Modeling and characterizing GPGPU reliability in the presence of soft errors

Jingweijia Tan, Yang Yi, Fangyang Shen, Xin Fu

EECS Department, University of Kansas, Lawrence, KS, United States
CSEE Department, University of Missouri, Kansas City, MO, United States
New York City College of Technology, Brooklyn, NY, United States

Abstract
The general-purpose computing on graphic processing units (GPGPUs) becomes increasingly popular due to its high computational throughput for data parallel applications. Modern GPU architectures have limited capability for error detection and fault tolerance since they are originally designed for graphics processing. However, the rigorous execution correctness is required for general-purpose applications, which makes reliability a growing concern in the GPGPU architecture design. With CMOS processing technologies continuously scaling down to the nano-scale, on-chip soft error rate (SER) has been predicted to increase exponentially. GPGPUs with hundreds of cores integrated into a single chip are prone to manifest high SER. This paper explores a first step to model and characterize GPGPU reliability in light of soft errors. We develop GPGPU-SODA (GPGPU Software Dependability Analysis), a framework to estimate the soft-error vulnerability of GPGPU microarchitecture. By using GPGPU-SODA, we observe that several microarchitecture structures in GPGPUs exhibit high soft-error susceptibility, and the structure vulnerability is sensitive to the workload characteristics (e.g. branch divergences, memory access pattern). We further investigate the impact of several architectural optimizations on GPU soft-error robustness. For example, we find that increasing the number of threads supported by GPU significantly affects the GPGPU soft-error robustness. However, changing the warp scheduling policy has little impact on the structure vulnerability. The observations made in this study provide designers the useful guidance to build resilient GPGPUs: a comprehensive resiliency solution for GPGPUs should consider the entire GPGPU design instead of solely focusing on a particular structure.

1. Introduction
The performance of microprocessors has been improved tremendously as more and more cores are integrated into a single chip. However, the throughput of multi/many-core processors is limited to computing workloads exhibiting intensive data-level parallelism. With hundreds of on-chip cores, GPU (graphic processing unit) supports thousands of light-weight threads and provides high computational throughput for parallel applications. For example, NVIDIA’s GeForce 8800 [1] can provide up to 197× higher throughput than Intel’s Core2Duo processors for data parallel applications. AMD’s GPU (HD4870) provides 1.2 Tflop/s while Core2Quad only delivers 100 Gflop/s on peak performance [2]. There are new programming models such as NVIDIA CUDA™ [3], AMD Brook+ [4], and OpenCL [5] that greatly reduce the programmers’ efforts in

* Corresponding author.
E-mail addresses: jtan@ittc.ku.edu (J. Tan), cindyyi@umkc.edu (Y. Yi), fangyangshen@gmail.com (F. Shen), xinfu@ittc.ku.edu (X. Fu).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.01.001
writing general-purpose applications using GPUs. With their strong computing power and improved programmability, the general-purpose computation on GPUs (GPGPUs) emerge as a highly-efficient device for a wide range of parallel applications.

Existing GPU processors lack error detection capability and fault tolerant features. Historically, they are mainly designed for image and video processing, which do not require complete computation correctness. [6] observed that errors in video applications are usually masked since they only impact a few pixels, or the corrupted image is quickly recomputed in the next frame. However, the strict execution correctness is required in GPGPUs as they support the execution of general-purpose applications such as scientific computing, financial applications, and medical data processing. This makes reliability a growing concern in GPGPU architecture design. Even worse, the shrinking of feature sizes allows the manufacture of billions of transistors on a single GPGPU processor chip that concurrently runs thousands of threads. This further urges the need for characterizing and addressing reliability in GPGPU architecture design.

As the CMOS processing technologies keep scaling toward nano-scale, devices are facing higher environmental susceptibility. Soft errors, also called transient faults or single-event upsets (SEUs), are failures caused by high-energy neutron or alpha particle strikes in integrated circuits. These failures are defined as soft errors because they do not permanently damage the circuit but do destroy data. On-chip soft error rate (SER) has been predicted to increase exponentially [7,8] due to the smaller feature size, lower supply voltage, and larger integration scale. GPGPUs with hundreds of cores integrated into a single chip are prone to manifest high SER [9]. If left unattended, the reliability issue will soon become growth impediment for future GPGPU either by preventing them from scaling down to smaller feature sizes or by resulting in imprecise operation of these devices.

The emerging GPGPU architecture has motivated various studies on its performance and power issues. Ariel et al. [10] developed AerialVision to provide insights into the performance bottleneck in GPGPU architectures. Hong et al. [11] proposed an integrated power and performance prediction model for GPGPUs. However, there is little work reported on analyzing GPGPU architecture from the reliability perspective. In order to improve GPGPU throughput, architects have explored numerous GPGPU architectural design optimizations (e.g. dynamic warp formation, various warp scheduling policies) [12,25]. However, their impacts on GPGPU soft-error vulnerability are largely unknown. A performance-oriented optimization may become a poor choice if it severely degrades the GPGPU robustness. Unfortunately, lacking the insights into GPGPU reliability characteristics, designers have limited hints on building resilient GPGPUs.

This study is the first step to model and characterize the soft-error vulnerability on GPGPU architectures. To this end, we develop a reliability-aware GPGPU processor simulator, which provides cycle-accurate, microarchitecture level vulnerability estimation on GPGPUs. By using a set of GPU benchmarks (e.g. Nvidia CUDA SDK, Parboil, Rodinia), we evaluate the GPGPU vulnerability with various architecture designs and optimizations.

