Analyzing the Sensitivity of GPU Pipeline Registers to Single Events Upsets

Josie E. Rodriguez Condia*, Marcio M. Goncalves†, Jose Rodrigo Azambuja†, Matteo Sonza Reorda*, Luca Sterpone*
*Politecnico di Torino - Department of Control and Computer Engineering (DAUIN)
{josie.rodriguez, matteo.sonzareorda, luca.sterpone}@polito.it
†Federal University of Rio Grande do Sul (UFRGS) - Institute of Informatics - PGMICRO
{mmgoncalves, jose.azambuja}@inf.ufrgs.br

***Abstract*—Graphics processing units are available solutions for high-performance safety-critical applications, such as self-driving cars. In this application domain, functional-safety and reliability are major concerns. Thus, the adoption of fault tolerance techniques is mandatory to detect or correct faults, since these devices must work properly, even when faults are present. GPUs are designed and implemented with cutting-edge technologies, which makes them sensitive to faults caused by radiation interference, such as single event upsets. These effects can lead the system to a failure, which is unacceptable in safety-critical applications. Therefore, effective detection and mitigation strategies must be adopted to harden the GPU operation. In this paper, we analyze transient effects in the pipeline registers of a GPU architecture. We run four applications at three GPU configurations, considering the source of the fault, its effect on the GPU, and the use of software-based hardening techniques. The evaluation was performed using a general-purpose soft-core GPU based on the NVIDIA G80 architecture. Results can guide designers in building more resilient GPU architectures.**

***Index Terms*—Fault tolerance, graphics processing units, pipeline registers, single event upsets**

I. Introduction

Graphics Processing Units (GPUs) were originally designed as accelerators for data-intensive applications, such as multimedia and graphics processing. However, in the last decade, the GPUs have evolved into widely-used general-purpose devices thanks to their high computation power and programming support. These General-Purpose Graphics Processing Units (GPGPUs) are feasible solutions also for safety-critical applications, e.g., in the automotive and robotics domains. In these domains, the use of fault tolerance techniques is mandatory to detect or correct faults arising during the operation of the system. Moreover, the fault detecting and correcting solutions are crucial, since in the safety-critical applications, GPUs must operate correctly even in the presence of faults. In conclusion, how to effectively guarantee the reliability and functional-safety of GPUs is still an open issue.

Recent GPU devices are designed and implemented employing high operating frequencies, the latest technology scaling approaches, and reduced operating voltages to comply with performance and energy requirements. However, some studies have found that due to the advanced semiconductor technologies they are based on, these devices may be prone to suffer by the effects of faults during the operative life [1], such as those caused by radiation interference. Radiation-induced errors are mainly caused by energized particles and can affect the correct operation of a device, even at ground level, where neutrons are the primary source of soft-errors [2]–[4]. The most common error in a device is caused by a fault forcing the change in the logical state (bit-flip) of flip-flops, registers, or memories. These faults are known as *Single Event Upsets* (SEUs) [5].

The architecture of a GPU is based on the Single-Instruction Multiple-Data (SIMD) paradigm and cores divided in pipeline stages including pipeline registers (PRs). In these cores, one instruction is fetched from memory, and then decoded and processed in parallel. Multiple execution units are used to operate the same instruction employing independent tasks (threads). The GPU employs large Register Files (RFs) to provide and retrieve data operands and results efficiently to and from the execution units. Thus, most instructions directly interact and operate with the RF to mask the latency during the execution of a parallel application. Nevertheless, the previous conditions allow the easy propagation into the system of an error caused by a fault coming from other critical structures, such as the PRs, which are distributed internally across the main executing core. The PRs store control signals of the instructions and a fault can significantly compromise the operation of a running application [6]. Moreover, these structures are hidden for the programmer and cannot be controlled or protected with conventional methods. Therefore, from the reliability viewpoint, the Architectural Vulnerability Factor (AVF) analysis and a fault propagation evaluation are mandatory for designing countermeasures to protect the execution of an application.

