Reliability evaluation of LU decomposition on GPU-accelerated System-on-Chip under proton irradiation

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

***Abstract*—Graphic Processing Units (GPU) have become a basic accelerator both in high-performance nodes and low-power SoC. They provide massive data parallelism and very high performance per watt. However, their reliability in harsh environments is an important issue to take into account, especially for safety-critical applications. In this paper we evaluate the influence of the parallelization strategy on reliability of LU decomposition on a GPU-accelerated SoC under proton irradiation. Specifically we compare a memory bound and a compute bound implementation of the decomposition on a K20A GPU embedded on a TK1 SoC. We leverage the GPU and CPU clock frequencies both to highlight the radiation sensitivity of the GPU where we are running the benchmark, and also to apply both algorithms to solve problems with the same size when exposed to the same radiation dose. Results show that a more intensive use of the resources of the GPU increases the cross-section. We also observed that most of the radiation-induced errors hang the operating system and even the rebooting process. Finally, we present a preliminary study of the error propagation of the LU decomposition algorithms.**

***Index Terms*—fault tolerance, GPU, LU decomposition, System-on-Chip, proton irradiation.**

I. Introduction

Current System-on-Chip (SoC) that include several cores with a low-power GPU are very attractive in many environments. They combine low cost and weight, high power efficiency, relatively high performance and, high flexibility in the use of their various components, as shown for multiple applications, such as Advanced Driver-Assistance Systems (ADAS) [1], avionics [2] and space [3]. However, in these environments there is an additional factor that is very important to consider: the radiation sensitivity of these devices along with the applications running on them. In many cases, especially in space, specific rad-hard devices are used to withstand radiation effects [4]. However, it is increasingly relevant to study the sensitivity of Commercial off-the-shelf (COTS) components, which are currently considered as an alternative for the design of spacecraft electronics [5].

Our work aims to assess how the radiation effects depend on the parallelization strategy of applications executed on a GPU-accelerated SoC which, together with the problem size, defines their resource usage. To this end, we have tested two very different versions of the same application. Note that GPUs offer a large number of parallel computational cores, block and warp schedulers, as well as different kinds of memories that can be leveraged to increase the performance of the algorithms, but can also affect their radiation sensitivity.

We have selected Lower-Upper (LU) decomposition as a case study. This one-sided factorization is one of the benchmarks included for example in the Rodinia suite [6] and stands out because it is a basic computational kernel in multiple applications. Besides it combines a high computational load with intensive memory access, which may depend on the particular implementation used. In fact, LU decomposition has been used in both irradiation tests and fault injection experiments, such as [7], [8], but to the best of our knowledge, it has never been irradiated to study its sensitivity on GPU-accelerated SoC.

Our experiments have been carried out on the Tegra K1 (TK1) System-on-Chip (SoC), embedded in the Jetson development kit, which contains a quad-core CPU together with a small GPU [9]. The tasks of the CPU cores are limited to run a few operating system processes, prepare, launch, and verify the result of the LU decomposition. It is worth pointing out that no Single Event Latchups were observed in our DUT, which confirms the observations in [10], where the authors used much higher energy protons.

We want to highlight that our approach does not only focus on the parallelization strategy or the size of the problem, but also, leverages the GPU and CPU frequencies. Properly combining both kind of frequencies allows us to adjust the utilization of the CPU and GPU, and also to apply both LU algorithms to solve the problems with the same size when exposed to the same radiation dose.

Our results show that a more intensive utilization of the GPU resources, such as its shared memory or schedulers, produces higher computational performances. However, it also increases the probability of suffering radiation induced errors

This work has been supported by the Valencian Regional Government through PROMETEO/2019/109, the University Jaume I project UJIB2019-36, the Spanish Ministry of Science and Innovation under project PID2019-106455GB-C21, and the project PID2020-113656RB-C21 supported by MCIN/AEI/10.13039/501100011033 as well as the Regional Government of Madrid throughout the project MIMACUHSPACE-CM-UC3M (2022/00024/001).

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

Jose A. Belloch, Almundena Lindoso, Mario Garcia-Valderas and Luis Entrena are with the Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Av. de la Universidad 30, 28911 Leganés, Madrid, Spain (e-mail:{jbelloc,alindoso,mario.garcia,entrena}@ing.uc3m.es).

Yolanda Morilla is with the Centro Nacional de Aceleradores (CNA), Centro Nacional de Aceleradores, CSIC, JA, Universidad de Seville, E-41092 Seville, Spain, SPAIN (e-mail: ymorilla@us.es).

and so the cross-section of the applications.

