A dynamic reconfiguration mechanism to increase the reliability of GPGPUs

Josie E. Rodriguez Condia†, Pierpaolo Narducci*, M. Sonza Reorda‡, L. Sterpone§
Politecnico di Torino, Torino, Italy
*pierpaolo.narducci@studenti.polito.it, {†josie.rodriguez, ‡matteo.sonzareorda, §luca.sterpone}@polito.it

Abstract[1]— **General Purpose Graphic Processing Units (GPGPUs) are effective solutions for high-demanding data processing applications. Recently, they started to be used even in safety-critical applications, such as autonomous car driving systems. GPGPUs are implemented using the latest semiconductor technologies, which are more prone to faults arising during the lifetime operation. However, until now fault mitigation solutions were not extensively included in GPGPUs, due to the limited reliability requirements of the applications they were originally intended for (e.g., gaming or multimedia). This work proposes a dynamically configurable self-repairing mechanism aimed at mitigating the impact of permanent faults in the Scalar Processor (SP) cores in GPGPUs. The mechanism is based on spare modules that can be used to replace faulty SPs when a fault is detected. A configuration instruction allows dynamically controlling in software the selection of the set of active SPs in the SM. The method is extremely flexible since it does not require any change in the application software. Experimental results show that the solution introduces a moderate area overhead while allowing continue working even in the case of any permanent faults affecting the SPs.**

Keywords— **Fault mitigation, Fault tolerance, General Purpose Graphics Processing Units (GPGPUs), Graphics Processors.**

I. Introduction

In the last two decades, General Purpose Graphic Processing Units (GPGPUs) have become effective solutions mainly employed in data-intensive commercial applications, such as multimedia, multi-signal analysis and high-performance computing (HPC), thanks to their highly parallel architecture. Actually, this technology is a promising solution in many computationally-intensive applications requiring fast and real-time signal processing.

In the automotive field, GPGPUs are already commonly adopted in sensor-fusion systems and Advanced Driver-Assistance Systems (ADAS)[1]. These devices are designed for high-performance requirements and low power consumption. Thus, it is common to employ for their manufacturing the latest technology scaling approaches. Nevertheless, it is well-known that these technologies are more prone to suffer from faults during both production and lifetime operation, raising some concerns in terms of reliability [2]. More in detail, the new devices may suffer from critical effects, such as wear-out, causing the dropping in long-term reliability of the device, and aging, increasing the number of permanent faults in the device. In fact, these technologies introduce new reliability challenges in the long-term, where traditional end-of-production test approaches are not enough [3].

[1] This work has been partially supported by the European Commission through the Horizon 2020 RESCUE-ETN project under grant 722325.

Most commercial GPGPUs include fault mitigation mechanisms for memories based on Error Correcting Codes (ECCs). In most industrial applications, these ECCs provide the required reliability. In the worst case, when the GPGPU stops working, it may be replaced.

A different scenario exists in the automotive industry. For this kind of functional safety applications, the effects of a fault may cause unacceptable operational failures. Thus, GPGPUs may require complementary fault mitigation solutions to be applied during real-time operation.

Traditional solutions to increase the reliability of digital design are based on hardware, software, and hybrid approaches. The hardware mechanisms are considered as feasible solutions in applications with strong requirements in terms of functional safety and reliability, such as the automotive one. In this case, an additional cost can be justified by the improved features and capabilities. Hardware solutions include Duplication with Comparison (DWC), Double and Triple Modular Redundancy (DMR, TMR), ECC and the hardening of selective logic gates [4]. The adoption of these solutions requires a careful evaluation of the involved area and power consumption overhead. Moreover, some of these techniques are mainly intended to mitigate the effects of transient faults in a system. In contrast, mitigation of permanent faults requires strategies, such as Built-In Self-Repair (BISR), replacing a faulty block with a spare one.

In BISR, the granularity of the block depends on the target module, and the complexity and criticality of the device [5]. In the past, BISR has been successfully applied in the memory blocks of processor-based systems by adding spare rows, columns, and additional controller structures to correct faults during the production phase and also during in-field execution [6, 7]. Other works [8-10] targeted data-path units, such as the register file, and some internal components of the execution units (EUs)[11]. Similarly, some works proposed reconfiguration solutions targeting computational blocks in GPGPUs [12] or other modules in the GPGPU, such as the memories [13], and functional units [14], or combinations of both aligning the system to the specific workload requirements [2, 15]. In [16], the author proposes selective hardware redundancy to correct defective blocks during the production stage. This method is intended to increase the production yield during manufacturing.

