The Journal of Supercomputing (2021) 77:11625–11642
https://doi.org/10.1007/s11227-021-03751-2

DYRE: a DYnamic REconfigurable solution to increase GPGPU's reliability

Josie E. Rodriguez Condia[1] · Pierpaolo Narducci[1] · Matteo Sonza Reorda[1] · Luca Sterpone[1]

Accepted: 15 March 2021 / Published online: 29 March 2021

Abstract

General-purpose graphics processing units (GPGPUs) are extensively used in high-performance computing. However, it is well known that these devices' reliability may be limited by the rising of faults at the hardware level. This work introduces a flexible solution to detect and mitigate permanent faults affecting the execution units in these parallel devices. The proposed solution is based on adding some spare modules to perform two in-field operations: detecting and mitigating faults. The solution takes advantage of the regularity of the execution units in the device to avoid significant design changes and reduce the overhead. The proposed solution was evaluated in terms of reliability improvement and area, performance, and power overhead costs. For this purpose, we resorted to a micro-architectural open-source GPGPU model (FlexGripPlus). Experimental results show that the proposed solution can extend the reliability by up to 57%, with overhead costs lower than 2% and 8% in area and power, respectively.

Keywords In-field test · Fault mitigation · Fault-tolerance · General-purpose graphics processing units (GPGPUs) · Reliability · Testing

✉ Josie E. Rodriguez Condia
josie.rodriguez@polito.it

Pierpaolo Narducci
pierpaolo.narducci@studenti.polito.it

Matteo Sonza Reorda
matteo.sonzareorda@polito.it

Luca Sterpone
luca.sterpone@polito.it

[1] Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, Turin, Italy

1 Introduction

Currently, GPGPUs are a major workhorse in applications involving data-intensive operations, such as multimedia and high-performance computing (HPC). Moreover, GPGPUs are now widely used in the electronics equipment for safety-critical systems, for example, in the automotive and robotics fields [1]. In all these cases, the reliability can be limited by the effects of transient and permanent faults affecting the GPGPU hardware [2]. In fact, GPGPUs are implemented using the latest technology scaling to increase performance and reduce power consumption. However, some studies [3] show that devices manufactured with cutting-edge technologies can be particularly prone to faults arising during the operational-life caused by aging, wear-out, or external effects (e.g., radiation) [4], so compromising their reliability [5]. As a result, traditional end-of-line testing is no longer sufficient to address these emerging reliability challenges properly.

In order to tackle reliability issues, some solutions have been proposed in the literature [6]. These can be divided into three main categories: software, hardware, and hybrid.

The software solutions rely on modified versions of the application code to harden and mitigate fault effects [7]. These solutions are noninvasive, flexible, and have been proven in GPGPUs [8], but can be very costly in terms of performance [9]. In [10], the authors developed fault-tolerance solutions for parallel processors by adjusting the instruction-level parallelism, increasing the reliability at the cost of workload performance. On the other hand, authors in [11] propose a reduced precision *Duplication with Comparison* (DWC) approach to increase the reliability in GPUs by replicating instructions and operating them in execution units at different precision, so obtaining redundancy at zero cost, but degrading performance and output precision.

The hardware solutions are based on special structures devoted to verifying the correct operation and mitigate errors in the modules. They may use design for testability (DfT) structures, e.g., *Built-In Self-Test* (BIST), to detect faults, hardware redundancy, and spare modules to provide fault-tolerance (*Built-in Self-Repair* or BISR). Among the possible hardware solutions, a popular one relies on including redundancy to guarantee the long-term reliability of GPGPU devices. Special strategies, such as *Error Correcting-Codes* (ECCs), contribute to reducing the sensitivity to faults in some large structures, such as memories, register files, and communication interfaces [12]. However, the mitigation of faults in other modules, e.g., execution cores, scheduling controllers, and dispatchers, is more complex. Authors in [13] employed a dual-lockstep structure to provide fault-tolerance capabilities to processor-based devices against transient fault effects. In contrast, Authors in [14] explore several fault-tolerant strategies to harden a processor against faults. The results show that complex devices can take advantage of several strategies depending on the affected module to reduce the total overhead, which in principle can also be adopted into GPU devices. Other strategies, such as DWC [15], *Triple Modular Redundancy* (TMR), BISR or combinations of redundancy, custom controllers and hardware diversity [16] [17], are

adopted for elaborated modules with considerable costs in terms of hardware and power overhead [18]. However, the additional cost in terms of design and manufacturing of these detection and mitigation strategies may be unaffordable. In the past, several solutions have been proposed (at different levels of granularity) for hardening processor-based systems. These solutions include the repair of pipeline stage modules [19], reconfigurable structures for processing elements in a device [20], repair of embedded SRAM memories [21] and the test and repair of different modules in parallel architectures [22]. However, their extension to GPGPUs has not been fully explored.

Hybrid mechanisms are optimized solutions that combine hardware structures and software mechanisms to detect [23] and mitigate faults [24, 25]. The hardware and hybrid solutions must be adopted during the design stages of a device and may significantly impact the devices. Nevertheless, their main advantage is the low-performance overhead during the operation [26].

A compelling case is when we aim at protecting the execution cores of a GPGPU. These are regular structures that represent a considerable percentage of the area and are the principal operative elements inside the GPGPU. In [27], the authors propose a fault detection and mitigation technique for large modules by employing a DWC mechanism. In [28], a hybrid approach called Dynamic Duplication with Comparison (DDWC) is presented aimed to detect faults in the execution cores during the in-field operation. Similarly, in [29], and [30], the authors propose mitigation solutions for similar structures by adapting the BISR mechanism to replace faulty modules during the manufacturing process and the in-field operation, respectively. Nevertheless, most currently adopted fault-tolerance solutions for GPGPUs do not provide the detection and the mitigation of faults using the same architecture. Moreover, only in rare cases, the solutions are intended to operate and be flexibly activated during the in-field execution of a device.

