MirrorCache: An Energy-Efficient Relaxed Retention L1 STTRAM Cache

Kyle Kuan and Tosiron Adegbija
{ckkuan,tosiron}@email.arizona.edu
Department of Electrical & Computer Engineering
University of Arizona, Tucson, AZ, USA

ABSTRACT

Spin-Transfer Torque RAM (STTRAM) is a promising alternative to SRAMs in on-chip caches, due to several advantages, including non-volatility, low leakage, high integration density, and CMOS compatibility. However, STTRAMs' wide adoption in resource-constrained systems is impeded, in part, by high write energy and latency. A popular approach to mitigating these overheads involves relaxing the STTRAM's retention time, in order to reduce the write latency and energy. However, this approach usually requires a dynamic refresh scheme to maintain cache blocks' data integrity beyond the retention time, and typically requires an external refresh buffer. In this paper, we propose *mirrorCache*—an energy-efficient, buffer-free refresh scheme. MirrorCache leverages the STTRAM cell's compact feature size, and uses an auxiliary segment with the same size as the logical cache size to handle the refresh operations without the overheads of an external refresh buffer. Our experiments show that, compared to prior work, mirrorCache can reduce the average cache energy by at least 39.7% for a variety of systems.

CCS CONCEPTS

• **Computer systems organization → Processors and memory architectures**;

KEYWORDS

Spin-Transfer Torque RAM (STTRAM); cache; retention time; non-volatile memory; energy efficient systems; write energy; write latency; emerging memory technologies.

ACM Reference format:
Kyle Kuan and Tosiron Adegbija. 2019. MirrorCache: An Energy-Efficient Relaxed Retention L1 STTRAM Cache. In *Proceedings of ACM Great Lakes Symposium on VLSI, Tysons Corner, VA, USA, May 9–11, 2019 (GLSVLSI '19),* 4 pages.
https://doi.org/10.1145/3299874.3318022

1 INTRODUCTION

The Spin-Transfer Torque RAM (STTRAM) has emerged as a promising alternative for replacing traditional SRAMs in future on-chip caches. STTRAMs offer several attractive characteristics, such as non-volatility, low leakage, high integration density, and CMOS compatibility. However, some of STTRAM's most important challenges include its long write latency and high write energy [1, 2]. These challenges are attributed, in part, to the STTRAM's long *retention time*, which refers to how long data is retained in the memory in the absence of power [3]. Intrinsically, the retention time can be as long as 10 years. However, long retention time is typically over-provisioned for most cache data blocks [3]. Data blocks typically only need to remain in the cache for less than 1s before they are either replaced or invalidated. Thus, a viable STTRAM optimization is to substantially relax its retention time, thereby reducing the STTRAM's write latency and energy [3].

Even though relaxed retention time enables substantial energy and latency savings, several blocks may still need to remain in the cache beyond the retention time. To maintain the data correctness beyond the retention time, different techniques [1, 2] have been proposed to dynamically refresh the data block after the retention time has elapsed. Without loss of generality, we collectively refer to these techniques as the *dynamic refresh scheme (DRS)*. DRS typically writes a data block to a refresh buffer and back into the STTRAM cache to restart its retention clock in the cache. DRS also imposes overheads as a result of the buffer (leakage and area overheads) and the refresh operations (dynamic energy and time overheads). Our goal in this work is to mitigate these overheads.

We propose and explore the idea of *mirrorCache* as an energy-efficient relaxed retention L1 STTRAM cache that substantially mitigates the refresh buffer overheads. MirrorCache leverages STTRAM's density to enable an in-cache refresh function that eliminates the need for an external buffer. MirrorCache features a *main segment* and an equal-sized *auxiliary segment*. The cache's effective capacity is the size of the main segment. Rather than writing/reading refreshed data blocks to/from an external buffer as in prior work, mirrorCache writes to its auxiliary segment, and cache blocks can be fetched from either the main or auxiliary segments. Thus, mirrorCache eliminates the overheads concomitant to the external buffer (including its peripheral circuitry), and also substantially reduces the energy accrued during each refresh operation.