Other works introduced functional tests [17, 18], fault detection [19-22], and mitigation [23-25] strategies only based on software mechanisms. These solutions are effective in detecting most faults and tolerating a high percentage of them. Moreover, the added area overhead is zero. Nevertheless, their cost in terms of performance degradation and memory overhead may be relevant due to these solutions are implemented by instrumenting the application code with custom functions. Similarly, in [26], the authors proposed a software-based redundant multithreading mechanism multiplying the threads to be executed. However, the performance overhead is directly dependent on the workload and the behavior of the application. Moreover, it is required a program translation and recompilation, thus limiting the in-field operation for embedded systems. Considering the previous works, the combination of software mechanisms and additional hardware modules in a hybrid structure to achieve the same results may be attractive.

In the present work, we propose a BISR strategy mainly aiming at addressing problems related to permanent faults effects, during the in-field operation, in the EUs (or Scalar Processor (SP) cores) inside the Streaming Multiprocessor (SM) of a GPGPU. This BISR strategy leverages on the high regularity of the SPs in the GPGPU architecture. We leverage the techniques recently described in [17], which allow the detection of permanent faults in the SP cores of a GPGPU resorting to software self-test procedures.

The basic idea behind our work lies in introducing a given number of Spare SP (SSP) cores, which may substitute any faulty SP as soon as a permanent fault affecting it is detected. In our proposal, the reconfiguration can be activated via software with an additional instruction (*Config_SPs*), which has been purposely introduced in the GPGPU Instruction Set. Moreover, this instruction is compatible with the original programming language of the GPGPU. Apart from the execution of *Config_SPs* instruction, the mechanism is completely transparent to the programmer. The method only allows tolerating faults affecting SP cores, which correspond to a significant fraction of the total SM area.

The adapted BISR mechanism is intended to add as minimal as possible structural changes into the existing hardware of the GPGPU, and thus on its performance. Finally, the proposed solution does not require any change in the application code. The proposed BISR mechanism (together with the related test) can be activated during power-on or at reset when timing constraints for fault detection and hardware reconfiguration are not so relevant.

The proposed BISR solution has been implemented and evaluated resorting to an extended version of the FlexGrip model, which represents a simplified version of the NVIDIA GPU architecture. Extensive experimental results showing its cost and effectiveness have been gathered referring to that model.

Although the usage of spare units is a well-known solution in dependable architectures, to the best of our knowledge this is the first work proposing its adoption at the SP core level in a GPGPU, and exploring in a comprehensive way the costs/benefits of its integration in a hardware model representing a real GPGPU.

The paper is organized as follows. Section II provides an overview of the open-source GPGPU model employed for our work. Section III describes the proposed fault mitigation method, as well as a summary of the software test mechanisms which can be used to detect faults in the SP cores and how they can be integrated in the proposed solution. Section IV describes the implementation of the proposed method in the extended FlexGrip model. Section V reports the experimental results and analysis, and Section VI draws some conclusions.

II. THE FLEXGRIP GPGPU MODEL

FlexGrip is an open-source GPGPU model fully described in VHDL. This model was initially developed by the University of Massachusetts and optimized for Xilinx FPGAs [27]. FlexGrip implements the Nvidia G80 micro-architecture, and it is also compatible with the CUDA programming environment under SM_1.0 compatibility.

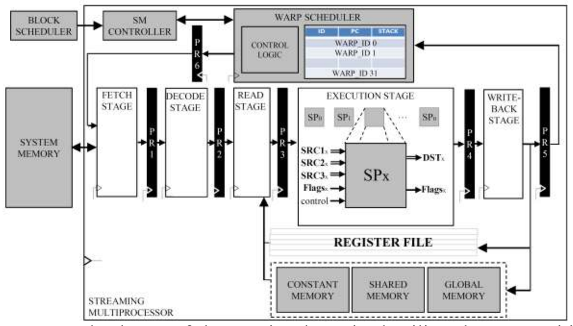

Fig. 1. A general scheme of the SM in FlexGrip detailing the composition and interconnections of the SP in the execution/control-flow pipeline stage

The original FlexGrip version was substantially improved to remove several functional limitations related to compiler restrictions, structural bugs, and instruction format support. Additional detail can be found in [28]. The improved FlexGrip model fully supports 27 instructions in 64 formats.

The model accepts a set of configuration parameters, such as the kernel parameters, including the Grid dimension, Block dimension, Blocks per core, number of registers per thread, and the number of blocks per SM core. Similarly, for simulation purposes, the data memory (*global*) and the benchmark instructions should be specified for the model.

The internal architecture of FlexGrip is based on the SIMT (*Single-Instruction Multiple-Thread*) paradigm and exploits a custom SM core with a five stages pipeline (*Fetch*, *Decode*, *Read*, *Execution/Control-flow* and *Write-back*), as shown in Fig. 1. This special-purpose parallel processor executes the same instruction (*warp instruction*) for a set of threads. A warp is defined as a group of 32 threads. Moreover, the SM employs a warp scheduler controller (WSC) for thread management. In the SIMT paradigm, one warp instruction is fetched, decoded, and distributed to be processed on an independent SP within the SM. The *Read* and *Write-back* stages load and store data operands from/to Register Files, shared, global or constant memories. Fig. 1 shows a scheme of the GPGPU detailing the interconnections affecting the SPs. These input and output interconnections are static for each SP and can be divided into data-path and control-path ones.