The rest of the paper is structured as follows. Section II summarizes the related work. Section III describes the experimental environment and methodology. Section IV presents the experimental results and discussion. Finally, section V summarizes the main conclusions.

II. Related work

A considerable number of papers have been published in recent years analyzing the fault tolerance of GPUs. Two main techniques have been employed to carry out those analysis: fault injection and radiation [11], [12]. Recent papers combine both techniques trying to model the behaviour of this kind of device [13], [14].

It is important to consider what kind of benchmark to use to study the fault tolerance of such devices. A widely used benchmark is the product of matrices, but others have also been used, such as FFT, matrix transposition or some algorithms included in the Rodinia suite [6], [11]. Recently, more complex applications related to neural networks have also been used to assess GPUs reliability [15]. LU decomposition can be a very suitable benchmark because it is a basic computational kernel in multiple applications and also because it combines a high computational load with intensive memory access, which may depend on how it is implemented.

A number of papers that use fault injection to study the version of the LU included in Rodinia can be found in the literature [7]. There are also several papers that design and analyse various fault-tolerant versions (ABFT) of this decomposition [16]–[18]. However, to the knowledge of the authors, LU decomposition has never been irradiated to study its sensitivity on GPUs.

Most radiation reliability analysis of GPUs comparing the behaviour of different benchmarks have been performed on high performance devices under neutron radiation [8], [11], [12].

Nevertheless, some radiation tests have also been performed with different GPU-accelerated SoC, mainly with NVIDIA Jetson and Qualcomm Snapdragon devices [19]. For example, results of testing several Snapdragon devices with different scaling technology and including different Adreno GPUs are included in [20], [21]. Experiments use proton, neutron and heavy ion radiation with different energy levels and analyze the SEE sensitivity of the platforms.

Jetson TX1 and TX2 platforms were tested in [22], [23] using proton irradiation to provide a baseline assessment of their radiation susceptibility. Jetson TX1 and Snapdragon 820 were irradiated in [24] using protons, heavy ions and also laser testing to characterize long-term radiation effects and determine their dose-rate sensitivity. Radiation evaluations show that different types of Jetson boards are acceptable for many low earth orbit short duration missions using the proper mitigation techniques. For example, in [25] gamma-ray photons were employed to evaluate the tolerance to radiation effects of a Jetson Nano board, which includes a 128-core NVIDIA Maxwell GPU. Preliminary results suggest operation beyond 20 krad(Si). Similar conclusions were reached in [26] for a Jetson AGX Xavier board, including a 512-core NVIDIA GPU, using proton irradiation. High-energy protons were used in [10] to evaluate the cross-section and the total dose performance of the Tegra K1 SoC by running different versions of the FFT in the CPU and GPU.

Radiation experiments using different kinds of particles on COTS GPU-accelerated SoC show that Single-Event Functional Interrupt (SEFI) errors are a common problem in this type of device. However, in almost all cases the error is solved by rebooting the device and Single Event Latchups (SEL) are very rare.

III. Experimental environment and methodology

A. *Device Under Test*

The Tegra K1 (TK1) System-on-Chip (SoC) is fabricated on a 28 nm Complementary Metal-Oxide-Semiconductor (CMOS) process technology. This system comprises a quad-core ARM Cortex A15 processor (or CPU), an ARM Cortex A15 battery-saving shadow core, and an NVIDIA "Kepler" K20A GPU with 1 Streaming Multiprocessor (SM) containing 192 CUDA cores [9]. The device has 2 GiB of global memory which is the same DDR3 memory for both the CPU and GPU. It also includes 128 KiB of L2 cache, 48 KiB of shared memory and a register file with 32768 registers. Recall that, following the CUDA memory model, the shared memory included in the GPU is a fast memory allocated per thread block that can be only accessed by all the threads of that block. It is worth noting that neither the device's DDR3 memory nor any of the GPU's internal memory support Error Correction Codes (ECC). Therefore, even single bit flips in the memory cells can produce errors in the applications.

The K20A GPU can be classified as a high-end COTS embedded GPU [3]. This kind of GPU does not offer as much computational power as high-performance GPUs, but it combines low cost, very high-performance per watt with the flexibility offered by programming frameworks such as CUDA or OpenCL. It is worth pointing out that OpenCL is not supported by the GPU included in the TK1 SoC or any other embedded NVIDIA GPU. This device is an ideal accelerator for SoC that can be included in embedded systems with a wide range of applications such as ADAS, avionics or space, among many others.

