Red-Shield: Shielding Read Disturbance for STT-RAM Based Register Files on GPUs

Hang Zhang[†‡], Xuhao Chen[‡], Nong Xiao[†‡], Fang Liu[†‡], Zhiguang Chen[†‡]
[†]State Key Laboratory of High Performance Computing, College of Computer,
National University of Defense Technology
[‡]College of Computer, National University of Defense Technology
Changsha, Hunan 410073, China
{hangzhang, xuhaochen, nongxiao, liufang, chenzhiguang@nudt.edu.cn}

ABSTRACT

To address the high energy consumption issue of SRAM on GPUs, emerging Spin-Transfer Torque (STT-RAM) memory technology has been intensively studied to build GPU register files for better energy-efficiency, thanks to its benefits of low leakage power, high density, and good scalability. However, STT-RAM suffers from a reliability issue, read disturbance, which stems from the fact that the voltage difference between read current and write current becomes smaller as technology scales. The read disturbance leads to high error rates for read operations, which cannot be effectively protected by SECDEC ECC on large-capacity register files of GPUs.

Prior schemes (e.g. read-restore) to mitigate the read disturbance usually incur either non-trivial performance loss or excessive energy overhead, thus not applicable for the GPU register file design which aims to achieve both high performance and energy-efficiency. To combat the read disturbance on GPU register files, we propose a novel software-hardware co-designed solution, i.e. *Red-Shield*, which consists of three optimizations to overcome limitations of the existing solutions. First, we identify dead reads at compiling stage and augment instructions to avoid unnecessary restores. Second, we employ a small read buffer to accommodate register reads with high access locality to further reduce restores. Third, we propose an adaptive restore mechanism to selectively pick the suitable restore scheme, according to the busy status of corresponding register banks. Experimental results show that our proposed design can effectively mitigate the performance loss and energy overhead caused by restore operations, while still maintaining the reliability of reads.

Keywords

GPU, STT-RAM, Register File, Read Disturbance

1. INTRODUCTION

General-purpose graphics processing units (GPGPUs) have been widely adopted for high performance computing during the last decade, owing to its high throughput and energy-efficiency [14]. Its single instruction multiple thread (SIMT) architecture enables thousands of threads running concurrently to hide long latency of memory operations and hence achieve high throughput. To maintain hardware contexts of a huge number of threads, GPGPUs usually employ a large-capacity register file for seamlessly context switching. Unfortunately, large-capacity register files also consume a large portion of power that usually accounts for 15-20% of total power in modern GPUs [7], as the register files are currently built with SRAM technology that has high leakage power in submicron technology nodes. To combat the high energy consumption of SRAM, there have been extensive studies on replacing SRAM with emerging STT-RAM to build energy-efficient register files of GPUs, due to its benefits of low leakage power, high density, and good scalability [6, 8, 9].

However, `read disturbance` issue has become a major design challenge for STT-RAM based on-chip memory [16, 17] recently. The read disturbance lies in the decreasing difference between write current and read current that are applied to read/write STT-RAM cells. Due to performance concerns, GPU register files usually employ simple ECC protections against SRAM errors [11], however it is not sufficient to shield the high error rate caused by the read disturbance issue [17]. Therefore, how to cope with the read disturbance issue is a key concern that decides whether the emerging STT-RAM can be employed to build energy-efficient register files on GPUs, and it has not been addressed well by prior works.

Restoring data after each read can be a simple yet effective solution to suppress the read disturbance on GPU register files [16, 17]. However, this restore operations lead to unaccepted performance loss and energy overhead. Consequently, it is imperative to judiciously tackle the read disturbance for STT-RAM based register file while preserving the high throughput and energy-efficiency of GPUs. To this end, we propose a novel software-hardware co-design, namely ***Red-shield*** (***Re***ad ***D***isturbance ***Shield***), to alleviate the performance loss and extra energy consumption incurred by the read disturbance issue on GPUs. In summary, we make the following contributions in this work.

- We devise a compiler assisted dead read identification scheme to identify dead reads that do not need restores, and remove unnecessary restore operations related to these reads.
- Since successive reads to the same register lead to temporal locality, we employ a small SRAM read buffer to accommodate the reads with high locality.
- We propose an adaptive restore mechanism to selectively pick the optimal restore scheme, according to the status of register banks.

