Analyzing Soft-Error Vulnerability on GPGPU Microarchitecture

Jingweijia Tan
EECS Department, University of Kansas
jtan@ittc.ku.edu

Nilanjan Goswami
ECE Department, University of Florida
nil@ufl.edu

Tao Li
ECE Department, University of Florida
taoli@ece.ufl.edu

Xin Fu
EECS Department, University of Kansas
xinfu@ittc.ku.edu

***Abstract*—The general-purpose computation on graphic processing units (GPGPU) becomes increasingly popular due to their high computational throughput for data parallel applications. Modern GPU architectures have limited capability for error detection and tolerance since they are originally designed for graphics processing. However, the rigorous execution correctness is required for general-purpose applications. This makes reliability a growing concern in 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 in a single chip are prone to manifest high SER. This paper explores a first step to 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 workload characteristics (e.g. branch divergences, memory coalescing). We further investigate several architectural optimizations. We find that both dynamic warp formation and increasing the number of threads supported by GPU largely affect the GPGPU soft-error robustness. However, changing the warp scheduling policy has minor 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 just focusing on a particular structure.**

I. Introduction

The performance of microprocessors has 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, NVIDA'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 100Gflop/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 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 called 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] have developed AerialVision to provide insights into the performance bottleneck in GPGPU architectures. Hong et.al. [11] have proposed an integrated power and performance prediction model for GPGPUs. However, there is little work done 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, increasing number of threads per streaming multiprocessor, various warp scheduling policies) [12, 13, 25]. But their impact on GPGPU soft-error vulnerability is 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 robust GPGPUs.

This study is the first step to perform the soft-error vulnerability analysis on GPGPU architectures. To this end, we

978-1-4577-2064-2/11/$26.00 ©2011 IEEE

develop a reliability-aware GPGPU processor simulator, it 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. To our best knowledge, this is the first work on analyzing the GPGPU reliability characteristics in the light of small-scale processing technology.

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 coalescing 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. We find that both dynamic warp formation (a technique to handle branch divergence) and increasing the thread quantity per SM are able to improve the warp scheduler and registers reliability, but meanwhile 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.

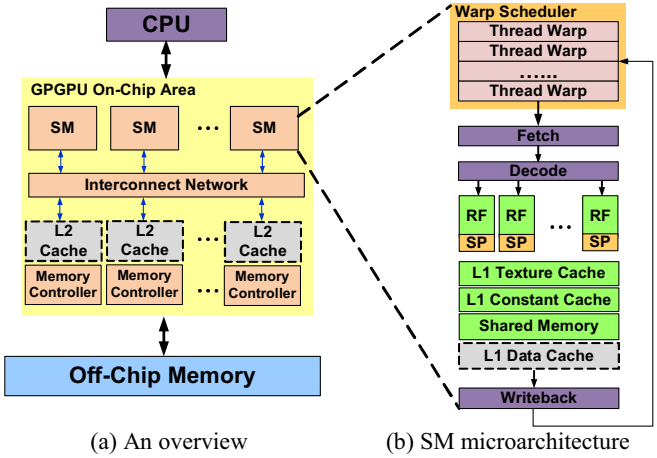
The rest of this paper is organized as follows: Section II provides background on state-of-the-art GPGPU architecture, and soft errors. Section III presents our developed GPGPU-SODA framework. Section IV describes our experimental methodologies. Section V analyzes the GPGPU soft-error vulnerability. We discuss the related work in Section VI, and conclude the paper in Section VII.

II. BACKGROUND

A. General-Purpose Computing on Graphic Processing Units (GPGPU) Architecture

Figure 1(a) shows an overview of the state-of-the-art GPU Architecture [3]. It consists of scalable number of in-order streaming multiprocessors (SM) that can access multiple memory controllers via an on-chip interconnection network. Figure 1(b) illustrates a zoom-in view of the SM. It contains the warp scheduler, register files (RF), streaming processors (SP), constant cache, texture cache, and shared memory. In addition to 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 Figure 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 for 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.



(a) An overview (b) SM microarchitecture

Figure 1. General-Purpose Computing on Graphic Processing Units (GPGPU) architecture

Threads in the SM execute in the 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. Figure 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 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.

B. Microarchitecture Level Soft Error Vulnerability Analysis

A key observation of soft error behavior at microarchitecture level is that a SEU may not affect processor states required for correct program execution. At 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 study, we use AVF as the major metric to estimate structure soft-error vulnerability.

III. 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 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 input/output which requires a more complicated analysis considering all the 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. Hence 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 input/output 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], it 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 into 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 AVF for address-based structure, GPGPU-SODA summarizes the ACE components [18] of each bit's lifetime in the structure. In order to calculate 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. Because it ignores the data dependencies across threads caused by the inter-thread data sharing, an ACE instruction may be incorrectly identified as un-ACE. 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 V, 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.

