Soft-Error Reliability and Power Co-Optimization for GPGPUs Register File using Resistive Memory

Jingweijia Tan
ECE Department,
University of Houston
Houston, TX, USA
jtan12@uh.edu

Zhi Li
EECS Department
University of Kansas
Lawrence, KS, USA
zli@ku.edu

Xin Fu
ECE Department
University of Houston
Houston, TX, USA
xfu8@central.uh.edu

***Abstract*—The increasing adoption of graphics processing units (GPUs) for high-performance computing raises the reliability challenge, which is generally ignored in traditional GPUs. GPUs usually support thousands of parallel threads and require a sizable register file. Such large register file is highly susceptible to soft errors and power-hungry. Although ECC has been adopted to register file in modern GPUs, it causes considerable power overhead, which further increases the power stress. Thus, an energy-efficient soft-error protection mechanism is more desirable. Besides its extremely low leakage power consumption, resistive memory (e.g. spin-transfer torque RAM) is also immune to the radiation induced soft errors due to its magnetic field based storage. In this paper, we propose to LEverage reSistive memory to enhance the Soft-error robustness and reduce the power consumption (LESS) of registers in the General-Purpose computing on GPUs (GPGPUs). Since resistive memory experiences longer write latency compared to SRAM, we explore the unique characteristics of GPGPU applications to obtain the win-win gains: achieving the near-full soft-error protection for the register file, and meanwhile substantially reducing the energy consumption with negligible performance loss. Our experimental results show that LESS is able to mitigate the registers soft-error vulnerability by 86% and achieve 60% energy savings with negligible (e.g. 4%) performance loss.**

Keywords—GPGPU; Register File; Soft Error; Reliability; Energy Efficiency; Resistive Memory

I. INTRODUCTION

With the strong computing power and improved programmability, graphics processing units (GPUs) become a highly attractive platform for HPC applications [8]. GPUs are originally designed for graphics applications which are able to tolerate faults by nature, thus the error detection and fault tolerance features are largely ignored. Different from graphics applications, HPC applications have rigorous demand on execution correctness and raise the serious reliability challenge in the general-purpose computing on GPUs (GPGPUs) [2, 4, 5, 36]. Among various emerging reliability concerns, soft errors, which are the failures caused by the high-energy neutron or alpha particle strikes in integrated circuits, receive major attention [1]. GPGPUs with hundreds of cores integrated in a single chip are highly vulnerable to the soft errors [2, 36].

GPGPUs generally support thousands of simultaneously active threads. The extreme multithreading requires a sizeable register file (RF), which is much larger than that in CPU processors: e.g., the RF size is 2MB in NVIDIA Fermi [6] and 6MB in AMD Cayman [3]. RF holds contexts of all the parallel threads and exposes them to the particle strikes, leading to high SER. It has been found that such large yet high vulnerable RF emerges the major contributor to the overall SER of GPUs [4, 5]. Meanwhile, RF is known as the power-hungry structure in GPGPUs [8-9, 35, 38-39] because accessing such a large structure consumes substantial dynamic power and its leakage power is non-trivial as well. In this paper, we focus on the soft-error reliability and power co-optimization for GPGPUs RF.

Although the single-error-correction double-error-detection ECC (SEC-DED ECC) has been applied to registers in recent GPUs products (e.g. Nvidia Fermi [6] and Kepler [7]), it causes considerable area and energy overhead [2, 21]. This is because extra energy is required for every single register write to generate the ECC and read to check the ECC. Since the GPGPUs register file is frequently accessed, there is large ECC generating and checking energy overhead during the program execution. Moreover, the overhead further increases when the multi-bit error correction is requested. As can be seen, ECC fails to achieve the goal of co-optimizing both soft-error reliability and power in GPGPUs RF.

Recently, resistive memory, such as spin-transfer torque RAM (STT-RAM), emerges as the promising candidate for future universal memory. It has been proposed in the GPU microarchitecture design [34-35]. STT-RAM offers several benefits such as extremely low leakage power, high density, non-volatility, and competitive read time as compared to SRAM. Besides these, its advantage of the immunity to soft error attacks [13, 15, 30-31] provides the great opportunity to build robust and low-power storage-cell based structures, such as the register file, in GPGPUs.

In this study, we propose to LEverage reSistive Memory to effectively mitigate both the registers Soft-error vulnerability and energy consumption (LESS). Although it exhibits the power and reliability advantages, STT-RAM experiences significantly slower write latency than SRAM [10-12]. Since register file is the crucial structure on exploring the thread-level parallelism, its access time affects the GPU performance [38]. In addition, the long-latency write on RF will occupy the result bus resources for multiple cycles, which can cause pipeline stalls [22]. Therefore, building a pure STT-RAM based register file will cause significant performance loss. We explore *the hybrid STT-RAM and SRAM based register file* in LESS, and explore *the unique characteristics of GPGPU applications* to hide the long write latency to STT-RAM and obtain the win-win gains*: achieving the near-full soft-error protection for the register file, and meanwhile substantially reducing the power consumption with negligible performance loss.*

LESS is composed of two techniques (lifetime-aware LESS and narrow-width-aware LESS), and our experimental results exhibit that the integrated LESS technique successfully builds a soft-error robust (86% vulnerability mitigation) and low-power (60% energy savings) register file in GPGPUs with only 4% performance loss. To our best knowledge, this is the first work to leverage the advantage of STT-RAM on particle strikes in building the soft-error robust and low-power GPGPUs.