GLSVLSI '16, May 18-20, 2016, Boston, MA, USA

DOI: http://dx.doi.org/10.1145/2902961.2902988

Table 1: Read Disturbance Error Rate

Tech(nm)	45	32	22	15	11
BER	1.38E-8	3.38E-7	3.07E-6	2.16E-5	1.2E-4
LER	1.42E-5	3.50E-4	3.17E-3	2.21E-2	1.17E-1
LER SECDEC	1.02-10	6.12E-8	5.04E-6	2.46E-4	7.11E-3

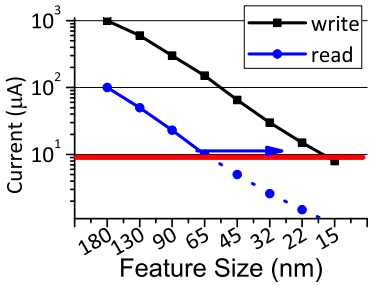

Figure 1: The scaling of read and write currents [17]

To the best of our knowledge, this is the first work to address the read disturbance issue for large-scale STT-RAM based register files on GPUs.

2. MOTIVATION

A STT-RAM read operation have the same operating mechanism as a write operation but with smaller voltage. Fig. 1 illustrates the current amplitude comparison between write current and read current at different technology nodes.

The decreasing difference between read current and write current leads to read operations with a high error rate. Therefore, the read disturbance has become a major design concern for STT-RAM based register files. We use the model from [17] to calcuate the read disturbance error rate. The 1024-bit register entry is evaluated as a whole block. Due to performance concerns, GPUs usually employ SECDEC ECC to protect register files [12]. The modeled error rate is shown in Table 1. The BER denotes the raw bit error rate, the LER denotes the line error rate, and the LER SECDEC denotes the line error rate after adopting SECDEC ECC. We can see that LER SECDEC is so high that it cannot guarantee the reliability of reading a register entry. For instance, the LER SECDEC at 22mm is about 5.04E-6, which is much greater than 2.5E-11, the acceptable error rate of DRAM [17, 13].

Additionally, the state-of-the-art design [17] that aims to suppress the read disturbance issue is not applicable to GPU register file design, because it is specifically designed for STT-RAM based LLC (L2 cache) whose features are quite different from those of the register file. It requires help information from the upper-level memory hierarchy (L1 cache), and however there is no such upper-level layer above register files on GPUs. Therefore, it is necessary to devise effective mechanisms to obviate the read disturbance issue on STT-RAM based GPU register files while not incurring much performance and energy overhead.

3. RED-SHIELD DESIGN

In this section, we present our proposed design, ***Red-shield***, which incorporates several software and hardware optimizations to address the read disturbance issue of STT-RAM based register files.

3.1 Dead Read Identification

To understand the characteristic of accessed register values on GPUs, we conduct program profiling on different benchmarks. The result is shown in Fig. 2. Most of these benchmarks have similar distributions. On average, nearly 49.3% of the generated register values are referenced (read) only once, while nearly 27.9% and 11.6% of them are referenced twice and three times, respectively. Those are referenced more than threes times only constitute 0.09% of total references. A similar result has been observed in previous work [5]. In fact, it implies that we do not need to restore those register values that will not be re-referenced in the future.

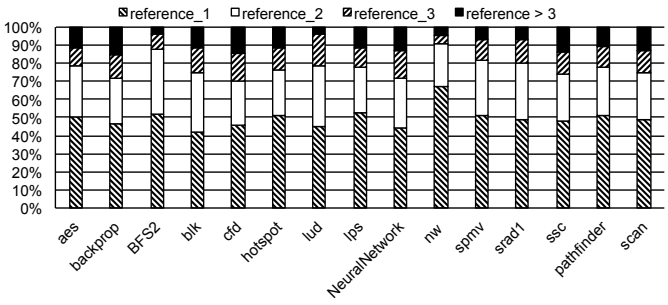

Figure 2: Register read access count distribution. It shows the number of read accesses to registers.