IV. EXPERIMENTAL METHODOLOGY

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

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), *BN* (binomial options), *BP* (back propagation), *FWT* (fast walsh transform), *HS* (hot spot), *HY* (hybrid sort), *LPS* (3D laplace solver), *MM* (matrix multiplication), *MRIF* (Magnetic resonance imaging FHD), *MT* (matrix transpose), *PNS* (petri net simulation), *SLA* (scan of large arrays), *SP* (scalar product), *SRAD* (Speckle Reducing Anisotropic Diffusion), *SS* (similarity score), *ST3D* (stencil 3D). The workloads show significant diversity 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.

TABLE I
HARDWARE CONFIGURATION

Number of SMs	28
SIMD pipeline width	8
Warp size	32
Number of threads/SM	1024
Maximum blocks/SM	8
Number of registers/SM	16384
Shared memory/SM	16KB
Constant cache /SM	8KB
Texture cache/SM	64KB
Branch divergence method	Immediate post-dominator
Warp scheduling policy	Round robin among ready warps
DRAM controller	8
DRAM queue size	32
Bus width	8 bytes/cycle
Interconnect topology	Mesh
Routing algorithm	Dimension order
Virtual channels	2
VC buffer size	4
Flit size	16B

We use AVF as the basic metrics to estimate how susceptible a GPGPU microarchitecture structure is to soft error attacks. Only focusing on the reliability domain will mislead the evaluation on a GPGPU architecture design. In this study, we further use the Mean Instructions to Failure (MITF) metric to describe the trade-off between performance and reliability [8]. MITF represents the amount of work a processor can complete, on average, between two failures. At a fixed frequency and raw error rate, MITF is proportional to the ratio of IPC to AVF (*IPC/AVF*). A higher MITF, and correspondingly, a higher *IPC/AVF* implies a greater amount of work accomplished between errors, which is desirable.

V. SOFT-ERROR VULNERABILITY ANALYSIS ON GPGPUS

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

A. Soft-Error Vulnerability Profile of the Baseline GPGPU

A.1. Instruction Level Vulnerability Profile

Figure 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 into 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 68% ACE, 29% un-ACE, and 3% vulnerability-unknown instructions across the benchmarks. Several benchmarks have considerable amounts of un-ACE instructions, for instance, 60% instructions are un-ACE in *BN*. Generally, threads running in GPGPUs are mainly composed of loops, un-ACE instructions appear in the loop will be repeatedly executed and largely contribute to the total number of un-ACE instructions in the workload. Our GPGPU-SODA framework reports 18% FDD and 11% 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, they are usually inserted for pipeline flushing, or instruction alignment in VLIW processors [16, 17, 22].

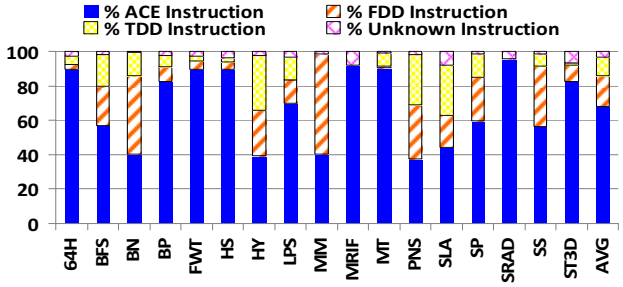


Figure 2. The percentage of ACE, FDD, TDD and unknown instructions

A.2. Soft-Error Vulnerability of GPGPU Microarchitecture Structures

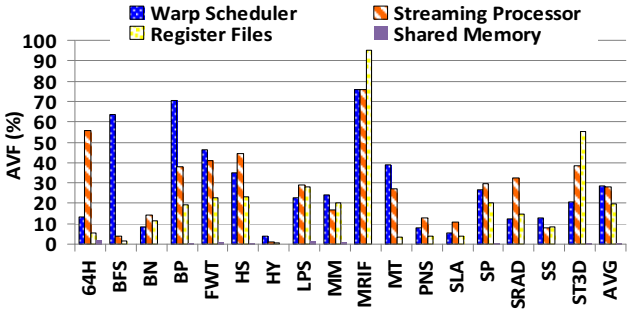


Figure 3. The AVF of GPGPU SM microarchitecture structures

Figure 3 shows the soft-error vulnerability of several key structures in GPGPU streaming multiprocessors 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 Figure 3. Because the workloads we studied either do not or rarely use those two structures and their AVF is lower than 4%. On the contrary, several 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 76%, 95%, and 75%, respectively.