The contributions of this work are:

- We develop GPGPU-SODA (GPGPU Software Dependability Analysis), a unified framework to evaluate the vulnerability of major microarchitecture structures in the GPGPU Streaming Multiprocessor (SM). It will facilitate researchers in exploring robust GPGPUs to meet the increasing demands for high-performance and reliable GPU computing, and hence benefit numerous real-life applications.
- By using GPGPU-SODA, we observe that several structures in GPGPU microarchitecture (e.g. warp scheduler, registers and streaming processors) are highly susceptible to soft errors. Moreover, the structure vulnerability is sensitive to workload characteristics such as branch divergences, memory access patterns, and so on. We further find that SMs in the same GPU processor may exhibit divergent vulnerability characteristics.
- We investigate the impact of architectural optimizations on GPGPU microarchitecture vulnerability. For example, we find that dynamic warp formation (a technique to handle branch divergence) reduces the structure vulnerability, and increasing the thread quantity per SM is able to improve the warp scheduler and registers reliability but degrade streaming processors robustness. Moreover, compared to those two optimizations, changing the warp scheduling policy introduces little impact on the structure vulnerability.
- Based on our comprehensive experiments, we find that a GPGPU reliability-optimization technique should consider the entire GPGPU design. Because a technique targeting on the vulnerability mitigation for one particular structure may degrade other structures’ soft-error robustness.

The rest of this paper is organized as follows: Section 2 provides background on state-of-the-art GPGPU architecture, and soft errors. Section 3 models the GPGPU microarchitecture soft-error robustness and presents our developed GPGPU-SODA framework. Section 4 describes our experimental methodologies. Section 5 analyzes the GPGPU soft-error vulnerability. We discuss the related work in Section 6, and conclude the paper in Section 7.

2. Background

2.1. General-purpose computing on graphic processing units (GPGPUs) architecture

Fig. 1(a) shows an overview of the state-of-the-art GPU architecture [3]. It consists of a scalable number of in-order streaming multiprocessors (SM) that can access multiple memory controllers via an on-chip interconnection network.
Fig. 1(b) illustrates a zoom-in view of the SM. It includes the warp scheduler, register files (RF), streaming processors (SP), constant cache, texture cache, and shared memory. In addition to the CPU main memory, the GPU device has its own off-chip external memory (e.g., global memory) connected to the on-chip memory controllers. Some high-end GPUs also have a L1 local cache and L2 cache (shown as dotted line in Fig. 1), and error-checking-correction unit for memory.

To facilitate GPGPU application development, several programming models have been developed by NVIDIA and AMD. In this paper, we study the NVIDIA CUDA programming model but some of the basic constructs are similar to most programming models. In CUDA, the GPU is treated as a co-processor that executes highly-parallel kernel functions launched by the CPU. The kernel is composed of a grid of light-weighted threads; a grid is divided into a set of blocks (referred as cooperative thread arrays (CTA) in CUDA); each block is composed of hundreds of threads. Threads are distributed to the SMs at the granularity of blocks, and threads within a single block communicate via shared memory and synchronize at a barrier if needed. Per-block resources, such as registers, shared memory, and thread slots in a SM are not released until all the threads in the block finish execution. More than one block can be assigned to a single SM if resources are available.

Threads in the SM execute in the Single Instruction Multiple Data (SIMD) fashion. A number of individual threads (e.g., 32 threads) from the same block are grouped together, called warp. In the pipeline, threads within a warp execute the same instruction but with different data values. Fig. 1(b) also presents the details of SM microarchitecture. Each core interleaves multiple warps (e.g., 32) on a cycle-by-cycle basis: the warp scheduler holds those warps, and every cycle it selects a warp with a ready instruction (i.e., the same instruction from all the threads within the warp are ready to execute) to feed the pipeline. The execution of a branch instruction in the warp may cause warp divergence when some threads jump while others fall through at the branch. Threads in a diverged warp have to execute in a serial fashion which greatly degrades the performance. Immediate post-dominator reconvergence [14] has been widely used to handle the warp divergence. Recently, several mechanisms, such as dynamic warp formation [12], have been applied to further improve the efficiency of branch handling. Due to SIMD lock-step execution mechanism, a long latency off-chip memory access from one thread would stall all the threads within a warp, and the warp cannot proceed until all the memory transactions complete. The load/store requests issued by different threads can get coalesced into fewer memory requests according to the access pattern. Memory coalescing improves performance by reducing the requests for memory access.

2.2. Microarchitecture-level soft-error vulnerability analysis

A key observation of the soft error behavior at the microarchitecture level is that a SEU may not affect processor states required for correct program execution. At the microarchitecture level, the overall hardware structure's soft error rate is decided by two factors [15]: the FIT rate (Failures in Time, which is the raw SER at circuit level) per bit, mainly determined by circuit design and processing technology, and the architecture vulnerability factor (AVF) [16]. A hardware structure's AVF refers to the probability that a transient fault in that hardware structure will result in incorrect program results. Therefore, the AVF, which can be used as a metric to estimate how vulnerable the hardware is to soft errors during program execution, is determined by the processor's state bits required for architecturally correct execution (ACE). At the instruction level, an instruction is defined as ACE (un-ACE) instruction if its computation result affects (does not affect) the program final output, and AVF is primarily determined by the quantity of ACE instructions per cycle and their residency time within the structure [16]. In this paper, we use AVF as the major metric to estimate structure soft-error vulnerability.
3. GPGPU-SODA: GPGPU Software Dependability Analysis

