Analyzing the Influence of Memory and Workload on the Reliability of GPUs Under Neutron Radiation

German Leon, Jose M. Badia, Jose A. Belloch, *Member, IEEE*, Mario Garcia-Valderas, *Member, IEEE*, Almudena Lindoso, *Senior Member, IEEE*, and Luis Entrena, *Member, IEEE*

***Abstract*— Evaluating the impact of utilizing different GPU resources is crucial for gaining insights into the reliability of GPUs when exposed to radiation. In this study, we employed various versions of a microbenchmark to investigate the effect of different memory types on the performance of a low-power GPU integrated into the Tegra X1 (TX1) system on a chip (SoC) of a Jetson Nano board. Additionally, we explored the tradeoff between enhanced computational performance and the occurrence of failures over time by optimizing the utilization of GPU resources. Our findings demonstrate that maximizing the utilization of the device's cores enables the completion of a greater number of computations without errors. By fully harnessing the computational potential of the GPU cores, we effectively increase the work that we can complete between failures. Moreover, we observed that the use of the different memory types has a significant influence on the overall reliability of the GPU. The outcomes of this research contribute to a comprehensive understanding of the interplay between GPU resources, irradiation effects, and reliability. This knowledge is instrumental in guiding the development of robust GPUs for applications in radiation-prone environments.**

***Index Terms*— Fault tolerance, GPU, microbenchmark, neutron, radiation, soft error.**

I. Introduction

MODERN GPUs have become increasingly complex and powerful devices, employing technologies with large-scale integration. They are utilized across a wide range of applications, each with varying requirements in terms of size, price, reliability, computational performance, and power consumption. High-performance GPUs consist of thousands of cores, encompassing integer and floating-point computing units with varying precision, along with multiple control units like warp schedulers and instruction dispatchers. These devices also feature a very large number of registers and multiple levels of memory with differing performance characteristics. With the rise of neural network applications, GPUs have even started incorporating specialized components such as tensor cores [1]. This heightened complexity and diversity of components pose significant challenges in analyzing the reliability of GPUs and identifying the sources of potential errors. These errors can arise from various factors, including manufacturing defects, voltage scaling, hardware wear-out, or radiation effects [2], [3]. In data centers, which increasingly deploy high-end GPUs in their nodes, these errors have become more prevalent [4], [5]. Furthermore, safety-critical environments such as space missions [6], autonomous driving [7], [8], and railway signaling [9] have also incorporated SoCs with increasingly powerful GPUs [10]. Understanding and addressing the reliability challenges posed by these errors is crucial for ensuring the robustness and dependability of GPUs in diverse applications and demanding environments.

This article focuses on the investigation of soft errors induced by terrestrial radiation, which refers to transient errors that occur in the processing logic and memory components of GPUs. Accelerated radiation is a widely used method for analyzing the reliability of such devices. However, pinpointing the software and hardware sources of radiation-induced errors presents significant challenges [11]. Microbenchmarks that stress specific GPU components offer a promising solution for narrowing down these sources [12], [13]. For example, they allow us to test the behavior of different types of memories, such as registers, L1 cache, or L2 cache, or to evaluate the fault tolerance of using one resource of the GPU (i.e., floating point unit and LD/ST unit) when repeatedly executing some assembly instructions. We have designed a simple microbenchmark that iteratively performs different arithmetic operations per iteration and thread. We have used CUDA to implement it [14]. One of our goals is to increase the number of instructions executed per cycle in order to stress the arithmetic and control units of the GPU. Besides, by modifying the size of the grid and thread block, we can systematically change the computational load of the cores, warp schedulers, dispatch units, and other control components of the GPU. Additionally, we have implemented three versions of each microbenchmark that differ in the type of memory used to store the result of the operations on each thread. Namely,

Manuscript received 11 January 2024; revised 15 February 2024; accepted 27 March 2024. Date of publication 11 April 2024; date of current version 16 August 2024. This work was supported in part by the University Jaume I under Project UJIB2019-36; in part by MCIN/AEI/10.13039/501100011033; in part by ERDF a way of making Europe through the Spanish Ministry of Science and Innovation under Project PID2020-113656RB-C21 and Project PID2022-138696OB-C21; and in part by the Regional Government of Madrid under Project MIMACUHSPACE-CM-UC3M. *(Corresponding author: Jose M. Badia.)*

German Leon and Jose M. Badia are with the Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, 12071 Castellón de la Plana Castellón, Spain (e-mail: leon@uji.es; badia@uji.es@uji.es).

Jose A. Belloch, Mario Garcia-Valderas, Almudena Lindoso, and Luis Entrena are with the Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, 28911 Leganés, Spain (e-mail: jbelloc@ing.uc3m.es; mario.garcia@uc3m.es; alindoso@ing.uc3m.es).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNS.2024.3387490.

Digital Object Identifier 10.1109/TNS.2024.3387490

we have used registers, shared memory, and global memory to store the result. This way, we can also assess the effect of using different types of memories in the error rate of the microbenchmark. The microbenchmark was also designed to facilitate the task of identifying errors in the result on each thread due to radiation. To perform the radiation experiments and facilitate the analysis of the results, we used a small low-power GPU integrated into the Tegra X1 (TX1) system on chip (SoC), which is part of the NVIDIA Jetson Nano platform [15]. By using this experimental setup, we aimed to gain insight into the behavior and vulnerabilities of low-power GPUs under radiation, contributing to a better understanding of their reliability in radiation-sensitive environments.

The main contributions of the article can be summarized as follows.