The rest of this paper is organized as follows: Section II provides background on GPGPUs pipeline, multi-banked RF, and STT-RAM. Section III presents the proposed LESS technique. Section IV describes our experimental methodologies. Section V evaluates the proposed mechanisms. We discuss the related work in Section VI, and conclude the paper in Section VII.

II. Background

A. GPGPUs Pipeline and Multi-Banked RF

GPGPUs typically consist of a number of in-order cores, called streaming multiprocessor (SM) [14]. GPGPUs execute highly-parallel kernels, and a kernel is composed of a grid of light-weighted threads. Threads in the SM execute in SPMD model. A number of individual threads (e.g. 32 threads) are grouped together, called warp. At any given cycle, a warp that is ready for execution is selected and issued by a warp scheduler for the register access and execution. All threads within a warp access the RF simultaneously, and the operand buffer collects the operands read out from registers and sends them to the execution stage (as described in Figure 2).

Register file (RF) in the SM is highly-banked (e.g. 16 banks) to provide high bandwidth, so that multiple register operands required by one instruction can be accessed simultaneously [8, 9, 14, 38-39]. Each RF bank is equipped with dual ports to support 1 read and 1 write per cycle. Each entry in the bank is composed of 32 same-named registers in a warp. In order to reduce the possibility of bank conflicts, registers are interleaved across the RF banks.

B. Resistive Memory's Immunity to Soft Errors

When a particle strikes the storage cell, the device state flips and a soft error occurs once the amount of the charges caused by the particle is higher than a certain threshold. STT-RAM uses the Magnetic Tunnel Junction (MTJ) as the data storage device, instead of latching circuitry [10-12]. It has been observed that a particle is not able to generate sufficient charges and switch the state of a MTJ under various altitudes [15]. In other words, different from SRAM which is susceptible to the particle strikes, STT-RAM is immune to the radiation induced soft errors. There have been many studies leveraging STT-RAM to build the soft-error resilient microarchitecture [13, 15]. The thermal fluctuation and process variation can also cause soft errors in STT-RAM. However, Sun et al. [15] find that the switching possibility of an STT-RAM under thermal fluctuation or process variation is much lower than that of an SRAM caused by particle strikes. Moreover, the variation induced errors can be easily controlled via circuit parameter tuning techniques [37]. So STT-RAM is generally recognized as soft-error free [13, 15, 30, 31].

III. LESS: Leveraging STT-RAM to Build Soft-Error Robust and Low-Power Register File

In this section, we propose LESS to take advantage of STT-RAM's immunity to soft errors, and explore the energy-efficient and reliable RF. A straightforward approach to protect RF is building the pure STT-RAM based registers, named as *all STT-RAM* in this paper. We investigate a large number of GPGPUs benchmarks (detailed experiment methodologies are in Section IV), and observe that all STT-RAM degrades the performance by around 70% because the long-latency STT-RAM register writes cause pipeline stalls. Even worse, all STT-RAM increases the dynamic power consumption remarkably as a large voltage is requested during the STT-RAM write operation. The overall energy increases about 14% even though the leakage energy is substantially reduced by using STT-RAM. Relaxing the data-retention time to reduce the STT-RAM write delay and energy has been proposed in both CPU [10, 11] and GPGPUs [35]; however, it hurts the STT-RAM's soft-error robustness [17] and is not applicable to our study. We explore two soft-error protection mechanisms that are energy efficient and performance friendly in Section III.A and III.B.

A. Lifetime-Aware LESS

1) Observation: The Lifetime of GPGPUs Register Values vs. RF Soft-Error Vulnerability

Previous investigations on GPGPUs workloads have shown that majority register values have short lifetime [8, 9]. Lifetime is defined as the number of instructions between the *register value* write and its last read. Note that each physical register is written with different values multiple times during the program execution, and we consider the lifetime of each unique register value instead of the lifetime of each physical register. We further find that those short-lived register values have little impact on the RF soft-error vulnerability. Figure 1(a) presents the distribution of register values with different lifetime. As it shows, for instance, 80% of the register values are alive no more than 10 instructions. Interestingly, the remaining 20% long-lived register values contribute to about 87% of the overall register soft-error vulnerability as shown in Figure 1(b). (The studied benchmarks and vulnerability estimation methodologies are introduced in Section IV.) Using STT-RAM to provide the shield for those long-lifetime register values would achieve near-full protection for the RF.