In this work, we propose a solution called *DYnamic REconfigurable structure for in-field detection and mitigation of faults* (DYRE) based on the coalescence of the classical DWC mechanism and the BISR approach. DYRE is intended to increase the reliability and operative-life, by supporting both detection and mitigation of permanent faults in the execution cores of a GPGPU. This mechanism allows the reconfiguration of the GPGPU to identify (through comparisons) and mitigate (by module replacement) possible faults arising during the in-field operation. The architecture of a GPGPU architecture adopting DYRE can be dynamically re-configured using custom instructions purposely added to the instruction set. Finally, the DYRE architecture is designed to avoid significant changes in the original GPGPU design and minimize its impact on execution performance.

The main contributions of this work are given as follows:

1. The proposal of one architecture to detect permanent faults in the execution units of GPGPU cores and mitigate their effects during the in-field operation;
2. The evaluation of the hardware, power, and performance cost involved by the DYRE architecture and of its benefits in terms of reliability enhancement.

The results and analyses show that the overall GPGPU's reliability in the execution cores is improved by up to 57% when the DYRE architecture is used. Moreover, DYRE introduces less than 1% of performance degradation, less than 5% of hardware (Area) costs, and less than 8% of additional power consumption. Hence, we claim that the proposed approach may represent a viable and promising solution to develop highly reliable GPGPUs with minimum design and overhead costs with respect to commercially available GPUs.

The manuscript is organized as follows: Sect. 2 introduces the architecture of a GPGPU, also detailing the model employed to implement the proposed approach (FlexGripPlus). Section 3 describes the proposed fault-tolerance technique. Section 4 reports the experimental results and the performance features of the proposed fault-tolerance mechanism. Finally, Sect. 5 draws the main conclusions of the work and highlights some future works.

2 Background

2.1 Classical fault-tolerance mechanisms

This subsection describes the two classical approaches to provide fault detection (Duplication with Comparison) and fault mitigation (Build-In Self-Test) into digital devices. These approaches are the basis for the development of the proposed DYRE structure.

2.1.1 Duplication with comparison

DWC is a classical fault-tolerance approach and can be employed at several levels of abstractions. For this work, we describe the DWC approach used at the hardware abstraction level. However, the same concept can also be applied from the hardware up to the system level. In these levels, DWC employs the concept of *Sphere of Redundancy/Replication* [31], which is based on replicating one or more components (modules or systems) with the purpose of increment the fault-tolerance capabilities of a device or system.

The replication of components is used to perform simultaneous parallel operations in the original and redundant components. The output results of both (original and redundant ones) are compared to identify any mismatch, which is employed to determine errors or indicate the presence of a fault. Thus, the active use of DWC increases the fault detection capabilities during the system's operation. The DWC structure is commonly complemented by other modules, including input and output selector switches, one comparison unit, and one general controller.

2.1.2 Built-in self-repair

BISR is a fault-tolerance mechanism that allows the mitigation of faults in a system by replacing an affected component (module or system) with a spare and fault-free copy. The BISR follows a *regular switching strategy* that consists of adding one or

more spare copies of a component and activating them (switching the input and outputs of the component) when a faulty component is detected, so correcting the fault effect and extending the operative life and reliability of the system.

The BISR strategy includes input and output switching modules attached to the target components and the spare ones, which are located in parallel. One general redundancy manager is employed to manage the active components' operation in a system and perform the swapping among the components in the system.

2.2 Fundamentals of GPGPU organization

A GPGPU is organized following the single instruction multiple data (SIMD) paradigm and its adaptations, such as the single instruction multiple thread (SIMT) by NVIDIA. The microarchitecture of a GPGPU is based on multiple processing modules (also known as *Streaming Multiprocessors*, or SMs). The SMs are composed of local controllers, schedulers, cache memories, a register file, and several execution units (EUs, CUDA cores, or Scalar/Streaming Processors (SPs)) devoted to simultaneously executing one instruction on multiple data.

One general controller (Block scheduler controller) submits tasks to each SM in the GPGPU. The local controllers in the SM manage and trace the operation of the assigned tasks by submitting groups of threads (Warps) to the available SPs for parallel operation. Each SP can be composed of an integer and a floating-point module. Modern SMs also include hardware accelerators for specific tasks, e.g., matrix operations and neural networks processing.

2.3 FlexGripPlus

FlexGripPlus [32] is an open-source RT-level model of a soft-GPGPU described in VHDL and is an extended version of the FlexGrip model [33] initially designed by the University of Massachusetts. FlexGripPlus corrected some significant architectural restrictions and programming bugs presented in FlexGrip, meanwhile preserving the original descriptions.

FlexGripPlus implements the G80 architecture of NVIDIA and is also compatible with the CUDA programming environment with the compute architecture SM_1.0. The architecture of FlexGripPlus is based on a custom SM core composed of five pipeline stages (*Fetch*, *Decode*, *Read*, *Execute*, and *Write*), as shown in Fig. 1. The SM includes a warp scheduler controller that manages the execution of the instructions and controls the operations of each thread. One warp instruction is fetched, decoded, and dispatched to be executed into the available SPs in the SM. In the *Read* and *Write* stages, the operands are loaded and stored from/to one of the memory resources (register file, or shared, constant and global memory) in the system. The number of SPs in the SM can be selected among 8, 16, or 32.

The SPs are composed of multiple sub-modules to operate signed and unsigned arithmetic and logic operations. The inputs to each SP core are organized as data channels (iDCx) consisting of 32 bit-size input data operands (SRC1, SRC2, and SRC3) and predicate flags (4 bit-size). The output data channel (oDCx) is composed