- We assess the soft error reliability of the GPU integrated into a low-cost and low-power SoC, specifically the TX1 SoC featured in the NVIDIA Jetson Nano board. Platforms of this nature prove highly valuable in environments with constraints on power, weight, and size, as seen in applications like space missions and other embedded systems, which are increasingly prevalent in the IoT era. Therefore, it is important to assess the soft error reliability of this kind of platform, which has been less studied that other discrete and higher performance GPUs not included in SoC.
- We design and implement a simple microbenchmark using CUDA that can run a high number of instructions per cycle (IPC). Besides, it allows us to easily increase the utilization of the main resources of the GPU by changing the grid and thread block sizes. This allows us to assess the relation between the observed error rates and the use of the resources of the GPU including, for example, cores, warp schedulers, dispatch units, and other control components.
- We analyze the effect of using the different memories of the GPU in the error rates. In particular, we compare the effect of using registers, shared memory, and global memory and, in some cases, the cache memories. To do this, we have developed three versions of the microbenchmark that stress the use of each particular type of memory.
- We leverage the profiling metrics of the GPU to examine the relationship between resource utilization, computational performance, and the reliability of the different versions of the microbenchmark. Results show a strong correlation between the SDC failures and the performance of the code.

Experimental results show that maximizing core utilization reduces reliability issues caused by radiation and that the choice of memory type has a significant impact on GPU reliability.

The use of a low-power GPU with only one streaming multiprocessor (SM) simplifies the experiments and results analysis, as all thread blocks are executed in the same SM. However, similar results and conclusions have been obtained using other benchmarks and microbenchmarks to stress the resources of GPUs with more SMs. For example, Rech et al. [16], use matrix multiplication, Fast Fourier Transform, and two synthetic benchmarks where each thread performs floating point sums or multiplications. They test the codes on two high-performance GPUs with 14 and 15 SMs. They study the effect of varying the grid and thread block size and find that increasing the degree of parallelism, and thus the resource utilization, increases the cross section. However, optimizing core utilization can compensate for this increased error rate and allow more work to be completed without errors.

The remaining sections of the article are organized as follows. In Section II, we provide an overview of related work in the field of GPU reliability, including studies on microbenchmarking and the utilization of profiling metrics. Section III presents detailed descriptions of the device under test (DUT) and the experimental environment used in our study. In Section IV, we present and discuss the experimental results obtained from our investigations. Finally, in Section V, we summarize the main conclusions drawn from our research.

II. Related Work

Over the past decade, there has been extensive research conducted on the radiation reliability of various types of GPUs. The majority of these studies focus on high-performance GPUs that consume substantial power, as indicated by previous works such as [17], [18], [19]. However, fewer studies have explored the behavior of low-power GPUs integrated into commercial off-the-shelf (COTS) SoCs, as highlighted by the works showed in [20], [21], and [22]. To the best of our knowledge, experiments using radiation with our DUT, which comprises a Jetson Nano board, have only been published twice. Slater et al. [23] tested it using gamma ray photons to assess its suitability for space missions by determining its tolerance to radiation using total ionizing dose (TID). Serrano-Cases et al. [24], show that using redundant kernel execution is an effective way of reducing the SDC cross section of the Jetson Nano when exposed to proton irradiation. They also show that the CPU in the SoC is the main source of the functional interrupts detected.

A wide range of benchmarks has been employed to assess the radiation reliability of GPUs. Among these, the matrix product benchmark stands out as one of the most commonly used methods, as evidenced by studies such as the ones included in [16] and [25]. Additionally, various other codes, both memory-bound and compute-bound, with distinct characteristics in terms of GPU resource utilization, have been utilized, as highlighted in works like [13], [18], [21]. Furthermore, in recent years, numerous studies have focused on evaluating the fault tolerance of neural networks on such devices, as demonstrated by research included in [26] and [27].

Sometimes, it is essential to evaluate the reliability of specific components within a device, such as memory, arithmetic units, load/store units, and others. In such cases, it is advantageous to implement simplified benchmarks that solely utilize the specific component of the architecture that we wish to assess. This approach allows for a more focused evaluation of the targeted component's reliability and performance.

For instance, in the studies showed in [4] and [28], neutrons were employed to evaluate the neutron sensitivity of GPU L2 caches and register files. These memory structures are responsible for the majority of failures in modern GPUs. The researchers focused on two different GPU architectures: Fermi (40 nm technology) and Kepler (28 nm technology). Special attention was given to the occurrence of multiple bit and multiple cell upsets. The experiments revealed that the employment of the ECC mechanism, capable of detecting double bit errors and correcting single bit errors, proved sufficient to detect and correct all observed errors. Additional experiments were documented in [17], wherein the same architectures were tested, including the L1 cache memory of the GPU. The results confirmed that the GPU with the Kepler architecture, which is newer than the GPU with the Fermi architecture, exhibited improved memory reliability. Moreover, the reliability of the GPU's arithmetic units was also evaluated under radiation in [16]. The authors utilize two microbenchmarks to assess the behavior of two GPUs during repeated execution of floating-point sums and multiplications. The degree of parallelism was varied by adjusting the number of threads while increasing the number of operations per thread.

At a lower level, exploring the behavior of specific assembler instructions under neutron radiation can provide valuable insights. The study showed in Hari et al. [12], utilized both error injection and radiation techniques to evaluate seven commonly used low-level assembly SASS instructions executed by workloads from the Rodinia benchmark suite [29]. These instructions encompass various operations, including integer and floating-point operations (IADD, FADD, IMAD, and FFMA), shared memory load (LDS), branching (BRA), and the ISETP instruction, which performs a comparison and stores the result in a predicate register. For each microbenchmark, the authors implemented a CUDA kernel that repeatedly executed the targeted instruction. The experiments were conducted on an NVIDIA K40 Tesla GPU with the Kepler architecture. A similar methodology, combining injection and radiation experiments, was employed in the study presented in [13] to evaluate the silent data corruption (SDC) and detected unrecoverable error (DUE) FIT rates of six low-level assembler instructions. Additionally, the authors implemented a microbenchmark to evaluate the FIT rate of the Register File storage. The experiments were conducted on two GPUs with Kepler and Volta architectures.