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

As shown in Figure 3, the structure vulnerability varies dramatically across the workloads. For instance, the warp scheduler's AVF is 4% in *HY* but 76% in *MRIF*. In general, the warp scheduler and registers show comparatively low AVF in workloads containing a large number of un-ACE instructions that are immune to the soft error strikes, such as *BN*, *HY*, *PNS*, and *SLA*. However, the AVF of those two structures is not always high even in programs with a significant amount of ACE instructions. For example, the AVF of warp scheduler is 20% in *ST3D,* although 83% of the instructions are ACE. As

introduced in Section II.A, 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, they 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 determining 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.

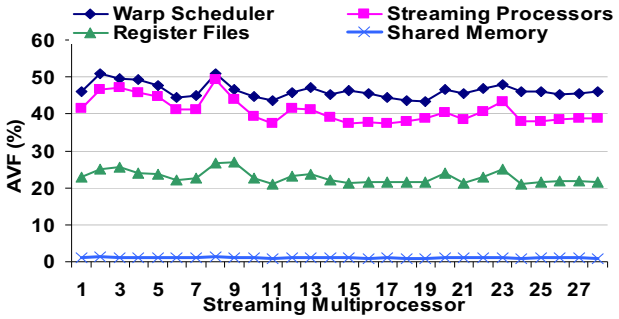
In the SM, shared memory is mainly designed for inter-thread communication; it is not used in kernels without thread communication/synchronization. Therefore, as Figure 3 shows, the AVF of shared memory is zero in several benchmarks (e.g. *BFS*, *MRIF*). To improve the performance, program developers use the shared memory as software managed on-chip cache for the global memory and reduces 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 low which is less than 2% as shown in Figure 3. 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. Table II presents the percentage of used entries in the shared memory for each benchmark. On average, it is only 33%. In workloads whose block resource allocation is limited by the share memory (e.g. *64H, LPS, SP, SRAD*), more than 50% entries are never be 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.

TABLE II
SHARED MEMORY UTILIZATION IN PERCENTAGE

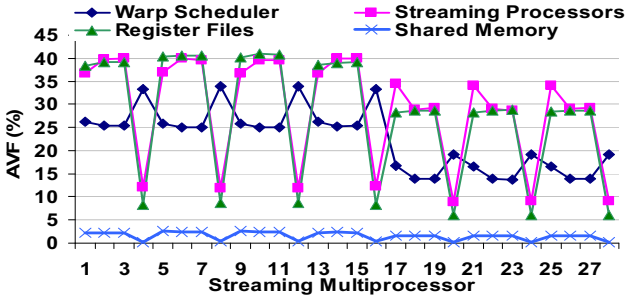
64H	35%	***BFS***	0%	***BN***	14%	***BP***	26%	***FWT***	99%	***HS***	26%
HY	75%	***LPS***	40%	***MM***	49%	***MRIF***	0%	***MT***	25%	***PNS***	11%
SLA	6%	***SP***	50%	***SRAD***	37%	***SS***	37%	***ST3D***	25%	***AVG***	33%

As shown in Figure 3, in most benchmarks, streaming processors' AVF has strong correlation to the number of ACE instruction. Because the instruction execution time in streaming processors is constant, and the ACE instruction quantity per cycle becomes the major factor to determine the AVF. However, streaming processors' AVF is only 4% in *BFS* even 60% instructions are ACE. *BFS* has heavy branch divergences. Since diverged threads within the warp have to execute sequentially, it causes tremendous performance loss, and the parallel streaming processors have very low utilization. In our baseline machine configuration, the immediate post-dominator based reconvergence [14] 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. The parallelized streaming processors are still highly under-utilized in *BFS*, and the AVF is low.

(2) Vulnerability Variations at Streaming Multiprocessor Level



(a) *FWT*



(b) *LPS*

Figure 4. The AVF of microarchitecture structures in each SM while running (a) *FWT* (b) *LPS*

A GPGPU processor contains tens of SMs. Generally, they are equally utilized and exhibit similar vulnerability behavior. Figure 4 (a) shows the AVF of structures in each SM when executing *FWT*. As can be seen the figure, there is small AVF divergence among the SMs. However, this is not the case for some benchmarks. Figure 4 (b) shows an example of *LPS*. SM 4, 8, 12, 16, 20, 24, and 28 exhibit opposite vulnerability behavior compared to other SMs. Take SM 3 and 4 for examples, the warp scheduler's vulnerability is low in SM 3 but high in SM 4. In contrast, the streaming processors, registers and shared memory have much higher vulnerability in SM 3 than SM 4. This happens due to the threads running in each SM have different memory access patterns. It also determines the quantity of memory requests that can get coalesced. In this example, 86% (59%) of memory transactions are merged in SM 3 (SM 4). The un-coalesced off-chip memory accesses have to complete sequentially; it dramatically increases the warp waiting time and ACE instruction residency time in the warp scheduler. Therefore, the scheduler's reliability degrades in SM 4 with many un-coalesced memory transactions. Meanwhile, there are less ready warps to feed into the SIMD pipeline, and structures in the pipeline (e.g. streaming processors, registers) have fewer ACE instructions per cycle. They become more robust to the soft error strikes. On the contrary, when most memory requests

get coalesced in SM 3, warp stall time reduces so that the warp scheduler vulnerability decreases. The pipeline structures are filled by instructions and suffer high vulnerability. In addition, the blocks in *LPS* kernels cannot be evenly distributed to the SMs since the total number of blocks is not an integer multiple of the number of SMs. So, there are fewer blocks running in SM 17 ~ 28 and they are more reliable compared to SM 1 ~ 16 (as shown in Figure 4 (b)). Note that different amounts of branch divergences in SMs could also contribute to the SM vulnerability variation.

