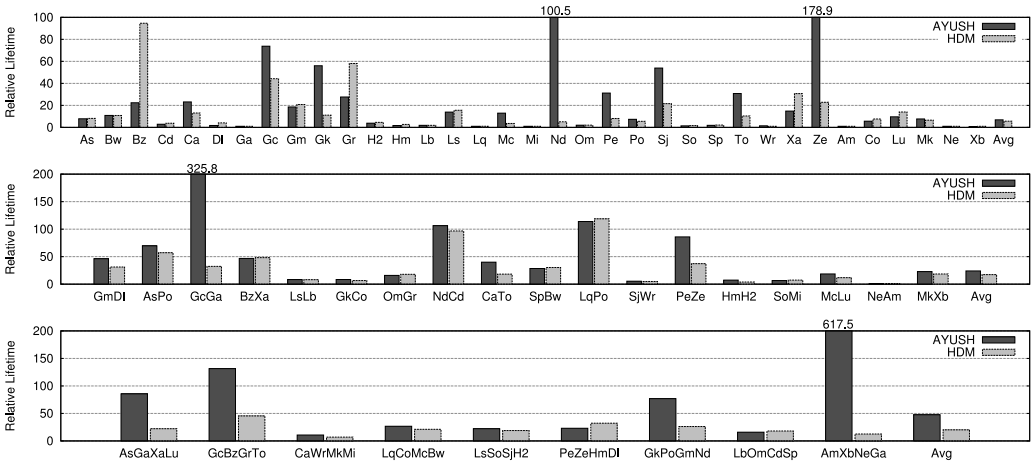

Fig. 1. Results on improvement of lifetime with AYUSH (More is better)

TABLE 3
Parameter Sensitivity Study (Rel. LfT. = Relative Lifetime)

	Rel. LfT.	⇓ WPKI	IntraV Base	IntraV AYUSH	Energy Loss %	nMigr--ation
Single-core System						
Default	*5.66*	*0.77*	*93.0*	*64.8*	*−0.97*	*154 K*
Default	6.90	0.70	93.0	60.4	−0.58	624 K
$I = 250$ K	7.61	0.72	93.0	60.1	−0.47	610 K
$I = 1$ M	6.60	0.66	93.0	60.6	−0.55	641 K
$\alpha = 50\%$	7.08	0.69	93.0	60.2	−0.51	644 K
$\alpha = 70\%$	6.75	0.71	93.0	60.3	−0.51	602 K
$L = 4$	6.90	0.70	93.0	60.4	−0.58	624 K
$L = 5$	3.49	0.58	93.0	71.5	−0.68	580 K
$W_S = 2$	*6.24*	*0.84*	*118.3*	*86.7*	*−0.70*	*160 K*
$W_S = 2$	7.73	1.02	118.3	85.5	0.04	643 K
$W_S = 4$	*4.96*	*0.69*	*88.8*	*63.0*	*−1.09*	*142 K*
$W_S = 4$	6.70	0.39	88.8	59.0	−0.75	602 K
8-way	6.61	−0.06	64.3	48.7	−0.53	561 K
32-way	8.97	1.14	133.2	87.6	−0.90	637 K
2 MB	6.61	0.50	78.6	53.2	−0.16	662 K
8 MB	9.33	0.47	128.2	95.9	−0.68	408 K
Dual-core System						
Default	*17.30*	*0.88*	*81.7*	*42.2*	*−1.42*	*324 K*
Default	24.06	0.72	81.7	41.1	−0.15	1753 K
$I =$125 K	26.14	0.74	81.7	41.1	−0.06	1743 K
$I =$500 K	23.34	0.73	81.7	41.1	0.01	1784 K
$\alpha =$50%	23.76	0.73	81.7	40.8	0.02	1813 K
$\alpha =$70%	22.32	0.71	81.7	41.3	−0.48	1682 K
$L = 4$	24.13	0.72	81.7	41.1	−0.14	1756 K
$L = 5$	9.59	0.65	81.7	48.2	−0.63	1576 K
$W_S = 2$	*17.67*	*0.94*	*87.6*	*45.1*	*−1.12*	*354 K*
$W_S =$2	24.21	0.98	87.6	41.3	0.12	1779 K
$W_S = 4$	*14.74*	*0.78*	*76.6*	*41.4*	*−1.28*	*301 K*
$W_S =$4	24.51	0.45	76.6	42.1	0.09	1734 K
8-way	24.76	−0.13	62.7	41.5	0.21	1615 K
32-way	22.74	1.16	106.0	42.3	−0.28	1747 K
4MB	14.52	0.44	56.6	22.2	0.04	1847 K
16 MB	35.25	0.94	117.8	77.1	0.17	1359 K
Quad-core System						
Default	*20.14*	*0.75*	*66.5*	*34.7*	*1.94*	*943 K*
Default	47.62	0.63	66.5	30.7	2.29	4253 K
$I = 62.5$ K	46.12	0.63	66.5	30.7	2.28	4228 K
$I = 250$ K	49.68	0.63	66.5	30.7	2.30	4295 K
$\alpha = 50\%$	50.40	0.64	66.5	30.7	2.34	4387 K
$\alpha = 70\%$	48.23	0.59	66.5	30.9	2.15	3851 K
$L = 4$	48.64	0.63	66.5	30.7	2.30	4259 K
$L = 5$	16.15	0.61	66.5	35.4	2.06	4080 K
$W_S = 2$	*21.86*	*0.83*	*73.4*	*36.6*	*1.61*	*1009 K*
$W_S =$2	40.53	0.81	73.4	32.3	1.91	4312 K
$W_S = 4$	*19.86*	*0.70*	*62.5*	*31.4*	*1.51*	*855 K*
$W_S = 4$	52.27	0.42	62.5	30.4	2.45	4175 K
8-way	36.52	0.37	48.7	30.0	2.82	3136 K
32-way	46.36	0.94	89.5	34.4	1.46	4265 K
8 MB	30.47	0.46	53.2	21.7	2.21	4516 K
32 MB	65.99	0.81	91.4	47.2	−1.46	3266 K