Profiling techniques have been employed by researchers to obtain behavior and performance metrics of different codes, enabling the exploration of their relationship with reliability. In the study showed in [13], various higher level benchmarks were used to assess the reliability of GPUs. To gain insights into the contributions of low-level instructions within these benchmarks, the authors utilize profilers such as `nvprof` and `nsight-compute` [30]. By profiling these codes, they were able to analyze the performance characteristics of individual instructions. Additionally, the authors proposed a model to estimate the FIT rate of different codes. This model leverages two well-known performance metrics provided by the profilers: IPC and Achieved Occupancy of the GPU. In the subsequent sections, we will also employ these metrics to evaluate how our microbenchmark utilizes the resources of the GPU.

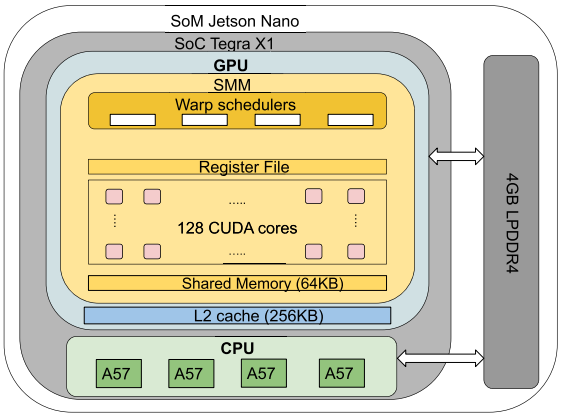

Fig. 1. DUT. TX1 SoC included in a Jetson Nano SoM.

Finally, in the study presented in Topçu and Öz [31], the authors put forward a machine learning methodology to predict the vulnerability of GPU applications to soft errors. They employed regression and classification models to forecast the rates of masked faults, SDC, and crashes for various benchmarks. These predictions were based on metrics collected through simulation and the profiling tool `nsight-compute`. The classification models achieved impressive maximum prediction accuracy rates ranging from 82.6% to 96.6%, depending on the specific type of error.

III. Experimental Environment and Methodology

A. Device Under Test

The NVIDIA TX1 SoC was used as the DUT in this study [15]. This SoC is built using 20 nm planar technology and consists of a quad-core ARM Cortex-A57 CPU that implements the Armv8-A architecture. Each core is equipped with a 48 KiB L1 instruction cache, a 32 KiB L1 data cache, and a 2 MiB L2 unified cache that is shared by all cores. Additionally, the cores feature the advanced SIMD NEON extension that supports vector operations with both integer and floating point elements.

The TX1 SoC also includes an NVIDIA Maxwell GPU with 1 symmetric multiprocessor (SM) that is partitioned into four processing blocks, each with its own instruction buffer, warp scheduler, and 32 CUDA cores. The GPU has a 256 KiB L2 cache, 64 KiB *shared-memory*, and 64 K (32-bit) *registers*. While the L2 cache of the CPU supports error-correcting codes (ECCs) and they were enabled during all our experiments, GPU memories does not support this mechanism.

To support the CPU and GPU, the TX1 SoC is integrated into an NVIDIA Jetson Nano system-on-module (SoM) that also includes 4 GiB of external DDR4 memory. This memory is shared by both the CPU and GPU. Fig. 1 shows the main components of the DUT.

Fig. 2. Setup of the experiments showing the six irradiated Jetson-Nano boards.

B. Setup and Procedure

The beam was focused on a spot with a diameter of 2 in plus 1 in of penumbra, which provided uniform irradiation of the GPU chip without directly affecting nearby board power control circuitry and DDR memories. Even if the beam is collimated, scattering neutron may be found outside the beam spot.

The experiments were conducted in November 2022 at the ChipIR facility of the Rutherford Appleton Laboratory in the U.K., where a beam of neutrons with an energy spectrum similar to atmospheric radiation was used [32]. The average neutron flux was maintained at approximately 3.14×10^6 n/(cm$^2 \cdot$s), and the total flux during the irradiation campaign was 2.28×10^{11} n/cm^2. Six Jetson boards, including the TX1 SoC under test, were irradiated as shown in Fig. 2. The entire SoC was affected by the irradiation. That includes the four CPU cores and the GPU with its shared, L1, and L2 cache memories. The beam spot was focused on the SoC and the position of the board was chosen to reduce the effect of radiation on the DDR memory. However, even when the beam is collimated, scattered neutrons can be found outside the beam spot and can also affect this type of external memory, which contains the operating system and all the files with our codes and results. All experiments were conducted at room temperature.

The Jetson Nano Development Kit comes with a heat sink attached to the SoM, which provides effective cooling for high-performance computing, and we added two fans to improve heat dissipation. We followed the recommendations in [33] and monitored the temperature and power consumption of the units throughout the experiments by accessing the internal sensors provided by the board.

In our setup, the six Jetson Nano boards sent test logs to a host controller via their serial communications port. The host controller was located outside the shielded irradiation room and was not affected by the radiation. The controller was also connected to the GPIO pins of the DUTs, allowing for remote reset of each Jetson Nano board if the system hung. The entire test was managed remotely from a laptop connected to the host controller via Ethernet. We used power over ethernet (PoE) technology to provide power and access to the boards over the same Ethernet cable. Fig. 2 illustrates the radiation test setup used during the experiments. We ran the Ubuntu 20.04 operating system with the CUDA 10.2 driver from an external memory, which was connected to each Jetson Nano board with a 30 cm cable and protected under several paraffin blocks to reduce the effects of radiation on the operating system and application files.