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 un-coalesced memory accesses, the barrier instructions could increase the warp scheduler's vulnerability. 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, and the memory coalescing. Especially, the branch divergence reduces the streaming processors AVF. The memory coalescing decreases warp scheduler's AVF, but increases the registers and streaming processors AVF.

A.3. Trade-off between IPC and AVF

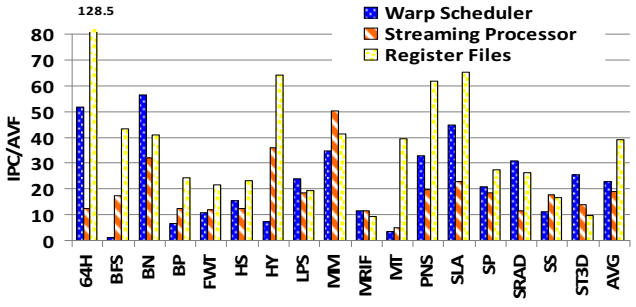


Figure 5. The IPC/AVF of GPGPU SM microarchitecture structures

Figure 5 demonstrates the IPC/AVF statistics of warp scheduler, streaming processors and registers for the studied benchmarks. It describes the trade-off between reliability and performance. Note that the shared memory and L1 caches are infrequently used and exhibit quite low AVF, they are not presented in the Figure. As Figure 5 shows, *BFS*, *MT*, and *ST3D* show the lowest IPC/AVF efficiency for warp scheduler, streaming processors, and registers, respectively. *BN*, *MM*, and *64H* show the highest IPC/AVF efficiency for those three structures respectively. At the SM level, the IPC/AVF is high in *BN* and *MM*. Because they contain a large portion of un-ACE instructions leading to the low structure vulnerability, meanwhile, they gain the high performance by efficiently leveraging the SIMD pipeline.

There is no strong correlation between structure IPC/AVF and AVF statistics. As shown in Figure 3, all the three structures suffer extremely high vulnerability in *MRIF*, but their IPC/AVF efficiency is still better than some benchmarks. Especially, the warp scheduler's IPC/AVF is higher in *MRIF* than that in *BFS*, *BP*, *FWT*, *HY* and *MT*. In *MRIF*, the parallel threads are well organized to efficiently use the resources and achieve the high performance to enhance the trade-off between IPC and AVF. In contrast, structures in *HY* and *SS* obtain low IPC/AVF even though they exhibit strong soft-error robustness. The performance is seriously degraded in both benchmarks. In *HY*, the number of threads simultaneously running in the SM is limited by the shared memory. In *SS*, several kernels have very few threads in a block, since the SM can only launch up to 8 blocks, there are inadequate parallel threads to execute. Both the two benchmarks lack the capability to take advantage of the parallel units to obtain high performance. And the low throughput hurts the IPC/AVF substantially.

B. 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.

B.1. Dynamic Warp Formation (DWF)

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 streaming multiprocessors 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 improves the performance and reduces 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 and shared memory are written/read faster which helps to reduce their lifetime. Similar to the warp scheduler, the DWF could increase/decrease those two structures' AVF. However, it is much more certain that DWF increases the streaming processors' AVF. This is mainly

caused by the increased utilization and the increased quantity of ACE instructions per cycle in streaming processors when DWF is enabled.

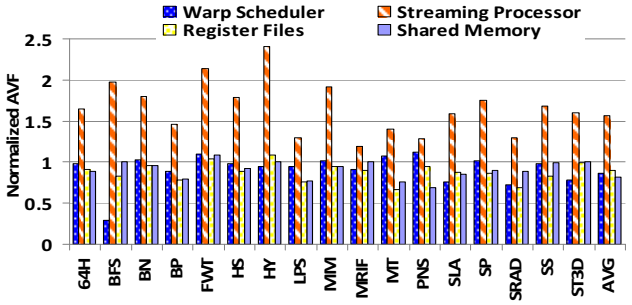


Figure 6. The normalized AVF of GPGPU streaming multiprocessor microarchitecture structures under the impact of dynamic warp formation.