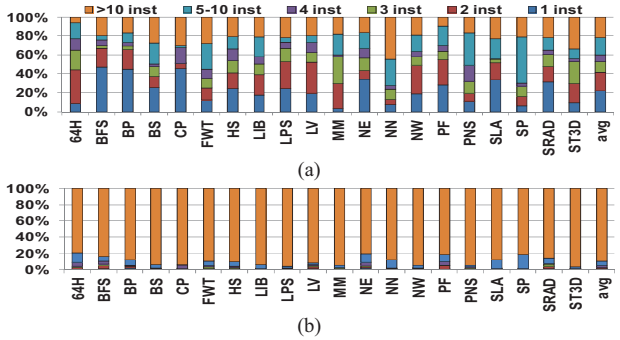


Figure 1. (a) The distribution of register values with different lifetime in instructions. (b) The contribution of register values in each lifetime category to the overall registers soft-error vulnerability.

2) The Idea of Lifetime-Aware LESS and Its Implementation

We explore the hybrid SRAM and STT-RAM based RF, and propose lifetime-aware LESS to designate the long-lifetime (e.g., longer than 10 instructions) register values to STT-RAM while other short-lifetime register values to the SRAM-based RF. By doing this, the number of STT-RAM writes decreases dramatically, which effectively alleviates both the performance loss and dynamic power overhead compared to the all STT-RAM case. Meanwhile, lifetime-aware LESS still exhibits the strong

capability in saving the leakage power due to the contribution of the STT-RAM based RF, and the overall RF power consumption drops substantially. Therefore, our technique optimizes both the RF soft-error robustness and power consumption.

Note that our proposed RF design is completely different from the register file caching (RFC) [8] and operand register file (ORF) [9] techniques, both introduced by Gebhart et al. The fundamental idea of RFC and ORF is using a small register file cache to keep the frequently read or soon to be read values and reduce the main RF access frequency for power saving. To further improve the soft-error robustness, one can keep the SRAM based implementation of the small register file cache, while using STT-RAM to build the main RF. We name it as enhanced operand register file (E-ORF). However, no matter if the value is long or short lived, its first read can occur immediately once it is produced. Therefore, E-ORF does not have the capability to differentiate the value lifetime, not mention to designate register values with different lifetime into different RF segments to maximize the fault coverage with minimal STT-RAM writes. We evaluate E-ORF and compare it with our lifetime-aware LESS in section V.A.2.

In lifetime-aware LESS, the lifetime of register values can be obtained with the help of the compiler through live variable analysis [8, 20]: the compiler analyzes the data flow graph in backward order and thus figure out the number of instructions between the register value's write and its last read. Furthermore, the register namespace is partitioned into two sets of architectural register names representing the SRAM and STT-RAM based RF, respectively. The long-lifetime (short-lifetime) register values will be properly mapped to the STT-RAM (SRAM) registers during the compilation time. The amount of SRAM and STT-RAM registers required by each thread will be sent to the GPUs during the kernel launch time.

Figure 2 shows the architectural design of our hybrid RF. In the default RF design, the warp ID is combined with the RF ID and the number of registers per thread to index the bank and the entry holding the register value. To keep the same register mapping mechanism, the SRAM and STT-RAM RF each has 16 banks with 1 read and 1 write ports, banks with the same bank number from the two RF segments share the data bus linked to the operand buffer and the execution units. As Figure 2 shows, the register index is based on the RF ID in certain name space (i.e., SRAM or STT-RAM) and the number of that kind of registers required by each thread. And one multiplexer is used to choose SRAM or STT-RAM bank for one register access. There is no other hardware required to support the STT-RAM or SRAM register accesses.

The SRAM and STT-RAM RF size partition largely impacts the effectiveness of lifetime-aware LESS, and an ideal partition will match the ratio between the maximum requirements of the two RF types through the entire execution time. We collect the ratio between the numbers of per-thread SRAM and STT-RAM register requirements provided by the compiler in various GPGPU benchmarks. The averaged ratio across all benchmarks is 75% STT-RAM and 25% SRAM when defining the register value lifetime that is longer than 10 instructions as long-lived register values (Based on our sensitivity analysis, setting the lower limit of the long lifetime as 10 instructions leads to the optimal results). We then use this ratio to partition 128KB RF in a default RF configuration into 96KB STT-RAM and 32KB SRAM RF segments. In the case that there are insufficient SRAM registers for threads in an SM, some free STT-RAM registers will be used to hold register values which are originally mapped to the SRAM RF, and vice versa.

In lifetime-aware LESS, the write operations to STT-RAM registers cannot be fully eliminated, and the 20% long-lifetime register value writes into the STT-RAM could still cause considerable performance loss. Thus, we further explore to absorb the STT-RAM write delay in the following subsection.