In order to characterize and optimize the soft-error vulnerability of emerging GPGPU architecture, a tool to estimate the impact of soft errors on GPGPU is highly desired. Recently, Sim-SODA [17] has been developed to compute AVF of CPU microarchitecture structures. Compared to the general-purpose CPU cores, GPGPU SMs implement in-order SIMD pipeline and have significantly different architecture and data/control flow. Sim-SODA is not applicable to the GPGPU architecture. For instance, in order to calculate AVF of structures buffering in-flight instructions (e.g. issue queue and reorder buffer in CPU core, the warp scheduler in the SM), the framework needs to identify ACE instructions by tracing the data dependence chains on the instruction output(s). Sim-SODA is built upon Alpha ISA, and only supports the analysis on instructions with two inputs and one output. However, instructions in GPGPU are likely to consume/produce vector inputs/outputs, which requires more complicated analysis considering all data dependencies among the vector inputs and outputs. Moreover, Sim-SODA mainly targets on single-thread uniprocessor in which the AVF computation is relatively simple. On the contrary, GPGPU contains tens of SMs with thousands of parallel threads running simultaneously and threads within a block share data via the per-core shared memory. Therefore, the error propagation from one thread to another thread is possible. In other words, the instruction output from one thread could affect the correct execution of a different thread. When estimating the GPGPU microarchitecture vulnerability, a comprehensive methodology is required to consider the vector inputs/outputs and thread-level error propagation in ACE instruction identification, which is obviously far beyond the capability of Sim-SODA.

We build GPGPU-SODA (GPGPU Software Dependability Analysis) on a cycle-accurate, open-source and publicly available simulator GPGPU-Sim [13], which supports CUDA Parallel Thread Execution (PTX) ISA. GPGPU-SODA is capable of estimating the vulnerability of the major microarchitecture structures in SM including warp scheduler, streaming processors, registers, shared memory, and L1 texture and constant caches. The structures in SM are classified as two types: address-based structures keeping computation data values (e.g. registers, shared memory, and L1 caches), and structures buffering instructions (e.g. warp scheduler). To compute the AVF for address-based structure, GPGPU-SODA summarizes the ACE components [18] of each bit’s lifetime in the structure. In order to calculate the AVF for structures buffering instructions, GPGPU-SODA implements an instruction analysis window [16] for each thread. It explores data dependence chains for instructions with multiple inputs and outputs to perform the comprehensive ACE instruction identification. Note that even threads within a warp share the same PC, an analysis window on one thread for the entire warp is not sufficient. Since it ignores the data dependencies across threads caused by the inter-thread data sharing, an ACE instruction may be incorrectly identified as an un-ACE instruction. In CPU processors, an analysis window with a size of 40,000 instructions is required to determine un-ACE instructions. Since GPGPU workloads are composed of light-weighted threads, the window size is largely reduced to 1000 instructions in GPGPU-SODA. We classify the instruction as vulnerability-unknown if its vulnerability cannot be decided by our 1000-instruction analysis window. As our simulation results shown in Section 5, the percentage of unknown instructions is mostly lower than 5%. We use synthetic micro-benchmarks that have small amounts of instructions to validate the analysis windows. Their reports on un-ACE instructions match our expectations.

4. Experimental methodology

We use the developed GPGPU-SODA to obtain the GPGPU reliability and performance statistics. Table 1 shows the simulated baseline GPGPU configuration. It includes both SM and interconnect configurations.

Table 1

<table>
<thead>
<tr>
<th>Hardware configuration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of SMs</td>
</tr>
<tr>
<td>SIMD pipeline width</td>
</tr>
<tr>
<td>Warp size</td>
</tr>
<tr>
<td>Number of threads/SM</td>
</tr>
<tr>
<td>Maximum blocks/SM</td>
</tr>
<tr>
<td>Number of registers/SM</td>
</tr>
<tr>
<td>Shared memory/SM</td>
</tr>
<tr>
<td>Constant cache /SM</td>
</tr>
<tr>
<td>Texture cache/SM</td>
</tr>
<tr>
<td>Branch divergence method</td>
</tr>
<tr>
<td>Warp scheduling policy</td>
</tr>
<tr>
<td>Memory controller</td>
</tr>
<tr>
<td>DRAM controller</td>
</tr>
<tr>
<td>DRAM queue size</td>
</tr>
<tr>
<td>DRAM timing</td>
</tr>
<tr>
<td>DRAM banks/chip</td>
</tr>
<tr>
<td>Bus width</td>
</tr>
<tr>
<td>Interconnect topology</td>
</tr>
<tr>
<td>Routing algorithm</td>
</tr>
<tr>
<td>Virtual channels</td>
</tr>
<tr>
<td>VC buffer size</td>
</tr>
<tr>
<td>Flit size</td>
</tr>
</tbody>
</table>
We collect a large set of available GPGPU workloads from Nvidia CUDA SDK [19], Rodinia Benchmark [20], Parboil Benchmark [21] and some third party applications. We list them as follows: 64H (64 bin histogram), BFS (breadth first search), BP (back propagation), BS (Black Scholes option pricing), CP (Columbic Potential), FWT (fast walsh transform), HS (hot spot), KM (K-means), LIB (LIBOR), LPS (3D laplace solver), LV (Levenshtein Edit-distance calculation), MRIF (Magnetic resonance imaging FHD), MT (matrix transpose), NE (nearest neighbor), NN (neural network), NW (Needleman Wunsch), SLA (scan of large arrays), SP (scalar product), SRAD (Speckle Reducing Anisotropic Diffusion), ST3D (stencil 3D), STO (store GPU). The workloads show significant diversities according to their kernel characteristics, divergence characteristics, memory access patterns, and so on. They are compiled into PTX assembly, and GPGPU-SODA takes the produced PTX assembly instructions associated with the information on thread-level register, shared memory, and memory usages to perform the simulation. We simulate most benchmarks to completion except few causing extremely long simulation time. AVF is used as the basic metrics to estimate how susceptible a GPGPU microarchitecture structure is to soft error attacks.