Recently, Software-based fault tolerance techniques for GPUs were proposed based on the hardening of the RF structure, so protecting the system against data flow errors caused by SEUs [7], [8]. Moreover, these techniques can be adapted to the source code of a program and simplify the task of software developers in the safety-critical domain [9].

In this work, we indirectly evaluate the sensitivity to SEUs of the pipeline registers in a GPU when applications are hardened with low-level software-based hardening techniques

This work has been partially supported by the European Commission through the Horizon 2020 RESCUE-ETN project under grant 722325 and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001, CNPq, and FAPERGS.

developed to detect SEUs in the register files. The program codes are adapted at assembly-level and implemented in a soft-core GPGPU based on the NVIDIA G80 architecture.

II. Background

A. Software-based hardening techniques for GPUs

Software-based hardening techniques have been adapted to GPUs in the past, aiming to detect faults and increase reliability. Software-baed solutions are mainly classified as *i*) program duplication, when the code of the entire program is duplicated, *ii*) selective duplication, when only crucial parts of the code are duplicated, and *iii*) Algorithm-Based Fault Tolerance (ABFT) [10], when the hardening directly depends on the architecture of the algorithm used in the program code.

Program duplication was employed in the past by duplicating the execution of the entire program [11]. Similarly, the authors in [7] targeted faults affecting the general-purpose register files in the GPU by replicating the whole assembly code in an intertwined fashion and reaching up to 99% error reduction at a performance cost of up to 78%. On the other hand, the authors in [12] showed that selective duplication can reduce costs in execution time and resource overhead by lowering the detection coverage. Finally, ABFT techniques can achieve high detection rates with moderate execution time and resource overhead [13]. However, ABFT is limited by the type of application to which hardening approaches can be applied.

This work extends the evaluations proposed and introduced in [7] and [14] by evaluating the pipeline registers structures located in the datapath of the GPUs. We employ the FlexGrip-Plus model to perform the experiments [15] [16].

B. FlexGrip Architecture

The FlexGrip model is an open-source soft-core GPGPU fully described in VHDL that implements the G80 architecture from NVIDIA [17]. A new version, called *FlexGripPlus*, removed some operative and programming constraints by verifying and correcting the descriptions. The new version can be fully programmed using the CUDA programming environment and supports up to 28 instructions [15] [16].

The structure of FlexGripPlus is mainly composed of an array of Streaming Multiprocessors (SMs) that executes groups of threads, also called warps, in parallel. Each SM executes instructions following variations of the SIMD and Single-Instruction Multiple-Thread (SIMT) taxonomies [18].

The workload in a SM core is assigned by a Block Scheduler Controller. Internally, the SM includes a Warp Scheduler Controller (WSC) that distributes the tasks as warps and dispatches them into the available executions units, also known as Scalar Processors (SPs). The SM is organized in a five-stage pipeline (see Fig. 1).

The datapath of a SM is mainly composed of the SPs, the PRs and the memory modules in the GPU. The memory hierarchy in the GPGU is composed of the General-Purpose Register File (GPRF), the local memory, the constant memory, the shared memory, and the global or main memory. All these memory modules are visible to the programmer and

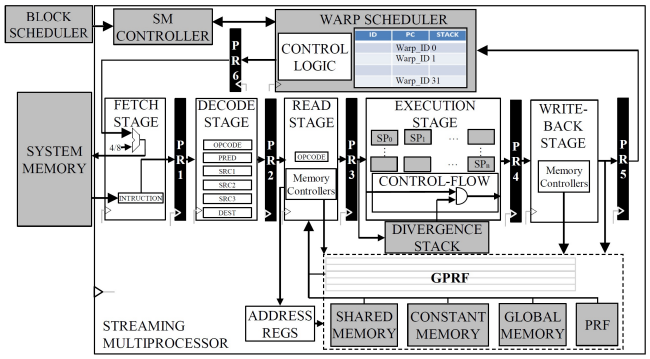

Fig. 1. A general scheme of the SM in FlexGrip.

can be configured and accessed directly or indirectly by the programmer. However, other modules, such as the PRs, are hidden to the programmer, which reduces the available methods to control or observe the operational status of these modules. The PRs are also part of the datapath and serve as interconnections among the pipeline stages. Each PR is located between two pipeline stages to temporary store operands for the execution of an instruction in the SM. However, PRs also store control signals related to the status of the instruction under operation.