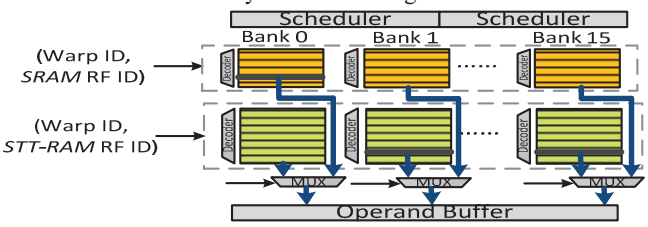


Figure 2. The SRAM (top) and STT-RAM (bottom) hybrid register file in lifetime-aware LESS.

B. Narrow-Width-Aware LESS

1) Observation: Narrow-Width Values in STT-RAM Writes

Generally, not all register values will be 32-bit long in general-purpose applications [16]. In this study, a narrow-width write is defined as the case that the upper 16 bits of all the 32 same-named register values are zeros during the writeback stage. There are numerous narrow-width writes in GPGPUs applications [18], and we observe that 63% of the STT-RAM writes are narrow-width writes on average across the investigated benchmarks, as shown in Figure 3. Since a narrow-width write only requires half of the result bus bandwidth, two narrow-width STT-RAM writes can be combined to share the bus. This will greatly reduce the performance loss caused by STT-RAM writes.

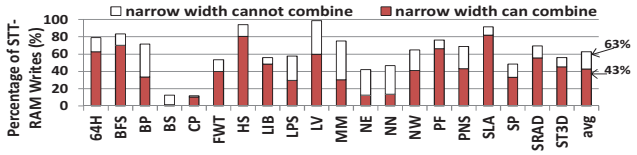


Figure 3. Percentage of STT-RAM writes that are narrow width and can be combined, and that are narrow width but cannot be combined.

To keep the design simple, we only consider combining narrow-width STT-RAM writes from two warps that (i) occur in two continuous cycles, and (ii) designate to different banks as there is only one write port in each bank. Note that the two warps can execute totally different instructions since the combination only happen at the write back stage. We observe that there are still 43% STT-RAM writes satisfying those two requirements, as shown in Figure 3. This is because threads in a kernel all execute the same code and warps usually proceed at similar rate under the fine-grained multithreading in the SM. Moreover, the same instruction code tends to write data with the same data width. Thus two continuous warps tend to both perform narrow-width writes. In summary, there are considerable amount of STT-RAM writes that can be combined to further improve the performance on top of lifetime-aware LESS.

2) The Idea of Narrow-Width-Aware LESS and Its Implementation

We propose narrow-width-aware LESS, which intelligently assigns the free data bus resources induced by the narrow-width STT-RAM write from one warp to another warp's narrow-width STT-RAM write during the writeback stage. Therefore, the result bus resources are well utilized to tolerate the negative performance impact caused by the long STT-RAM writes, and meanwhile, the error coverage obtained in lifetime-aware LESS is still maintained.

Our proposed idea is orthogonal to the traditional write buffer design [28], which attaches a SRAM based write buffer to the STT-RAM bank to hide the STT-RAM write latency. Note that all narrow-width writes will proceed as long as there are available bus resources. In other words, the previous narrow-width write will not wait to combine with the following narrow-width write.

Figure 4 describes the implementation of the proposed technique at the writeback stage. We leverage the zero detection logic in execution units to detect the narrow-width values [16], and add one bit (i.e., narrow bit) to indicate whether all the 32 same-named register values are narrow-width or not. Since two narrow-width STT-RAM writes can be performed simultaneously, we need to keep the destination register IDs for both warps. They are named as high RF ID and low RF ID, respectively, as shown in Figure 4. The high (low) RF ID keeps the destination RF ID for data using the higher (lower) 16 bits of each 32-bit data slot in the result bus. In addition, two narrow-width valid bits (i.e., high-NV, and low-NV) are attached to those two destination register IDs to indicate if the higher/lower part of each 32-bit data slot contains narrow-width value. As Figure 4 shows, a tri-state logic and a demultiplexer are also used to control the writes of the higher and lower 16 bits of each 32 same-named register values into the result bus, respectively.

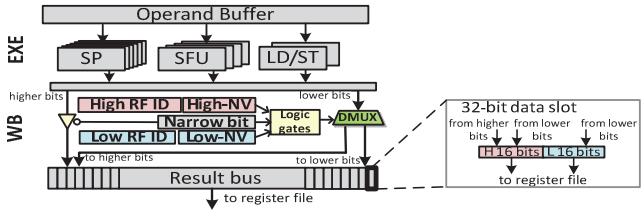


Figure 4. The implementation of narrow-width-aware LESS.