Every SP has three input data operands of 32 bits (*SRC1*, *SRC2*, and *SRC3*), and predicate flags (4 bit-size) forming a data channel. *SRC1*, *SRC2,* and *SRC3* are selected depending on the instruction type and are loaded during the *Read* stage. Moreover, some control signals select and configure the SP during execution. As output, the SP produces the result (*DST*) signals and the changes in the predicate flags. The storing location for *DRT* is determined in the *Write-back* stage. The input and output data channels are independent for each SP. In contrast, the control-path connections are shared among all SPs.

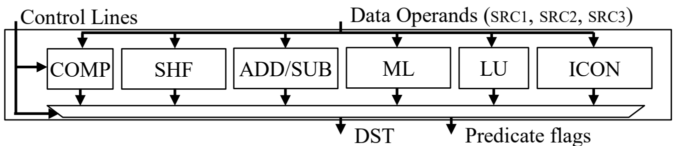

Fig 2. A general scheme of the internal architecture of the integer SP core

The SPs process signed and unsigned integer operands and include hardware modules for addition/subtraction (ADD/SUB), multiplication (ML), integer conversion (ICON), comparison (COMP), shifting (SHF), and logic unit (LU) with basic logical operations (AND, OR, XOR and NOT). As shown in Fig 2, FlexGrip was designed to support configurations composed of 8, 16 and 32 SPs.

III. PROPOSED FAULT MITIGATION TECHNIQUE

Given the complexity of a GPGPU, different mitigation methods should be used addressing the different composing parts. This work proposes a fault mitigation strategy targeting the SPs in the *Execution/Control-flow* stage of a GPGPU. This method aims at increasing the reliability of this stage by disabling an SP once it has been labeled as faulty due to a permanent fault, and substituting it with a Spare SP core (SSP). The solution is based on a hybrid approach.

The hybrid approach combines some mechanism to detect permanent faults in the SPs, based for example, on Design for Testability (DfT) or Software-Based Self-Test (SBST) test programs. For instance, in [21] we showed that suitable test programs can detect a high percentage of permanent faults within a single module. Once a faulty SP has been identified, a re-configuration process is launched. This process executes an ad-hoc instruction, which replaces the faulty SP by a spare one. For the purpose of this paper, we do not focus on the fault detection and localization phases but focus on the hardware changes to be introduced to support the reconfiguration phase (see Fig 3). It is worth noting that in this work, we did not consider fault administration structures (FAS) to be activated after a device shut-down and recover a previous configuration state in the device. These FAS could be composed of flash memories and controllers to store the state of SPs and SSPs in the device.

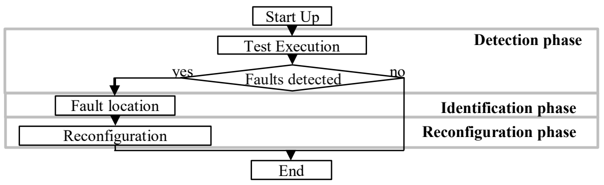

Fig 3. A general scheme of the detection, identification, and configuration approaches for fault mitigation

A. *Fault Mitigation Architecture*

The BISR architecture is based on the addition of a given number of SSP modules in parallel to the existing SPs. These SSPs are cold stand-by modules, thus reducing the power consumption during inactivity. Two switching modules are added to control the data-path signals in the existing SP and SSPs. The switching units are placed in the *Execution/Control-flow* stage of the GPGPU following the circular switching scheme, see Fig. 4. The switching units are based on meta-crossbar structures targeting the control of the data-path input and output interconnections in the SPs. A controller module is added to control SPs and SSPs in the system. Fig. 4 shows the general scheme of the configuration structure composed of n SP cores and m SSPs added and connected to the SM. It is worth noting that the number of input and output signals on each SPs is different. Thus, the size of the input and output switches may differ. Nevertheless, the same mechanism is employed to manage each SP core.

In order to minimize the impact of the changes on the existing architecture, we preserved the original memory hierarchy and WSC modules. Thus, each input Thread Data Channel (ThDC) is kept at the input of the execution stage.

The input switching unit includes, as outputs, the additional SP-data channels (SPDCs) and also connects them to the SPs and SSPs. In this way, each input ThDC is connected with one SPDC in the configurable scheme. The output SPDCs, coming from the SPs and SSPs, are connected with the output switch unit and the original output ThDC of the execute pipeline stage.