The nine rows with gray background are for HDM.

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. On increasing the *cache associativity*, IntraV of baseline is increased since few ways absorb most of the writes and hence, the scope of enhancing lifetime is also increased. On increasing the *cache size*, capacity misses reduce and the hit-rate increases since the applications have a fixed working set size. Hence, IntraV and lifetime enhancement are increased. Clearly, for future caches of high capacity, AYUSH will be even more effective.

6 Conclusion

The limited write endurance of NVMs presents major bottleneck in their use. We presented AYUSH, a technique for improving the lifetime of SRAM-NVM hybrid caches. Our future work will focus on synergistically integrating AYUSH with cache-access level and bit-level write-minimization techniques to improve the cache lifetime even further.

References

[1] 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.

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

[3] 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. Design, Autom. Test Eur. Conf. Exhib.*, 2010, pp. 136–141.

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

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

[6] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *Proc. 9th IEEE Symp. Embedded Syst. Real-Time Multimedia*, 2011, pp. 19–28.

[7] Q. Li, M. Zhao, C. J. Xue, Y. He, "Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache," in *Proc. 13th ACM SIGPLAN/SIGBED Int. Conf. Lang., Compilers, Tools Theory Embedded Syst.*, 2012, pp. 109–118.

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

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

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

[11] B. Quan, T. Zhang, T. Chen, J. Wu, "Prediction table based management policy for STT-RAM and SRAM hybrid cache," in *Proc. 7th Int. Conf. Comput. Convergence Technol.*, 2012, pp. 1092–1097.

[12] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin, C.-H. Lin, H.-Y. Lee, P.-Y. Gu, S.-M. Wang, F. T. Chen, K.-L. Su, C.-H. Lien, K.-H. Cheng, H.-T. Wu, T.-K. Ku, M.-J. Kao, and M.-J. Tsai, "A 4Mb embedded SLC Resistive-RAM macro with 7.2ns read-write random-access time and 160ns MLC-access capability," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest Tech. Papers*, 2011, pp. 200–202.

[13] G. Sun, X. Dong, Y. Xie, J. Li, Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, 2009, pp. 239–249.

[14] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 329–338.

[15] A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato, "Analyzing the optimal ratio of SRAM banks in hybrid caches," in *Proc. IEEE 30th Int. Conf. Comput. Design*, 2012, pp. 297–302.

[16] 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 19th Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 234–245.

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