When a warp instruction finishes execution, its narrow bit is checked. If it is set as 0, it means that values to be written back to the RF are normal-width. Thus both higher and lower 16 bits of each value will be written into the corresponding slots in the result bus when it is free. Moreover, the low RF ID is set as the instruction's destination RF ID, and both low-NV and high-NV are reset to 0, indicating this is a normal-width (i.e., 32-bit) write. When the narrow bit is set as 1, both high-NV and low-NV are checked, and there are three possible scenarios as follows: (i) if they are both 0 and the result bus is available, it implies that no previous write is occupying the result bus, and the lower 16 bits of each register value are directed to the lower half of each 32-bit data slot; (ii) in the case that only one narrow valid bit is 0, it implies that there is one previous narrow-width write currently using the bus, and the current narrow-width write will be designated to the free resources according to the narrow valid bit. In the above two scenarios, the corresponding destination register ID and narrow valid bit will be updated to record the information of the current narrow-width write. Note that the higher 16 bits of each value are disabled via the tri-state logic as they are all zeros; (iii) when both narrow valid bits are 1s, it means that the result bus is already fully occupied by two previous narrow width writes, thus the current narrow-width write has to be stalled and wait for the available bus resources. Finally, when writing data to the STT-RAM banks, the destination RF IDs combined with the narrow valid bits are used to write each half of the 32-bit data into the appropriate banks. Once one warp instruction's write finishes, its information saved in the destination RF ID and narrow valid bits is cleared.

C. LESS: Putting It All Together

The merge of LA-LESS and NWA-LESS delivers a unified low-power and error protection scheme, LESS, for register file. Although some simple logic gates are introduced by LESS, it does not introduce additional area overhead thanks to the high-density of STT-RAM based RF, our gate-level modeling also proves this. Note that LESS aims to use STT-RAM to provide shield for registers, and the RF soft errors do not migrate to any other GPGPU structures.

IV. Experimental Setup

We use GPGPU-Sim (v3.1.0) [22] and simulate PTXPlus instruction set, which is a one-to-one mapping to the native hardware instruction set. We focus on the 40nm process technology which is adopted in recently released GPGPU products. Our baseline GPGPUs configuration models the Nvidia Fermi style architecture: the warp size is 32; the GPUs contain 16 SMs; each SM supports 1536 threads and 8 blocks at most; Each SM contains 128KB RF, 48KB shared memory, 16 KB L1 data cache, and 128KB L2 data cache. The scheduler applies the round robin scheduling policy. We collect workloads from Nvidia CUDA SDK [25], Rodinia Benchmark [26], and Parboil Benchmark [27]. We simulate all kernels in each benchmark to complete.

We use error coverage to evaluate the soft-error reliability of GPGPUs RF, which is calculated based on architecture vulnerability factor (AVF) [23]. A hardware structure's AVF refers to the probability that a soft error in that hardware structure will result in incorrect program results. The sum of AVF and error coverage is equal to one. As discussed in Section II.B, STT-RAM is considered as soft-error free in our experiment. Thus the error coverage for STT-RAM based register file is 100%. We build our power model based on the energy analysis tool CACTI [32]. The SM frequency is 600 MHz, and the supply voltage is 0.9V. We obtain the access time and energy of the SRAM and STT-RAM based RF with various size designs (listed in Table 1) based on a modified NVSim [24] simulator. In the experiment, the read access latency for SRAM and STT-RAM are both 1 cycle. Note that one STT-RAM write causes 4-cycle delay in our study, which is usually tens of cycles in previous studies on STT-RAM based L1/L2 caches in CPUs [10]. It is because SM runs at much lower frequency than CPU processors. The number of read/write operations to the two RF segments, and the total execution time are collected from the modified GPGPU-Sim to evaluate both RF dynamic and leakage energy. Our energy estimation is consistent with previous studies [8, 9, 39]. The energy consumed by the hardware introduced by LESS is also included in the model.

V. Evaluation

A. Lifetime-Aware LESS (LA-LESS)

1) The Effectiveness of LA-LESS

Figure 5 shows the normalized (a) execution time, (b) RF error coverage, and (c) RF energy (including both dynamic and leakage) obtained by LA-LESS when running the investigated benchmarks. The results are normalized to the baseline case without any optimization.

As it shows, on average, LA-LESS achieves the error coverage as high as 86% since the major RF vulnerability contributors (i.e., long-lived register values) are well protected by STT-RAM. Excitingly, LA-LESS gains 60% energy reduction on average, because the extra write energy caused by the infrequent STT-RAM RF writes is trivial when compared with the energy savings

brought by the reduced per access energy on the small SRAM RF and the extremely low leakage on the STT-RAM RF.

TABLE 1. ACCESS TIME AND ENERGY FOR STT-RAM AND SRAM BASED RF. OUR HYBRID RF IN LESS IS COMPOSED OF 96KB STT-RAM AND 32KB SRAM RF SEGMENTS AS DISCUSSED IN SECTION III.A.2.