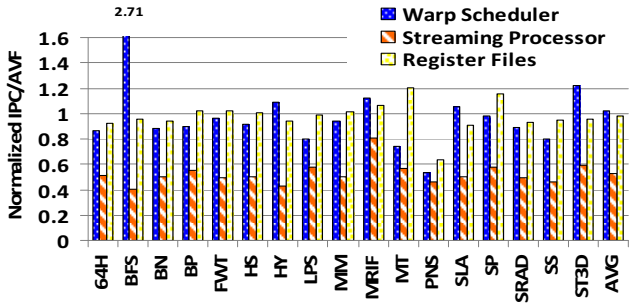


Figure 7. The normalized IPC/AVF of GPGPU streaming multiprocessor microarchitecture structures under the impact of dynamic warp formation.

Figure 6 presents the structures' AVF under the impact of DWF, they are normalized to the baseline case with PDOM applied for comparison. On average, DWF optimizes the warp scheduler, registers, and shared memory reliability by 14%, 11%, and 18%, respectively. Obviously, most benchmarks studied here fall into the category where DWF introduces the positive effect on structure reliability. On the other hand, as shown in Figure 6, DWF increases streaming processors' AVF by 55%. Note that the capability of DWF on optimizing structure reliability is largely 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, its effect on reliability is trivial. For example, the warp scheduler AVF only decreases 9% in *MRIF* with 2% branch divergence operations. On the other hand, DWF helps to improve warps scheduler reliability by 71% in *BFS* with 13% branch divergence operations.

Figure 7 presents the normalized IPC/AVF of the three major microarchitecture structures when DWF is applied. Intuitively, DWF should improve the IPC/AVF of warp scheduler and registers since it targets on optimizing performance and reduces their vulnerability as shown in Figure 6. Interestingly, their IPC/AVF efficiency under DWF is similar to the baseline case. During the random thread re-grouping 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. In most benchmarks, DWF decreases the IPC around 6%, so that it introduces 2% improvement on warp scheduler's IPC/AVF and 3% reduction on registers' IPC/AVF. In *BFS*, the warp scheduler's AVF decreases substantially so that DWF achieves 171% IPC/AVF improvement. In contrast, the IPC/AVF of warp scheduler drops dramatically in *PNS*. Because the percentage of coalesced memory transactions reduces from 86% to 41%, it introduces great performance loss. The benefit of DWF on reducing the quantity of warps and shrinking warp residency time becomes trivial; the warp residency time is extended by the long latency memory accesses instead. As a result, the warp scheduler's AVF even increases 12% in *PNS*. With the decreasing IPC and increasing AVF, *PNS* performs poorly when considering the trade-off between reliability and performance. Finally, as Figure 7 shows, the IPC/AVF of streaming processors is around 57% lower than the baseline case. Since the streaming processors' AVF already increases 55% under the impact of DWF, its IPC/AVF hurts considerably when taking the performance loss into account.

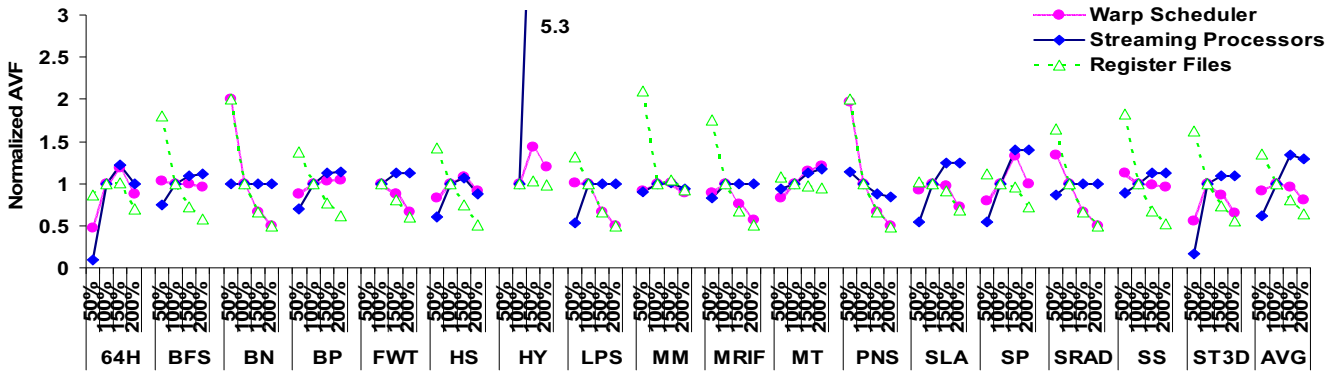
B.2. Number of Threads per Streaming Multiprocessors (SM)

In previous works, the effect of varying 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. Figure 8 shows the AVF and IPC/AVF results with different amounts of threads per SM. It is normalized to the baseline case. The results of L1 caches and shared memory are not shown since they always exhibit strong capability in fault tolerance across various configurations. When the per-block resources shrink to 50%, the kernels in *HY* and *FWT* fail to launch into the GPGPU processor due to the in-sufficient resources for even one block. Therefore, the results for *HY* and *FWT* are missing for 50% reduced thread count (512) in Figure 8. As Figure 8 (a) and (b) show, both AVF and IPC/AVF vary significantly across the benchmarks with the increasing number of threads per SM. We characterize the benchmarks based on their reliability behaviors as follows.