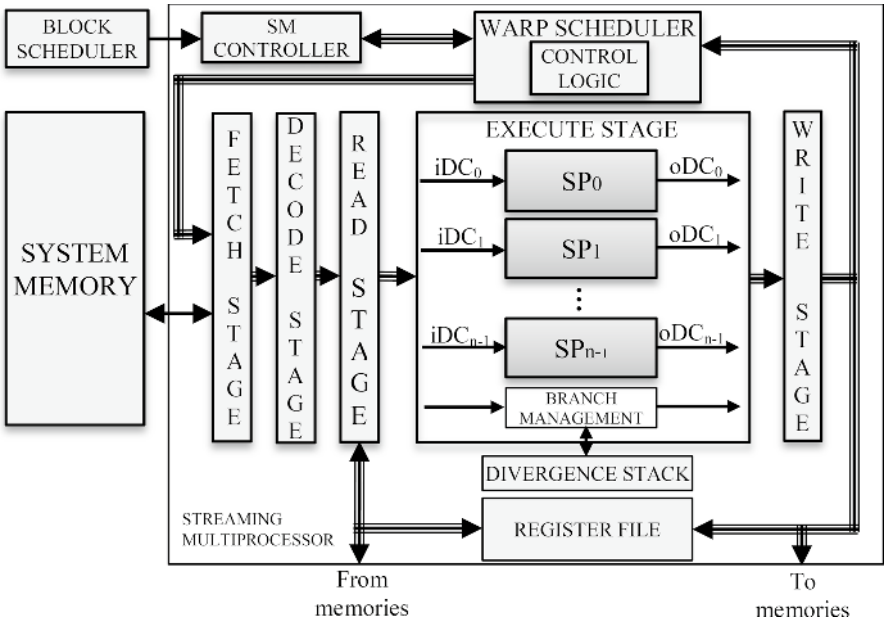

Fig. 1 A general scheme of the FlexGripPlus model

of the 32 bit-size result and the output predicate flag from each SP. These output channels are connected statically to the next pipeline stage in the SM. Similarly, the SPs are statically assigned to any thread/task by the controller in the SM. In the fault-tolerant architecture proposed in this work, the external control signals are redundantly used to configure the SP's operation with the instruction to operate.

3 Proposed solution

DYRE is a fault-tolerance architecture intended to detect permanent faults in the SP cores of an SM (in the GPGPU) and mitigate their effects. This mechanism takes advantage of the high regularity and homogeneous composition of the SP cores, the parallel execution of the thread/tasks on the SPs, and the distribution of the tasks among the SPs to reduce the cost in terms of hardware and performance. The DYRE architecture is based on the addition of one or more spare SPs (SSPs) in the Execute stage of the SM. Each additional SSP can be employed for results comparison or replacement purposes. It includes a mechanism for dynamically deciding how to use the available SSPs, thanks to the introduction of two additional instructions in the GPGPU instruction set. The execution of these instructions allows for pairing an SSP with a given SP (comparing the results they produce) or substituting one SP with a given SSP, respectively.

In particular, an SSP can

1. be paired to an SP, so that it performs the same operations on the same input data; hence, the results produced by the paired SP and SSP can be compared, and this allows detecting possible faults affecting one of the two modules
2. replace a faulty SP core.

The architecture of a DYRE GPGPU differs from a normal one only in the *Execute* stage (see Fig. 2). It includes one or more SSP cores, three crossbar units (input, middle, and output), some configuration registers, one comparator block (COMP), a controller unit, and some decoding logic. This structure provides flexibility allowing two non-exclusive operational features: (1) the in-field detection of faults and (2) the in-field mitigation of faults in the SPs.

The specific architecture of a DYRE GPGPU can be flexibly and dynamically decided by executing ad hoc assembly instructions introduced in the GPGPU instruction set to activate the fault detection and fault mitigation features. The *DETection Trigger* (DETT) instruction enables the DYRE comparison structure and configures and selects an SSP to be paired with an SP. The SSP and SP to be compared are included as part of the instruction format. Thus, when active, the DETT instructions enable comparing the selected pair (SP and SSP) results for fault detection purposes. Similarly, the *MITigation Trigger* (MITT) instruction enables the replacement structure of DYRE and reconfigures the GPGPU, substituting one SP with an SSP for mitigation purposes. The instruction format in MITT includes fields

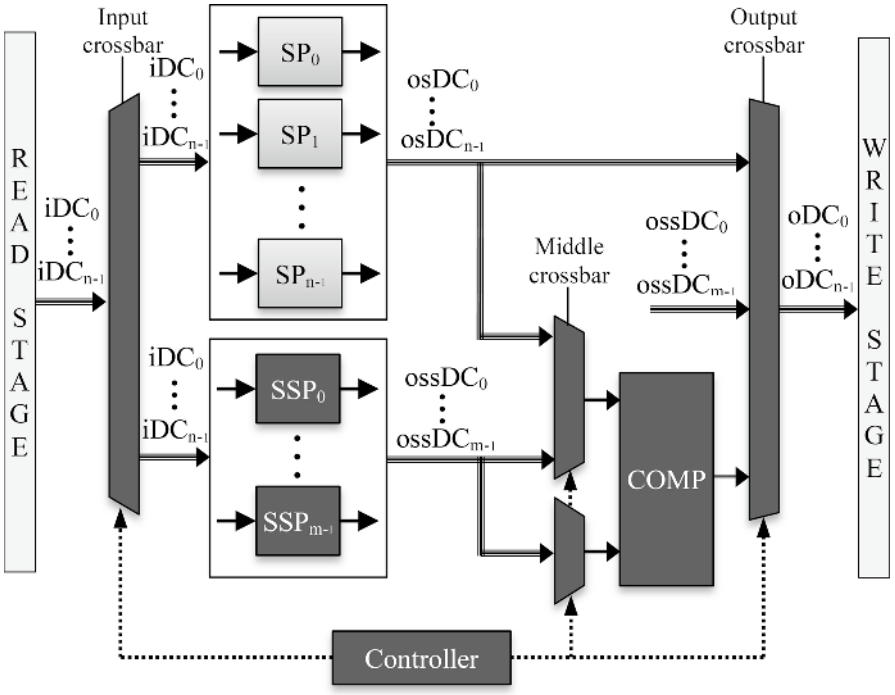

Fig. 2 A general scheme of the Execute stage of a GPGPU with the DYRE architecture

to select the SP and SSP to be commuted. For both cases, a programmer can employ any selection policy to control the comparison and replacement among the available SPs and SSPs.

Moreover, both instructions can reconfigure the DYRE architecture with the cost of only one instruction cycle and are intended to be included in a running application, so dynamically enabling both features of a DYRE GPGPU for in-field operation. Both operational features (*detection* and *mitigation*) are intended to use the same hardware structures, thus reducing the overall hardware cost. However, more than one SSP is required to use both operational features in DYRE simultaneously.