Furthermore, for a register value that will be re-referenced several times, the last reference also does not need to be restored, since there is no more read after the last reference. For instance, if we know the register value A will be referenced three times in total after it is generated, the 3rd read operation on A does not need a restore operation. Here, we refer to the last read operation as the ***dead read***, since the register value is not useful anymore after the last read operation. Apparently, the only read operation to the register value that has only one reference is also a ***dead read***.

If we can identify these dead reads before run-time, then the unnecessary restore operations to these dead reads can be avoided and both performance and energy-efficiency can be improved. To this end, we propose a dead read identification scheme that can effectively recognize dead reads at the compiling stage. The standard register analysis algorithm [1] is employed to analyze register liveness coverage.

3.2 Read Buffer Promotion

Although the dead read identification scheme can effectively reduce the number of unnecessary restore operations, for register values that have multiple references, it is still necessary to restore the data after each read operation (except for the last one). An example of the register liveness analysis result of `backprop` is shown in Fig. 3. 'R' denotes a register read; 'W' denotes a register write; 'L' indicates a register is alive. For instance, r_2 is referenced three times (line 2,4,8) in this snippet code. Though the last read operation can be identified as a dead read, there is still a need to restore the first two read operations after reading.

For the register values that have many references (e.g. more than 3 times), if the reference count information is available, we can devise smart schemes to leverage this temporal locality. To further reduce the number of restore operations, we propose a read buffer promotion scheme to make full use of this observed high temporal locality. To be specific, we build a reference tracking algorithm by leveraging the register liveness analysis. Based on the liveness information, this algorithm collects the reference count for each read operand of each instruction. The reference count for a specific register is re-collected only if the corresponding operand is dead or written. This reference count is also encoded into instructions. We

```
                          register number-> 0 1 2 3 4 5 6
1     cvt.u32.u16 $r2, $r0.hi;               R L W L L
2     mul.half.lo.u16 $r5,$r3.lo,$r2.hi;     L L R R L W
3     mul.half.lo.u16 $r6,$r4.lo,$r1.hi;     L R L L R L W
4     mad.wide.u16 $r5,$r3.hi,$r2.lo,$r5;    L L R R L W L
5     mad.wide.u16 $r6,$r4.hi,$r1.lo,$r6;    L R L L R L W
6     shl.u32 $r5,$r5,0x00000010;            L L L L L W L
7     shl.u32 $r6,$r6,0x00000010;            L L L L L L W
8     mad.wide.u16 $r3,$r3.lo,$r2.lo,$r5;    L L R W L R L
```

Figure 3: A snippet of the PTX source code for the backprop benchmark

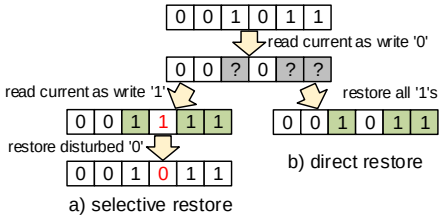

Figure 4: Comparison between the previous restore operation and our fast-restore operation

then deploy a small SRAM read buffer to store these register values with high temporal locality (e.g. reference count > 3).

3.3 Adaptive Restore Design

A STT-RAM read operation is similar to a write operation except that the read voltage is smaller than write voltage. To restore a data value in STT-RAM cells in an efficient way, people propose the selective restore (SR) [17] that involves two read operations and one write operation. After first read operation, SR requires another read that employs a current with an opposite direction compared to the first read. This second read operation aims to identify which specific cells have been disturbed. Then a successive write operation can write only the cells that have been destructed at the first read. The SR scheme is shown in Fig. 4(a). The SR can effectively reduce the number of restored data bits, and therefore save energy consumption caused by unnecessary restores. However, since SR requires extra operation and causes non-trivial performance overhead, it can not be naively applied to the STT-RAM based register file design on which performance is more critical.