The output switch reduces the total number of data-path channels in order to keep the same pipeline interconnections. In the adapted mechanism the input switch behaves as a data

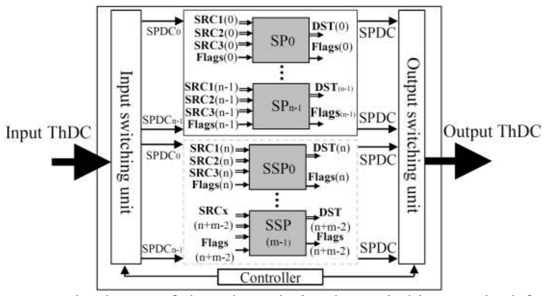

Fig 4. A general scheme of the adopted circular switching method for SP core configuration

channel de-multiplexer. Similarly, the output switch acts as a data channel multiplexer.

The placement of the two switching modules at the input and output of the SP cores contributes to maintaining the original memory hierarchy for each thread execution without relevant changes in the design. This is achieved considering that each ThDC does not include information related to the direct association of a ThDC to a specific SP core to perform operations. Thus, each thread process uses the original registers and memory locations, even when a spare core is active. Hence, the proposed solution is entirely transparent to the software, which must not be modified in any way.

The solution we followed does not impact the WSC existing in the SM. The WSC traces the execution and the state of each thread in a warp, but it does not include information related to the SP core allocation, thus remaining without changes.

The switch controller manages the configuration of both switching units using the same input control signals. The custom *Config_SPs* instruction generates the input control signals and activates the re-configuration of the SPs.

More in detail, the instruction selects one SP and one SSP core and forces the GPGPU to substitute the former with the latter for all the following activities. In the current version, the selected configuration is not saved anywhere: hence a test and possible re-configuration should be performed at each power-on or reset. The introduction of a small Non Volatile Memory could allow storing the configuration. The format of the instruction is selected avoiding any overlapping with the original instruction set of the GPGPU. The instruction format is divided into two parts. The first part of the instruction picks and enables one of the available SSPs in the system. The second part manages the switching units by selecting the correct input and output data channels for each SP and SSP core.

B. *Fault detection, fault identification, and reconfiguration*

For the purpose of this paper, we assume that the test and possible reconfiguration steps are both performed during the device power-on (or reset), so the fault detection and location phases, as well as the reconfiguration one, can be executed without any strict time and memory constraints.

For the sake of completeness, we summarize here how the fault detection and location phases could be implemented. More details about this solution can be found in [17]. However, other solutions (e.g., based on DfT) could be used as well.

At the power-on, the BISR structure is inactive. Thus the SP cores are initially connected with each ThDC and the SSP cores remain in cold standby mode. Then, a set of test patterns is applied to the SPs. These patterns are based on the execution of well-defined operations to test in parallel each SP in the SM.

The strategy employed is based on targeting all sub-modules in the SP cores and forcing them to execute suitable test patterns using instructions. The partial thread results are stored in the global memory for later analysis. As each thread executes the same instructions on different SP cores, a Signature-per-Thread (SpT) mechanism is used. Every SpT is compared with a set of previously stored results, and depending on the comparison each SP core can be labeled as faulty. This method allows for quick identification of the faulty SP.

Assuming that *n* SPs and *m* SSPs are available, the test is then repeated after reconfiguring the GPGPU so that *m* SP cores are substituted with *m* available SSP cores. At the end of this phase, a full map of the faulty and fault-free cores is available. Based on this map, if at least *n* cores are available, the GPGPU can be reconfigured accordingly and can continue working correctly.

For the sake of simplicity and to avoid reconfiguring the code, we assume that the minimum number of cumulative fault-free SPs and SSPs in the SM is *n*. This value is compared each time to verify when the SM cannot operate anymore.

The above procedure assumes that the system does not include any Non Volatile Memory (NVM). Hence, at each power-on, a complete test is required to build the map of faulty / fault-free cores. If an NVM is available, this map can be stored there and used to reconfigure the GPGPU accordingly at each power-on. The map is then updated with a given frequency, depending on the reliability targets and scenario parameters.

IV. Implementation

The improved version of the FlexGrip model was used to experimentally evaluate the performance of the proposed BISR fault mitigation strategy. This mechanism is implemented in the *Execution/Control-flow* pipeline stage of the GPGPU. Nevertheless, additional changes were made in the *Decode* and *Read* stages by the introduction of the configuration instruction. The *Decode* stage was modified by adding new combinational logic to decode the added instruction.