PRs are grouped and named according to their location in the design. The PR(W-F), between the *Warp* and *Fetch* stages, stores control signals related with the status of the operation of an instruction in the SM. These PRs include the following parameters: Warp Program Counter (WPC), Active Thread Mask (AThM), shared memory address, and general-purpose address. Similarly, the PR(F-D), between the *Fetch* and *Decode* stages, includes the same information of the PR(W-F) and only adds information concerning the operational code of the instruction.

The PR(D-R), between *Decode* and *Read* stages, stores the decoding signals of instructions, which are employed to activate the operational modules in the next stage, such as the memory sources, SPs, and the divergence management mechanisms. The *Read-Execution* PR(R-E) includes the Temporary Registers (TRs), which handle the parallel operands and predicate conditions for every SP core in the execution stage. Finally, the *Execution-Write* PR(E-Wr) includes a similar structure of the TRs, but handles the results of the operations performed in the execute stage. One additional PR(W-Wr) is located in the SM to interconnect the *Write* stage and the *Warp* controller of the SM. The PR(W-WR) is only composed of control signals describing the status and operation of an instruction in the SM.

From the reliability point of view, as the PRs hold sensitive information for the operation of the SM (data and control signals), SEUs affecting them can lead to data corruption or the interruption of the execution flow. The size of each PRs in FlexGripPlus are reported in Table I, considering 8, 16, and 32 SP configuations. The size of the controlpath does not depend on the number of SPs, but the size of the datapath does. Therefore, for a 8-SP configuration, the sizes of controlpath

and datapath are almost the same, but the datapath for a 32-SP configuration if four times the size of the controlpath.

TABLE I
SIZE OF THE PRS ACCORDING TO THE NUMBER OF SPS IN THE SM.

Pipeline Registers	Controlpath	Datapath (SPs)		
		8	16	32
Warp to Fetch (W-F)	140	–	–	–
Fetch to Decode (F-D)	237	–	–	–
Decode to Read (D-R)	408	–	–	–
Read to Execute (R-E)	302	1,024	2,048	4,096
Execute to Write (E-W)	251	512	1,024	2,048
Write to Warp (Wr-W)	133	–	–	–
Total	1,471	1,536	3,072	6,144

III. EVALUATION OF THE PRS TO SEUS

In this Section, we present the mechanism to evaluate the SEU sensitivity. Then, we outline the implemented software-based hardening technique.

A. Fault injection environment

The custom fault injector is based on the *ModelSim* simulator, which hosts the FlexGripPlus model. The environment is based on the translation of a fault target into simulator commands in the hosting framework to represent the injection of a transient fault in the model. The process of translating and applying a fault are based on the guidelines introduced in [6], [19]. Moreover, a parallel multi-thread approach is employed to reduce the execution time of a fault campaign [20].

The reduction of the fault locations in a target module also contributes to improve the performance. This reduction (which does not impact the results accuracy) is based on observing and analyzing the switching activities and coding styles of the target application. These analyses are obtained after performing multiple fault-free simulations. Finally, structural analyses to identify untestable faults in a target module can be performed [21].

The injector environment is composed of a fault injector controller (FIC), a fault decoder (FD), and a fault injector checker and classifier (FICC). The FIC configures the GPU model and the simulation framework on ModelSim. Moreover, it orchestrates the whole fault simulation campaign. The FD translates one fault into a equivalent sequence of commands for the simulator. The FD refers to the fault list to load a random target location for fault injection. Moreover, the FD computes a random injection time considering the time intervals when the GPU kernel is operating. The generated commands are applied in the model before the simulation starts. The FICC checks the fault effect in the model and the method used for finishing the simulation. Finally, the FICC classifies the fault effect.

