This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2016.2645207, IEEE Computer Architecture Letters

Addressing Read-disturbance Issue in STT-RAM by Data Compression and Selective Duplication

Sparsh Mittal*, Jeffrey S. Vetter§, Lei Jiang†
*IIT Hyderabad, §Oak Ridge National Laboratory, †Indiana University
Email: sparsh0mittal@gmail.com, vetter@ornl.gov, jiang60@iu.edu

Abstract—In deep sub-micron region, STT-RAM (spin transfer torque RAM) shows read-disturbance error (RDE) which presents a crucial reliability challenge. We present SHIELD, a technique to mitigate RDE in STT-RAM LLCs (last level caches). SHIELD uses data compression to reduce cache-write traffic and restore requirement. Also, SHIELD keeps two copies of data blocks compressed to less than half the block size and since several LLC blocks are only accessed once, this approach avoids several restore operations. SHIELD consumes smaller energy than two previous RDE-mitigation techniques, namely high-current restore required read (HCRR, also called restore-after-read) and low-current long latency read (LCLL) and *even an ideal RDE-free STT-RAM cache*.

1 Introduction

STT-RAM is a promising memory technology for designing LLCs due to its close-to-SRAM read latency and high write endurance. Since the write latency/energy of STT-RAM are higher than those of SRAM, previous work has mainly focused on addressing its write overhead [1], however, a more severe issue of RDE in STT-RAM caches has not been adequately addressed. In STT-RAM, both reads and writes are performed by applying a voltage between bit line and source line and the only difference between them is that reads use smaller voltage than writes [2]. With feature size scaling, the write current reduces exponentially such that halving feature size leads to 25% magnetic tunnel junction (MTJ) area and 75% write current reduction [2]. However, read current does not reduce as much since sensing correct data using low-current ($< 20\mu A$) is challenging [3]. For sub-32nm feature size, the magnitude of read current becomes so close to the write current that a read operation is likely to modify the stored data and this is referred to as RDE. At small feature sizes, RDE causes very high error rates, such that even with strong error-correcting codes (e.g. 5EC6ED which can correct 5 errors and detect 6 errors), the error rate remains higher than that acceptable for on-chip caches [2]. It is expected that with ongoing process scaling, *readability* and *not writability* will become the most crucial bottleneck for STT-RAM [4], [5], [2].

In this paper, we present a technique, named 'SHIELD', which shields STT-RAM caches against both RDE and high write latency/energy overhead. SHIELD works by using data compression to reduce the number of bits written to cache (§2). Firstly, SHIELD does not write any bits to STT-RAM block for all-zero data, since during next read operation, such data can be reconstructed from compression encoding bits. This avoids RDE and high write latency/energy issue for all-zero data. Secondly, SHIELD keeps two copies of data in the block if the data have compressed width (CW) of at most 32B (assuming 64B block size). On the next read operation, one copy gets RDE which is not corrected since the second copy still remains free of RDE. This avoids one restore operation for such narrow data ($0 < CW \leq 32B$). Since many LLC blocks are only read once (§3), this approach can effectively avoid many restore operations. Thirdly, for any data with $CW > 32B$, including an uncompressed data-item, a single copy is written which is restored after each read operation.

A key benefit of SHIELD is that it avoids read, write and restore operations for all-zero data and restore operations for narrow-compressed data. Thus, SHIELD addresses both write latency/energy overhead and RDE issue by using data compression. By comparison, previous techniques [2], [4], [6] reduce some restore operations only and do not reduce read operations or address the overhead of write operations (§2.4). SHIELD stores only one cache line in a block and thus, forgoes the capacity advantage of compression. By virtue of this, SHIELD does not suffer from overheads which generally accompany compression techniques such as extra tags, compaction, fragmentation, changes in cache replacement policy, etc. Further, some techniques for reducing soft-errors in SRAM caches (e.g., [7]) keep two or three copies of a cache block (64B) in a cache set which leads to sharp degradation in cache capacity. By comparison, SHIELD duplicates compressed data of at most 32B size within a 64B cache block itself, and thus SHIELD does not degrade cache capacity. Compared to HCRR (also called refresh (or restore) after read [2], [6]) baseline, for single-core ($N = 1$) and dual-core ($N = 2$) workloads, SHIELD saves 7.8% and 9.2% energy, whereas an ideal RDE-free cache only saves 6.1% and 6.4% energy, respectively.