Using experiments with STTRAM caches designed with different retention times, our results reveal that mirrorCache reduces the overall cache energy by at least 39.7% on average, compared to prior work, without introducing substantial overhead. Our results also show that mirrorCache reduces the energy as compared to SRAM by at least 19.8%, on average, while trading off the latency for energy savings.

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$$15.00
https://doi.org/10.1145/3299874.3318022

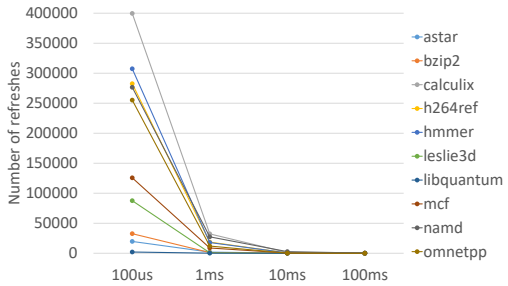

Figure 1: Number of cache block refreshes for SPEC CPU2006 benchmarks

2 BACKGROUND AND RELATED WORK

STTRAM uses a magnetic tunnel junction (MTJ) cell as the binary storage cell. MTJ contains two ferromagnetic layers separated by an oxide barrier/tunnel layer [4, 5]. Updating the MTJ cell's data bits relies on the magnetization switching of MTJ's free layer [4]. As a result, it takes more time and energy than conventional SRAM to change the free layer's magnetization state [5].

An approach for reducing the write energy and latency is to substantially relax STTRAM's data retention time, since it directly impacts the write energy and latency. The retention time can be relaxed by decreasing the MTJ cell's thermal stability or reducing the planar area [3]. Given a relaxed retention time STTRAM cache, *dynamic refresh schemes (DRS)* have been proposed to refresh data blocks that must remain beyond the retention time [1–3].

DRS incurs energy overheads due to the large number of refreshes and the need for a refresh buffer. Several techniques have been proposed to reduce the number of refreshes. For example, Jog et al. [1] refreshed only the first eight most recently used (MRU) blocks, and used a write buffer to bridge the surge of refresh requests and long STTRAM write time. Other techniques used compiler-assisted techniques to make refresh more efficient (e.g., [6]). However, DRS still requires a buffer to temporarily hold data blocks during refresh. Our work aims to minimize the overheads of DRS by leveraging the STTRAM's high density to eliminate the refresh buffer and its overheads. In effect, mirrorCache substantially reduces the leakage power and overall cache energy compared to DRS.

3 MIRRORCACHE

3.1 Cache block refresh analysis

To demonstrate the significance of the number of refreshed blocks and interplay with cache retention time, we analyzed the number of cache block refreshes required to maintain data correctness in different SPEC CPU2006 benchmarks. Figure 1 depicts the number of refreshes for different SPEC CPU2006 benchmarks running on 32KB L1 caches with retention times: 100μs, 1ms, 10ms, and 100ms.

As expected, smaller retention times, which consume less write energy and write latency, require substantially more refreshes. However, even though a longer retention time reduces the number of refreshes (e.g., the 100ms cache reduces the number of refreshes by 15x, on average), this comes at the expense of higher write energy and latency. Thus, it is important to minimize the overheads associated with the refresh operations.

3.2 Overview of mirrorCache

Figure 2 illustrates mirrorCache's operation. MirrorCache comprises of two equal-sized segments with identical organizations: the *main* and *auxiliary* (or *mirror*) segments. When a block must be refreshed, it is written to the auxiliary segment, and future references to the block are serviced from the auxiliary segment. If a block in the auxiliary segment needs to be refreshed, it is written back to the main segment and referenced from there. Block replacements are performed using normal replacement policies, e.g., least-recently used (LRU) or pseudo-LRU [7].

MirrorCache trades off physical area in favor of refresh efficiency, and offers a few advantages compared to DRS. First, mirrorCache eliminates the need for a refresh buffer, thereby eliminating the buffer's leakage power. Second, since the auxiliary segment is of equal size to the main segment, mirrorCache eliminates any chances of an oversubscribed buffer when several blocks must be refreshed simultaneously [8]. Third, since data can be directly fetched from the auxiliary segment, pipeline stalls are unnecessary when accessing data that is being refreshed. To prevent the overheads of tag comparisons for a larger cache, the tag array remains in 32KB setting, wherein a single status bit determines whether a block request is serviced from the main or auxiliary segment. Finally, mirrorCache reduces the dynamic energy incurred by each refresh operation since the writes to and reads from the external buffer are eliminated.