	SRAM based RF		STT-RAM based RF	
	32KB	*128KB*	*96KB*	*128KB*
Write Lat (ns)	0.497	1.06	5.124	6.036
Write Lat (cycles)	1	1	4	4
Read Dyn. Eng (nJ)	0.049	0.131	0.082	0.092
Write Dyn. Eng (nJ)	0.043	0.123	0.529	0.645
Leakage power (mW)	31.2	130	3.21	4.283

As its main drawback, LA-LESS causes around 11% performance loss. Take NN as an example, the execution time increases by 60% because the long-lifetime register values account for 40% of the register values, and the frequent STT-RAM writes hurt the throughput. Fortunately, the energy savings for NN is promising because its warps contain few threads, the RF utilization in NN is low and leakage is the main component of the total energy. The reduced leakage in the hybrid RF far outweighs the considerable STT-RAM write energy. ST3D exhibits similar behavior as NN.

2) Comparison with Enhanced Operand Register File (E-ORF)

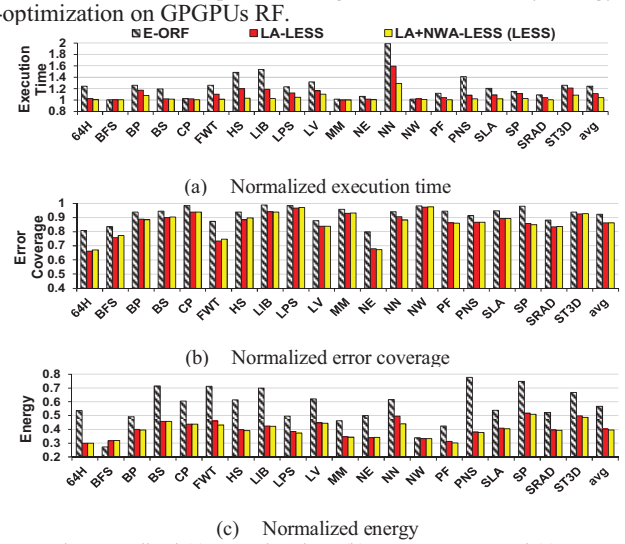
Figure 5 also compares LA-LESS with the enhanced operand register file (E-ORF) technique that applies STT-RAM to build the main RF and uses SRAM to build the ORF (as discussed in Section III.A.2). We follow the ORF configuration presented in [9] and set its size as 18KB. As it shows, LA-LESS achieves slightly (i.e. 6%) lower error coverage than E-ORF. In LA-LESS, the short-lived register values are always directed to the SRAM based RF. But in E-ORF, some short-lived register values whose first read occurs a few instructions away from the value generation will be recognized as exhibiting weak temporal locality and directed to the STT-RAM based main RF. The improved error coverage obtained by E-ORF is mainly from the additional protection for that kind of short-lived register values. As the serious negative impact of gaining such small reliability improvement, ORF consumes 17% more energy and extends the execution time by 13% compared to LA-LESS because of the increased write operations to the STT-RAM based main RF.

B. LESS: Combining LA-LESS and Narrow-Width-Aware LESS (NWA-LESS) Together

As we described in Section III, NWA-LESS can optimize 43% of the STT-RAM writes in LA-LESS. Instead of evaluating NWA-LESS technique separately, we focus on the aggregated impact and evaluate the effectiveness of LESS, which combines LA-LESS and NWA-LESS together, as shown in Figure 5. It is also named as LA+NWA LESS in Figure 5. Thanks to NWA-LESS, LESS further reduces the performance penalty when compared with LA-LESS and meanwhile, it maintains the high error coverage and low energy consumption. The performance benefit of NWA-LESS is obvious in benchmarks that contain numerous narrow-width STT-RAM writes. For example, the performance loss of ST3D reduces from 19% to 8%. This is because 45% of the STT-RAM writes are narrow-width values. On average, LESS performs well in almost all the benchmarks, and it obtains 86% RF error coverage, 60% RF energy reduction, with only 4% performance loss.

C. Comparing LESS with Other Soft-Error Protection Techniques for GPGPU RF

We further compare LESS with several other RF protection mechanisms in Figure 6: *all STT-RAM* that builds the pure STT-RAM based RF to achieve full error protection; *ECC* [6] that attaches the SEC-DED ECC (7-bit long) to every register for the error protection; *Shield* [33] that explores an ECC table to only protect the long-lifetime register values. As it shows, all STT-RAM has the highest (70%) performance loss and consumes 14% energy overhead when writing every value into the pure STT-RAM based RF. ECC has strong capability in gaining full coverage without hurting performance, however, it needs 38% extra energy to keep the ECC bits, and generate/check ECC for each register access. Shield significantly reduces the ECC's energy overhead by 30% as it uses a relatively small ECC table to protect long-lived register values and achieve good error coverage. As can be seen, all these techniques introduce extra energy overhead. Excitingly, LESS saves tremendous amount of energy with negligible performance penalty. Moreover, the energy overhead increases in ECC related mechanism when the multi-bit error tolerance is desired; while LESS still keeps its benefit since STT-RAM is immune to soft errors. Thus, LESS is more efficient than ECC related techniques with regard to the reliability-energy co-optimization on GPGPUs RF.