5. Soft-error vulnerability analysis on GPGPUs

In this section, we perform a detailed reliability analysis on GPGPU streaming multiprocessors. Subsection A profiles the soft-error vulnerability of SM structures with the baseline GPGPU configuration. Section 2.2 estimates the impact of several architecture optimizations on GPGPU vulnerability.

5.1. Soft-error vulnerability profile of the baseline GPGPU

5.1.1. Instruction-level vulnerability profile

Fig. 2 profiles the instruction vulnerability characteristics in various benchmarks. It shows the percentage of ACE, un-ACE, and vulnerability-unknown instructions for each workload. The un-ACE instructions are further classified as first dynamically dead (FDD) instructions whose results are not consumed by any other instructions [16], transitively dynamically dead (TDD) instructions whose results are only read by FDD or TDD instructions [16]. On average, there are 88% ACE, 8% un-ACE, and 4% vulnerability-unknown instructions across the benchmarks. As shown in Fig. 2, most benchmarks have a large amount of ACE instructions, while benchmark SLA is the exception which only contains 41% ACE instructions. Generally, threads running in GPGPUs are mainly composed of loops, un-ACE instructions appear in the loop will be repeatedly executed and contribute to a large number of un-ACE instructions in the workload. This is the case in SLA whose un-ACE instructions mainly exist in the loops. Our GPGPU-SODA framework reports 1% FDD and 8% TDD instructions which dominate the un-ACE instructions. This is different from the general-purpose workloads in CPU processors that have 10–30% NOP instructions, which are usually inserted for pipeline flushing, or instruction alignment in VLIW processors [16,17,22].

5.1.2. Soft-error vulnerability of GPGPU microarchitecture structures

Fig. 3 shows the soft-error vulnerability of several key structures in GPGPU SMs with the baseline GPGPU configuration. We present the averaged result across the SMs. The AVF of L1, constant and texture caches are not shown in Fig. 3. This is because the workloads we studied either do not or rarely use those structures, their AVF is even lower than 4%. As shown in Fig. 3, the primary microarchitecture structures (e.g. warp scheduler, registers) that are heavily utilized exhibit high vulnerability. The AVF of the warp scheduler, registers, and streaming processors achieves up to 92%, 77%, and 95%, respectively.

(1) Analysis on Structure’s AVF in the Streaming Multiprocessors (SM)

As shown in Fig. 3, the structure vulnerability varies dramatically across the workloads. For instance, the warp scheduler’s AVF is 2% in NW but 92% in BP. In general, the GPU microarchitecture structures show comparatively low AVF in workloads containing a large number of un-ACE instructions that are immune to the soft error strikes, such as SLA. However, the AVF of
structures is not always high even in programs with a significant amount of ACE instructions. For example, the AVF of warp scheduler, streaming processors, and register files is around 5% in NE, NN, and NW benchmarks although more than 90% of the instructions are ACE. This is because they contain quite limited number of threads, most pipeline resources are idle leading to low soft-error vulnerability. As introduced in Section 2.1, the per-block resources (e.g. registers, thread slots in the warp scheduler, and shared memory) will not be released until the block completes execution, which limit the maximum number of blocks that can be simultaneously assigned to an SM. Different per-block resources become the bottleneck for kernels that have different resource requirements. The bottleneck structure is prone to be fully utilized and manifest high vulnerability, while other structures are usually underutilized and show strong capability in fault tolerance. In ST3D, register file size is the one to determine the number of blocks; and the warp scheduler has numerous idle thread slots at run time. Hence, it has low AVF even though most of the instructions sitting there are ACE. Due to the same reason, some benchmarks (e.g. BP) have reliable registers but vulnerable warp scheduler which is the resource bottleneck.

In the SM, shared memory is mainly designed for inter-thread communication; and it is not used in kernels without thread communication/synchronization. Therefore, the AVF of shared memory is zero in BFS, BS, CP, KM, LIB, MRIF, NE, and NN, as shown in Fig. 3. To improve the performance, program developers use the shared memory as software managed on-chip cache for the global memory and reduce the off-chip memory accesses. Intuitively, the shared memory turns to be vulnerable when it is used for thread communication or performance enhancement purpose, and becomes the resource bottleneck during the block allocation. However, its AVF still keeps as low as 12% on average across the benchmarks that use the shared memory. Shared memory is highly banked. The bank selected to hold a data value is determined by the data address, which leads to the unbalanced bank usage in a block. The number of blocks that each bank can support is different, and the minimum number finally limits the number of blocks the shared memory can support. Even though shared memory becomes the resource bottleneck, most banks in it may be underutilized, and the vulnerability of the entire structure is low. Fig. 4 presents the percentage of used entries in the shared memory for each benchmark. On average, it is only 24%. In workloads whose block resource allocation is limited by the shared memory (e.g. 64H, LPS, NW, SP, SRAD, and STO), more than 40% entries are never used during the entire execution time. For the used entries, they are written/read in a very short period and become free in majority of the time. Therefore, the shared memory has quite low vulnerability.