3.3 Identifying blocks to refresh

For simplicity, we opted to simply refresh all the blocks with longer lifetimes than the retention time (details in Section 4.2). To determine if and when blocks must be refreshed, we employ a *refresh counter* for each block, similar to prior work [2]. We implement the counter as a four-state finite state machine (FSM), resulting in a 2-bit per block overhead. The counter has a clock period $C = \frac{1}{P} * R$, where R is the retention time, $P = S - 1$, and S is the number of counter states (four in our case). When a block is written into the cache, the cache controller initializes the counter to zero, enables it, and then writes the block to the mirror segment when the counter's state reaches P.

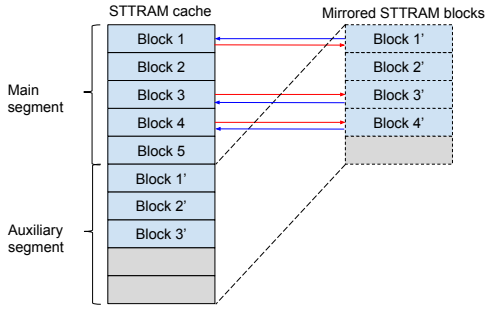

Figure 2: Illustration of mirrorCache

Table 1: Cache parameters of SRAM and STTRAM with different retention times

Cache Configuration	SRAM: 32KB, STTRAM: 64KB physical size, 64B line size, 4-way; Buffer: 1KB					
Memory device	SRAM	STTRAM-100μs	STTRAM-1ms	STTRAM-10ms	STTRAM-100ms	STTRAM-buffer
Write energy (per access)	0.125nJ	0.095nJ	0.107nJ	0.122nJ	0.141nJ	0.156nJ
Hit energy (per access)	0.494nJ	0.3nJ	0.3nJ	0.3nJ	0.3nJ	1.089nJ
Leakage power	186.264mW	154.686mW				285.666mW
Hit latency (cycles)	2	1	1	1	1	1
Write latency (cycles)	2	3	4	5	7	1

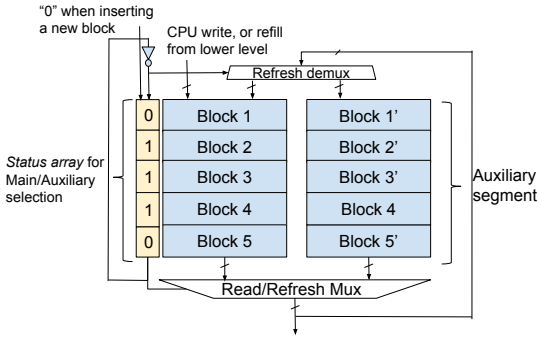

Figure 3: mirrorCache architecture

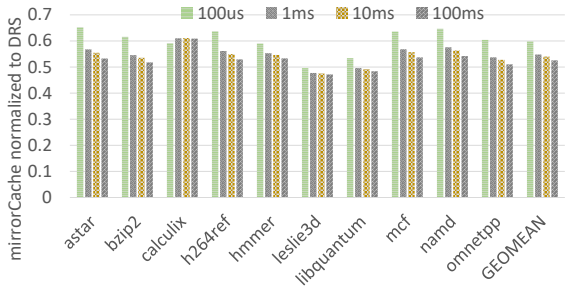

Figure 4: mirrorCache energy normalized to DRS (baseline of one)

3.4 MirrorCache architecture and overhead

Figure 3 illustrates mirrorCache's architecture and datapath for the main and auxiliary segments. To indicate the segment holding a cache block, we use a *status array* comprising of a status bit per block. Thus, the array size is equal to the number of logical cache blocks. For example, a 32KB cache would have a 512-element array, with one bit per element. When a block is inserted by CPU or refilled from lower level memory, the block would be allocated to the main segment, and the status bit is set to 0. When the counter FSM triggers a refresh, the status bit is read and inverted, indicating that the block is in the auxiliary segment; the datapath is then established for the block to be written to the auxiliary segment. Similarly, a refresh from the auxiliary segment writes the block into the main segment, and the status bit is set back to 0. For a fetch request, the status array indicates the active segment for the block; tag comparison is then performed as usual. The status array can be implemented in relaxed retention time STTRAM in order to easily integrate into the STTRAM cache. Overall, the overheads of implementing mirrorCache are from the status array and the muxes, as shown in Figure 3.