(a) Normalized execution time

(b) Normalized error coverage

(c) Normalized energy

Figure 5. The normalized (a) execution time, (b) error coverage, and (c) energy of enhanced operand register file (E-ORF) [13], the proposed lifetime-ware LESS (LA-LESS), and LA-LESS + the proposed narrow-width-aware LESS (LA+NWA-LESS).

We further use McPAT [19] to estimate the energy consumption of GPGPUs, and find that register file consumes ~14% of the total GPGPUs chip energy (including both dynamic and leakage). Therefore, LESS achieves ~8.4% energy savings for the entire GPGPUs chip; while ECC technique increases the chip's energy consumption by ~5.3%. The energy-efficiency of LESS outperforms ECC even it causes 4% performance loss.

D. Die Cost Analysis

Building STT-RAM based RF in GPUs chip requires additional manufacturing cost. In this subsection, we estimate the die cost of our LESS technique. Eq. 1 describes the estimation model [29] used in this study:

$$die\_cost = \frac{wafer\_cost}{\frac{wafer\_area}{die\_area} * yield} \quad \text{Eq. 1}$$

where *yield* is the good die percentage in a wafer. LESS increases the wafer cost by 5% due to the additional manufacturing process of STT-RAM [29]. We calculate the RF area via using McPAT [19], and find that it occupies around 13% of the GPGPUs die area. Since LESS reduces the RF area by 55%

compared to the baseline case, it reduces the overall GPGPUs die area by 7%. Given the above information about wafer cost and die area, we find that the die cost of GPGPUs integrated with LESS is still slightly (e.g., 2%) lower than that of the baseline case under various yields ranging from 40% to 100%. Therefore, the additional manufacturing cost introduced by LESS does not have negative impact on the die cost.

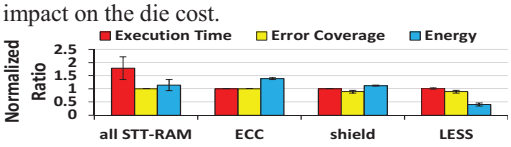


Figure 6. Comparison among all STT-RAM, ECC, shield, and LESS techniques, the standard deviations across benchmarks are also shown.

VI. Related Work

There have been works on investigating, detecting, and recovering the soft errors on GPUs [2, 4, 5, 36]. For example, Farazmand et al [5] propose a statistical fault injection technique to analyze the soft-error vulnerability of GPU structures. In our study, we explore the soft-error immunity of STT-RAM to protect GPU registers, and all those previously proposed schemes are orthogonal to LESS.

STT-RAM has been widely investigated as a replacement for SRAM based structures [10-12]. For instance, Smullen et al. [11] redesign the STT-RAM cell to improve its write time and energy by trading-off the retention time. Besides using STT-RAM to build stand-alone memory, there have been many studies integrating STT-RAM into both CPU and GPGPUs microarchitecture design [12, 35]. For example, Guo et al. [12] explore storage and combinational logic with STT-RAM to resist the power wall of CMPs. All these studies focus on the high density, low leakage features of STT-RAM to deliver energy-efficient CPUs/GPUs. And the studies of using STT-RAM soft-error resilience for reliable computing limit in CPUs [13, 15]. To our knowledge, this is the first work to use STT-RAM combined with the GPGPU unique characteristics to build soft-error robust and low-power GPGPUs RF.

VII. Conclusions

The increasing adoption of GPGPUs for HPC applications raises their reliability challenge. GPUs require a sizable RF to support thousands of parallel threads, making RF highly susceptible to soft errors and power-hungry. Although ECC has been added in modern GPUs, it causes considerable energy overhead. An energy-efficient protection mechanism is desired for the vulnerable and power-hungry register file. STT-RAM exhibits the benefit of low leakage and high density, and more interestingly, it is immune to the radiation induced soft errors. In this paper, we propose LESS, which leverages STT-RAM and the unique characteristics of GPGPU applications to explore the soft-error robust and low-power RF in GPGPUs without hurting the performance. LESS is composed of two mechanisms: LA-LESS that designs a SRAM and STT-RAM hybrid RF and protects the long-lived register values via STT-RAM RF; and NWA-LESS that combines two narrow-width STT-RAM writes. LESS achieves 86% RF soft-error vulnerability improvement, 60% RF energy savings with only 4% performance loss.

Acknowledgement

This work is supported in part by NSF grants CCF-1320730, CCF-1351054 (CAREER), EPS-0903806 and matching support from the State of Kansas through the Kansas Board of Regents.