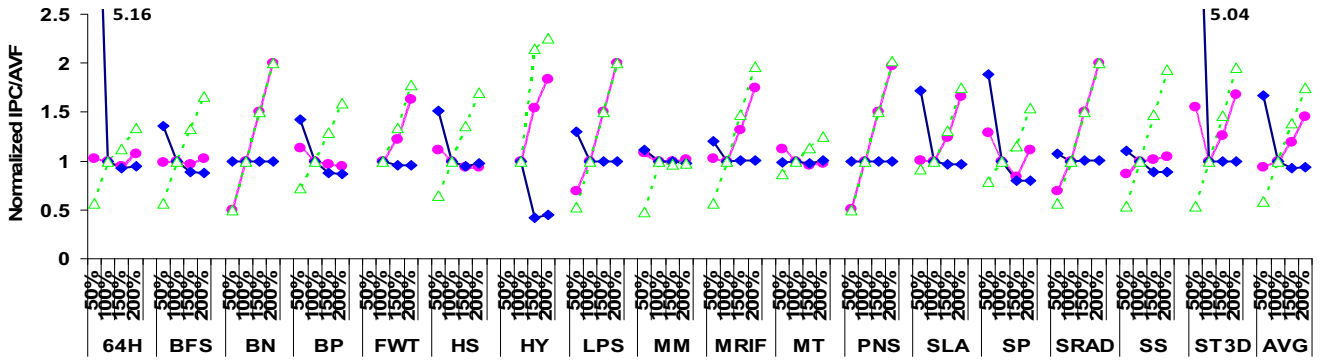
In *BN* and *PNS*, the 50% thread configuration is already sufficient for all the blocks to execute simultaneously, additional thread slots and resources are not used and do not contribute to performance. In that case, the size of warp scheduler and registers scales up with the increasing thread count, but their utilization reduces due to the increasing idle entries. Therefore, as Figure 8 (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 little change in the streaming processors' AVF. When considering both reliability and performance factors, the warp scheduler's and registers' IPC/AVF increases linearly with the number of threads, while streaming processors' IPC/AVF is slightly affected by the various configurations (as shown in Figure 8(b)).

Similarly, *LPS*, *SRAD*, and *MRIF* are unable to leverage the

additional resources beyond the baseline configuration; they show the same behavior as *BN* from the 100% to 200% configurations. However, in the 50% configuration, there are in-sufficient resources to hold blocks and feed the pipeline. Streaming processors are highly under-utilized, performance and the streaming processors' AVF both decrease. Since the reduction of AVF outweighs performance, the streaming processors' IPC/AVF in *LPS*, *SRAD*, and *MRIF* is higher than the baseline according to Figure 8(b). For *FWT*, *SLA*, *SP* and *ST3D*, the performance reaches the peak for the 150% configuration; they show the same behaviors as *BN* from the 150% to 200% configurations. In *ST3D*, the IPC/AVF of warp scheduler and streaming processors is high, but registers' IPC/AVF is low for the 50% configuration. Because the 50% registers control the number of blocks to be launched in *ST3D*. It causes under-utilized warp scheduler and streaming processors, but fully utilized registers. Hence, the AVF of both scheduler and streaming processors is low, but registers' AVF is high (as can be seen in Figure 8(a)).



(a) Normalized AVF



(b) Normalized IPC/AVF

Figure 8. The normalized (a) AVF and (b) IPC/AVF 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.

In *BFS*, *BP*, MM, and *SS*, there are enough blocks for the simultaneous execution with the increasing number of threads supported by the SM. The performance improves from 50% to 100% configuration, but starts to degrade beyond the 100% configuration due to the increased interconnect and memory contentions among threads. In that case, the warp scheduler is quickly filled up by the stalled warps and exhibits high vulnerability to soft error strikes. Moreover, *64H*, *HS* and *HY* exhibit the similar behavior except their performance peak is at 150% configuration. In *64H*, 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. For *HY*, performance jumps 3X from 100% to 150% configuration. The streaming processors are heavily used for the 150% configuration, and suffer high vulnerability.