2 SHIELD: Key Idea and Architecture

SHIELD uses data compression for reducing the amount of data written to cache and making scope for selective duplication. While SHIELD can work with any compression technique, in this paper, we illustrate its working using BDI (base-delta-immediate) compression algorithm [8]. A BDI-compressed block is represented as $B_p\Delta_q$, where p and q represent the size of base and $Delta$ (difference), respectively. BDI uses 6 such patterns, viz. $B_8\Delta_1$, $B_8\Delta_2$, $B_8\Delta_4$, $B_4\Delta_1$, $B_4\Delta_2$ and $B_2\Delta_1$. BDI also checks if the block has repeated 8-byte values or is all-zero [8]. Of these 8 compression states, the one providing the smallest compressed size is finally chosen.

Defining consecutive reads: We now define a metric which is helpful in understanding the scope of mitigating RDE. For any cache block, *consecutive-read (CRead)* measures the average number of consecutive reads (i.e., no intermediate writes) seen by it during its residency in LLC. Also, for any application, the CRead is defined as the average CRead for all its cache blocks. Clearly, higher the value of CRead, higher is the restore requirement of an application since automatic restores through write happen infrequently. Note that CRead is different from the 'reuse-count' which measures both read and write opera-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2016.2645207, IEEE Computer Architecture Letters

tions in one generation of a block, e.g., in Figure 1, CRead for the block is 2 (= (2+1+3)/3), whereas reuse-count is 8.

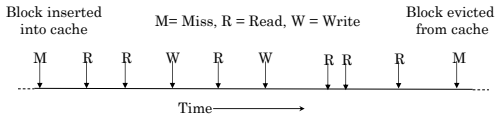

Fig. 1: One generation of a cache block.

2.1 SHIELD: Key Idea

For an RDE-affected STT-RAM cache, data are written to cache on both actual write operations and restore operations. To reduce the data written to STT-RAM during both these operations, SHIELD writes data in compressed form. This reduces the STT-RAM cache write energy. We perform two optimizations to BDI algorithm [9] for avoiding restore operations and further reducing the amount of data written to cache.

First, for data with all-zeros, no bit is stored in the STT-RAM block since the original uncompressed data can be easily recovered based on compression state encoding bits. Second, the first delta in BDI algorithm is always zero since it is the difference between the non-zero base with itself. Hence, we do not store this delta. The corresponding element is generated from the non-zero base. Based on our optimizations and LLC access pattern, SHIELD takes the following steps:

1. For all-zero data ($CW = 0$), no bits are written to or read. Thus, for such data, RDE does not occur and the restore operations are *completely* avoided.

2. It is well-known that due to filtering from L1 cache, LLC sees much smaller data locality than L1 cache. Hence, most LLC blocks are read only few times. In fact, our experiments have shown that average value of CRead for single and dual-core systems is 1.61 and 1.32, respectively (Figure 3(c) and (f)). Clearly, after any write operation, cache blocks see less than 2 read operations. Based on this, SHIELD keeps two copies of the data with $0 < CW \leq 32B$ within the same block. On a later read operation, one copy gets RDE, whereas the second copy still remains free from RDE. Thus, keeping two copies is equivalent to restoring the data at the time of original write operation to avoid *one* future restore operation. Thus, if $CRead \leq 2$, keeping two copies can avoid at least half of the restores for such narrow-width ($0 < CW \leq 32B$) blocks.