3.1 Fault detection

This operational feature is inspired by the DWC mechanism and uses a sphere of redundancy composed of the active SPs in the SM. The DYRE architecture uses this feature to detect faults through the comparison of results. When DETT instruction is executed, the local controller enables the fault detection feature, and one SSP and one SP are selected to perform all the following instructions in parallel. This procedure is transparent for the execution of the application. The SP and the SSP can be paired by a time interval or the entire execution of the application. Moreover, the target SSP or SP can be replaced with another core at any moment of the in-field operation by executing a new DETT instruction.

More in detail, the DYRE architecture uses two crossbars (*input* and *middle*) to select a target SP. Both crossbars select and duplicate the input and output data channels to feed the SSPs core and the comparator block, respectively.

After each operation, the results of the SP and the SSP are compared. The comparator triggers a faulty flag when a mismatch is detected. The flag is propagated to the next stage and sent to the exceptions unit in the GPGPU or the Host.

3.2 Fault mitigation

This operational feature is based on an adaptation of the BISR mechanism, and it is intended to mitigate the effect of faults in the cores by disabling and replacing one affected SP core with one of the available SSPs in the system. The SSPs are organized as cold standby modules and are active only when required. Correspondingly, the inactive SP cores are disabled to reduce the power consumption during inactivity.

The static distribution of tasks among the SPs allows the correction of faults by switching the input data from a faulty unit to a fault-free unit. This behavior also reduces further changes in other modules of the GPGPU. For this purpose, it is possible to mask the replacement of a faulty SP by an SSP. Thus, the fault-mitigation structure operates transparently from the memory and scheduling controller's point of view.

More in detail, the execution of the MITT instruction activates two crossbars (*input* and *output*, as depicted in Fig. 2) to redirect the data-flow of the data channel

from one active SP (faulty core) to the selected SSP (fault-free), so mitigating the fault effect. The effect of the MITT remains active for all subsequent instructions.

3.3 Suggested methods of use

The DYRE architecture is intended to operate in two cases: (1) in the Power-on/reset phase of the device and (2) during the in-field operation of an application.

At the power-on, the DYRE architecture is inactive. Hence, the SSPs are initially idle as cold standby modules. A specially crafted test program applies patterns to check the possible presence of permanent faults in each SP. This program includes several DETT instructions that activate one SSP and swap the available SPs to perform comparisons when executing the same instructions on the same data. If a mismatch is found, the SP is labeled as faulty. The program replaces the faulty SPs with SSPs through MITT instructions, and the application starts. It is worth noting that the generation of suitable test programs for the SPs is out of the scope of this work. However, previous works [34] showed that generating them is feasible.

Nevertheless, the use of DYRE during in-field operation requires adding one or several DETT instructions in the application code. Each DETT instruction selects one SSP, so activating the fault detection through comparisons. When comparison produces a mismatch between the results, during the execution of the application's instructions, a fault is identified as detected. Then, a subroutine activates the MITT instruction to replace the faulty SP with one SSP. This subroutine can be launched when a mismatch is generated or during the idle times of an application. The replacement subroutine (with MITT) is intended to substitute the faulty core with minimal latency in the execution of the application, considering the low reconfiguration cost of the mitigation feature.

It is worth noting that the DYRE architecture does not include any fault administration structure to store the actual configuration state and possibly be restored after a device power-off or reset. This fault administration structure could be composed of a non-volatile memory (NVM) and some controllers to store the state and role of the SPs and SSPs of both operational features. Hence, at each power-on, a complete test is required to build the map of faulty/fault-free cores. Alternatively, the map can be updated with a given frequency, depending on the characteristics of the application and parameters in the structure, such as the number of SSPs.

3.4 Implementation

DYRE was implemented in FlexGripPlus, modifying the *Decode*, *Read*, and *Execute* stages. The hardware to support the DETT and MITT instructions was added in the *Decode* stage. Similarly, a bypass mechanism and some changes in the memory controllers were performed in the *Read* stage to add flexibility to the instructions. The implementation allows the adoption of the DYRE architecture with any of the three SP configurations (8, 16, and 32) of the model.

The *Execute* stages include the additional SSPs, the crossbars, and the controllers of the DYRE architecture. The main purpose of the crossbars is the

selection of the input and output data channels (iDCx and oDCx) to feed the SPs and SSPs in the system. The input crossbar selects one of the iDCx feeding the active SP cores and can duplicate or switch the input data to one of the SSPs. In case of duplication, the selected SSP redundantly executes precisely the same operation of the selected SP. In contrast, in the case of switching, the input crossbar substitutes the iDCx of one SP core and feeds a selected SSP. The control signals of the SP cores are statically shared among the SP and SSPs in the system.

The middle crossbar is composed of two independent crossbars used to feed the two inputs of the COMP module. COMP is only used during the fault detection operation and is composed of a bitwise comparator that compares the results and output flags from two execution units (SPs or SSPs). On the other hand, the output crossbar manages the results coming from the active SPs and SSPs. This crossbar is used to select the output channels (osDCx and ossDCx) from the active SPs and SSPs and feed the next pipeline. The flexibility of the middle crossbar allows the comparison of two SSPs when the mitigation and duplication mode are simultaneously activated.

The input and output crossbars are indeed meta-crossbars and multiplexer structures used to preserve the same type of input and output data channels in the *Execute* stage and from and to other stages of the SM.

Some configuration registers are employed to select among the operational features (detection, mitigation, or both). The local controller configures the DYRE architecture using decoded commands that came from the DETT and MITT instructions. Some decoding logic is included to manage the two operational features when controlling the crossbar structures.

The DETT and MITT instructions were designed to select the channels or target cores using operands coming from an immediate value or a general-purpose register. This flexibility in the instruction format allows the dynamic selection of the target core during the in-field operation. Both instructions use a format composed of six bits stating the instruction type. The other five bits select the input data channel to be switched for duplication or replacement, and five bits select the target SSP core to be used.

In order to use DETT and MITT, a programmer only needs to add any or both instructions, as part of the application, to activate the detection (DETT) or mitigation (MITT) features of DYRE. In the first case, DETT and MITT can be added before the original application code, so activating static detection and mitigation. Advanced use of DYRE requires the application's adaptation to include the instructions, so one or several DETT instructions enable different comparisons among the cores and spare cores. Once a fault is detected, the MITT instruction replaces the faulty core with one available spare one. Finally, there is also the possibility of developing special test routines, including DETT and MITT, to perform functional testing on the cores before starting an application's execution. This alternative is intended for the Power-on/off stages in the system.