The *Read* stage includes a bypass register to keep the pipeline coherence during instruction execution and also to store the configuration information for the SP cores. This information is then decoded and employed in the *Execution/Control-flow* stage. In the *Execution/Control-flow* stage, the description of the switching structures is the same for the input and output switches: both switching units were designed using the same basic descriptions, i.e., multiplexer blocks and bypass register structures (see Fig. 5). Moreover, an automatic generation mechanism is described to interconnect the SPs and busses with the input and output of the multiplexers in the same stage. Additional decoding, concatenating and de-concatenating blocks control the activation of the module and reduce the number of multiplexers in the system. In this way, a big multiplexer module is attached to each additional SSP. Similarly, the output switch uses multiplexers to interconnect with the output ThDCs. The ThDCs are selectable depending on the number of SPs in FlexGrip (8, 16 or 32).

Both switches (input and output) are activated under the same control condition, thus reusing the controller structure. This switching controller includes additional registers, to store the configuration information, and decoders to reduce the total number of control bits. This controller information is used to design the instruction op-code composed of eleven active bits fields. Moreover, these registers maintain the inactive SSPs and SPs in a cold standby mode, thus avoiding any unnecessary switching activity and reducing the power consumption.

The management of the information flow (ThDCs and SPDCs), allows the usage of the same registers and memory locations employed by the SPs and the replacing SSPs, when active. In this way, the memory is virtualized from the mitigation modules. Instead of a restriction, this condition was exploited to add the BISR infrastructures in the GPGPU by employing the same memory hierarchy modules. The control-path lines on each SP core are not considered as inputs in the switches due to these interconnections are shared on all SP cores and can be directly assigned on the SSPs.

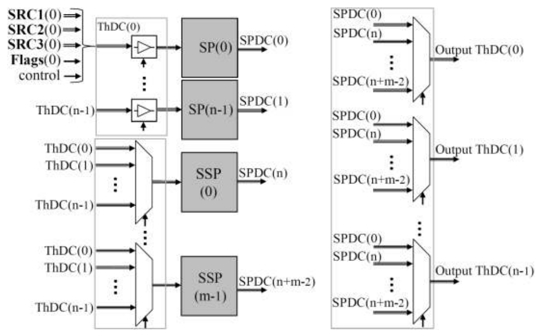

Fig. 5. A general scheme of the implemented structure in the FlexGrip model

The capabilities and flexibility of the FlexGrip model for selecting the number of SPs in the SM are also employed in the description of the mitigation modules. The same code style is used to describe the BISR modules. These structures are parametrically generated depending on the total number of SP and SSP cores aiming to reduce the hardware overhead in the system among configurations.

V. Experimental Results

The FlexGrip model was configured in 3 modes (8, 16, and 32 SPs), so it is possible to analyze the benefits and limitations of the proposed BISR strategy under different SPs in the device. We implemented the proposed strategy in the model and ran extensive simulations to validate its correctness. Moreover, we quantitatively evaluated the cost and the benefits of the strategy. For each considered case, the number of introduced SSPs ranged from 0 to 7. The analyses were performed resorting to a gate-level version of the FlexGrip model. The estimation of the hardware overhead was done resorting to the *Design Vision* tool by Synopsys using the ultra-compiler configuration. The NanGate Open-cell library was employed for the experiments [29].

A. Hardware overhead

The modules modified for implementing the BISR strategy are the *Decode, Read,* and *Execution* stages. These modules were modified at the RT level. Then, the GPGPU model was synthesized at gate level and compared in size with respect to the original design. Table 1 reports the hardware overhead results: for each configuration, we reported the required number of cells and the percent of area overhead, computed concerning the corresponding configuration in the original version of the model.

The hardware overhead introduced by the BISR strategy can be split into two parts: from one side, there is the cost to implement the instruction, the switching modules, and the switching controller. The "0 SSPs" configuration is used to quantify the hardware cost in the BISR structure. The hardware overhead of these structures represents a low percentage of the whole hardware: for all SP core configurations, the hardware cost is in the range from 0.8% to 1.7%. From the other side, there is the cost for the SSPs, which linearly grows with their number and becomes largely dominant when the SSPs increase. In fact, the addition of one SSP core (6,623 cells) introduces hardware overhead greater than 3% in all SP configurations. Clearly, the hardware overhead rate grows with more SSPs and is higher when the SPs are lower. The optimum choice of both parameters depends on the design requirements, e.g., connected to the computation required by the application, by the probability of faults (given by the operating environment and by the semiconductor technology), by the target reliability and by the duration of the mission. In any case, it is worth noting that the hardware overhead remains below 20% for all the considered GPGPU configurations.

The last two columns in Table 1 report some figures allowing the evaluation of the relative size of the SPs with respect to the total size of the FlexGrip model. From the results, it is shown that the percent area of the whole SM that can be protected resorting to the BISR strategy ranges from about 25%, in the 8 SPs configuration, to about 55% with 32 SPs. It is worth noting that the adopted BISR mechanism was aimed to mitigate faults in the SP cores, only. Other solutions can be used to mitigate faults in other modules.