The microbenchmarks were implemented in C, and we used Python scripts to run and test them. We used the Python module `Pexpect` to spawn and control a subprocess responsible for executing each microbenchmark. We included three watchdogs to detect and recover from various hangs of the tests and the operating system. The first timeout, associated with the spawned process, was set to a time greater than the maximum expected duration of each microbenchmark. The second watchdog used the watchdog Linux API to reboot the system if it hung for more than 20 s. Finally, a watchdog was implemented on the host controller to reset the device if the Jetson system hung and did not send a log result for more than 25 s.

C. Programming Model and Microbenchmarks

The compute unified device architecture (CUDA), defined by NVIDIA, is based on an array of SMs, each of which contains multiple CUDA cores that can execute multiple threads in parallel [14]. Threads are logically grouped into thread blocks, which are dispatched to an SM and can utilize its *shared-memory*. Thread blocks are further divided into "warps" of 32 threads, which are scheduled to execute on the cores of the SM. Thread blocks are organized in a grid. CUDA programs combine host code, run on the CPU, with one or more kernel functions that are executed on the CUDA cores using a single instruction multiple threads (SIMTs) model.

Since the DUT's GPU contains only one SM, the defined thread blocks will execute sequentially. Each thread block will be divided into warps of 32 threads, which will execute the same instructions in lockstep, using the 128 available cores in parallel. Starting with the Volta architecture, newer than the Maxwell architecture of the GPU of the DUT, NVIDIA introduced a feature called "thread-level warp," that allows threads within a warp to execute instructions independently, rather that strictly in lockstep, thus improving the efficiency of the GPU by reducing the impact of divergent branching within warps. However, the entire warp still executes in lockstep when it comes to certain operations, such as arithmetic instructions. The GPU's occupancy, use of instruction dispatchers and warp schedulers, and the number of instructions executed per cycle will depend on various factors, such as the number and size of thread blocks, the code executed by each thread, and the utilization of computational resources, such as registers and memories, available on each core or on the GPU.

```
1: void micro(int num_iter, volatile int *global) {
2: int out, in~= mem = thread_id;
3: #pragma unroll 2
4: for (int iter=0; iter<num_iter; ++iter) {
5: out = in*mem + thread_id;
6: mem = (out-mem)/in;
7: }
8: }
```

Listing 1. Microbenchmark kernel executed by every thread in the GPU.

To assess the radiation reliability of the DUT, we use a simple microbenchmark run by each of the threads using the GPU cores. As can be seen in the Listing 1, each thread performs a given number of iterations in which it performs four basic arithmetic operations on integers. We have chosen to use the integer data type since it allows us to more easily achieve a higher number of IPC. We leverage both instruction-level parallelism (ILP) and thread-level parallelism (TLP) to increase the IPC and also the occupancy of the GPU [34]. We have used `#pragma unroll 2` to unroll the loop, increase the instructions that can be dispatched, and better hide the memory access latency. Results show that this unrolling increases the IPC of the code.

The microbenchmark basically uses three variables that it stores in registers local to each thread (`in`, `out` and `thread_id`). To study the effect of using different types of memory, we use an additional variable `mem`. In the register-only version of the kernel, this variable is stored locally in one additional register per thread. In the version that uses *shared memory*, the variable is stored in a dynamically reserved block of *shared-memory* when the kernel executes. Each thread accesses a different *shared-memory* location to work with its variable `mem`. Finally, in the version that uses *global-memory*, each thread accesses that variable in a different location in *global-memory* stored out of the GPU, shared with the CPU and passed as the second argument of the function. The microbenchmark version utilizing *global-memory* incorporates an extra parameter pertaining to the global memory employed by the threads. To prevent the compiler from optimizing the code by storing data in a register, we employ the `volatile` keyword. This ensures that, during each iteration, each thread reads the data directly from its original global memory location. In the case of the version employing *shared memory,* we similarly use the `volatile` keyword when declaring the shared memory vector responsible for storing data used by each thread.

We have designed the microbenchmark to facilitate the detection of errors during its execution. If everything has worked correctly, at the end of the execution, each thread should store in its variable `mem` its unique global index (`thread_id`). Therefore, we can easily detect whether an error has occurred due to radiation and the number of threads that have been affected by it by counting the number of threads for which the above condition is not met. The microbenchmark code is publicly available at https://github.com/josembadia/microbench.

IV. Experimental Results and Discussion

To investigate the impact of computational load on the reliability of the DUT, we conducted three versions of the microbenchmark using various grid sizes (representing the number of thread blocks) and varied the number of threads per block. The test configurations were denoted by the notation `gxby`, where `x` indicated the grid size (i.e., the number of thread blocks) and `y` represented the number of threads per block. For instance, an example configuration would be `g32b64`, which employed a grid consisting of 32 thread blocks, each containing 64 threads, resulting in a total of 2048 threads.

The number of iterations run by the microbenchmark in each variant was adapted so that the duration of all iterations was 2 s. Thus, each type of experiment was subjected to the same amount of radiation. Each variant was run continuously under irradiation until a total of 100 errors of any type were counted. In our experimental evaluation, we distinguish two main types of detected errors due to radiation.

- *Silent Data Corruption:* The result of the microbenchmark is not correct, that is, at least one thread finished with a value different that its global thread index in its variable `mem`. The radiation has affected the result of one or more threads.
- *Detected Unrecoverable Error:* The program crashes for example due to an exception while accessing outside its memory segment, or the system hangs and produces a system reboot.