References

[1] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor, ISCA, 2004.
[2] K. Yim, C. Pham, M. Saleheen, Z. Kalbarczyk, and R. Iyer. Hauberk: Lightweight Silent Data Corruption Error Detectors for GPGPU, IPDPS, 2011.
[3] D. Kanter. AMD's Cayman GPU architecture, 2010. http://www.realworldtech.com/cayman/.
[4] J. Tan, N. Goswami, T. Li, and X. Fu, Analyzing Soft-Error Vulnerability on GPGPU Microarchitecture, IISWC, 2011.
[5] N. Farazmand, R. Ubal, D. Kaeli, Statistical Fault Injection-Based AVF Analysis of a GPU Architecture, SELSE, 2012.
[6] NVIDIA's Next Generation CUDA™ Compute Architecture: Fermi™, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
[7] NVIDIA's Next Generation CUDA™ Compute Architecture: Kepler™ GK110, http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf
[8] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron. Energy-Efficient Mechanisms for Managing Thread Context in Throughput Processors, ISCA, 2011.
[9] M. Gebhart, S. W. Keckler, and W. J. Dally. A Compile-Time Managed Multi-Level Register File Hierarchy, MICRO, 2011.
[10] Z. Sun, X. Bi, H. Li, W. Wong, Z-L Ong, X. Zhu, and W. Wu, Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme, MICRO, 2011.
[11] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches, HPCA, 2011.
[12] X. Guo, E. Ipek, T. Soyata, Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-RAM Based Computing, ISCA, 2010.
[13] H. Sun, C. Liu, W. Xu, J. Zhao, N. Zheng, and T. Zhang, Using Magnetic RAM to Build Low-Power and Soft Error-Resilient L1 Cache, IEEE Trans. on VLSI, 2010.
[14] NVIDIA. CUDA Programming Guide Version 3.0., Nvidia Corporation, 2010.
[15] G. Sun, E. Kursun, J. Rivers, Y. Xie, Exploring the Vulnerability of CMPs to Soft Errors with 3D Stacked Non-Volatile Memory, ICCD, 2011.
[16] D. Brooks and M. Martonosi, Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance, HPCA, 1999.
[17] C. W. Smullen, Designing Giga-Scale Memory Systems with STT-RAM, Ph.D. Dissertation, 2011.
[18] S. Z. Gilani, N. S. Kim. M. Schulte, Power-Efficient Computing for Compute-Intensive GPGPU Applications, HPCA, 2013.
[19] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures MICRO, 2009.
[20] J. Whaley, D. Avots, M. Carbin, and M. S. Lam, Using datalog with binary decision diagrams for program analysis, APLAS, 2005.
[21] R. Walker, R. Betz, An Investigation of the Effects of Error Correcting Code on GPU-Accelerated Molecular Dynamics Simulations, XSEDE, 2013.
[22] A. Bakhoda, G.L. Yuan, W. W. L. Fung, H. Wong, Tor M. Aamodt, Analyzing CUDA Workloads Using a Detailed GPU Simulator, ISPASS, 2009.
[23] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor, MICRO, 2003.
[24] X. Dong, C. Xu, Y. Xie, N. P. Jouppi, NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory, IEEE Trans. on CAD, 2012.
[25] http://www.nvidia.com/object/cuda_sdks.html
[26] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing, IISWC, 2009.
[27] Parboil Benchmark Suite. URL: http://impact.crhc.illinois.edu/ parboil.php
[28] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs, HPCA, 2009.
[29] M. T. Chang, P. Rosenfeld, S. L. Lu, and B. Jacob, Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM, HPCA, 2013.
[30] Freescale, Document Number: Brmramslscltrl-Freescale MRAM Technology, 2007.
[31] S. Tehrani, Status and Prospect for MRAM Technology, 2010.
[32] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.1. HP Labs, Tech. Rep. 2008.
[33] P. Montesinos, W. Liu, and J. Torrellas, Using Register Lifetime Predictions to Protect Register Files Against Soft Errors, DSN, 2007.
[34] P. Satyamoorthy, STT-RAM for Shared Memory in GPUs, Master's Thesis, The University of Virginia, 2011.
[35] N. Goswami, B. Cao, and Tao Li, Power-performance Co-optimization of Throughput Core Architecture using Resistive Memory, HPCA, 2013.
[36] D. Palframan, N. S. Kim, M. H. Lipasti, Precision-Aware Soft Error Protection for GPUs, HPCA, 2014.
[37] Y. Emre, C. Yang, K. Sutaria, Y. Cao, and C. Chakrabarti, Enhancing the Reliability of STT-RAM through Circuit and System Level Techniques, SiPS, 2012.
[38] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, X. Liang, An Energy-Efficient and Scalable eDRAM-Based Register File Architecture for GPGPU, ISCA, 2013.
[39] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, V. J. Reddi, GPUWattch: Enabling Energy Optimizations in GPGPUs, ISCA, 2013.