In *MT*, the performance improves smoothly as the number of threads increases from 50% to 200%.Considering that the warp scheduler size scales up with the thread quantity, and it is fully utilized to provide enough warps for the parallel execution, one would expect a stable warp scheduler AVF across the various configurations. However, as Figure 8(a) shows, it increases when more threads are running in the SM. Because there are heavy memory/interconnect contentions with the increasing thread count, the memory access time is extended, and warps have to reside longer in the scheduler. Due to the same reason, the registers have lower utilization and their AVF reduces gradually. In addition, the streaming processors become more vulnerable because the usage increases. When taking the improved performance into consideration, the IPC/AVF of both warp scheduler and streaming processors has little change, and registers' IPC/AVF increases.

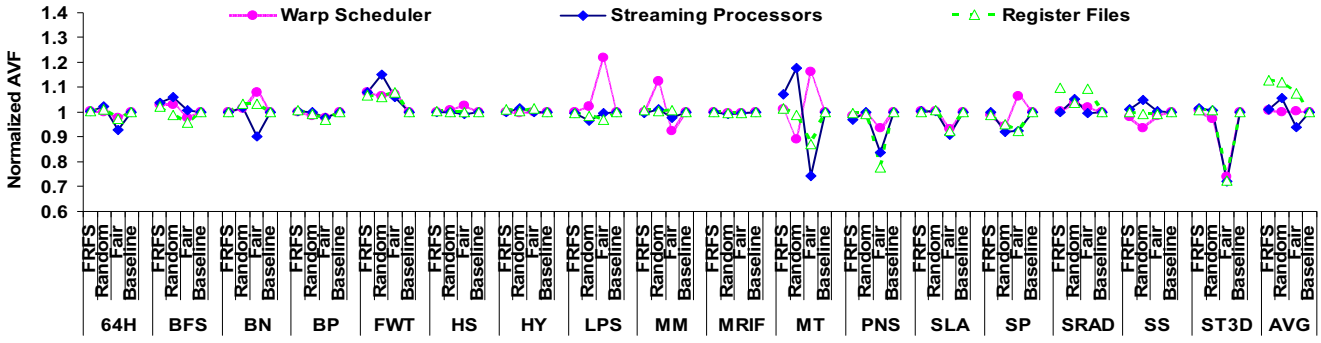
In summary, on average across the benchmarks, increasing the quantity of threads largely affects the GPGPU

microarchitecture soft-error robustness. It improves the warp scheduler and registers soft-error robustness, but meanwhile, degrades the streaming processors reliability.

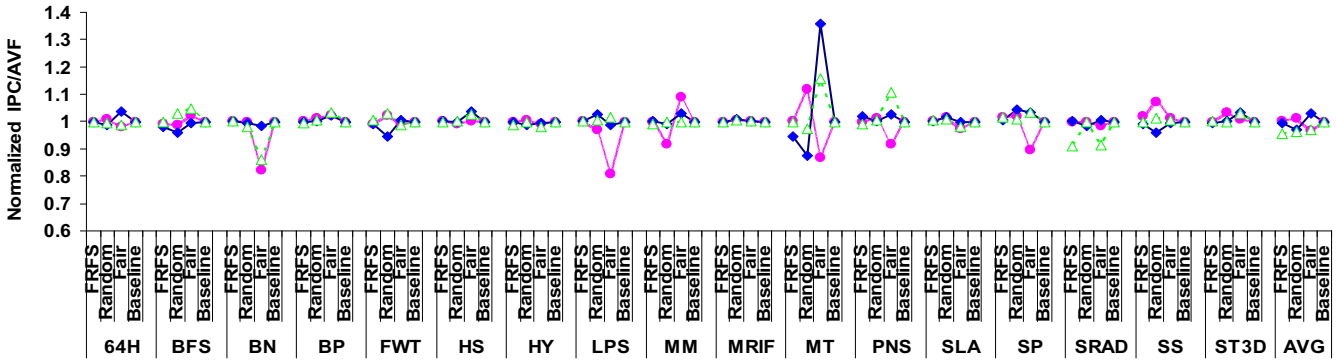
B.3. Warp Scheduling

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 three warp scheduling policies: First-Ready First-Serve (FRFS), Random, and Fair (issuing a warp with the minimum instructions executed). They are compared against the baseline case of round-robin scheduling. Figure 9 presents the normalized AVF and IPC/AVF under the impact of various policies.

As Figure 9 shows, on an average across the benchmarks, the structure AVF varies less than 10% under different scheduling policies when compared to the baseline case, and the IPC/AVF variation is even less than 5%. Most benchmarks are insensitive to the warp scheduling policies with the exception of *LPS*, *MT*, *PNS*, and *ST3D* in case of Fair scheduling policy. 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, such as *LPS* and *MT* shown in Figure 9 (a). The IPC/AVF of warp scheduler in those two benchmarks decreases due to the little change in the performance. When a significant amount of off-chip memory transactions fail to get coalesced under the impact of Fair policy, the workload execution time severely extends. As a result, both structure AVF and performance decreases considerably (e.g. *PNS* and *ST3D*).