Each fault is classified as (i) Silent Data Corruption fault (SDC), when a fault generates mismatches in the memory results, (ii) Detected Unrecoverable Error (DUE), when a fault corrupts the execution of an application or affects the correct termination, or (iii) Masked, when a fault does not affect the system execution and the results. During the fault injection campaign, one fault was injected per application execution.

B. Software-based hardening technique

The software-based hardening technique is an adapted version of the Full Program Duplication (FPD) mechanism. In principle, the FPD is a software-based fault-tolerance technique that can be applied at all abstraction levels of the software stack. However, in our case, the GPU assembly language is preferred because we can maintain a deterministic control during the code adaptation and also guarantee that the final bytecode retains the modifications applied. Moreover, as previously mentioned, the user does not have access to the PRs. Therefore, it is impossible to design a software-based technique that directly targets the PRs and explicitly modify its normal behavior. On the other hand, even though the FPD hardening technique only targets the GPRF, we explore the possibility of detecting faults in the PR through the program transformation indirectly. Still, they are built to detect faults in the register files.

The adaptation of the FPD as software-based technique includes three main code transformations: (1) datapath duplication, (2) consistency checking, and (3) host notification.

Datapath duplication (1) starts by duplicating all used registers over spare ones. A static code analysis is used to identify the number of used and spare registers per application. Then, a hash table assigns a spare register as a copy register to each used register in the application. In case there are not enough spare registers, selective hardening or register spilling can be used. Then, all instructions are duplicated, with the replica operating over replicated registers. Instruction duplication forces the re-execution of the datapath in an intertwined fashion, completely separating the original and duplicated datapaths, and taking advantage of Instruction Level Parallelism (ILP) to speed up program execution. By exploiting ILP, we are able to absorb a portion of the execution time overhead caused by the duplication.

The consistency checking (2) is responsible for evaluating the coherence between the values in the registers and their replicas. A comparison instruction is used to compare and set an error flag in case of mismatch. The checking of consistency decreases the performance obtained by the ILP and also introduces flow-dependency when the comparison is performed between the two datapaths. In order to optimize consistency checks, they are only performed after memory accesses. By doing so, we guarantee that memory access are correct. On the other hand, as memory addresses are not replicated, a fault in the store instruction may still lead to an undetected error.

The host notification (3) informs the host that a fault has been detected. The implementation may vary as it may correspond to a memory write instruction or an exception signal to the host. Implemented through conditional instruction execution, these additional code blocks are not executed on

a fault-free execution of an application, and therefore do not cause any performance degradation under normal circumstances.

Combined, these code transformations are able to detect close to 100% SDC faults affecting the FlexGripPlus GPRF registers. It is not able to detect all faults because the memory addresses are not replicated. Therefore, store instructions are also not replicated, becoming a point of failure. On the other hand, memory (and store instruction) duplication could be easily implemented to cover those faults.

IV. Experimental results

For the experiments, we employed four case-study applications to evaluate the SEU susceptibility in the PRs. Moreover, the experiments are also used to observe the effects of the adapted software-based hardening technique.

The selected benchmarks are matrix multiplication (MxM), Fast Fourier Transform (FFT), Vector Sum (VectorSum), and Bitonic Sort (Sort). These benchmarks are selected considering the different workload and coding styles, generating different patterns of use in the PRs. MxM and VectorSum are mostly data-flow oriented, with few conditional deviations, while FFT and Sort are mostly control-flow oriented, with many conditional deviations.

Twenty four fault injection campaigns were performed at register transfer level using the FlexGripPlus model and targeting the PRs. All application code and their hardened versions have had fault simulated under the 8, 16, and 32 SP cores configurations of the GPU model. A total of 10,000 faults were injected per application under one of the SP configurations. In total, 240,000 faults were injected in the experiments.