TABLE 1. HARDWARE OVERHEAD OF THE BISR STRATEGY FOR MULTIPLE CONFIGURATIONS OF THE GPGPU

Version	SP cores	SSPs	Total Cells in design	Area overhead (%)	Total SP cells in the design	SP/SSP cores cells (%)
Original	8	0	229,515	-	52,984	23.08
	16	0	280,132	-	105,968	37.82
	32	0	386,100	-	211,936	54.89
Fault Tolerant	8	0	231,343	0.8	52,984	22.90
		1	237,279	3.4	59,607	25.12
		2	243,063	5.9	66,230	27.24
		4	254,692	11.0	79,476	31.20
		6	266,182	16.0	92,722	34.83
		7	271,757	18.4	99,345	36.55
	16	0	283,160	1.1	105,968	37.42
		1	290,034	3.5	112,591	38.81
		2	296,164	5.7	119,214	40.25
		4	309,318	10.4	132,460	42.82
		6	321,529	14.8	145,706	45.31
		7	335,139	19.6	152,329	45.45
	32	0	392,476	1.7	211,936	53.99
		1	400,902	3.8	218,559	54.51
		2	410,280	6.3	225,182	54.88
		4	425,172	10.1	238,428	54.07
		6	440,576	14.1	251,674	57.12
		7	460,372	19.2	258,297	56.10

B. *Performance and power overhead*

Since the BISR strategy requires the introduction of some complex switching modules to flexibly interconnect all the SPs and SSPs with the rest of the system, it clearly impacts the GPGPU overall performance. We performed an experimental analysis of this phenomenon resorting to the data produced by the synthesis tool. In particular, the impact on the performance of the adapted BISR strategy has been evaluated by analyzing the changes in the critical path delay for all configurations.

Results showed that for a large number of SSPs (6 or 7), the performance degradation reaches up to 20%. This is mainly caused by the logic included in the input and output switches and inside the switch controller. More in detail, for one SSP, the timing degradation is up to 15% and up to 16% for 2 SSPs. Clearly, it should be noted that the reported results have been obtained without executing any specific optimization to reduce such a performance overhead.

The power overhead can be neglected for this BISR strategy by considering that all inactive SSPs and SPs act as cold standby modules. Moreover, other structures remain in a configuration state. Thus, only static power by leakage current is consumed during operation. In a real implementation of the strategy, the transistor technology (i.e., 12 or 7nm) presents leakage currents in the order of 10nA/µm to 12nA/µm. Thus, the final power overhead of the BISR strategy is negligible in comparison with the dynamic power consumption produced in the GPGPU.

C. *Reliability advantages*

The goal of the adapted BISR strategy is to allow a GPGPU-based system to continue working even after one or more faults arose within the SPs. This strategy is independent of the considered fault model, provided that a suitable technique to detect it and identify the affected SP core is available.

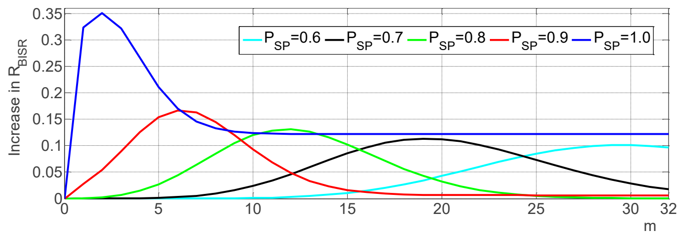

Fig. 6. Improvement in the reliability of the BISR structure for multiple probabilities of correct execution under multiple configurations of the SSPs (m).

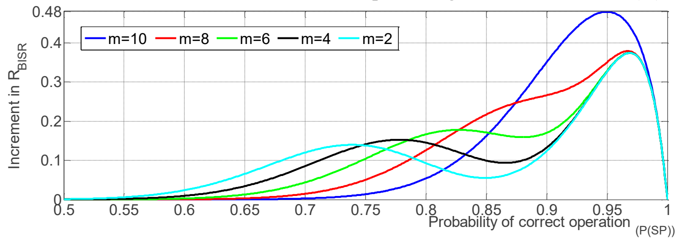

Fig. 7. Improvement in the reliability of the system R_{BISR} with respect to the probability of correct execution for various values of SSPs (m).