On the other hand, the direct restore (DR) [16] can be also employed in our STT-RAM register file design. The idea is to skip the second read operation and restore all the "1"s, regardless whether each of these "1"s is disturbed or not. Thus we can save the time for performing one read operation for each restore. The downside of this scheme is that it might write the "1"s that are not disturbed and might consume unnecessary energy. However, this downside is not a problem because of two reasons: 1) Our previous dead read identification and read buffer promotion schemes can effectively reduce the number of restores. Thus, the number of required direct restores are relatively smaller compared to the baseline that employs SR scheme. 2) as technology scales, the read current and write current are getting closer, and therefore the direct restore (under the inversion optimization) only consumes a reasonable amount of energy close to the second read operation in the SR scheme.

Furthermore, we introduce an adaptive restore scheme that combines the benefits of the selective restore and the direct restore together, and can selectively choose to use SR or DR, according to the serving state of register banks. To be specific, the operand collector only performs DR when there is a bank conflict, since in this situation the latency of read operations has a dominant impact on performance; If there is no bank conflict, SR is preferred to minimize the energy consumption of restore operations.

4. EXPERIMENTAL EVLUATION

In this section, we present the simulation configuration, and evaluate our proposed design in terms of performance and energy. Then we give a brief estimation on the hardware area.

4.1 Simulation Configuration

We use a cycle accurate GPGPU simulator, GPGPU-Sim v3.2 [2], to model the detailed GPU architecture. We augment the simulator with the STT-RAM hybrid register file, with respect to the circuit level parameters of both SRAM and STT-RAM. Our baseline GPU is configured as NVIDIA Fermi GTX480. We modify the PTX parser to perform dead read identification.

We select 18 representative benchmarks from Rodina [3], Parboil [15], Mars [4] and NVIDIA CUDA SDK [10] benchmark suites to evaluate our proposed design. These benchmarks cover a wide range of applications with various characteristics.

We compare different configurations as follows.

- base: the baseline register file built with SRAM.
- STT: the register file design built with STT-RAM.
- WB: the STT-RAM based register file with a SRAM write buffer, configured as in previous work [9]. Since there is no read disturbance protection in this scheme, the line error rate is so high that it is not a feasible for GPUs.
- RD: the STT-RAM based register file design with the selective restore scheme [17].
- CO: Configured similar with RD, but with the dead read identification technique.
- CORB: Configured similar with CO, but with the read buffer promotion design.
- CORBAR: Configured similar with CORB, but with the adaptive restore scheme.

4.2 Performance Evaluation

Fig. 5 shows the GPU throughput under different configurations, all normalized to the baseline. It is clear to see that employing the selective restore scheme (RD) to suppress read disturbance incurs unacceptable performance loss, achieving only 83.4% of the SRAM baseline performance on average. This is because longer read operations (plus restore time) significantly increases the probability of stalling pipelines. Fortunately, our proposed dead read identification scheme (CO) can effectively improve throughput and achieve 90.6% of the baseline performance, due to the capability of avoiding unnecessary restore operations. For some applications, such as BAC, NEU and MON, CO can even alleviate the performance loss to achieve 96.26% of the baseline performance. Moreover, CORB further improves performance to 91.9% of the baseline on average, due to the capability of accommodating read request with multiple accesses. Some benchmarks like LAV, LPS and LUD can significantly benefit from the high temporal locality. Finally, CORBAR achieves 93.1% of the baseline performance on average, since it can adaptively choose when to use DR or SR according to whether the corresponding bank is busy or not. This adaptive feature is especially helpful for benchmarks like LPS, SRA and PAT, which have higher number of bank conflicts.