A. Streaming multiprocessors

Table II, reports the fault rate and the Architectural Vulnerability Factor (AVF) of the original applications and the hardened ones. Although results do not provide evidence of a clear trend that could be observed for all applications, the changes in the hardware organization by modifying the number of SPs show that an increment in the number of SPs also increases the PR susceptibility to SEUs. This trend is observed when moving from 8 to 16 SPs in the FFT and VectorSum applications. However, the behavior shown in the MxM and Sort benchmarks are different. In both cases, the susceptibility to SEUs is reduced when increasing the number of SPs in the configuration of FlexGripPlus.

An explanation for the increment in susceptibility to SEUs can be found analysing the kernel configurations of the applications and the effects in the architecture when increasing the SPs in the GPU. As initially reported in Table I, the increment in the number of SPs also increases the number of registers devoted to store operands and results for the additional SPs. Moreover, the FFT and VectorSum are configured to use the number of available SPs in the system, so the simultaneous workload (number of threads) executed in parallel is proportional to the number of SPs. The previous behavior is also seen in the Sort application. However, this application has a dependency between the data operands and the executed instructions that could explain the contradictory behavior. Finally, MxM is a bi-dimensional parallel program and is configured to execute 8 threads per warp only. Thus, increasing the number of SPs has no direct effect in the number of executed operations. Nevertheless, the kernel configuration seems to be responsible for the reduction in the SEU effect.

On the other hand, a general observation of the results of the hardened-code versions of the applications shows a reduction in the percentage of faults affecting the applications. For most cases, the hardended applications seem to be partially effective in detecting faults and reducing the total percentage of the fault effects (from 16.2% to 100.0%). However, for some configurations of the FFT and Sort applications, the hardening techniques increase the susceptibility to SEU effects. In this case, both FFT and Sort applications are mainly control-flow based applications and the coding style seems to play an important role in the effectiveness of the software-based technique when changing the configuration of a GPU. Thus, we can conclude that the effectiveness of a hardening technique depends on the the coding style of a target application and the configuration of the GPU. It is worth noting that the AVF analyses were performed using the same hardware configuration to reflect the effect of the description and workload of an application in the GPU.

B. Pipeline registers

A second evaluation is performed dividing the PRs into datapath and controlpath. Table II reports the results for each application. The detected faults by the hardening technique are analysed for each application.

For all applications, the trend shows that the highest percentage of SEU effects are caused by the controlpath part of each PR. This behavior can be explained when considering that the controlpath fields in the PRs are active permanently during the execution of each instruction, while the datapath is only active when processing instructions involving operands. Moreover, parts of the datapath are inactive when executing intra-warp divergence paths or executing instructions that require only one or two input operands.

The reported results show that the distribution of the SDCs and DUEs in the controlpath and datapath for the analyzed applications directly depends on the dominant coding style of the applications.

For the FFT and Sort applications, which are control-oriented, the hardening technique was effective in reducing the number of SDCs in the controlpath. On the other hand, the number of DUEs increased. For the VectorSum and MxM applications, which are data-oriented applications, the hardening technique was partially effective in reducing the percentage of SDC affecting the controlpath of the PRs. A similar but less effective trend can be observed for the detected DUEs. Both results show that the hardening technique is more effective in data-oriented programs to reduce the effect of faults, in the PRs, propagating their effects to the register file.