A. Computation Performance and Profiling Metrics

The different variants examined use more or less GPU components with varying degrees of intensity. The use of GPU computing resources and memory can be assessed using the `nvprof` profiler. Table I contains two of the most important performance metrics provided by the profiler. These are the average number of IPC and the GPU occupancy, which is the ratio of active warps to the maximum number supported by the SM. Three of the execution variants (`g1b32`, `g1b128`, and `g1b1024`) use a single block of threads and contain 32, 128, and 1024 threads per block, respectively. In the case of 32 threads, we only use one of the four GPU core blocks and only one warp scheduler and two instruction dispatchers. This means that in the best case, using registers, the IPC in the four core blocks is only 0.25 and the utilization is $1/64 = 1.56\%$, as the maximum number of active warps in the DUT is 64. In the case of 128 threads, we can use all the cores of the GPU, but in practice, we only maintain an average of 0.96 IPC. It is well known that in order to hide the latency between dependent instructions, it is necessary to keep many more warps running than the four running at any given time in the DUT [35]. By using a block with 1024 threads, we double the number of instructions executed per cycle (2.04). In order to maximize the workload on the core components of the GPU, we use two additional variants (`g32b64` and `g64b32`) that increase the number of threads to 2048, which is the maximum allowed by SM in the GPU architecture of our DUT. These two variants make it possible to increase the average number of instructions executed per cycle to more than 4.

Fig. 3 allows us to evaluate the computational performance of each of the microbenchmark variants used. We can clearly see how the number of arithmetic operations with integers increases when we modify the size of the grid and of the thread blocks. This happens independently of the type of memory used. The increase correlates with the behavior of metrics such as the average number of IPC, the issue instruction slot utilization, and the average number of eligible warps per cycle, among others. Obviously, using more efficiently the resources

TABLE I
PERFORMANCE METRICS OF THE DIFFERENT VARIANTS OF THE MICROBENCHMARK. TBS REFERS TO THE THREAD BLOCK SIZE IN CUDA

			Occupancy (%)			IPC		
	grid	tbs	reg	shm	glb	reg	shm	glb
g1b32	1	32	1.56	1.56	4.49	0.25	0,23	0.04
g1b128	1	128	6.25	6.25	6.25	0.96	0,90	0.07
g1b1024	1	1024	50	50	50	2.04	2,06	0.17
g64b32	64	32	50	50	50	4.07	4,12	0.17
g32b64	32	64	100	100	100	4.08	4.13	0.41

of the GPU increases the computational performance of the code. The figure also allows us to assess the effect of using different types of memory in the microbenchmark. It is clear that continuous access to *global-memory* slows down code execution enormously. This can also be seen by comparing the value of the IPC metric in the Table I when using different types of memory.

Table II shows the number of memory transactions for the three versions of the microbenchmark with four types of memory: global, GPU L2 cache, GPU unified L1/Texture cache, and GPU shared memory. The results are shown with three of the variants of the test with an increasing number of threads. The three versions of the algorithm use a similar quantity of registers per thread and use the L2 cache. Specifically, the *shared memory* version uses 19 registers per thread, the *global memory* version uses 20 registers per thread, and the *registers* version uses 18 registers per thread. Note that in the last version, only one of these registers is used to store the variable `mem`, while the rest are used to store other local variables or to compute memory addresses. The *shared memory* version is the only one to use this kind memory. The metrics show that accessing the microbenchmark variable `mem` in global memory significantly increases the number of global memory transactions (`glb_trans`) and also the use of the both cache memory levels of the GPU, that is the L2 cache (`l2_read_trans`) and the unified L1/Texture cache (`tex_cache_trans`). As a matter of fact, the microbenchmark using *global memory* is the only one using the unified cache memory of the GPU and increases its use with the number of threads. The `global_hit_rate` metric reflects that it has a 50% hit rate for global loads in unified cache. The performance using *registers* and *shared memory* is quite similar. Their use of the L2 cache is similar and the *shared memory* transactions (`shm_ld_trans`) only slightly reduce the operations per second. We can also see that when we use global memory, both global and shared memory transactions increase with the number of threads, while these metrics remain constant for the versions of the microbenchmark that use registers and shared memory. Obviously, the only version that increases shared memory transactions with the number of threads is the one that uses this type of memory.

B. DUT Reliability

First of all, it is important to point out that radiation affects not only the GPU running the microbenchmark, but the entire SoC, including the CPU and its associated cache memory.

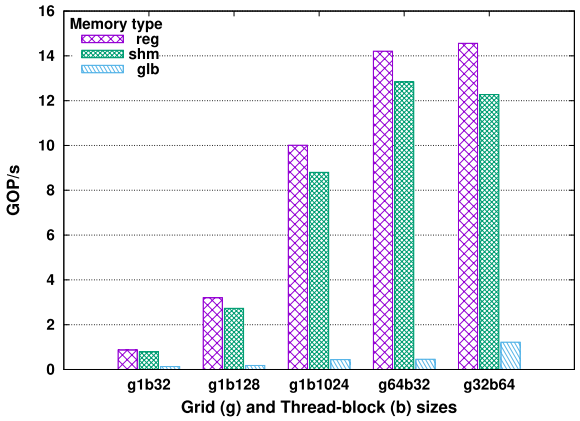

Fig. 3. Effect of the memory type and computational load in the performance of the microbenchmark measured as billions of integer arithmetic operations per second (GOP/s).

Radiation can also affect the process in charge of launching the kernel to the GPU, collecting the results of the different threads and checking their correctness, as well as all the operating system processes that may be running during the tests. Although the GPU is running the microbenchmark most of the time during the tests, some of the errors detected may be due to the effect of radiation on the CPU and the processes it runs.

Second, it is worth remembering that although the overall memory (DDR) of the device is less affected by radiation, the CPU and GPU cache memories are included in the focus of the beam. As we can see in the Table, there are a large number of transactions with the L2 cache memory of the GPU and this number is especially high in the case of the version of the microbenchmark based on global memory, since the variable accessed by each thread in the different iterations is temporarily stored in this type of cache memory. Thus, the effect of radiation on this type of cache may particularly affect this version of the microbenchmark.

The modification of the grid and thread block size influences not only the computational performance obtained by the microbenchmark, but also the number of failures produced. This is because it will increase the area used to perform the calculations and the stress to which the different components of the platform susceptible to particle impact are subjected.