As shown in Fig. 3, streaming processors’ AVF has strong correlation to the number of ACE instructions in most benchmarks. Since the instruction execution time in streaming processors is constant, the ACE instruction quantity per cycle becomes the major factor to determine the AVF. However, streaming processors’ AVF is only 10% in BFS even 92% instructions are ACE. BFS has heavy branch divergences and long latency off-chip memory accesses. The diverged threads within the warp have to execute sequentially, which causes tremendous performance penalty. In our baseline machine configuration, the immediate post-dominator based reconvergence is applied to reconverge threads at the immediate post-dominator (detailed explanation can be found in [14]) to handle the branch divergences. However, it has limited capability to efficiently recover the performance loss. Furthermore, the long latency memory accesses increase the streaming processors idle time. Accordingly, the parallelized streaming processors are highly under-utilized in BFS, and the AVF is low.
(2) Vulnerability Variations at Streaming Multiprocessors (SM) Level

A GPGPU processor contains tens of SMs. Generally, they are equally utilized and exhibit similar vulnerability behavior. Fig. 5(a) shows the AVF of the major microarchitecture structures in each SM when executing HS. As it shows, there is small AVF divergence among the SMs. However, this is not the case for some benchmarks.

Fig. 5(b) demonstrates an example of BP with its streaming processors’ AVF varies significantly across SMs. For example, the streaming processors’ AVF is 80% in SM 9, but drops down to 29% in SM 27. This happens because the threads running in each SM have different memory access pattern, which determines the quantity of memory requests that can get coalesced (more coalesced requests implies fewer off-chip memory transactions) and the severity of memory contentions among requests affect the memory access latency. SM 27 experiences more off-chip memory accesses compared to SM 9, the streaming processors in SM 27 are idle in majority of the time, and correspondingly, they are more robust against the soft error strikes. Interestingly, the AVF of warp scheduler, register files, and shared memory does not show much difference among SMs. During the kernel launch process, all its blocks may not be able to distribute to the SMs and start the execution simultaneously since the maximum number of blocks an SM can support is limited. The block which cannot be allocated to the SM at the kernel launch time has to wait until there is one block finishes and its resources in the SM are released. By doing this, the execution time of each SM tends to be balanced. Furthermore, SMs with long block execution time caused by the frequent memory accesses execute fewer blocks, and have smaller quantity of ACE instructions in the storage-based pipeline structures (e.g. warp scheduler) through the entire execution time. However, instructions have to remain in those structures during the off-chip memory accesses, and their residency time increases. As a result, the AVF of warp scheduler, register files, and shared memory are nearly the same across SMs.

![AVF of microarchitecture structures in each SM while running HS](image1)

![AVF of microarchitecture structures in each SM while running BP](image2)

![AVF of microarchitecture structures in each SM while running SP](image3)

*Fig. 5.* The AVF of microarchitecture structures in each SM while running (a) HS, (b) BP, and (c) SP.
Fig. 5(c) shows another example of SP, the structure AVF differs substantially across SMs, and structures in the same SM show similar vulnerability behaviors. Take SM 7 and 9 for example, the vulnerability of the major structures including warp scheduler, streaming processors, and registers is low in SM 7 but it is much higher in SM 9. This is caused by the uneven distribution of blocks among SMs. SM 7 is assigned 3 blocks for execution; while the number of blocks allocated to SM 9 is doubled leading to much larger amount of ACE instructions. Moreover, different from BP, the execution time of SMs in SP varies significantly, SM 7 finishes its workloads much earlier than SM 9, it implies that the residency time of ACE instructions in both SMs are similar. Therefore, structures in SM 9 are more vulnerable to soft errors due to their high quantity of ACE instructions.

When a thread encounters a barrier instruction, it has to wait until the synchronization in the block finishes. In other words, warps will stall upon the synchronization requests. Similar to the long latency off-chip memory accesses, the barrier instructions could increase execution time, thus affect the structure soft-error robustness. Since warps are issued to the pipeline in the round-robin order in our baseline GPGPU machine, threads within the block usually proceed at similar rate; the warp waiting time at the barrier is short compared to that for the off-chip memory access. Furthermore, barrier instructions only account for 2.5% of the total instructions in most benchmarks. Their impact on structure vulnerability is trivial.

In summary, we observe that the GPGPU microarchitecture vulnerability is highly related to the workload characteristics such as the percentage of un-ACE instructions, the per-block resource requirements, the branch divergences, the memory access pattern (e.g. memory coalescing, memory resource contentions, and the memory access latency), the block allocation which determines the execution time and the amount of workload executed in each SM.

5.2. Impact of architecture optimizations on GPGPU microarchitecture-level soft-error vulnerability

In this subsection, we evaluate the impact of several popular architectural design optimizations on the soft-error vulnerability of GPGPU microarchitecture.

5.2.1. Branch divergence

Branch divergence is a major cause to the performance degradation in GPGPUs. As we discussed earlier, the immediate post-dominator (PDOM) lacks the capability to reconverge threads at the beginning for branch divergence to further improve the performance. Dynamic warp formation (DWF) is proposed in [12] to efficiently handle the threads divergence. It groups threads from multiple warps but branching to the same target into a new and complete warp, and issue it into the SIMD pipeline. Therefore, the parallel streaming processors in the SM are fully utilized and the performance is enhanced.

