Improving GPU Register File Reliability With a Comprehensive ISA Extension

Marcio Gonçalves, Josie Esteban Rodriguez Condia,
Matteo Sonza Reorda, Luca Sterpone and
Jose Rodrigo Azambuja

November 23, 2020

Improving GPU register file reliability with a comprehensive ISA extension

M.M. Gonçalves, J.E.R. Condia, M. Sonza Reorda, L. Sterpone, J.R. Azambuja

Abstract – This work proposes a comprehensive ISA extension to improve GPU reliability to transient effects. Three additional instructions are proposed, implemented, and combined with software-based datapath duplication. Modified program codes are compared to state-of-the-art software-based fault tolerance techniques in terms of execution time, the circuit area is evaluated against the original GPU architecture, and a fault injection campaign is performed to assess reliability. Results show that the proposed ISA extension improves the performance of software-based approaches while maintaining fault detection capabilities at negligible costs in the circuit area. This work can help engineers in designing more efficient and resilient GPU architectures.

1. Introduction

Over the past years, Graphics Processing Units (GPUs) have been used in safety-critical applications such as self-driving cars. Newest GPUs are designed with cutting-edge technology that combines high transistor density with high operating frequency and low voltage supply, making them prone to experience radiation-induced transient effects [1, 2]. Recently, GPUs have even been shown to experience radiation effects on applications running at ground level, where neutrons are the primary source of soft errors [3, 4]. The use of fault tolerance techniques is mandatory because safety-critical applications must work properly, even in the presence of faults.

Fault tolerance techniques can be applied by means of software or hardware modifications. Software-based techniques require program code transformation, while hardware-based techniques require hardware modifications. Software-based approaches provide high detection rates at the cost of performance degradation [5] but can be applied to any GPU architecture. Hardware-based approaches, on the other hand, can be applied with no performance degradation, but require access to GPU architecture description. Open-source GPUs have allowed developers to study the effects of radiation, as well as to design and evaluate fault tolerance techniques.

The authors in [6] proposed ISA extensions to GPUs. They proposed to compare registers with a new XOR instruction that notifies the host via hardware in case of divergence. They also propose a strategy, where a signature register is updated as each instruction executes, adding or subtracting its destination register values based on whether the instruction is original or a duplicate. The technique includes an extra meta-data bit in the instructions to inform the hardware when the signature register must be updated. Experiments indicate that these ISA extensions could lower the average runtime overhead to 30%, but without achieving 100% fault detection.

This work proposes a comprehensive ISA extension to improve software-based fault tolerance techniques. Resilient atomic load, store, and set predicate instructions are developed and evaluated in a low-level software-based hardening approach. The proposed resilient instructions are deployed in tandem with datapath duplication to optimize resources and reduce costs in performance degradation.

The main contribution of this work is to decrease the overhead in execution time caused by datapath duplication by optimizing consistency checking and memory access.

2. Proposed ISA Extension

We propose three additional instructions to the SASS 1.0 ISA, used by NVIDIA, as a comprehensive ISA extension. The new proposed instructions are atomic ones, being able to check the consistency of read registers and replicate the data to written registers in single instruction execution. By doing so, this extended ISA is able to absorb multiple consistency checking instructions and memory access duplication into a single instruction execution.

Faults affecting the register file of GPU architectures can cause data or control flow effects on the system. The first group includes faults in the data structures of the system, such as input and output data registers. These faults can induce the system to an incorrect operation result, but do not change the GPU's program flow. Faults affecting the control path, on the other hand, such as registers storing conditional branch data, can cause incorrect branch decisions, modifying the correct GPU execution flow.

To harden the register file against data flow effects, we can assume that, as long as the memory between the host and the GPU is correct, there is no data flow error. Therefore, software-based techniques usually focus on checking the consistency of registers used by memory access instructions. For the control flow errors, they must guarantee that predicate registers have been correctly written. An example of a software-based hardened code can be seen in Fig. 1. As one can see on the second column (SW-based), instructions 3, 10, and 11 are used to check memory accesses for the load and store instructions, while instructions 7 and 8 are used to check a predicate set instruction.

Our ISA extension includes load, store, and predicate set instructions (raLOAD, raSTORE, and raISET, respectively), which are able to absorb the checking instructions and also the load replication. Predicate registers have not been replicated, but one could extend the raISET to also write to a second predicate register. The third column of Fig. 1 (ISA Extension) shows the software transformation when using our approach. While the pure software-based approach requires eight extra instruction, our ISA extension is able to reduce it to one.

The implementation of our proposed comprehensive ISA extensions requires software and hardware support. The software must be able to generate program code considering the new instructions, while the hardware must be able to execute the new instructions without affecting its performance.

Hardware improvements have been made to the following pipeline stages of the target GPU: Decode, Read, and Write. We have also implemented a hardware exception to notify the host about possible fault detections. We adapted these modules to consider a second source and destination registers and included a consistency check to compare original and replicated address registers and notify the hardware exception in case of divergence. Software modifications have been applied mainly to the compilation flow, where the CUDA binaries are generated. We modified the compilation flow to duplicate used registers into spare ones and datapath instructions. We also added support for the compilation flow to replace load, store, and iset instructions with our extended ISA.