To assess the radiation reliability, we ran more than 33 000 executions of the different variants of the microbenchmark, exposing the Nano boards to a total fluence of 2.28×10^{11} n/cm^2 for more than 21 h of irradiation. We use the failures in time (FIT) metric to evaluate the radiation sensitivity of the SoC while running the microbenchmark. This metric is the number of failures detected in 10^9 h having into account that the terrestrial radiation flux is 13 n/(cm$^2 \cdot$ h) [36].

Fig. 4 depicts the SDC and DUE FIT rates obtained from various variants of the microbenchmark run. The figure includes the confidence intervals computed with a confidence of 95%. Unfortunately, due to time constraints during the radiation campaign, sufficient data could not be obtained for three specific variants: `g1b32` with *shared-memory* and

TABLE II
MEMORY USE METRICS

	g1b32			g1b1024			g32b64		
	reg	shm	glb	reg	shm	glb	reg	shm	glb
glb_trans	2	2	6.79×10^7	2	2	2.57×10^8	2	2	6.31×10^8
l2_read_trans	5.49×10^6	5.50×10^6	2.68×10^7	5.50×10^6	5.50×10^6	8.23×10^7	5.50×10^6	5.49×10^6	$1,99 \times 10^8$
shm_ld_tr	0	2.46×10^7	0	0	2.17×10^8	0	0	4.35×10^8	0
tex_cache_trans	0	0	1.70×10^7	0	0	6.45×10^7	0	0	1.58×10^8
global_hit_rate	0%	0%	50%	0%	0%	50%	0%	0%	50%

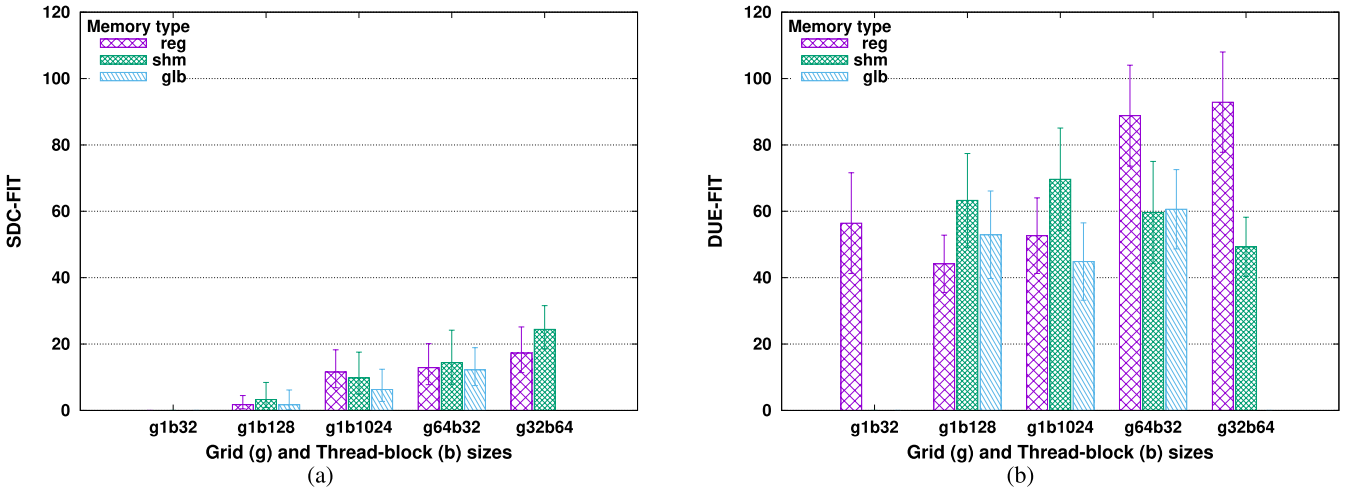

Fig. 4. Effect of the memory type and computational load in the FIT of the DUT. (a) SDC_FIT rate. (b) DUE_FIT rate.

global-memory, as well as `g32b64` with *global-memory*. Moreover, as the number of events of each type that we were able to accumulate for each test is small, the level of confidence in the results is not very high. However, it is of sufficient quality to allow us to analyze the behavior of the algorithms.

Comparing the FIT rates of both error types, it becomes evident that the SDC rate is significantly lower than the DUE rate. Neutron radiation primarily results in program crashes, system hangs, and reboots, rather than producing incorrect results in the code executed by the GPU cores. To illustrate, the `g1b32` test using registers experienced no SDC errors, but the code crashed 27 times, requiring 26 system reboots due to system hangs. Furthermore, it can be observed that SDC errors increase with the use of GPU resources across all three memory types. That is, as we increase the occupancy, IPC and other performance metrics, and so the code performance, the SDC FIT rate increases. This can be due to the increase in the number of cores being used or the number of instructions they execute per second and so the use of the instruction scheduler. In the last two variants (`g64b32` and `g32b64`), we are also increasing the number of warps and so the use of the warp schedulers.

The behavior of the DUE rates is quite different. This type of error only increases when we increase the use of GPU resources in the microbenchmark that does not use global or shared memory, except in the case of 32 threads. On the contrary, the DUE rates of the other two versions of the microbenchmark does not seem to depend on the size of the grid or the thread blocks, even if the use of the GPU's computational and memory resources is quite different, as we can see in Tables I and II.

The behavior of both types of errors seems to confirm that the DUE FIT can arise from errors that affect the instruction cache or the code itself, but most of them are dependent solely on hardware features such as control logic, synchronization, and interfaces [21]. On the other hand, the SDC FIT rate depends on how the device's resources are utilized and the architecture vulnerability factor (AVF), which is the probability that a fault in the architecture will propagate to the application's result [37].