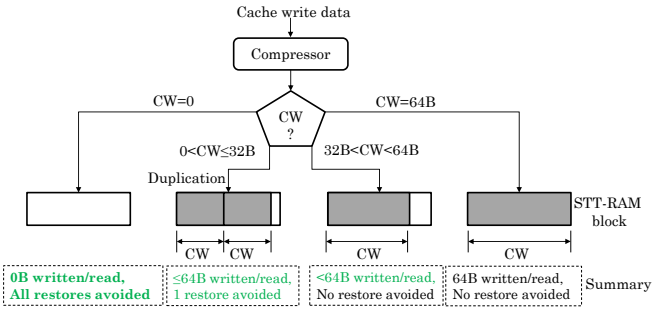

Fig. 2: Action taken by SHIELD on a cache write operation

2.2 Actions taken by SHIELD on read and write operations

Write operations: On any write operation, the cache controller compresses the data and computes CW. For $0 < CW \leq 32$, two copies of data are stored and for other CW values, only one data copy is stored. Figure 2 summarizes the action taken on a write operation and benefit/cost incurred for different CW values. Table 1 shows the 4-bit encoding used for different compression states for BDI algorithm. Out of 16 combinations from 4 bits, 13 combinations are utilized and the remaining 3 are not utilized. For each block, this 4-bit encoding is stored at the time of a write operation using a memory technology that does not suffer from RDE (e.g. SRAM). Due to our optimizations, the sizes of BDI-compressed blocks are different in our work (shown in Table 1) than those in the original BDI algorithm [8].

TABLE 1. Encoding and compressed sizes for different states (Enc. = encoding, Uncomp. = uncompressed)

Enc.	State	Copies	Size (B)	Enc.	State	Copies	Size (B)
0000	Zeros	1	0	1100	$B_4\Delta_1$	1	19
0001	Repeat	1	8	1101	$B_4\Delta_1$	2	19×2
0011	Repeat	2	8×2	0100	$B_4\Delta_2$	1	34
0010	$B_8\Delta_1$	1	15	1110	$B_2\Delta_1$	1	33
0110	$B_8\Delta_1$	2	15×2	1000	$B_8\Delta_4$	1	36
0101	$B_8\Delta_2$	1	22	1111	Uncomp.	1	64
0111	$B_8\Delta_2$	2	22×2				

Read operations: On any read operation, first the encoding (E) is consulted. If $E = 0000$, then the zero-data are reconstructed without accessing the block which avoids RDE. If $E = 0011$, 0110, 1101 or 0111, it implies that $0 < CW \leq 32B$ and 2 copies of data are stored. Hence, one copy is read, no restore is performed and the encoding is changed to 0001, 0010, 1100 and 0101, respectively, which indicates that now only one (error-free) copy of data is stored in the block. If $E = 0001$, 0010, 1100, 0101, 0100, 1110, 1000 or 1111, then data value is read, a restore operation is issued and the encoding remains unchanged.

2.3 Overhead assessment

Latency overhead: BDI compression and decompression take 2 and 1 cycles, respectively [8]. Given the high STT-RAM write latency, the BDI compression latency is easily hidden.

Energy overhead: We assume that a 1-byte adder (or subtractor) consumes 15 fJ energy [10]. Using this, we compute the overhead of compression by counting the number of subtractions in BDI algorithm. For $B_8\Delta_1$, $B_8\Delta_2$, $B_8\Delta_4$, $B_4\Delta_1$, $B_4\Delta_2$,$B_2\Delta_1$ and repeated value detection, a total of 406 byte subtractions required which consume 6.09 pJ. To account for other overheads and zero detection circuit, we finally assume the overhead of compression as 8 pJ. Decompression requires a maximum of thirty-one 2-byte additions, which consume 0.93 pJ. Hence, we assume the decompression overhead as 1 pJ.

We account for these overheads in our simulations.