4 Experimental evaluation

Two evaluations are performed on the proposed mechanism. Firstly, the overhead assessment determines the cost in terms of hardware, power, and performance of the DYRE architecture. For this purpose, the DYRE architecture is compared against the original design, DDWC, which is based only on fault-detection [7], and with BISR, which is based only on fault mitigation [9]. The original GPGPU and the three fault-tolerance mechanisms were synthesized using the *Design Compiler* tool using the 15 nm Nand gate Open-cell library and one clock of 500 MHz. It is worth noting that the internal memories were not synthesized. Figure 3 reports the results of the hardware and power overhead for each setup. Finally, a second evaluation analyses the reliability features of the proposed mechanism.

4.1 Hardware overhead analysis

Two cases were considered for the hardware overhead evaluation: (1) considering the affected modules, only and (2) considering the whole system. In the first case, the assessment was performed considering the modules affected by modifications when implementing DYRE. In the second case, the cost of the entire design is evaluated. All evaluations were performed using the three configurations with 8, 16, and 32 SPs.

According to the results, the hardware cost of implementing the instructions in the *Decode* stage is lower than 5% and almost negligible for the *Read* stage ($\approx 0.3\%$). Nevertheless, the SSPs' implementation directly affects the hardware cost in the *Execute* module. For a configuration of 8 SPs, the cost of using two SSPs is lower than 13%, but it increases to 42% when DYRE is configured to use the same SPs and SSPs. Among the SP configurations, it can be noted that in the *Execute* module and the entire design, the hardware overhead follows a proportional inverse relation. Thus, large SP configurations present low hardware overhead. In *Execute*, when adding 25% of SSPs, the cell and area costs are around 10% and 8%, respectively. In the case of adding 50% of SSPs, these costs are about 22% and 17%.

On the other hand, the hardware overhead in the design's logic is lower than 7% for all configurations. In the case of two SSPs, the cell and area overhead are lower than 2%, causing a minimum impact on the design when using DYRE. When the SM is configured with 32 SPs, the addition of one or two SSPs caused negative

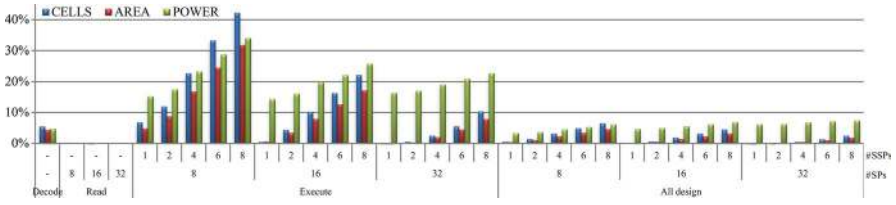

Fig. 3 Percentage of overhead cost of the DYRE architecture on each adapted module and in the entire GPGPU

percentages of hardware overhead. However, these values are due to the optimization constraint in the synthesis tool, and the effect is translated as power overhead for these configurations.

4.2 Power and performance analysis

From Fig. 3, the power consumption in the *Decode* module indicates a minimum overhead (< 5%), and it is almost negligible in the *Read* module for all SP and SSP configurations. In the *Execute* module, the addition of one or two SSPs in all SP configurations causes a moderate average cost of power from 14 to 17%. When DYRE is configured to include 50% of SSPs for each SP configuration, the power cost is moderate (around 23.7% and 25.9%). Moreover, the overhead reaches up to 34% when the number of SSPs and SPs is equivalent. Nevertheless, the entire logic cost remains stable and is lower than 8% in all configurations.

In terms of performance, the DYRE architecture does not introduce more than 1% of degradation in the critical path for all the evaluated configurations.

Although the synthesis of the model used only the clock constraint, the results in Fig. 3 show the distribution and the trend to consider when implementing the DYRE solution. In this way, the addition of two SSPs can be affordable in terms of hardware (< 2%) and power (< 8%) costs.

An overhead comparison of the DYRE structure with DDWC and BISR architectures is reported in Fig. 4. Each strategy was implemented and synthesized for the three possible SP configurations. In principle, results show that hardware overhead in DYRE is the lower of the three strategies (< 5%) and decreases when increasing the number of SPs in the design.

It must be noted that DYRE is a reconfigurable structure, so the power consumption of the solutions directly depends on the number of active features in the structure. When the comparison mechanism is active, the additional power cost is mainly caused by the active SSP. In contrast, when the mitigation feature

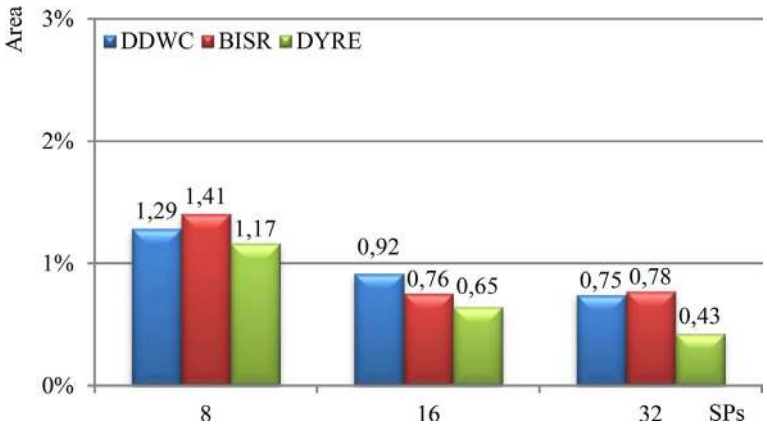

Fig. 4 Area overhead for the DDWC, BISR and DYRE architectures with respect to the original design evaluated in the 8, 16 and 32 SP cores configurations with one SSP

is employed, the power consumption remains the same as the original design. The replacement of one SP by one SSP does not add any power consumption load. It is worth noting that the DYRE structure uses a hot sparing strategy and the added SSPs remain as cold standby modules. However, during the configuration phase, a transient increment of power is presented when activating the controller and managing the switches.