We have used the Pearson correlation coefficient to statistically quantify the relation between the failure rate and the performance metrics. Table III shows the correlation between the failure rates of the two types of errors and the IPC performance metric. Similar results are obtained if we use other profiling performance metrics such as the issue slot utilization or the eligible warps per cycle, but also if we use the arithmetic operations per second of each version of the microbenchmark. Coefficients confirm that there is a strong correlation between the performance of the three versions of the benchmark and the SDC FIT, while the DUE FIT only shows this kind of correlation with the register-based version.

The FIT metric does not take into account the performance of the code being executed and the amount of work that can be carried out without being affected by any radiation-induced errors. It may happen that some variant has a higher number of FIT, but this problem is compensated by the fact that it has been able to perform a much higher number of calculations in the same time. To take this phenomenon into account,

TABLE III
PEARSON COEFFICIENT CORRELATION BETWEEN THE IPC PERFORMANCE METRIC AND THE ERROR RATES OF THE DIFFERENT VARIANTS OF THE BENCHMARK

	reg	shm	glb	Total
SDC FIT	0,93	0,89	0,84	0,81
DUE FIT	0,90	-0,69	0,01	0,55
Total FIT	0,95	0,39	0,39	0,75

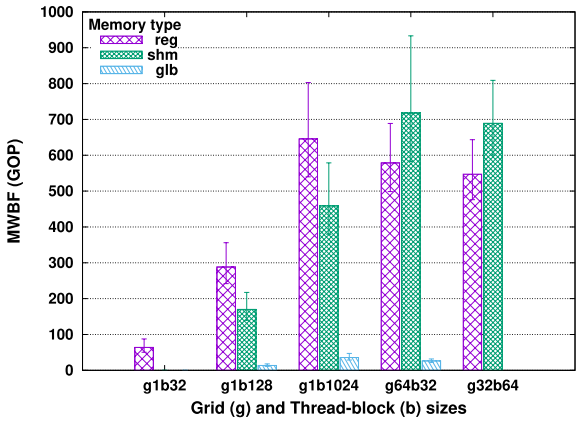

Fig. 5. Effect of the memory type and computational load in the MWBF measured as billions of integer arithmetic operations (GOP).

we use the mean work between failures (MWBF) metric [38]. We measure the work done by the microbenchmark as the number of arithmetic operations with integers performed. We have measured the number of arithmetic operations on each variant of the microbenchmark. Since the execution of all the variants used had a duration of 2 s, Fig. 3 shows the work done by each of them, if we double the scale of the vertical axis. The MWBF value for the different variants is shown in Fig. 5, which demonstrates that the benefits of using the GPU's computational resources more efficiently can compensate for a possible increase in errors. We can see, for example, that although the number of FIT in the *global-memory* version is similar to the other two versions, given the very low performance obtained when using *global-memory*, the amount of work that can be done before an error occurs is much smaller than with the versions using *registers* or *shared-memory*.

Therefore, if we want to increase the total amount of work done without errors, it is best to maximize the GPU resources use by launching the maximum number of threads supported by SM and to leverage *shared-memory* for computation. Since the amount of FIT does not increase substantially with increasing GPU resource usage, except when we use more registers, it is best to optimize our code implementation to get the most out of those resources [21].

Our analysis also includes an examination of the number of threads impacted by the SDC errors. The results reveal that in nearly all instances (93%), only one thread's outcome was affected by the neutron-induced error. This outcome aligns with expectations since individual threads perform independent computations utilizing mostly distinct resources such as arithmetic units, registers, and memory cells. Only in few instances were the results of several tens of threads impacted by SDC errors, and in just two cases were the results of almost all threads affected.

V. CONCLUSION

Evaluating the impact of utilizing different GPU resources is crucial for gaining insights into the reliability of GPUs exposed to radiation. In this study, we employed various versions of a microbenchmark to investigate the effect of different memory types on the performance of a low-power GPU integrated into the TX1 SoC of a Jetson Nano board. Additionally, we explored the tradeoff between enhanced computational performance and the occurrence of failures over time by optimizing the utilization of GPU resources. Results show that there is a strong correlation between the code performance and the SDC failures.

Our findings demonstrate that maximizing the utilization of the device's cores enables the completion of a greater number of computations without errors. By fully harnessing the computational potential of the GPU cores, we effectively reduce the reliability challenges posed by irradiation. Moreover, we observed that the choice of memory type has a significant influence on the overall reliability of the GPU.

The outcomes of this research contribute to a comprehensive understanding of the interplay between GPU resources, irradiation effects, and reliability. This knowledge is instrumental in guiding the development of robust GPUs for applications in radiation-prone environments.

ACKNOWLEDGMENT

The authors would like to thank Chris Frost, Carlo Cazzaniga, and Maria Kastriotou for their help with the experiment. ChipIR provided neutron beam time (DOI: 10.5286/ISIS.E.RB2200022).

REFERENCES

[1] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance & precision," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 522–531.

[2] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1226–1238, Apr. 2016.

[3] H. D. Dixit et al., "Silent data corruptions at scale," 2021, *arXiv:2102.11245*.

[4] D. Tiwari et al., "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2015, pp. 331–342.

[5] R. Canal et al., "Predictive reliability and fault management in exascale systems: State of the art and perspectives," *ACM Comput. Surv.*, vol. 53, no. 5, pp. 1–32, Sep. 2020.

[6] G. Furano and A. Menicucci, "Roadmap for on-board processing and data handling systems in space," in *Dependable Multicore Architectures at Nanoscale*. Berlin, Germany: Springer, Aug. 2018, pp. 253–281.

[7] J. Fickenscher, S. Reinhart, F. Hannig, J. Teich, and M. E. Bouzouraa, "Convoy tracking for ADAS on embedded GPUs," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, Jul. 2017, pp. 959–965.