(a) Normalized AVF



(b) Normalized IPC/AVF

Figure 9. The normalized (a) AVF and (b) IPC/AVF of GPGPU microarchitecture structures under the impact of FRFS, Random, and Fair

I. Related Work

Various performance and power analytical tools have been developed for GPGPU processors. Baghsorkhi el al. [27] have applied the work flow graph to analyze kernels and predict the GPU performance. Hong et al. [28] develop a 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 and characterizing 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 perform the soft error vulnerability analysis on GPGPU architecture 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 and the trade-off between reliability and performance. We further characterize the studied benchmarks based on their reliability behaviors under the impact of different GPU designs.

II. 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 degrade the warp scheduler reliability. 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. Both dynamic warp formation and 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 polices. 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.

REFERENCES

[1] GeForce 8800 & NVIDIA CUDA: A New Architecture for Computing on the GPU, NVIDA Corporation, 2006.

[2] ATI Mobility RadeonTM HD4850/4870 Graphics-Overview, http://ati.amd.com/products/radeonhd4800.

[3] NVIDIA CUDA™ Programming Guide Version 2.3.1, Nvidia Corporation, 2009.

[4] Advanced Micro Devices, Inc. AMD Brook+. http://ati.amd.com/technology/streamcomputing/AMD-Brookplus.pdf.

[5] Khronos. Opencl – the open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/

[6] J. Sheaffer, D. Luebke, and K. Skadron, "A Hardware Redundancy and Recovery Mechanism for Reliable Scientific Computation on Graphics Processors", In Proceedings of Graphics Hardware 2007.

[7] N. Wang and S. Patel, "ReStore: Symptom Based Soft Error Detection in Microprocessors", In Proceedings of DSN, 2005.

[8] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor", In Proceedings of ISCA, 2004.

[9] M. Dimitrov, M. Mantor, and H. Zhou, Understanding Software Approaches for GPGPU Reliability, The 2nd workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-2), 2009.

[10] A. Ariel, W.W. L. Fung, A. Turner, T. M. Aamodt, Visualizing Complex Dynamics in Many-Core Accelerator Architectures, In Proceedings of ISPASS, 2010.

[11] S. Hong, H. Kim, An Integrated GPU Power and Performance Model, In Proceedings of ISCA, 2010.

[12] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow, In Proceedings of MICRO, 2007.

[13] A. Bakhoda, G.L. Yuan, W. W. L. Fung, H. Wong, Tor M. Aamodt, Analyzing CUDA Workloads Using a Detailed GPU Simulator, In Proceedings of ISPASS, 2009.

[14] S. S.Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmanns, 1997.

[15] N. Soundararajan, A. Parashar, A. Sivasubramaniam, Mechanisms for Bounding Vulnerabilities of Processor Structures, In Proceedings of ISCA, 2007.

[16] 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, In Proceedings of MICRO, 2003.

[17] X. Fu, T. Li, and J. Fortes, Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis, Workshop on Modeling, Benchmarking and Simulation, 2006.

[18] A. Biswas, R. Cheveresan, J. Emer, S. S. Mukherjee, P. B. Racunas and R. Rangan, Computing Architectural Vulnerability Factors for Address-Based Structures, In Proceedings of ISCA, 2005.

[19] http://www.nvidia.com/object/cuda_sdks.html

[20] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing, In Proceedings of IISWC, 2009.

[21] Parboil Benchmark suite. URL: http://impact.crhc.illinois.edu/parboil.php.

[22] B. Fahs, S. Bose, M. Crum, B. Slechta, F. Spadini, T. Tung, S. J. Patel, and S. S. Lumetta, Performance Characterization of a Hardware Mechanism for Dynamic Optimization, In Proceedings of MICRO, 2001

[23] A. Kerr, G. Diamos, and S. Yalamanchilli, A Characterization and Analysis of PTX Kernels, In Proceedings of IISWC 2009.

[24] N. Goswami, R. Sankar, M. Joshi, T. Li, Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications, In Proceedings of IISWC, 2010

[25] N. B. Lakshminarayana, H. Kim, Effect of Instruction Fetch and Memory Scheduling on GPU Performance, Workshop on Language, Compiler, and Architecture Support for GPGPU, 2010.

[26] R. Nathan, D. J. Sorin, Argus-G: A Low-Cost Error Detection Scheme for GPGPUs, Workshop on Resilient Architectures (WRA), 2010

[27] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W. W. Hwu, An Adaptive Performance Modeling Tool for GPU Architectures, In Proceedings of PPoPP, 2010

[28] S. Hong, H. Kim, An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness, In Proceedings of ISCA, 2009.

[29] Y. Zhang, J. D. Owens, A Quantitative Performance Analysis Models for GPU Architecture, In Proceedings of HPCA, 2011.