TABLE II
FAULT RATE AND AVF RESULTS IN THE PRS OF THE GPGPU

Application	SPs	Original AVF (%)						Hardened AVF (%)						AVF Reduction (%)					
		Datapath			Controlpath			Datapath			Controlpath			Datapath			Controlpath		
		SDC	DUE	Total	SDC	DUE	Total	SDC	DUE	Total	SDC	DUE	Total	SDC	DUE	Total	SDC	DUE	Total
FFT	8	0.96	0.16	1.13	3.96	2.28	6.24	0.58	0.00	0.58	2.09	3.07	5.16	39.53	100.0	48.17	47.14	0.00	17.31
	16	1.06	0.08	1.14	4.88	3.92	8.80	0.65	0.08	0.73	3.73	5.93	9.67	38.76	0.49	36.03	23.50	-51.36	-9.85
	32	0.64	0.24	0.88	4.87	3.88	8.75	0.23	0.16	0.39	3.81	6.68	10.49	63.35	34.84	55.57	21.64	-72.16	-19.97
Matrix Mult.	8	0.47	0.08	0.55	3.27	2.95	6.21	0.24	0.00	0.24	0.35	2.40	2.75	49.40	100.0	56.63	89.39	18.55	55.79
	16	0.41	0.08	0.49	2.72	4.97	7.69	0.41	0.00	0.41	1.11	4.21	5.32	-0.66	100.0	16.12	59.31	15.28	30.85
	32	0.34	0.00	0.34	1.73	5.64	7.37	0.00	0.00	0.00	0.53	4.89	5.43	100.0	–	100.0	69.23	13.24	26.40
VectorSum	8	3.83	0.00	3.83	5.73	2.20	7.93	1.89	0.00	1.89	1.27	1.91	3.17	50.55	–	50.55	77.91	13.33	60.00
	16	4.09	0.00	4.09	5.93	4.43	10.36	1.98	0.00	1.98	2.40	4.12	6.52	51.60	–	51.60	59.55	6.93	37.07
	32	3.39	0.00	3.39	6.68	8.07	14.75	2.51	0.00	2.51	3.87	7.69	11.56	26.02	–	26.02	42.12	4.63	21.61
Sort	8	2.43	0.00	2.43	2.61	2.07	4.68	0.81	0.08	0.89	1.61	2.69	4.31	66.81	–	63.49	38.27	-30.32	7.98
	16	0.70	0.00	0.70	2.17	2.23	4.40	1.05	0.08	1.13	2.09	3.89	5.99	-50.27	–	-61.83	3.68	-74.85	-36.06
	32	0.40	0.08	0.48	1.68	2.27	3.95	0.08	0.08	0.16	1.43	4.27	5.69	79.82	-0.90	66.37	15.08	-88.24	-44.26

The Bitonic Sort application is a control-flow intensive, but also data-dependent application, so the register file is used to compare operands coming from memory. These comparisons are then employed to select the next instructions to execute. An in-depth analysis of the results shows that most of the faults in the controlpath are detected by one of the check point mechanisms, so detecting and notifying the fault.

Fig. 2 shows the AVF results for the controlpath in the PRs analysing the contribution of each field. Each PR includes registers devoted to store the instruction code (*Ins.code*), the thread mask (*TAM*), the starting address in the register file (*GPRS base*), the starting address in the shared memory (*Sh. base*), the Warp or line ID, the Program counter (*WPC*), and other control signals. The reported results are obtained with the SM configured with 8 SPs.

As it can be observed in the results, each application shows a different distribution of SEUs affecting each groups of control registers. However, it is possible to observe that the *Ins.code* and *TAM* are the most sensible fields in the controlpath of the PRs for all evaluated applications. In fact, the *Ins.code* field stores the signals used to activate the internal modules to process an instruction in parallel and these registers are distributed among all PRs. Moreover, this group of registers also includes the original and the decoded instruction-codes located in the (F-D) and (D-R) PRs. Similarly, the *TAM* fields are present in all PRs and store information regarding the status of the active threads executing an instruction. Although these registers are located in all PRs, the susceptibility is high due to the fact that they are massively used in each instruction, so any effect can easily propagate and compromise the execution of an instruction. A fault in the *Ins.code* field could cause the stop in the execution of an instruction or the access to unauthorized modules to process parts of the instruction as consequence of the SEU. Similarly, a fault in the *TAM* could cause the inactivation and stop of threads during instruction. Another effect caused by SEUs in the *TAM* field is the execution of additional threads. In both cases, the consequence is an error.