[8] K. V. Sakhare, T. Tewari, and V. Vyas, "Review of vehicle detection systems in advanced driver assistant systems," *Arch. Comput. Methods Eng.*, vol. 27, no. 2, pp. 591–610, Apr. 2020.

[9] I. Agirre, M. Azkarate-Askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla, "A safety concept for a railway mixed-criticality embedded system based on multicore partitioning," in *Proc. IEEE Int. Conf. Comput. Inf. Technol., Ubiquitous Comput. Commun., Dependable, Autonomic Secure Comput., Pervasive Intell. Comput.*, Dec. 2015, pp. 1780–1787.

[10] J. P. Cerrolaza et al., "Multi-core devices for safety-critical systems: A survey," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–38, Aug. 2020.

[11] H. Quinn, "Challenges in testing complex systems," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 2, pp. 766–786, Apr. 2014.

[12] S. Kumar Sastry Hari et al., "Estimating silent data corruption rates using a two-level model," 2020, *arXiv:2005.01445*.

[13] F. F. D. Santos, S. K. S. Hari, P. M. Basso, L. Carro, and P. Rech, "Demystifying GPU reliability: Comparing and combining beam experiments, fault simulation, and profiling," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2021, pp. 289–298.

[14] *CUDA C++ Program. Guide. PG-02829-001_V11.2. Design Guide*, NVIDIA, Santa Clara, CA, USA, Nov. 2021.

[15] *NVIDIA Tegra X1 Whitepaper, V1.0*, NVIDIA, Santa Clara, CA, USA, 2015.

[16] P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro, "Impact of GPUs parallelism management on safety-critical and HPC applications reliability," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2014, pp. 455–466.

[17] D. A. G. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 791–804, Mar. 2016.

[18] D. Oliveira et al., "High-energy versus thermal neutron contribution to processor and memory error rates," *IEEE Trans. Nucl. Sci.*, vol. 67, no. 6, pp. 1161–1168, Jun. 2020.

[19] A. Bustos et al., "Response of HPC hardware to neutron radiation at the dawn of exascale," *J. Supercomput.*, vol. 79, no. 12, pp. 13817–13838, Aug. 2023.

[20] S. M. Guertin, W. P. Parker, A. C. Daniel, and P. Adell, "Recent SEE results for snapdragon processors," in *Proc. IEEE Radiat. Effects Data Workshop*, Jul. 2019, pp. 1–5.

[21] V. Fratin, D. Oliveira, C. Lunardi, F. Santos, G. Rodrigues, and P. Rech, "Code-dependent and architecture-dependent reliability behaviors," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 13–26.

[22] J. M. Badia et al., "Reliability evaluation of LU decomposition on GPU-accelerated system-on-chip under proton irradiation," *IEEE Trans. Nucl. Sci.*, vol. 69, no. 7, pp. 1467–1474, Jul. 2022.

[23] W. S. Slater, N. P. Tiwari, T. M. Lovelly, and J. K. Mee, "Total ionizing dose radiation testing of NVIDIA Jetson Nano GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Dec. 2020, pp. 639–641.

[24] A. Serrano-Cases, S. Alcaide, M. Romero, Y. Morilla, and S. Cuenca-Asensi, "Analysis of kernel redundancy for soft error mitigation on embedded GPUs," *IEEE Trans. Nucl. Sci.*, vol. 70, no. 8, pp. 1700–1707, Jul. 2023.

[25] K. Ito, Y. Zhang, H. Itsuji, T. Uezono, T. Toba, and M. Hashimoto, "Analyzing DUE errors on GPUs with neutron irradiation test and fault injection to control flow," *IEEE Trans. Nucl. Sci.*, vol. 68, no. 8, pp. 1668–1674, Aug. 2021.

[26] F. F. Dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three GPU architectures," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2017, pp. 169–176.

[27] F. F. D. Santos et al., "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Rel.*, vol. 68, no. 2, pp. 663–677, Jun. 2019.

[28] P. Rech, L. Carro, N. Wang, T. Tsai, S. K. S. Hari, and S. W. Keckler, "Measuring the radiation reliability of SRAM structures in GPUS designed for HPC," in *Proc. IEEE 10th Workshop Silicon Errors Logic-Syst. Effects (SELSE)*, Apr. 2014, pp. 1–6.

[29] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.

[30] NVIDIA. (Apr. 2023). *Profiler User's Guide*. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide

[31] B. Topçu and I. Öz, "Predicting the soft error vulnerability of GPGPU applications," in *Proc. 30th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. (PDP)*, Mar. 2022, pp. 108–115.

[32] C. Cazzaniga, M. Bagatin, S. Gerardin, A. Costantino, and C. D. Frost, "First tests of a new facility for device-level, board-level and system-level neutron irradiation of microelectronics," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 1, pp. 104–108, Jan. 2021.

[33] E. Wyrwas, K. A. LaBel, M. Campola, and M. O'Bryan, "Guidance on standardizing GPU test approaches," in *Proc. Nucl. Space Radiat. Effects Conf. (NSREC)*, Dec. 2018, pp. 116–119.

[34] V. Volkov, "Understanding latency hiding on GPUs," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Univ. California Berkeley, Aug. 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-143.pdf

[35] NVIDIA. (Apr. 2023). *CUDA C++ Best Practices Guide*. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide

[36] C. Slayman, "JEDEC standards on measurement and reporting of alpha particle and terrestrial cosmic ray induced soft errors," in *Soft Errors in Modern Electronic Systems*. New York, NY, USA: Springer, Jan. 2011, pp. 55–76.

[37] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Jan. 2003, pp. 29–40.

[38] G. A. Reis, J. Chang, N. Vachharajani, S. S. Mukherjee, R. Rangan, and D. I. August, "Design and evaluation of hybrid fault-detection systems," in *Proc. 32nd Int. Symp. Comput. Architecture*, Jun. 2005, pp. 148–159.