3. Implementation and Evaluation

The ISA extension implementation has been done in the FlexGripPlus architecture [7], running four case-study applications. Table I shows the synthesis evaluation @15*nm*, considering the number of cells, circuit area (*mm²*), delay (*ns*), and the ISA extension overhead (%). The Decode pipeline stage showed the highest overhead, but as it is responsible for only 0.2% of the GPU circuit area, the final overhead was a negligible 0.15% in circuit area and 0.01% in the number of cells. Also, the delay had no overhead, showing that our hardware modifications did not affect the GPU's critical path, thus maintaining the maximum clock frequency.

When considering the execution time of the transformed code, presented in Table II, one can notice that pure software-based techniques showed an average overhead of 114.1%, while our ISA extension reduced overhead to an average 47.2% (a 58.6% reduction). Combined with hardware results, these data show that our approach is able to maintain the GPU performance and drastically reduce software-based techniques overheads in execution time.

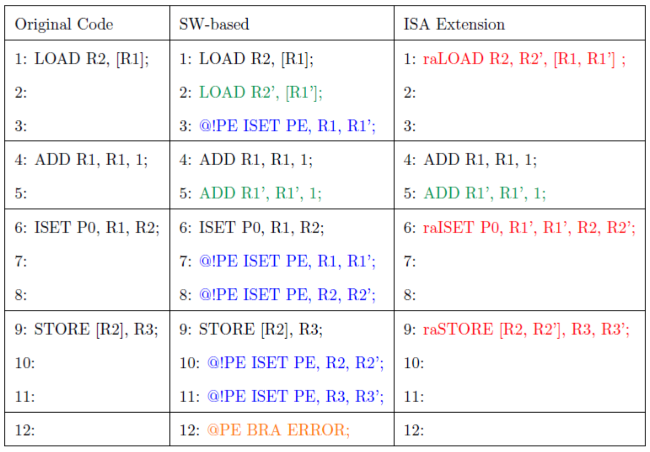
To evaluate fault detection capabilities, we performed a fault injection campaign by simulation at RTL level in ModelSim. Faults were injected during the execution of the original and the two hardened versions (software-based and ISA extension). For all case-study applications running in each program version, we injected 10,000 faults, one per program execution, adding up to 120,000 simulations. Faults have been distributed among application-used registers, achieving a 1% statistical error considering a 95% confidence level. Table III shows, for all types of errors, that both software-based techniques (SW-based) and our proposed ISA extension were able to detect all faults affecting the GPU register file.

4. Conclusions

This digest presented our proposed comprehensive ISA extension to improve the reliability of GPU register files. Results showed 58.6% improvements in performance at a negligible cost in circuit area and no degradation in reliability when compared to state-of-the-art software-based techniques. The final version will present additional data on the target GPU and discussions on implementation and fault injection result.

References

[1] Slayman et al., IIRW 2010, pp. 13-28.
[2] Dixit et al., IRPS, 2011, pp 1-7.
[3] Rech et al., Trans. on Nuclear Science, 2013, pp. 2797-2804.
[4] Azambuja et al., Trans. on Nuclear Science, 2013, pp. 4243-4250.
[5] Goncalves et al., Microelectronics Reliability, 2017, pp. 665-669.
[6] Mahmoud et al., SC18, 2018, pp 842-853.
[7] Du et al., DTIS, 2019, pp 842-853.



Original Code	SW-based	ISA Extension
1: LOAD R2, [R1];	1: LOAD R2, [R1];	1: raLOAD R2, R2', [R1, R1'] ;
2:	2: LOAD R2', [R1'];	2:
3:	3: @!PE ISET PE, R1, R1';	3:
4: ADD R1, R1, 1;	4: ADD R1, R1, 1;	4: ADD R1, R1, 1;
5:	5: ADD R1', R1', 1;	5: ADD R1', R1', 1;
6: ISET P0, R1, R2;	6: ISET P0, R1, R2;	6: raISET P0, R1', R1', R2, R2';
7:	7: @!PE ISET PE, R1, R1';	7:
8:	8: @!PE ISET PE, R2, R2';	8:
9: STORE [R2], R3;	9: STORE [R2], R3;	9: raSTORE [R2, R2'], R3, R3';
10:	10: @!PE ISET PE, R2, R2';	10:
11:	11: @!PE ISET PE, R3, R3';	11:
12:	12: @PE BRA ERROR;	12:

Fig. 1. Program code transformation.

Table I – Synthesis evaluation @15*nm* and ISA extension overhead

Pipeline Stage	FlexGripPlus			ISA extension (%)		
	Cells	Area	Delay	Cells	Area	Delay
Decode	1,793	793	0.11	10.48	8.17	0.0
Read	152,444	62,782	0.34	0.17	0.21	0.0
Write	65,315	29,192	0.65	0.81	1.14	0.0
GPU	612,368	280,480	35.56	0.01	0.15	0.0

Table II – Execution time and hardening techniques overhead

Application	Original (*ns*)	SW-based (%)	ISA extension (%)
Matrix Mult.	320.3	109.5	45.3
FFT	963.7	104.0	63.7
Vector Sum	140.6	105.2	45.3
Bitonic Sort	823.9	137.6	34.3

Table III – SDCs and hardening techniques detection

Application	Original	SW-based (%)	ISA extension (%)
Matrix Mult.	5,307	100.0	100.0
FFT	3,724	100.0	100.0
Vector Sum	3,096	100.0	100.0
Bitonic Sort	3,158	100.0	100.0