The power management strategy of the Jetson device uses dynamic frequency scaling with dynamic voltage scaling. The CPU and GPU frequency are dynamically adjusted depending on how busy the device is. Both the CPU and CPU provide a wide range of frequencies that affect the performance and energy consumption of the applications. The user can choose at runtime one of the available CPU and GPU processor frequencies by using some system commands or modifying some system files.

B. *Setup and procedure*

Experiments were performed in remote mode using the protons cyclotron accelerator of the Centro Nacional de Aceleradores (CNA) at Sevilla, Spain, in January 2021 [27]. In the used set-up on air, the proton beam reaches the DUT surface with $15.4 MeV$ and an estimated spread of 400 keV. The average proton flux was maintained in the order of $0.8 \times 10^7 p/(cm^2 s)$ into an homogeneous spot of $1.5 cm$ in diameter. The irradiation affected the whole SoC component of the board, containing the four CPU cores, and the GPU including its shared, L1 and L2 cache memories. The radiation did not affect the DDR memory common to the CPU and GPU the eMMC memory or the SD card. All our experiments were performed in the same radiation campaign and using only one Jetson TK1 board at room temperature.

Despite the low proton energy, the device was sensitive enough without thinning it, as happened in previous tests conducted by CNA on similar devices with technology of $28 nm$ and epoxy covering [27]. When we use $15.4 MeV$ incident protons in the DUT surface, the energy of the particles in the silicon active area is in the order of $10 MeV$ or below, so enough event rates are produced for the used technology [28].

In our setup the Jetson TK1 board sent the logs of the test to a host controller through the serial communication port. The host controller is near the DUT but not under direct beam exposure. The controller was also connected to the GPIO pins of the DUT so that it could be used to remotely reset the Jetson TK1 board when it hangs. The whole test was managed remotely from a laptop connected to the host controller through ethernet. Figure 1 displays the radiation test setup used during the experiments. The operating system Ubuntu 14.04 with the CUDA 6.5 driver was run from the SD card of the Jetson TK1 board, so that we avoided the radiation effect on the system files.

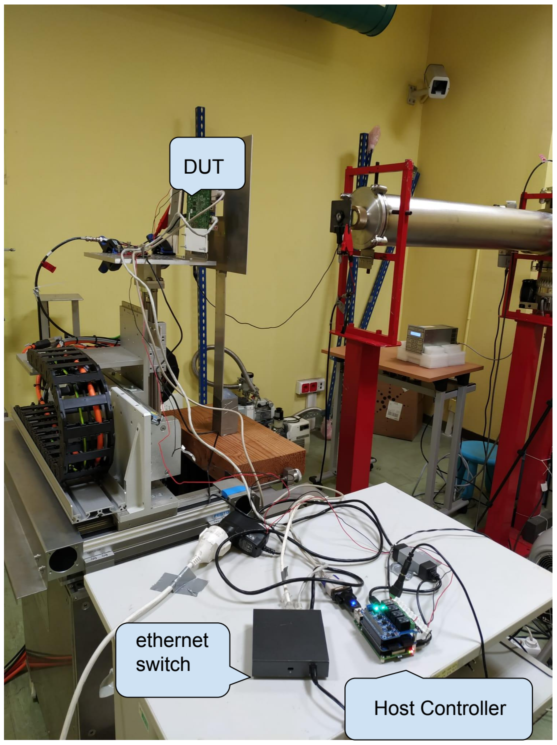

Fig. 1. Radiation test setup at CNA.

The LU decomposition algorithms were programmed using C and we used a Python scripts to run the different benchmarks. We employed the Python module `Pexpect` to spawn and control a subprocess in charge of running each multiplication. Our experiments included three watchdogs to detect and recover from different hangs of the tests and the operating system. The first timeout, associated with the spawned process, was set to a time larger than the maximum expected duration of one LU decomposition. The second watchdog used the watchdog Linux API to reboot the system if it hung for more than 20 seconds. Finally, a third watchdog was implemented on the host controller, so it could reset the device if the Jetson system hung and did not produce any log result during more than 25 seconds.