The reliability estimation of this strategy is based on the probability of correct operation of the system after a time *t*. Considering a GPGPU composed of *n* SPs and *m* SSPs, the execution of the system is correct if all thread instructions are operated without failures in the available execution units of an SM. Moreover, the probability of proper operation in the GPGPU can be described as the probability of GPGPU failure (when at most *(m+1)* SPs or SSPs fail). Both units (SPs and SSPs) are identical and operate independently among them, hence the probability of correct operation at time *t* in the SPs ($P_{SP(t)}$) and the SSPs is equal. In this way, the probability of proper execution, using the BISR mechanism (R_{BISR}), follows a cumulative distribution function as reported in equation 1.

$$R_{BISR} = \sum_{i=0}^{m+1} \binom{n+m}{i} \left[P_{SP(t)}\right]^{n+m-i} \left[1 - P_{SP(t)}\right]^{i} \qquad (1)$$

Extracting terms from equation 1 (see equation 2), the first one at right represents the probability of correct operation in the original GPGPU ($P_{GPU(t)}$) corresponding to the case where m=0, so tolerating a single faulty unit. On the other hand, the second term at right represents the added probability of a correct operation using the BISR strategy.

$$R_{BISR} = \sum_{i=0}^{1} \binom{n}{i} \left[P_{SP(t)}\right]^{n-i} \left[1 - P_{SP(t)}\right]^{i} + \sum_{i=2}^{m+1} \binom{n+m}{i} \left[P_{SP(t)}\right]^{n+m-i} \left[1 - P_{SP(t)}\right]^{i} \left[P_{sw(t)}\right]\left[P_{c(t)}\right] \qquad (2)$$

$$R_{BISR} = P_{GPU(t)} + \sum_{i=2}^{m+1} \binom{n+m}{i} \left[P_{SP(t)}\right]^{n+m-i} \left[1 - P_{SP(t)}\right]^{i} \left[P_{sw(t)}\right]\left[P_{c(t)}\right]$$

with:

$$P_{GPU(t)} = P_{SP(t)}{}^{n}[1-n] + n\left[P_{SP(t)}\right]^{n-1}$$

As $P_{GPU(t)}$ is a component of the probability for the BISR mechanism, it proves that $R_{BISR} > P_{GPU(t)}$. Thus, the GPGPU improves reliability according to the second term in equation 2. This term also includes the probability of the correct operation of the switching structures $P_{sw(t)}$ and in the controller $P_{c(t)}$. The dominant factor is the total number of SSP units (*m*) added in the BISR structure. Moreover, there is a direct relationship between the number of SSPs (*m*) and R_{BISR}. However, the BISR strategy may be feasible when considering a balance among overhead, cost, and reliability. In principle, *m* cannot be higher than *n*.

Fig. 6 represents the increment of the reliability of the BISR version (R_{BISR}) with respect to the original version for multiple values of $P_{ST(t)}$ and SSPs (*m*). From the graph, it can be noted that R_{BISR} is strongly dependent on the values of *m* and P_{ST}. In fact, R_{BISR} presents a maximum reliability peak whose position varies for each combination of P_{ST} and *m*. This peak value can be used to select the number of SSPs in the GPGPU considering a target probability of correct execution in the system. After this value, the effectiveness of the strategy drops down.

The graph in Fig. 7 describes the relation among $P_{ST(t)}$ and the increment of R_{BISR} for multiple values of m. This graph represents the gained benefits in terms of reliability for multiple BISR configurations. As expected, the increase in the number of SSPs (m) has a proportional positive impact on the reliability of the target structure. As can be seen from the graph, the BISR mechanism provides almost 10% of increased reliability, even when the probability of correct execution is dropped by up to 20%. Figures 6 and 7 can be employed to select the best trade-off among the parameters to reach a given target reliability.

D. Comparison with other techniques

A comparison with other well-known strategies such as lock-step and TMR can be performed. In principle, a TMR mechanism is highly reliable. However, this is not feasible to be used in the SPs due to the excessive hardware and dynamic power overhead. The lock-step strategy provides a high percentage of fault tolerance for most modules in a GPGPU. Nevertheless, it requires the duplication of each module. Thus, the hardware overhead is equal to or greater than 100%. A similar situation can be found in terms of power consumption. In contrast, the adapted BISR strategy takes advantage of the regularity of the SPs to reduce the hardware overhead to less than 20% even in the worst case. Moreover, the inactive SPs remain in cold stand-by mode reducing the power consumption of the mitigation strategy.

It must be also underlined that the proposed BISR strategy, based on dynamic reconfiguration, is particularly well suited for long-term missions (which are common for example in the automotive domain) since it allows avoiding the issues created by fault accumulation.

VI. CONCLUSIONS

A dynamic Built-In Self-Repair strategy was devised and evaluated targeting the mitigation of permanent faults possibly affecting the execution units (SPs) in the SM of a GPGPU. The BISR strategy is based on the introduction of a new instruction, which allows removing a faulty SP from the set of active ones, substituting it with one of the available spare SP cores. Results show that the structures required to implement the proposed technique introduce a relatively low hardware overhead (<4% with a single spare core). Moreover, we showed that the area of the modules where faults can be tolerated with the BISR structure can achieve about 55% of the total SM area.