Nevertheless, this transient power cost is almost negligible as most of the consumption is due to the active SPs in the SMs. From synthesis results, the SP cores consume about 70% of power in the Execute pipeline of the SM and about 55% for the entire design. Thus, the average increment of using the DYRE structure and one active SSP for fault detection purposes is equal to 15.75% of additional power in the Execute pipeline of the SM, see Fig. 3 (DYRE with 8 SPs and 1 SSP).

A way to balance the trade-off between the performance and power consumption needs to evaluate the power consumption in a workload. This consumption is used as the base for selecting a feasible switching period for the fault detection feature in the DYRE structure. Thus, the detection capabilities of DYRE remain active with a controlled cost in terms of performance (by the added instructions), and the power consumed.

More in detail, the DYRE architecture increases the overall power consumption of the system from 4.55% (8 SPs) to 8.72% (32 SPs) with respect to the original design. This behavior can be explained considering that the additional structures (controller, multiplexers, and the comparator block) remain active, so consuming static power even when the DYRE architecture is inactive. However, for synthesis purposes, this cost might be reduced by including power optimization strategies. It should be noted that power optimization techniques were not used during the experiments.

4.3 Reliability analysis

The reliability of the DYRE architecture is estimated by determining the probability of correct operation, which depends on the number of available and fault-free SP and SSP modules. The proper execution of the system is obtained when all thread operations are performed without failures affecting the execution cores. This probability of correct operations can be complemented and expressed as the probability of failure (when some SPs or SSPs fails). The dual-modules feature of the DYRE architecture influences the reliability calculation and the number of cumulative faults affecting SPs or SSPs before the overall architecture produces a failure.

During fault-free operations, both groups of SP and SSP modules are identical and operate in parallel independently among them. Considering this scenario, the probability of correct operation of the DYRE architecture (R_{DYRE}) can be computed by adopting a binomial distribution function using n SPs and m SSPs module, respectively. R_{DYRE} is composed of the probability of a fault in an SP($P_{\mathrm{core}(t)}$) at a given time t and a K limit related to the active operational features (mitigation and detection), as reported in Eq. 1.

$$R_{\mathrm{DYRE}} = \sum_{i=0}^{k} \left(\frac{n+m}{i} \right) \left[P_{\mathrm{core}(t))} \right]^{n+m-i} \left[1 - P_{\mathrm{core}(t))} \right]^{i} \tag{1}$$

In detail, when the fault mitigation feature is active, a failure in the overall system occurs when $k = (m + 1)$ execution units (SP or SSPs) are faulty, hence it is not possible to complete the thread operations without errors. However, if both features are active, the system produces a failure when $k = m$ execution unit fails since one SSP is used as a comparator during the in-field fault detection. Finally, in case the fault detection feature is enabled or both detection and mitigation features are disabled, there are no available SSPs dedicated to fault mitigation, therefore $k = 0$ and $m = 0$.

In order to determine the advantage, in terms of probability of correct operation, for the SM using the DYRE architecture, we introduce Eq. 2. Equation 2 is composed of two terms. The first term corresponds to the probability of correct operation of the SM without the DYRE architecture ($P_{SM(t)}$). In contrast, the second term represents the improved probability of correct operation by adopting the DYRE architecture ($\triangle R_{\mathrm{DYRE}}$). This term also includes the probabilities of correct operation for the switching modules ($P_{sw(t)}$) and the controller ($P_{c(t)}$).

$$\begin{aligned} R_{\mathrm{DYRE}} = & \sum_{i=0}^{0} \left(\frac{n}{i} \right) \left[P_{\mathrm{core}(t))} \right]^{n-i} \left[1 - P_{\mathrm{core}(t))} \right]^{i} \\ & + \sum_{i=1}^{k} \left(\frac{n+m}{i} \right) \left[P_{\mathrm{core}(t))} \right]^{n+m-i} \left[1 - P_{\mathrm{core}(t))} \right]^{i} \left[P_{sw(t))} \right] \left[P_{c(t))} \right] \end{aligned} \tag{2}$$

As it can be noted in Eq. 2, the number of SSPs (m) determines the probability of correct operation in the GPGPU. The behavior of $\triangle R_{\mathrm{DYRE}}$ concerning the probability of correct operation on SPs ($P_{\mathrm{core}(t)}$) is plotted in Fig. 5. The graph describes the relationship between $P_{\mathrm{core}(t)}$ and $\triangle R_{\mathrm{DYRE}}$ for multiple values of m. The almost stable behavior of about 20–40% of positive increment impacts $\triangle R_{\mathrm{DYRE}}$ when m increases and $P_{\mathrm{core}(t)}$ thoroughly decreases. Moreover, Fig. 5 reports the different benefits when selecting a limited number of additional SSPs (m). According to results and considering a probability of correct operation between 0.9 and 1.0, the

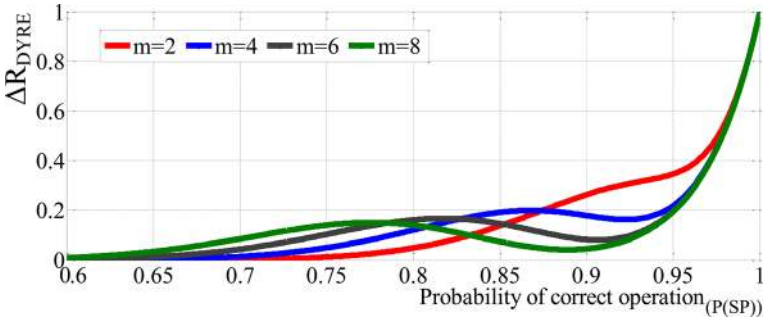

Fig. 5 Reliability benefit in the system for multiple probabilities of correct operation

best trade-off is observed when two additional SSPs are used in the DYRE structure. Thus, the reliability relation shown in Fig. 5 allows the design exploration of a potential DYRE composition.

Furthermore, the comparison between the reliability behavior of a standard GPGPU ($P_{SM(t)}$) and the one of an architecture adopting the two features of the DYRE architecture (*mitigation* only ($R1_{\mathrm{DYRE}}$) and *detection+mitigation* ($R2_{\mathrm{DYRE}}$)) is plotted in Fig. 6, using a typical probability function ($P_{\mathrm{core}(t)} = e^{-\alpha t}$) in both cases. This figure shows the reliability when adding two SSPs in the system. As it can be observed, the reliability of a DYRE GPGPU (R1 and R2) remains higher than without DYRE, so extending its operative life. A detailed analysis revealed that in some points the reliability is increased by up to 57%, when the mitigation and detection features are active, and 72% with the mitigation only feature.