C. CUDA programming model

The Compute Unified Device Architecture (CUDA) [29] was developed by NVIDIA to ease the programming of its General Purpose GPUs. CUDA programs combine a host code run on the CPU with one or several kernel functions to be executed in the CUDA cores using a Single Instruction Multiple Threads (SIMT) model. That is, every thread runs the same instructions on a different core of the GPU in lockstep, usually affecting different data. The host code is mainly in charge of transferring the data to the GPU, orchestrating the execution of the different kernels on the cores and getting and verifying the results.

In CUDA, GPUs are based on an array of Streaming Multiprocessors (SM) containing tens or hundreds of CUDA cores. Those cores are much simpler than CPU cores and are mainly devoted to perform basic arithmetic operations in a pipelined fashion. Latest GPUs also contain special tensor cores designed to optimize the basic operations performed during the training and inference steps of neural networks. Threads are grouped by the programmer in blocks that are dispatched to one SM. Thread blocks are organized in a grid that can have up to three dimensions. Besides, thread blocks are divided in warps usually containing 32 threads. Modern SMs contain several schedulers that can schedule several warps in parallel to groups of 32 cores. To increase the occupancy of the GPU, tens of warps can be kept active on each SM, so that whenever the threads of one warp have to wait for example while the data they need is loaded from memory, another warp can be executed.

The performance of the CUDA programs depends on several parameters [29]. One of the most important goals of the programmers is to maximize the GPU occupancy, that is, the ratio of active warps on an SM with respect to the maximum of active warps supported by each SM. This ratio is different on each GPU architecture. One way to increase the occupancy is to define thread blocks containing enough warps to keep the cores active while hiding the latencies between dependent instructions, synchronizations and other stalling factors. The main idea is to maximize the number of instructions per clock cycle during the execution of the programs. Another important factor that determines the performance is the efficient use of the fastest memories of the GPU. Whenever possible, data should be stored in the registers that are local to each thread, or in the fast memory shared by all the threads of each block.

Kernels should avoid accessing the global memory of the GPU, which is much slower than the shared memory. Moreover, data should be organized in memory to minimize the number of load operations. These accesses can be coallesced if adjacent memory positions are loaded by threads with consecutive identifiers in its block.

There are several limiting factors to the performance that depend on the resources available in the GPU. For example, individual threads should not use more than a given number of registers and there is also a limit to the number of registers or the shared memory that can be used by each thread block. Therefore, programmers should implement their codes to avoid any of the limiting factors being reached.

D. LU decomposition benchmark

LU is a Lower-Upper decomposition where a matrix is factorized as the product of a lower triangular matrix and an upper triangular matrix. It is usually performed as a first step towards the direct solution of square systems of equations, and it is also used to invert a matrix or compute its determinant. We have used CUDA to implement two very different parallel algorithms to perform LU decomposition. Our goal is to compare the behavior of two codes that make a different use of the resources of the SoC, including the memories of the GPU and also the warp schedulers or instruction dispatchers included in the SM.

The first LU decomposition algorithm, called `block`, is the block version included in the Rodinia benchmark suite [6]. Figure 2 shows the main panels of the first iteration of this algorithm. The algorithm starts from the top left corner and applies the following three steps:

$$A_{11} \leftarrow L_{11}U_{11} \tag{1}$$

$$L_{21} \leftarrow A_{21}U_{11}^{-1} \qquad U_{12} \leftarrow L_{11}^{-1}A_{12} \tag{2}$$

$$A_{22}^{'} \leftarrow A_{22} - L_{21}U_{12} \tag{3}$$

Then, the decomposition continues by applying the same steps to the $A_{22}^{'}$ panel obtained after finishing the first iteration. Three CUDA kernels are used to perform the three main steps of the algorithm, namely, `diagonal`, `perimeter` and `internal`. In Figure 2 we can see the panels affected by each kernel during the first iteration on a different color. The kernel `diagonal` performs the LU decomposition of the top left panel A_{11} by means of a simple method, using only one thread-block with `blk` threads. Every thread is in charge of computing one row of U_{11} and one column of L_{11} storing the intermediate results in the shared memory of the block. The last two kernels partition the panels A_{12}, A_{21} and A_{22} in square blocks of size $blk \times blk$ (dotted red lines in Fig. 2) and copy and update them in parallel by leveraging the shared memory of the GPU to reduce the accesses to global memory.

The second parallel algorithm that we have evaluated is called `rc`, and uses the same method that the kernel `diagonal`, but applied to the whole matrix. A very important difference with respect to the algorithm `block` is that all the elements of the matrix are loaded and stored from/to the global memory of the GPU. Besides, every thread may have to compute more than one row and column, depending on the size of the matrix. The algorithm `rc` is obviously much slower than the block version, as it uses less efficiently the cores and memories of the GPU.