4 SIMULATION RESULTS

4.1 Experimental Setup

We modified GEM5 [9] to model both mirrorCache and DRS, for comparison to prior work. We used configurations similar to the ARM Cortex A15 [10], featuring a 2GHz clock frequency, and a private L1 cache with separate instruction and data caches. We focused on the data cache, since our analysis revealed that the target refresh problem was predominant in the data cache.

To explore the benefits of mirrorCache in various retention time scenarios, in comparison to prior work (DRS) and SRAM, we used a 32KB cache with 64B line size and 4-way set associativity. For a rigorous comparison, we implemented DRS as a best-case amalgam of prior methods assuming an ideal refresh scenario. That is, blocks are only refreshed if they are reused, no blocks expire because of insufficient refresh buffer space, and buffer bandwidth does not constitute a bottleneck. DRS featured a fully-associative 1KB STTRAM buffer, which is appropriately scaled from prior work [1]. The buffer's dynamic energy was 1.245nJ per refresh and 285.67mW leakage power from the buffer's peripherals and high associativity.

We considered four retention times: 100μs, 1ms, 10ms, and 100ms, which we empirically found to be sufficient for the range of considered benchmarks. We modeled the retention times using MTJ modeling techniques proposed in [11] and used NVSim [12] to construct the STTRAM cache for the different retention times. Table 1 depicts the STTRAM and SRAM cache parameters used in our experiments as obtained from the modeling tools and techniques. We used ten SPEC CPU2006 benchmarks [13], compiled for the ARM instruction set architecture. Each benchmark was run using the *reference* input sets for 100M instructions after fast-forwarding for 100M instructions.

4.2 Results

In this section, we present the results of our experiments to illustrate the benefits of mirrorCache as compared to DRS. Throughout this section, we refer to our proposed work as mirrorCache, and a cache featuring the dynamic refresh scheme simply as DRS.

4.2.1 Energy Savings. Overall, mirrorCache substantially reduced the refresh energy compared to DRS. Unlike DRS, where the average refresh energy for each application was approximately 47% of the total energy, mirrorCache reduced the refresh energy to an infinitesimal portion (< 1%) of the total energy, thereby resulting in substantial cache energy savings.

Figure 4 depicts the overall cache energy of mirrorCache normalized to DRS. On average across all the benchmarks, mirrorCache

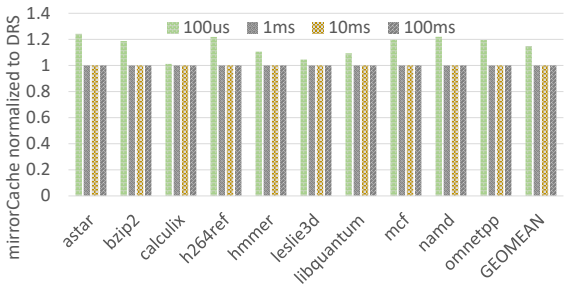

Figure 5: mirrorCache latency normalized to DRS (baseline of one)

reduced the cache's energy by 39.7%, 44.9%, 45.7%, and 47.2% for the 100μs, 1ms, 10ms, and 100ms caches, respectively. Energy savings were up to 52.8% for *leslie3d* on the 100*ms* cache. We observed that mirrorCache was more beneficial in higher retention time caches. Since fewer refreshes occur in high retention caches, the refresh buffer in DRS was over-provisioned and wasted energy.

We observed that *astar* and *namd* improved the least over DRS by 34.7% and 35.3% in the 100μs cache. These two benchmarks' data blocks were frequently updated, had high dynamic read and write activities, and low refresh activity. Thus, the energy consumption of these benchmarks was dominated by the dynamic energy. However, by eliminating the refresh buffer, mirrorCache still achieved substantial energy savings for these benchmarks despite their low refresh activity.