2.4 Comparison with Previous Works

Sun et al. [6] propose restoring recently-read value when the cache banks are idle by keeping them in a buffer. Their technique, however, requires consulting both the main cache and the buffer on each access and does not provide advantage in case of bursty reads. SHIELD actually reduces restore operations, whereas their technique merely reschedules them. The technique of Wang et al. [2] avoids restores for blocks which are expected to see a write operation in near future. To perform L2 restore operations and updating L1 or L2 metadata bits, their technique requires observing the state of the block in other (i.e., L2 or L1) cache. Also, their technique postpones restore operations and hence, in their cache design, a dirty L2 block may have RDE. Due to these, their technique complicates cache management and incurs overhead which increases with rising number of cores. Also, their technique would require significant modifications to work with different cache coherence schemes, however, they do not present these details. Unlike previous techniques [2], [6], SHIELD performs restore operations immediately and hence, always keeps the L2 cache RDE-free. Also, SHIELD works based on data-width only and hence, can be integrated with any cache coherence scheme.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2016.2645207, IEEE Computer Architecture Letters

Jiang et al. [4] propose selectively performing HCRR and LCLL read depending on whether a particular bank is idle. Their technique is proposed for main memory where the memory controller buffers read/write requests. Due to differences in cache and main memory architecture, their technique may not be applicable or effective for caches. Zhang et al. [11] use compression to reduce write-traffic and mitigate process variation in PCM main memory, whereas SHIELD reduces read- and write-traffic to mitigate RDE in STT-RAM LLC. Patel et al. [12] propose circuit-level techniques for reducing write-overhead of STT-RAM caches whereas we focus on addressing RDE in STT-RAM caches. In Free-ECC technique [13], a minimum compression ratio is required for making space for ECC, but SHIELD has no minimum requirement of compression ratio

3 Experimentation Platform

We use Gem5 simulator to perform simulations using detailed timing model. L1 data/instruction caches have 32KB size with 2-way associativity. For $N = 1$ and $N = 2$, L2 cache size is4MB and 8MB, respectively (N=number of cores). The L2 cache parameters for 16-way STT-RAM L2 are obtained using DESTINY tool for 32nm node, write EDP (energy-delay-product) optimized design and sequential tag-data access (Table 2).

TABLE 2. STT-RAM L2 cache parameters

	Latency (ns)			Energy (nJ)			Power (W)
	Hit	Miss	Write	Hit	Miss	Write	Leakage
4MB	3.737	1.567	4.97	0.304	0.105	0.389	0.044
8MB	4.058	1.805	5.003	0.333	0.112	0.427	0.072

Our single-core workloads come from 29 SPEC2006 benchmarks (with *ref* input) and 3 benchmarks from HPC field. By using each of these benchmarks exactly once, 16 dual-core multiprogrammed workloads are generated (refer Table 3).

TABLE 3. Workloads used in the 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), Pe(perlbench), Po(povray), Sj(sjeng), So(soplex), Sp(sphinx), To(tonto), Wr(wrf), Xa(xalancbmk), Ze(zeusmp), *Co(comd), Lu(lulesh), Xb(xsbench)*
Dual-core workloads: BwLu, PeLq, XaTo, CaMc, GkLb, GmWr, PoSp, NdGr, GcMi, BzZe, LsSo, SjXb, OmH2, CoCd, AsDl, HmGa

We simulate every workload till each application in the workload has executed nInst instructions, where nInst is 150M for single-core and 100M for dual-core workloads. This keeps the simulation turnaround time manageable since we perform detailed timing simulation with a full-system simulator and simulate a large number of workloads and techniques.

Our **baseline is HCRR scheme** which uses normal current ($20\mu A$) to read the STT-RAM cell and performs a restore (i.e. write) operation after every read operation to correct the RDE [4], [2]. It has been used in real prototypes [3]. Further, we show results with an **ideal RDE-free STT-RAM cache** and **LCLL read scheme** [14], [4]. LCLL scheme uses low-current ($\sim 10\mu A$) sense amplifiers (SAs) which do not create RDE but incur 3× the sensing latency of traditional SAs since they require extra sensing stages to minimize sensing margin degradation due to variations in STT-RAM cells.