Although the DYRE structure was validated using the FlexGripPlus model with the G80 architecture, we still claim that the proposed structure can be adopted into modern architectures of GPGPUs. DYRE targets the SP cores, which are also present in modern devices. Moreover, the implementation requires zero changes to the memory hierarchy and scheduling mechanisms. Finally, minimum effort is required to implement the custom control instructions. Furthermore, modern trends of GPGPU architectures include more SPs per SM, so increasing the volume of transistors in the device as the area and power consumption. However, as reported in results from Fig. 4, The DYRE structure requires a limited percentage of additional area, which follows a proportionally inverse relation with the number of SPs to harden. Thus, DYRE provides reliability benefits, and its cost drops when the number of SPs increases.

In contrast, DYRE may require special adaptations procedures (splitting or replication of structures) when the target hardening scope includes modules with different features, such as floating point units (FPUs), integer units (INT), and special function units (SFUs) or with different precision formats (i.e., 32 or 64 bits).

Finally, the DYRE structure employs active redundancy and hot sparing strategies to increase the fault-tolerance in the SP cores of GPGPU devices. A direct comparison with classical passive fault-tolerance strategies, such as DWC and TMR, can show that DYRE is less expensive in terms of hardware overhead and

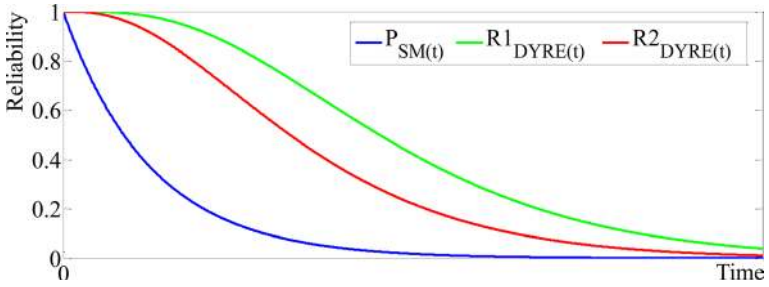

Fig. 6 Reliability comparison of a standard GPGPU and other using the two features of the DYRE architecture with two SSPs

power consumption. Moreover, DYRE can provide the main benefits of fault detection and fault mitigation simultaneously when activated.

5 Conclusions

We introduced an in-field dynamic architecture (DYRE) to detect and mitigate permanent faults affecting the execution units in GPGPUs. DYRE provides a solution that can be employed during the operative life of a GPGPU and extend the reliability capabilities by up to 57% for most configurations of the execution units of these devices. The proposed solution (targeting execution units, only) can be easily integrated by others targeting the remaining modules of a GPGPU.

The proposed strategy was implemented in a representative GPGPU, and the hardware and power overhead were measured. The results let us affirm that adding the proposed mechanism into a GPGPU design requires a minimum to moderate cost that directly depends on the number of additional cores included to support fault detection and mitigation.

As future works, we plan to extend the proposed mechanism, so exploring reliable architectures for in-field detection and mitigation of faults in other modules of GPGPU devices, including special function units (SFUs), controllers, and other unprotected structures. Moreover, the proposed architecture can also be adapted into other parallel architectures, so additional analyzes and evaluations can be performed as future activities.

Acknowledgements The European Commission has partially supported this work through the Horizon 2020 RESCUE-ETN Project under Grant 722325.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

References

1. Shi W, Alawieh MB, Li X, Yu H (2017) Algorithm and hardware implementation for visual perception system in autonomous vehicle: a survey. Integration 59:148–156. https://doi.org/10.1016/j.vlsi.2017.07.007
2. Gomez LB, Cappello F, Carro L, DeBardeleben N, Fang B, Gurumurthi S, Pattabiraman K, Rech P, Sonza Reorda M (2014) Gpgpus: how to combine high computational power with high reliability. In: 2014 Design, Automation Test in Europe Conference Exhibition (DATE), pp 1–9. https://doi.org/10.7873/DATE.2014.354