In our baseline GPGPU machine, a warp splits into multiple warps in the occurrence of the branch divergence; those spawned warps have to consume the issue bandwidth to leave the warp scheduler. They delay the issue time of other warps and increase the warp residency time. DWF re-groups the divergent threads into new warps. It reduces the number of warps competing for the issue bandwidth and shrinks the warp waiting time. Meanwhile, DWF targets to improve the performance and reduce the kernel execution time. Note that reducing the warp waiting time and correspondingly the ACE instruction residency time can help to decrease AVF, but shrinking the execution time would rather increase AVF as it increases the ACE instruction quantity per cycle [16]. Therefore, the impact of DWF on warp scheduler’s vulnerability can be negative or positive, depending on the warp residency time and the kernel execution time. When the reduction of warp residency time dominates that of kernel execution time, the warp scheduler AVF decreases, and vice versa. Since the warps enter into the SIMD pipeline earlier, registers are written/read faster which helps to reduce their lifetime. Similar to the warp scheduler, the DWF could increase/decrease registers AVF.

Fig. 6 presents the structures’ AVF under the impact of DWF, they are normalized to the baseline case with PDOM applied for comparison. Benchmarks KM, LIB, MRIF, NN, and ST3D cannot execute when DWF is enabled, we eliminate these benchmarks from this figure. The result of L1 caches and shared memory are not shown in Fig. 6 and the following vulnerability figures since they are always robust to the soft error strikes under different architectural optimizations. On average, DWF optimizes the warp scheduler, and registers reliability by 5% and 5%, respectively. Obviously, most benchmarks studied here fall into the category where DWF introduces the positive effect on those two structure reliability. Note that the capability of
DWF on optimizing structure reliability is significantly affected by the percentage of branch divergence operations in the benchmark. When there are few branch divergences, DWF has limited opportunities to explore the warp formation, whose effect on reliability is trivial. For example, the warp scheduler AVF only decreases 1% in CP with 2% branch divergence operations. On the other hand, DWF helps to improve warps scheduler reliability by 60% in BFS with 40% branch divergence operations.

The streaming processors’ AVF is highly related to the kernel execution time. Since the instruction computation time is mainly determined by the instruction type, a shorter kernel execution time infers the increased quantity of ACE instructions per cycle in the streaming processors. And the streaming processors are more susceptible to soft errors when the execution time shrinks. Intuitively, DWF would hurt the streaming processor’ soft-error vulnerability because it targets on optimizing performance and reduces the kernel execution time. Interestingly, DWF decreases streaming processors’ AVF by 10% on average as shown in Fig. 6. This is because DWF decreases the IPC in a number of benchmarks (as shown in Fig. 7). When randomly re-grouping threads into new warps, DWF may lose the opportunities to combine memory accesses originally from the same warp, and introduce extra off-chip memory transactions, which negatively affect the performance. When the benefit of DWF in branch divergences is outweighed by the increased memory access requests, the performance starts to degrade.

5.2.2. Off-chip memory access

In the GPGPU applications, all threads in a kernel execute the same code, and exhibit similar execution progress in the fine-grained multi-threading environment. When one thread encounters an off-chip memory access, other threads are likely to issue the requests at approximately the same time, leading to severe resource contentions on both on-chip-network and the memory controllers. The effect of interconnect topologies and request scheduling policies in the memory controller on GPU throughput has been studied in [13]. However, their impact on GPU microarchitecture soft-error vulnerability is largely ignored.

Fig. 8(a) shows the vulnerability of SM structures when applying the FIFO scheduling policy in memory controller, and two interconnect network topologies (i.e. Torus, Fly), respectively. Results are normalized to our baseline configuration with mesh topology and First-Ready First-Come First-Serve (FR-FCFS) scheduling policy. As it shows, even though the performance is hurt considerably while using the Torus network topology (Fig. 8(b) also presents the normalized IPC for the vulnerability analysis), in general, the interconnect network topology has little impact on the vulnerability of warp scheduler and register files. For instance, the two structures’ AVF is similar to the baseline case while the IPC decreases 23% in NE under the Torus topology. The reduced IPC implies fewer ACE instructions per cycle in the warp scheduler, however, the instructions spend longer time in the structure and its vulnerability remains almost the same as the baseline case. The similar case occurs in the register files. On the other hand, the streaming processors’ AVF varies noticeably in NE under the Torus topology, because it exhibits strong correlation with the GPU performance.

The FIFO scheduling policy simply serves the memory requests in the order of their arrival time in the memory controller. It causes the frequent switch among DRAM rows which increases the request service time and the memory access latency. FR-FCFS gives the memory requests that hit the open DRAM row higher priority to access the DRAM. Therefore, the IPC decreases 22% under FIFO scheduling policy compared to the baseline case that uses FR-FCFS. Similar to the case with different network topologies, the warp scheduler and registers vulnerability changes slightly under FIFO compared to the baseline case, while the impact of FIFO policy on SM vulnerability is considerable as shown in Fig. 8(a).

5.2.3. Number of threads per streaming multiprocessors (SM)

In previous works, the effect of altering the number of simultaneously running threads on performance has been well explored [13], but its impact on GPGPU microarchitecture soft-error vulnerability is still unknown. In order to change the number of parallel threads per SM, we scale the per-block resources (the amount of maximum blocks, shared memory, and registers) between 50% and 200% to the baseline case which supports 1024 threads.