4.2.2 Latency. We also evaluated mirrorCache's access latency benefits compared to DRS. In general, mirrorCache only achieved marginal latency benefits compared to DRS due to the additional write latency of the larger physical cache (Table 1) and the circuits to check the status bits.

Figure 5 depicts mirrorCache's latency normalized to DRS. For the 1*ms*, 10*ms*, and 100*ms* caches, mirrorCache achieved similar latency to DRS. For the 100μs cache, however, mirrorCache degraded the latency for majority of the benchmarks by 14.93% on average, and by up to 24.29% for *astar*. We observed that in the 100μs retention time, the latency overhead was dominated by the additional decoding circuits due to mirrorCache's increase in physical size. For the longer retention times, however, the overhead was dominated by the STT-RAM cell's write latency, which was equivalent for both DRS and mirrorCache, despite mirrorCache's increase in size. Across all benchmarks, *calculix* and *leslie3d* suffered the least latency degradation of 1.25% and 4.57%, respectively, in the 100μs cache. We observed that *calculix* had few writes compared to other benchmarks and *leslie3d*'s latency was dominated by the miss latency. As such, neither benchmark was significantly affected by mirrorCache's increase in write latency.

4.2.3 Comparison to SRAM. We summarize the results of mirrorCache's comparison to SRAM, but omit graphs due to space constraints. On average, mirrorCache reduced the overall cache energy by 34.7%, 31.2%, 27.2%, and 19.8% for the 100μs, 1ms, 10ms, and 100ms caches, respectively, as compared to SRAM. MirrorCache reduced the overall cache latency by 17.5% and 6.8% for the 100μs and 1ms, respectively, as compared to SRAM. For higher retention times (10ms and 100ms), however, mirrorCache *increased* the latency by 3.8% and 25.2%, respectively, with the most substantial increases occurring for write-intensive benchmarks like *omnetpp* and *h264ref*. We attribute these observations to STTRAM's long write latency (Table 1). We observed similar trends for DRS, so this was not a deficiency unique to mirrorCache.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed *mirrorCache* as an energy efficient design for mitigating the overheads of utilizing a refresh buffer in relaxed retention STTRAM caches. MirrorCache leverages the STTRAM's high density to feature a main segment and an auxiliary segment that enables in-cache refresh operations without the need for an external buffer. Results show that, compared to prior related work, mirrorCache reduced the cache energy, on average, by 39.7%, 44.9%, 45.7%, and 47.2% for the 100μs, 1ms, 10ms, and 100ms caches, respectively. Compared to an SRAM cache, mirrorCache reduced the cache energy by 34.7%, 31.2%, 27.2%, and 19.8% for the 100μs, 1ms, 10ms, and 100ms caches, respectively. For future work, we plan to explore mirrorCache's benefits in complex multi-level caches, and develop techniques for mitigating the latency overheads compared to SRAM.

REFERENCES

[1] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *DAC Design Automation Conference 2012*, June 2012, pp. 243–252.

[2] Z. Sun, X. Bi, H. Li, W. F. Wong, Z. L. Ong, X. Zhu, and W. Wu, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011, pp. 329–338.

[3] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 50–61.

[4] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, p. 165209, 2007.

[5] C. Xu, D. Niu, X. Zhu, S. H. Kang, M. Nowak, and Y. Xie, "Device-architecture co-optimization of stt-ram based memory for low power embedded systems," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2011, pp. 463–470.

[6] K. Qiu, J. Luo, Z. Gong, W. Zhang, J. Wang, Y. Xu, T. Li, and C. J. Xue, "Refresh-aware loop scheduling for high performance low power volatile stt-ram," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 209–216.

[7] H. Ghasemzadeh, S. Mazrouee, and M. R. Kakoee, "Modified pseudo lru replacement algorithm," in *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*. IEEE, 2006, pp. 6–pp.

[8] J. Ahn, S. Yoo, and K. Choi, "Dasca: Dead write prediction assisted stt-ram cache architecture," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014, pp. 25–36.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[10] "Cortex-A15 Processor." [Online]. Available: https://www.arm.com/products/processors/cortex-a/cortex-a15.php

[11] K. C. Chun, H. Zhao, J. D. Harms, T. H. Kim, J. P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular mtj based stt-mrams for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb 2013.

[12] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[13] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.