5. CONCLUSION

STT-RAM has been explored as a promising alternative for SRAM

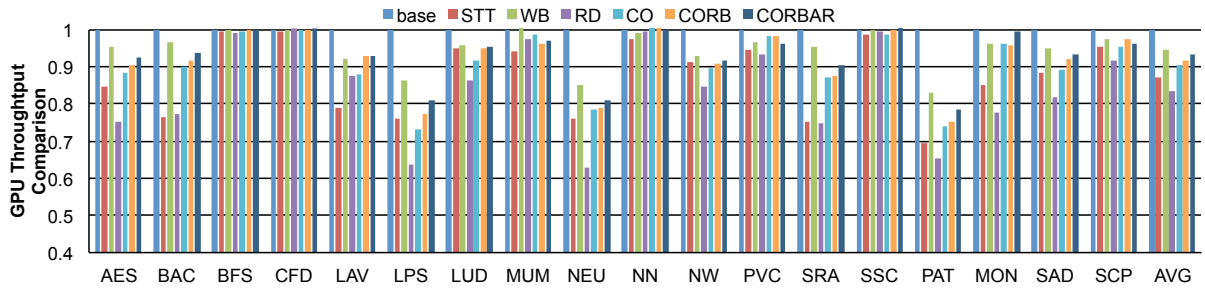

Figure 5: GPU throughput comparison between different configurations

to build the large-capacity register file on GPUs due to the advantage of high energy-efficiency. However, the read disturbance issue of STT-RAM imposes great challenges for register file design as technology further scales. To address the read disturbance issue, we present a novel design, *Red-shield*. It employs complier optimization to filter out short-lifetime reads so as to mitigate the performance and energy overhead incurred by the read-restore operations. Coupled with the read buffer promotion design as well as the optimized read-restore scheme, *Red-shield* can effectively alleviate pipeline stalls caused by read disturbance and improve the energy-efficiency. In total, our design can promote STT-RAM based GPU register files to become a feasible and effective solution for future GPU architectures.

6. ACKNOWLEDGEMENTS

We are grateful to our anonymous reviewers for their suggestions to improve this paper. This work is supported by National Natural Science Foundation of China, under grant Nos. 61433019, U1435217, 61232003, 61502514, 61402503, 61402501, 61120106005 and 61303073; National High Technology Research and Development 863 Program of China, under grant 2015AA015305.

7. REFERENCES

[1] A. W. Appel. *Modern compiler implementation in C*. Cambridge university press, 1997.

[2] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, ISPASS, pages 163–174, 2009.

[3] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron. A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads. In *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, IISWC, 2010.

[4] W. Fang, B. He, Q. Luo, and N. K. Govindaraju. Mars: Accelerating MapReduce with graphics processors. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):608–620, 2011.

[5] M. Gebhart, S. W. Keckler, and W. J. Dally. A compile-time managed multi-level register file hierarchy. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, MICRO, 2011.

[6] N. Goswami, B. Cao, and T. Li. Power-performance co-optimization of throughput core architecture using resistive memory. In *Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA, 2013.

[7] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, ISCA, page 487, 2013.

[8] G. Li, X. Chen, G. Sun, H. Hoffmann, Y. Liu, Y. Wang, and H. Yang. A STT-RAM-based low-power hybrid register file for GPGPUs. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, DAC, pages 1–6, 2015.

[9] X. Liu, M. Mao, X. Bi, H. Li, and Y. Chen. An efficient STT-RAM-based register file in GPU architectures. In *Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, ASP-DAC, pages 490–495, 2015.

[10] NVIDA. GPU Computing SDK.

[11] D. J. Palframan, N. S. Kim, and M. H. Lipasti. Precision-aware soft error protection for GPUs. In *The IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA, pages 49–59, 2014.

[12] D. Rossi, N. Timoncini, M. Spica, and C. Metra. Error Correcting Code Analysis for Cache Memory High Reliability and Performance. In *Design, Automation, and Test in Europe (DATE)*, HPCA, pages 1–6, 2011.

[13] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: a large-scale field study. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems (SIGMETRICS)*, SIGMETRICS, 2009.

[14] M. J. Schulte, M. Ignatowski, G. H. Loh, B. M. Beckmann, W. C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S. K. Reinhardt, and G. Rodgers. Achieving Exascale Capabilities through Heterogeneous Computing. *IEEE Micro*, 35(4):26–36, jul 2015.

[15] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *IMPACT Technical Report*, 2012.

[16] Z. Sun, H. Li, and W. Wu. A dual-mode architecture for fast-switching STT-RAM. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design (ISLPED)*, ISLPED, pages 45–50, 2012.

[17] R. Wang, L. Jiang, Y. Zhang, L. Wang, and J. Yang. Selective restore: an energy efficient read disturbance mitigation scheme for future STT-MRAM. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, DAC, pages 1–6, 2015.