Fig. 9 shows the AVF and IPC results with different amounts of threads per SM. It is normalized to the baseline case. When the per-block resources shrink to 50%, the kernels in FWT, KM, and STO fail to launch into the GPGPU processor due to the insufficient resources for even one block. Therefore, the results for FWT, KM, and STO are missing for 50% reduced thread count.
Fig. 8. The normalized (a) AVF and (b) IPC of GPGPU microarchitecture structures when applying the FIFO scheduling policy in the memory controller, the Fly, and the Torus network topology, respectively.

(512) in Fig. 9. As Fig. 9(a) and (b) show, both AVF and IPC vary significantly across the benchmarks with the increasing number of threads per SM. We characterize the benchmarks based on their reliability behaviors as follows.

In CP, LIB, and NW, the 50% thread configuration is already sufficient for all the blocks to execute simultaneously, additional thread slots and resources are not used and do not contribute to the performance. Therefore, the performance of these benchmarks remains the same as shown in Fig. 9(b). In that case, the size of warp scheduler and registers scales up with the increasing thread count, but their utilization reduces due to the increasing idle entries. Therefore, as Fig. 9(a) demonstrates, the AVF of warp scheduler and registers decreases as the number of threads increases. On the other hand, the streaming processors’ size and utilization do not change with thread count; hence there is a little change in the streaming processors’ AVF.

Similarly, LPS and SRAD are unable to leverage the additional resources beyond the baseline configuration; they show the same behavior as CP from the 100% to 200% configurations. However, in the 50% configuration, there are insufficient resources to hold blocks and feed the pipeline. Streaming processors are highly under-utilized, performance and the streaming processors’ AVF both decrease. The performance of these two benchmarks are the same from 100% to 200% configuration and much higher than 50% configuration as shown in Fig. 9(b). For FWT, SLA, SP and ST3D, the performance reaches the peak for the 150% configuration; they show the same behaviors as CP from the 150% to 200% configurations. NN is able to utilize the increased resources, so the performance is improved as the configuration increases from 50% to 200%. Correspondingly, the streaming processors’ AVF increases. Moreover, the vulnerability of warp scheduler and register files decreases slightly, because their usage decreases to some degree with regard to the continuously increased structure size from 50% to 200%.

In BFS, BP, BS, and NE, there are enough blocks for the simultaneous execution with the increasing number of threads supported by the SM. However, the performance of these benchmarks remains nearly the same from 50% to 200% configurations (shown in Fig. 9(b)). The contents on the interconnect network and memory resources among threads become server as the number of concurrent threads increases in the SM, and instructions spend longer time in the pipeline structures, which induces negative effect to the GPU performance. As a consequence, the structure utilization is similar across the different configurations, and the configuration variation has negligible impact on the SM structure vulnerability in those benchmarks except BFS as shown in Fig. 9(a). In BFS, majority SMs are idle while only few still execute blocks at the end of the kernel execution time. As the configuration changes from 50% to 200%, there are more free resources in SMs, and it helps to decrease the structures’ AVF. Moreover, KM exhibits similar behavior as BP except the performance and vulnerability decease in 150% configuration compared to the baseline case due to the serious workload imbalance among SMs.

As Fig. 9(b) shows, the IPC increases from 50% to 150% configurations in 64H and HS, it infers that the increasing resources are able to support more concurrent threads and improve the throughput. Take 64H as an example, the 50% configuration (especially, the 50% shared memory) significantly limits the number of blocks executed in the SM, warp scheduler and streaming processors are under-utilized and exhibit extremely low soft-error vulnerability. However, their performance degrades at 200% configuration because of the severer resource contentions among threads.

In STO, the IPC remains the same as the number of threads increases from 100% to 150%. Because the 150% configuration introduces uneven block distribution among SMs, several SMs are heavily loaded which determines the total kernel execu-
Streaming Processor
Register Files

of threads achieves up to 200%, meanwhile, the structure vulnerability increases as well. Because the SMs with 200% con-

Fig. 9. The normalized (a) AVF and (b) IPC of GPGPU microarchitecture structures when the number of threads per SM is 50% (512 threads), 150% (1536 threads) and 200% (2048 threads) to the baseline case.

Fig. 10. The normalized (a) AVF and (b) IPC of GPGPU microarchitecture structures under the impact of FRFS, Fair, Random, and Two-level warp scheduling policies, respectively.

tion time. Moreover, the vulnerability of warp scheduler and register files decreases in 150% configuration as their overall utilization (considering the increased structure size) across SMs decreases. The performance further improves as the number of threads achieves up to 200%, meanwhile, the structure vulnerability increases as well. Because the SMs with 200% con-
figuration have balanced workloads and the execution time is greatly reduced. The impact of the workload imbalance also exists in LV and MRIF. As Fig. 9(b) shows, the IPC is low in the baseline case (i.e. 100% configuration) compared with other configurations. This is because the 100% configuration results in the serious workload imbalance among SMs.

In MT, both the performance and structure vulnerability increases smoothly as the number of threads increases from 50% to 200%. This is because the utilization of each structure increases as the number of threads increases in each SM, it improve the throughput but meanwhile hurt the SM soft-error robustness.

In summary, on average across the benchmarks, increasing the quantity of threads largely affects the GPGPU microarchitecture soft-error robustness. It improves the warp scheduler and registers soft-error robustness, but meanwhile, it degrades the streaming processors reliability.