We use the following evaluation metrics: *1*) L2 cache energy (leakage+dynamic) *2*) Weighted speedup (referred to as relative performance), defined as $\Sigma_n(\text{IPC}_n(\text{technique})/\text{IPC}_n(\text{baseline}))/N$. *3*) Average CRead value for baseline (§2). *4*) ΔBWPKI (=$\text{BWPKI}_{\text{baseline}}$-$\text{BWPKI}_{\text{technique}}$) where BWPKI (bytes written to cache per kilo instruction) shows the cache write traffic. Thus, ΔBWPKI shows the reduction in write traffic compared to HCRR (baseline) scheme. For SHIELD, we also show *1*) *Compressed width (CW) of data on each cache write*. We classify the CW values in four ranges (refer Table 1): $CW = 0$, $0 < CW \leq 32B$ (i.e. {8B,15B,19B,22B}), $32B < CW < 64B$ (i.e. {33B,34B,36B}), $CW = 64B$ (i.e. uncompressed). *2*) Percentage of restore operations avoided (RstAvd). Let $Reads_{CW=0}$ be the reads to blocks with $CW = 0$ data and $Reads_{0<CW\leq 32\_2copy}$ be the reads to blocks with $0 < CW \leq 32B$ data having two copies. Then, $RstAvd = \dfrac{(Reads_{CW=0} + Reads_{0<CW\leq 32\_2copy}) \times 100}{TotalReads}$

4 Results and Analysis

Results are shown in Fig. 3 and summarized in Table 4.

TABLE 4. Summary of results (rel. perf. = relative performance)

	Single-core			Dual-core		
	SHIELD	Ideal	LCLL	SHIELD	Ideal	LCLL
Energy Saving	7.79%	6.10%	1.36%	9.23%	6.35%	1.55%
Rel. Perf.	1.05×	1.07×	1.03×	1.04×	1.08×	1.04×
ΔBWPKI	4065	2085	2089	3602	1877	1881

SHIELD: SHIELD consumes lower energy than HCRR, LCLL and *even the ideal RDE-free STT-RAM LLC*. Also, SHIELD provides higher performance than HCRR and LCLL. The efficacy of SHIELD for an application depends on the fraction of restores avoided, which depends on its data compressibility and the read/write access pattern, specifically its CRead value.

For $N = 1$ and $N = 2$, SHIELD reduces 50.7% and 48.5% of restore operations, respectively. For many applications, more than 95% of restores are avoided, e.g. Bw, Gk, Lb, Ls, Lq, Sj, Wr, Ze, Co, Lu, BwLu, GkLb, SjXb, etc. This happens because either these applications have highly compressible data and/or their CRead values are small. For some applications, more than 95% of blocks are all-zero and hence, only few restore operations are required since SHIELD avoids restores for narrow-data ($0 < CW \leq 32B$) and these applications are read-unintensive. For these applications, large performance improvement and energy saving are achieved and write traffic is significantly reduced.

For some applications CRead values are small, (e.g., Mi), but only few blocks are compressible. Opposite is true for applications where CRead values are high, (e.g., Bw, BwLu, PeLq), but most blocks are compressible. These effect partially cancel each other, and hence, these applications show only small energy gain/loss with SHIELD. For applications with high CRead values, e.g. Gc, Gm, Ca, Nd, Om, Pe, So, Xa, etc., the restore requirement is also high and thus, restore operations lead to performance and energy loss. Similarly, in some applications, most blocks are either incompressible or have compressed width greater than 32B, e.g., Gc, Gm, Hm, Mi, Pe, Po, To, Xa, XaTo, PoSp, HmGa, etc., and hence, SHIELD consumes comparable or larger energy than the RDE-free cache.