The strategy seems particularly suitable for long-term missions since it allows mitigating the effects of fault accumulation in the SP cores. Although the experiments were performed on a specific NVIDIA-based GPU architecture, we claim that the proposed solution can be easily extended to other architectures as well. As future work, we plan to extend and combine mitigation approaches for permanent and transient faults in different modules of a GPGPU.

REFERENCES

[1] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integration,* vol. 59, pp. 148-156, 2017/09/01/ 2017.

[2] L. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, *et al.*, "GPGPUs: How to combine high computational power with high reliability," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1-9.

[3] S. Hamdioui, *et al.*, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 129-134.

[4] I. Polian and J. P. Hayes, "Selective Hardening: Toward Cost-Effective Error Tolerance," *IEEE Design & Test of Computers,* vol. 28, pp. 54-63, 2011.

[5] T. Koal and H. T. Vierhaus, "Logic self repair based on regular building blocks," in *22th International Conference on Architecture of Computing Systems 2009*, 2009, pp. 1-6.

[6] C.-L. Su, Y.-T. Yeh, and C.-W. Wu, "An integrated ECC and redundancy repair scheme for memory reliability enhancement," in *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, 2005, pp. 81-89.

[7] C.-L. Su, R.-F. Huang, and C.-W. Wu, "A processor-based built-in self-repair design for embedded memories," in *2003 Test Symposium*, 2003, pp. 366-371.

[8] M. Schölzel, T. Koal, S. Müller, S. Scharoba, S. Röder, and H. T. Vierhaus, "A comprehensive software-based self-test and self-repair method for statically scheduled superscalar processors," in *2016 17th Latin-American Test Symposium (LATS)*, 2016, pp. 33-38.

[9] T. Koal and H. T. Vierhaus, "A software-based self-test and hardware reconfiguration solution for VLIW processors," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010.

[10] R. S. Ferreira and J. Nolte, "Low latency reconfiguration mechanism for fine-grained processor internal functional units," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1-6.

[11] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Precision-aware soft error protection for GPUs," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 49-59.

[12] K. Kwon, *et al.*, "Mobile GPU shader processor based on non-blocking Coarse Grained Reconfigurable Arrays architecture," in *2013 International Conference on Field-Programmable Technology (FPT)*, 2013.

[13] J. Zhao, G. Sun, G. H. Loh, and Y. Xie, "Energy-efficient GPU design with reconfigurable in-package graphics memory," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, 2012, pp. 403-408.

[14] W.-J. Lee*, et al.*, "A scalable GPU architecture based on dynamically reconfigurable embedded processor," *High Performance Graphics,* pp. 5-7, 2011.

[15] A. Dhar and D. Chen, "Efficient GPGPU Computing with Cross-Core Resource Sharing and Core Reconfiguration," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 48-55.

[16] J. R. Nickolls, "EFECT TOLERANT REDUNDANCY," Nvidia Corp., 2009.

[17] S. Di Carlo, *et al.*, "A software-based self test of CUDA Fermi GPUs," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1-6.

[18] D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, and L. Carro, "On the evaluation of soft-errors detection techniques for GPGPUs," in *2013 8th IEEE Design and Test Symposium*, 2013, pp. 1-6.

[19] B. Du, J. E. R. Condia, M. Sonza Reorda, and L. Sterpone, "About the functional test of the GPGPU scheduler," in O*n-Line Testing Symposium (IOLTS) 2018 IEEE 24th International*, 2018.

[20] S. Di Carlo, J. E. R. Condia, and M. Sonza Reorda, "On the in-field test of the GPGPU scheduler memory," in *22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2019)*, 2019.

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

[22] S. D. Carlo, J. E. R. Condia, and M. S. Reorda, "An On-Line Testing Technique for the Scheduler Memory of a GPGPU," *IEEE Access,* vol. 8, pp. 16893-16912, 2020.

[23] 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.

[24] S. Di Carlo*, et al.*, "Increasing the robustness of CUDA Fermi GPU-based systems," in *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, 2013, pp. 234-235.

[25] S. Di Carlo, *et al.*, "Fault mitigation strategies for CUDA GPUs," in *Test Conference (ITC), 2013 IEEE International*, 2013, pp. 1-8.

[26] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron, "Real-world design and evaluation of compiler-managed GPU redundant multithreading," *ACM SIGARCH Computer Architecture News,* vol. 42, pp. 73-84, 2014.

[27] 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.

[28] B. Du, J. E. R. Condia, and M. Sonza Reorda, "An extended model to support detailed GPGPU reliability analysis," in *14th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2019.

[29] J. Knudsen, "Nangate 45nm Open Cell Library," *CDNLive, EMEA,* 2008.