5.2.4. Warp scheduling

The warp scheduling policy largely affects the GPGPU throughput. In previous work, Lakshminarayana et al. [25] explore several policies and evaluate their impact on performance. In this paper, we analyze their effectiveness from the reliability perspective. We investigate four warp scheduling policies: First-Ready First-Serve (FRFS), Fair (issuing a warp with the minimum instructions executed), Random, and two-level round robin scheduling (Two-level) that distributes warps into several fetch groups, and applies round-robin policy at the group level and the warp level in each group to select appropriate warp for issuing [30]. They are compared against the baseline case of round-robin scheduling. Fig. 10 presents the normalized AVF and IPC under the impact of various policies.

As Fig. 10 shows, on an average across the benchmarks, the structure AVF varies less than 8% under different scheduling policies when compared to the baseline case, and the IPC variation is even less than 5%. The structure vulnerability under the FRFS and Random scheduling policies is similar to the baseline case, but the vulnerability variation is more noticeable in several benchmarks (e.g. LPS, LV, MRIF, ST3D, and so on) when Fair and two-level polices are enabled. The Fair policy keeps the uniform progress among warps. It may increase the opportunities for inter-warp memory coalescing when threads across warps have similar memory access patterns, and reduce the warp waiting time. However, when it fails to merge the inter-warp memory accesses, a large number of memory requests are generated in a very short time span, which results in heavy memory resource contention and extends the warp stall time. Hence, the warp scheduler’s AVF increases when the total execution time has minor increase, such as MT shown in Fig. 10(a). When a significant amount of off-chip memory transactions fail to get coalesced, and the workload execution time severely extends, the significantly reduced ACE instructions quantity per cycle dominates the increased instruction residency time in the pipeline structure, as a result, the structure AVF decreases considerably (e.g. SLA, SP, and ST3D). Two-level scheduling policy targets to hide the long latency memory accesses and improve the performance, therefore, the number of ACE instructions per cycle increases and structures become more susceptible to soft errors (e.g. LV and MRIF).

6. Related work

Various performance and power analytical tools have been developed for GPGPU processors. Baghsorkhi et al. [27] have applied the work flow graph to analyze kernels and predict the GPU performance. Hong et al. [28] develop an analytical tool to statically estimate the program execution time and further explore an integrated power and performance model [11] for GPU processors. Zhang et al. [29] explore a microbenchmark-based performance model to quantitatively analyze performance. However, there is little work done on modeling the GPGPU reliability. In this work, we explore GPGPU-SODA to estimate the GPGPU microarchitecture vulnerability in the presence of soft errors. Nathan et al. [26] develop Argus-G, a low cost error-detection scheme for GPGPUs. It mainly targets on error detection, which is orthogonal to our GPGPU-SODA framework.

There have been several studies on characterizing the GPU workloads on performance domain. Che et al. [20] characterize the diversity of Rodinia benchmarks. Kerr et al. [23] propose a set of metrics for GPU workloads and use them to analyze the behavior of GPU programs. Goswami et al. [24] explore a set of GPGPU workload characteristics to accurately capture workload behavior and use a wide range of metrics to evaluate the effectiveness of the characterization. However, those studies mainly focus on performance domain and largely ignore the reliability factor. We analyze the GPGPU soft-error vulnerability by using GPGPU-SODA, and observe that structure vulnerability is highly sensitive to workload characteristics such as branch divergences, off-chip memory accesses and so on. In addition, both [13] and [25] investigate the effect of various GPGPU architecture optimizations on GPGPU throughput and performance, while we evaluate their effectiveness on reliability. We further characterize the studied benchmarks based on their reliability behaviors under the impact of different GPU designs.

7. Conclusions

With their strong computing power and improved programmability, GPU has emerged as highly-efficient devices for a wide range of parallel applications. Modern GPU architectures lack thorough capability for error detection and tolerance since they are mainly designed for graphic processing. However, the rigorous execution correctness is required in GPUs for general-purpose computation. As the CMOS processing technology keeps on scaling down at nano-scale, the soft error
rate is predicted to increase exponentially, therefore, GPGPU with hundreds of cores integrated in a single chip are highly vulnerable to the soft error strikes.

In this paper, we first develop GPGPU-SODA to evaluate the GPGPU microarchitecture soft-error vulnerability. As our GPGPU-SODA suggests, majority structures in GPGPUs (e.g. warp scheduler, streaming processors, and register files) are highly vulnerable, a comprehensive technique is needed to protect them against the soft error attacks. We observe that the structure vulnerability is largely affected by the workload characteristics. For example, both branch divergences and long latency memory accesses decreases the streaming processors’ vulnerability. Moreover, SMS in a single GPGPU processor may manifest significantly different reliability characteristics. We further analyze the impact of several architecture optimizations on GPGPU microarchitecture vulnerability. For example, increasing the thread quantity per SM improve the warp scheduler and register files reliability, but degrade the streaming processors soft-error robustness. We also have found that the structure vulnerability is insensitive to the warp scheduling policies. Our observations provide designers the useful guidance in building the resilient GPGPU processors: a GPGPU reliability-optimization technique should consider all the GPGPU microarchitecture structures. A technique focusing on one particular structure would not be an efficient resilient solution to GPGPUs.

Acknowledgement

This work is supported by the National Science Foundation under Award No. EPS-0903806 and matching support from the State of Kansas through the Kansas Board of Regents. This research is also supported by PSC-CUNY Award 43.

References