SHIELD brings large reduction in cache write traffic by virtue of using compression and avoiding many restore operations. Especially for applications with compressible data, SHIELD brings large reduction in BWPKI, e.g., Bw(44451), Gk(9122), Sj(10711), Co(16420), Lu(9596), BwLu(26210), GkLb(8675) etc. Clearly, SHIELD addresses both RDE and write-overhead in STT-RAM. For $N = 1$ and $N = 2$, the compression and decompression energy (§2.3) together account for 0.14% and 0.15% of the total L2 energy. Clearly, the energy overhead of SHIELD is negligible.

HCRR: With HCRR, each read operation also leads to a write operation and thus, HCRR causes significant increase in write traffic compared to the RDE-free STT-RAM cache. The increase in BWPKI is especially high for read-intensive applications, e.g. Bw, Bz, Sj, Co, BwLu, BzZe, CoCd, etc. Since STT-RAM writes have high energy/latency overhead, use of HCRR can degrade efficiency and also create bandwidth issues. For some workloads, SHIELD provides much higher performance than HCRR, e.g., Bw(1.35×), BwLu(1.19×), BzZe(1.18×) and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2016.2645207, IEEE Computer Architecture Letters

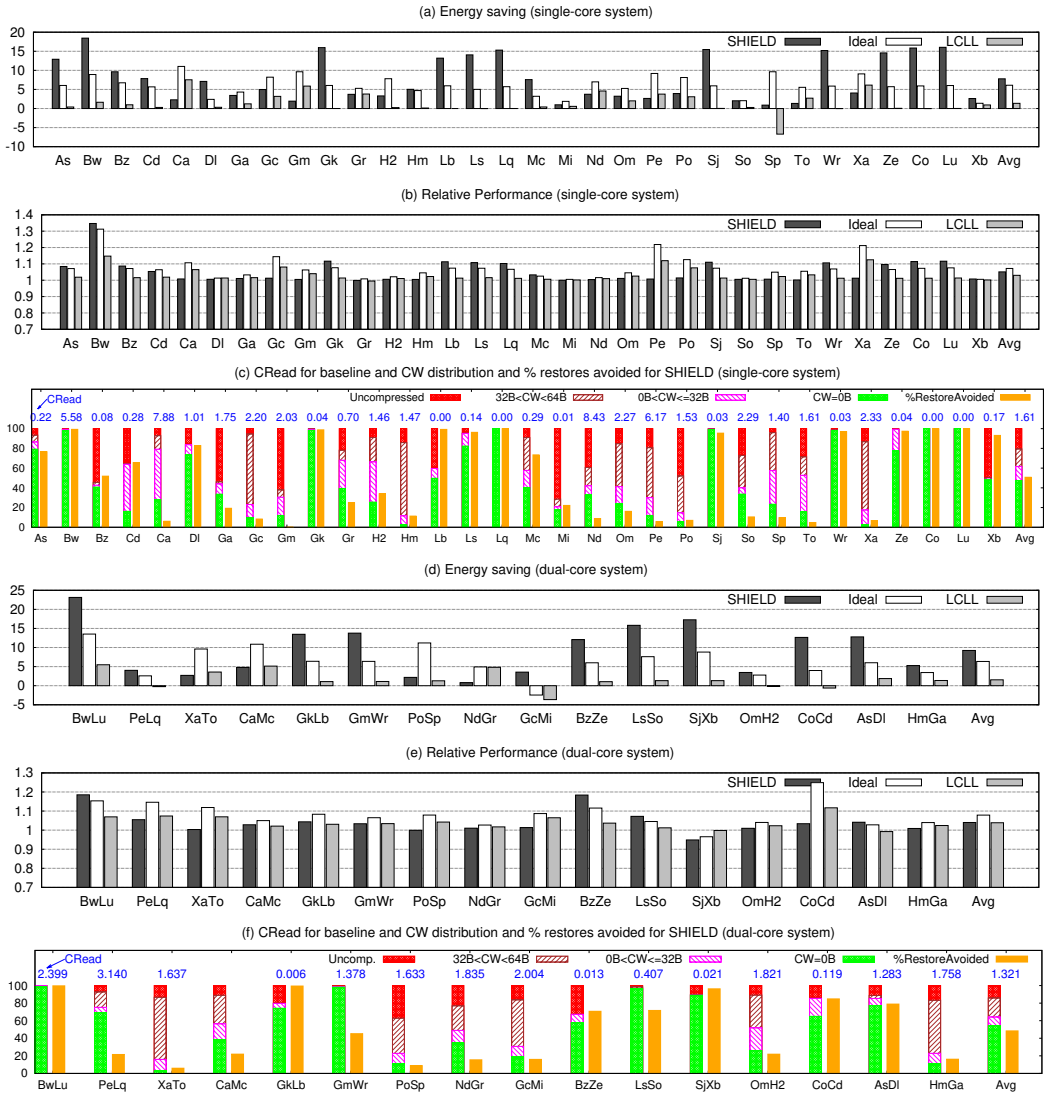

Fig. 3: (a)-(c) Results for single-core system, (d)-(f): Results for dual-core system

more than 1.10× for Gk, Lb, Ls, Sj, Wr, Co, etc. With increasing number of cores, the LLC access intensity increases and hence, restore operations will cause port obstruction and make the LLC unavailable for serving accesses. Clearly, use of HCRR in performance-critical systems is challenging.

LCLL: LCLL degrades performance by slowing down the read operations. Since read operations happen on critical access path, the large read latency may not be hidden. Unlike SHIELD, LCLL does not reduce write traffic to LLC. Although LCLL consumes smaller energy than HCRR, use of small read current in LCLL can lead to *decision failure* [5]. Also, since process variation affects device parameters, to guarantee RDE-free read operations, the value of read current needs to be set even lower than that assumed here; this, however, would further increase the read latency and cause performance loss. Clearly, SHIELD presents as a better technique for mitigating RDE than LCLL.