The difference in size of the reported results, in Fig. 2, clearly shows that in most of the cases, the hardening code by the hardening technique is effective in decreasing the susceptibility to SEUs in all groups of registers in the controlpath of the PRs. This behavior can be observed in the MxM application, which shows that for each group of registers, the hardening code effectively reduced most of the affecting fault in the controlpath.

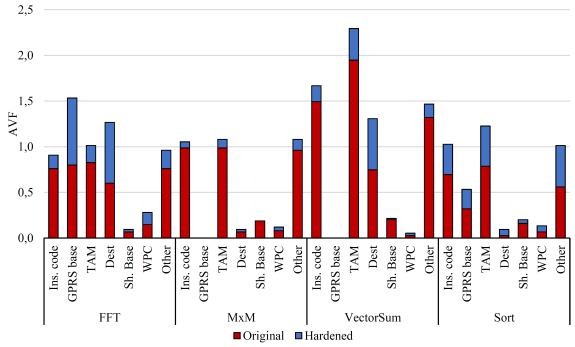

Fig. 2. AVF in Controlpath of pipeline registers for 8-core SM configuration.

C. Pipeline stages

Fig. 3 shows the pipeline stages' AVF classified according to the individual PRs in the SM for the original (red) and hardened (blue) applications.

Results for both applications show that each PR is affected differently by SEUs. It is clear that PRs (Wr-F), (F-D), and (D-R) are the most susceptible to SEUs registers for all analyzed applications. This behavior can be explained considering that these registers store the control signals used to start the execution of an instruction and also the signals devoted to activating parallel modules in the SM. Selective hardening of the 785 flip-flops in the (Wr-F), (F-D), and (D-R) PRs could be used to reduce the faults effect in a range between 59.0% and 87.9% of the detected faults for the analyzed applications.

A general overview of the fault effect in the PRs shows that all applications present the same trend. Also, the susceptibility of each PR is equivalent for the analyzed applications. Moreover, it shows that some fields in the controlpath of the PRs

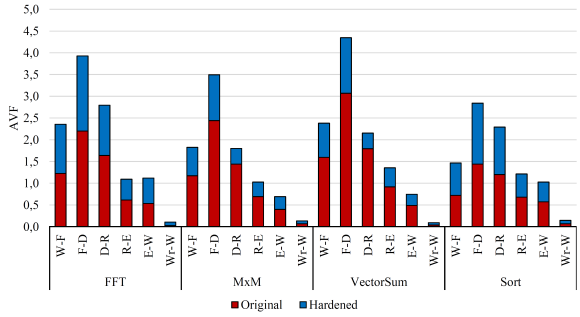

Fig. 3. AVF of the Pipeline stage registers for 8-core SM configuration.

are more susceptible to propagate fault effects than others. Interestingly, some of the most susceptible PRs are not the largest in size, so the propagation of a fault in the PRs seems to be directly related to the affected location in a PR instead of its size in the PR structure.

An in-depth analysis of a group of registers, storing control-signals, showed that a registers can produce different effects and propagate differently (as DUE or SDC) depending on the application and the time of injection. Previous results denote the relevance of the PRs in the operation of the GPU. The FFT and Sort applications are examples of a partial effectiveness of the employed hardening technique. As one can see in Fig. 3, all PRs show a significant reduction in AVF, meaning that most of them were detected by the hardening technique before becoming an error.

V. Conclusions and Future Work

In this paper, we evaluated the sensitivity of the PRs of a GPU to SEUs. A fault injection campaign was performed targeting the PRs of a GPGPU running four case-study application in three different SM configurations. Results were categorized according to SMs, PRs, and pipeline stage.

Resulting data showed that control signals are more sensitive to faults and, therefore, more critical in the pipeline. This happens mainly because a single fault affecting a control signal can compromise the operation of the whole GPU. The chosen software-based hardening technique, which was originally developed to protect register files in a GPU, was present as a feasible solution to increase GPU reliability. The correlation among the register file and the pipeline registers in the datapath allowed the indirect reduction of transient effects. The results also showed that, for some applications, the hardening solution also allowed the detection a significant percentage of faults. However, the coding style affects its effectiveness.