3. Hamdioui S, Gizopoulos D, Guido G, Nicolaidis M, Grasset A, Bonnot P (2013) Reliability challenges of real-time systems in forthcoming technology nodes. In: 2013 Design, Automation Test in Europe Conference Exhibition (DATE), pp 129–134. https://doi.org/10.7873/DATE.2013.040
4. Oliveira D, Blanchard S, DeBardeleben N, dos Santos F, Dávila GP, Navaux P, Favalli A, Schappert O, Wender S, Cazzaniga C, Frost C, Rech P (2021) Thermal neutrons: a possible threat for supercomputer reliability. J Supercomput 77:1612–1634. https://doi.org/10.1007/s11227-020-03324-9
5. Tiwari D, Gupta S, Rogers J, Maxwell D, Rech P, Vazhkudai S, Oliveira D, Londo D, DeBardeleben N, Navaux P, Carro L, Bland A (2015) Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp 331–342. https://doi.org/10.1109/HPCA.2015.7056044
6. Gizopoulos D, Psarakis M, Adve SV, Ramachandran P, Hari SKS, Sorin D, Meixner A, Biswas A, Vera X (2011) Architectures for online error detection and recovery in multicore processors. In: 2011 Design, Automation Test in Europe Conference Exhibition (DATE), pp 1–6. https://doi.org/10.1109/DATE.2011.5763096
7. Goncalves MM, Lamb IP, Rech P, Brum RM, Azambuja JR (2020) Improving selective fault tolerance in GPU register files by relaxing application accuracy. IEEE Trans Nucl Sci 67(7):1573–1580. https://doi.org/10.1109/TNS.2020.2982162
8. Rech P, Nazar GL, Frost C, Carro L (2014) GPUs reliability dependence on degree of parallelism. IEEE Trans Nucl Sci 61(4):1755–1762. https://doi.org/10.1109/TNS.2014.2303855
9. Laosooksathit S, Nassar R, Leangsuksun C, Paun M (2014) Reliability-aware performance model for optimal GPU-enabled cluster environment. J Supercomput 68(3):1630–1651. https://doi.org/10.1007/s11227-014-1128-7
10. Sartor AL, Lorenzon AF, Carro L, Kastensmidt F, Wong S, Beck ACS (2017) Exploiting idle hardware to provide low overhead fault tolerance for VLIW processors. J Emerg Technol Comput Syst. https://doi.org/10.1145/3001935
11. Dos Santos FF, Brandalero M, Sullivan M, Rech Junior RL, Martins Basso P, Hubner PM, Carro L, Rech P (2021) Reduced precision DWC: an efficient hardening strategy for mixed-precision architectures. IEEE Trans Comput. https://doi.org/10.1109/TC.2021.3058872
12. Lunardi C, Previlon F, Kaeli D, Rech P (2018) On the efficacy of ECC and the benefits of FinFET transistor layout for GPU reliability. IEEE Trans Nucl Sci 65(8):1843–1850. https://doi.org/10.1109/TNS.2018.2823786
13. de Oliveira B, Rodrigues GS, Kastensmidt FL, Added N, Macchione ELA, Aguiar VAP, Medina NH, Silveira MAG (2018) Lockstep dual-core arm a9: implementation and resilience analysis under heavy ion-induced soft errors. IEEE Trans Nucl Sci 65(8):1783–1790. https://doi.org/10.1109/TNS.2018.2852606
14. Constantinides K, Plaza S, Blome J, Zhang B, Bertacco V, Mahlke S, Austin T, Orshansky M (2006) Bulletproof: a defect-tolerant CMP switch architecture. In: The Twelfth International Symposium on High-Performance Computer Architecture (HPCA), 2006, pp 5–16. https://doi.org/10.1109/HPCA.2006.1598108
15. Sridharan V, Gurumurthi S (2015) Hardware based redundant multi-threading inside a GPU for improved reliability. US Patent No. 9,026,847
16. Baji T (2016) Nvidia AI driving platform and AI supercomputer Xavier. https://blogs.nvidia.com/blog/2016/09/28/xavier/. Accessed Feb 2021
17. Datla Jagannadha PK, Yilmaz M, Sonawane M, Chadalavada S, Sarangi S, Bhaskaran B, Bajpai S, Reddy VA, Pandey J, Jiang S (2019) Special session: in-system-test (IST) architecture for Nvidia drive-AGX platforms. In: 2019 IEEE 37th VLSI Test Symposium (VTS), pp 1–8. https://doi.org/10.1109/VTS.2019.8758636
18. Alcaide Portet S, Kosmidis L, Hernandez C, Abella J (2020) Software-only triple diverse redundancy on GPUs for autonomous driving platforms. In: 2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), pp 82–88. https://doi.org/10.1109/DSN-S50200.2020.00045
19. Luick DA (2008) Multiple parallel pipeline processor having self-repairing capability. US Patent No. 7,454,654 B2
20. Chattopadhyay A (2013) Ingredients of adaptability: a survey of reconfigurable processors. VLSI Des 2013:1–18. https://doi.org/10.1155/2013/683615

21. Ilyoung Kim, Zorian Y, Komoriya G, Pham H, Higgins FP, Lewandowski JL (1998) Built in self repair for embedded high density SRAM. In: Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270), pp 1112–1119. https://doi.org/10.1109/TEST.1998.743312
22. Koal T, Vierhaus HT (2010) 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 (DDECS), pp 40–43. https://doi.org/10.1109/DDECS.2010.5491821
23. Goncalves MM, Condia JER, Sonza Reorda M, Sterpone L, Azambuja J (2020) Improving GPU register file reliability with a comprehensive ISA extension. Microelectron Reliab 114:113768. https://doi.org/10.1016/j.microrel.2020.113768 (**31st European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ESREF 2020**)
24. Lindoso A, Entrena L, Garca-Valderas M, Parra L (2017) A hybrid fault-tolerant LEON3 soft core processor implemented in low-end SRAM FPGA. IEEE Trans Nucl Sci 64(1):374–381. https://doi.org/10.1109/TNS.2016.2636574
25. Lyu MR et al (1996) Handbook of software reliability engineering, vol 222. IEEE Computer Society Press, Los Alamitos, CA
26. Wilson C, Sabogal S, George A, Gordon-Ross A (2017) Hybrid, adaptive, and reconfigurable fault tolerance. In: 2017 IEEE Aerospace Conference, pp 1–11. https://doi.org/10.1109/AERO.2017.7943867
27. Sorensen SD, Sogaard S (2012) Failure detection and mitigation in logic circuits. US Patent No. 8,117,512B2
28. Condia JER, Narducci P, Sonza Reorda M, Sterpone L (2020) A dynamic hardware redundancy mechanism for the in-field fault detection in cores of GPGPUs. In: 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp 1–6. https://doi.org/10.1109/DDECS50862.2020.9095665
29. Nickolls JR (2005) Defect tolerant redundancy. US Patent No. 6,879,207B1
30. Condia JER, Narducci P, Sonza Reorda M, Sterpone L (2020) A dynamic reconfiguration mechanism to increase the reliability of GPGPUs. In: 2020 IEEE 38th VLSI Test Symposium (VTS), pp 1–6. https://doi.org/10.1109/VTS48691.2020.9107572
31. Mukherjee S (2011) Architecture design for soft errors. Morgan Kaufmann, Burlington
32. Condia JER, Du B, Sonza Reorda M, Sterpone L (2020) Flexgripplus: an improved GPGPU model to support reliability analysis. Microelectron Reliab 109:113660. https://doi.org/10.1016/j.microrel.2020.113660
33. Andryc K, Merchant M, Tessier R (2013) Flexgrip: a soft GPGPU for FPGAS. In: 2013 International Conference on Field-Programmable Technology (FPT), pp 230–237. https://doi.org/10.1109/FPT.2013.6718358
34. Di Carlo S, Gambardella G, Indaco M, Martella I, Prinetto P, Rolfo D, Trotta P (2013) A software-based self test of CUDA fermi GPUs. In: 2013 18th IEEE European Test Symposium (ETS), pp 1–6. https://doi.org/10.1109/ETS.2013.6569353