5 Conclusion

We presented a technique which uses compression and selective duplication of compressed-data to address both RDE and write overhead issue in STT-RAM LLCs.

References

[1] S. Mittal *et al.*, "A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-volatile On-chip Caches," *IEEE TPDS*, 2014.
[2] R. Wang *et al.*, "Selective restore: an energy efficient read disturbance mitigation scheme for future STT-MRAM," in *DAC*, 2015.
[3] R. Takemura *et al.*, "Highly-scalable disruptive reading scheme for Gb-scale SPRAM and beyond," in *IEEE IMW*, 2010, pp. 1–2.
[4] L. Jiang *et al.*, "Improving Read Performance of STT-MRAM based Main Memories through Smash Read and Flexible Read," *ASP-DAC*, 2016.
[5] W. Kang *et al.*, "Readability challenges in deeply scaled STT-MRAM," in *NVMTS*, 2014, pp. 1–4.
[6] Z. Sun *et al.*, "A dual-mode architecture for fast-switching STT-RAM," in *ISLPED*, 2012, pp. 45–50.
[7] A. Chakraborty *et al.*, "E < MC2: less energy through multi-copy cache," in *CASES*, 2010, pp. 237–246.
[8] G. Pekhimenko *et al.*, "Base-delta-immediate compression: practical data compression for on-chip caches," in *PACT*, 2012.
[9] S. Mittal *et al.*, "Reducing soft-error vulnerability of caches using data compression," in *ACM GLSVLSI*, 2016.
[10] X. Wu *et al.*, "Analysis of subthreshold FinFET circuits for ultra-low power design," in *IEEE International SOC Conference*, 2006.
[11] W. Zhang *et al.*, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *MICRO*, 2009, pp. 2–13.
[12] R. Patel *et al.*, "Reducing switching latency and energy in STT-MRAM caches with field-assisted writing," *TVLSI*, 2016.
[13] L. Chen *et al.*, "Free-ECC: An efficient error protection for compressed last-level caches," in *ICCD*, 2013, pp. 278–285.
[14] W. Kang *et al.*, "High reliability sensing circuit for deep submicron spin transfer torque magnetic random access memory," *EL*, 2013.