In the future, we plan to further evaluate the GPU's datapath and controlpath structures and propose hardware-based fault tolerance techniques targetting specific registers. We also intend to combine these techniques with software-based ones and provide a GPU fully resilient to SEU effects.

References

[1] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *Electron Devices, IEEE Transactions on*, vol. 57, no. 7, pp. 1527–1538, 2010.

[2] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *International Reliability Physics Symposium*, 2011, pp. 1–7.

[3] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on gpus," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2797–2804, 2013.

[4] J. R. Azambuja, G. Nazar, P. Rech, L. Carro, F. L. Kastensmidt, T. Fairbanks, and H. Quinn, "Evaluating neutron induced see in sram-based fpga protected by hardware- and software-based fault tolerant techniques," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4243–4250, 2013.

[5] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Transactions on nuclear Science*, vol. 50, no. 3, pp. 583–602, 2003.

[6] J. E. R. Condia and M. Sonza Reorda, "Testing permanent faults in pipeline registers of gpgpus: A multi-kernel approach," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2019, pp. 97–102.

[7] M. Gonçalves, M. Saquetti, F. Kastensmidt, and J. R. Azambuja, "A low-level software-based fault tolerance approach to detect seus in gpus' register files," *Microelectronics Reliability*, vol. 76, pp. 665–669, 2017.

[8] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing software-directed instruction replication for gpu error detection," in *International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2018, pp. 1–12.

[9] E. L. Rhod, C. A. L. Lisbôa, L. Carro, M. Sonza Reorda, and M. Violante, "Hardware and software transparency in the protection of programs against seus and sets," *Journal of Electronic Testing*, vol. 24, no. 1-3, pp. 45–56, 2008.

[10] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, pp. 518–528, 1984.

[11] S. Di Carlo, G. Gambardella, M. Indaco, I. Martella, P. Prinetto, D. Rolfo, and P. Trotta, "Increasing the robustness of cuda fermi gpu-based systems," in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, 2013, pp. 234–235.

[12] K. S. Yim, C. Pham, M. Saleheen, Z. Kalbarczyk, and R. Iyer, "Hauberk: Lightweight silent data corruption error detector for gpgpu," in *IEEE International Parallel Distributed Processing Symposium*, 2011, pp. 287–300.

[13] L. L. Pilla, P. Rech, F. Silvestri, C. Frost, P. O. A. Navaux, M. Sonza Reorda, and L. Carro, "Software-based hardening strategies for neutron sensitive fft algorithms on gpus," *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1874–1880, 2014.

[14] M. M. Goncalves, J. R. Azambuja, J. E. R. Condia, M. Sonza Reorda, and L. Sterpone, "Evaluating software-based hardening techniques for general purpose registers on a gpgpu," in *2020 IEEE 21th Latin-American Test Symposium (LATS)*. IEEE, 2020, pp. 1–6.

[15] B. Du, J. E. R. Condia, and M. Sonza Reorda, "An extended model to support detailed gpgpu reliability analysis," in *2019 14th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, 2019, pp. 1–6.

[16] J. E. R. Condia, B. Du, M. Sonza Reorda, and L. Sterpone, "Flexgripplus: An improved gpgpu model to support reliability analysis," *Microelectronics Reliability*, vol. 109, p. 113660, 2020.

[17] K. Andryc, M. Merchant, and R. Tessier, "Flexgrip: A soft gpgpu for fpgas," in *2013 International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 230–237.

[18] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, March 2008.

[19] W. Nedel, F. L. Kastensmidt, and J. R. Azambuja, "Evaluating the effects of single event upsets in soft-core gpgpus," in *2016 17th Latin-American Test Symposium (LATS)*, April 2016, pp. 93–98.

[20] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[21] J. E. R. Condia, F. A. Da Silva, S. Hamdioui, C. Sauer, and M. Sonza Reorda, "Untestable faults identification in gpgpus for safety-critical applications," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 570–573.