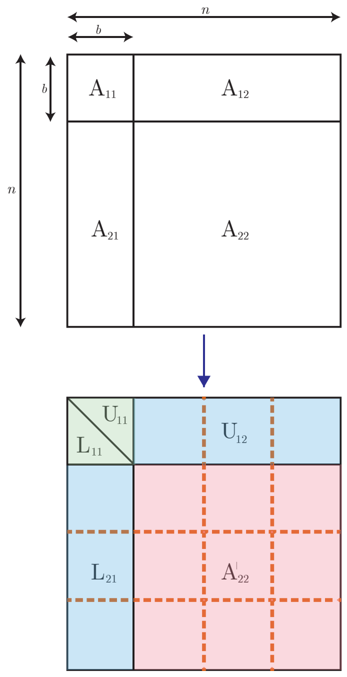

Fig. 2. Main panels and updates during the first iteration of the block LU decomposition.

E. GPU resources usage

Both algorithms make a very different use of the memories of the GPU, and also of other resources, such as the warp schedulers included in the SM of the GPU. On the one hand, algorithm `Block` successively executes each of its three kernels on each iteration. The size of the grid used to launch each kernel adapts to the size of the matrix panel factorized on each iteration. For example, suppose that Fig 2 represents a matrix of size 64×64, where each small square block is of size 16×16. Then, during the first iteration of the algorithm, kernel `diagonal` launches one block of threads of size 16; kernel `perimeter` launches 3 blocks of threads of size 32; and kernel `internal` launches 3×3 blocks of threads of size 16×16. On the other hand, algorithm `rc` uses a grid including only one block of threads of size 64. Therefore, the number of kernels launched when executing both algorithms is quite different, and thus, it is also the occupancy of the cores and the use of the 4 warp schedulers of the GK20 GPU included in the TK1 SoC.

Table I shows that the achieved occupancy of the kernel of the algorithm `rc` and the three kernels of the algorithm `block` is quite different. For example, with a matrix of size 1024×1024 and 1024 threads per block, the kernel `rc` can keep 32 warps active per cycle, which is 50% of the maximum number of warps that can be active in a K20A GPU (64). The kernel `diagonal` uses only one thread block with one warp and so it only gets an occupancy of $1/64 = 1.56\%$. On the

TABLE I
METRICS OF THE GPU RESOURCE USAGE WITH A MATRIX OF SIZE 1024×1024.

Metric	diagonal	perimeter	internal	rc
Achieved Occupancy (%)	1.56	18.65	91.96	50.00
Eligible Warps Per Cycle	0.06	1.00	6.26	2.01
Executed IPC	0,06	0.62	1.23	0.12
Number of Registers	30	25	17	24
Shared mem (B/th-blk)	1024	3072	2048	0
Global Load Trans. (M)	0.001	0.10	4.10	196.52
L2 Read Trans. (M)	0.02	0.25	8.31	215.51
Shared Load Trans. (M)	0.04	1.95	16.39	0

contrary, the kernel `internal` uses a large number of warps, which results in an occupancy close to the optimum (91.96%). The table also shows that the kernel `internal` achieves the maximum Instructions Per Cycle (IPC) and occupancy. As it clearly dominates the execution time of the algorithm `block`, this algorithm clearly overcomes the computational performance of the algorithm `rc`.

Radiation may have a quite different effect on both algorithms. For example, algorithm `rc` does not use the shared memory of the GPU and thus, a particle modifying the information stored in this memory will not produce any error. However, this algorithm accesses much more the global memory than the algorithm `block`. Even if this memory is not exposed to the radiation beam, the probability of a particle modifying the data being loaded and stored from/to the L2 cache memory notably increases. Table I shows that the algorithm `rc` performs a much larger number of load transactions from global memory and read transactions from L2 cache than the three kernels of the algorithm `block`. Besides, algorithm `block` makes a much more intensive use of the warp schedulers of the SM, as it has to launch three different kernels with different block sizes on the successive iterations of the algorithm. This fact can be seen if we compare for example the number of eligible warps per cycle of the kernels of both algorithms shown in Table I. All the data included in this table has been obtained using the CUDA profiler `nvprof` [30].

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Radiation experiments

Our first radiation experiments were devoted to determine an appropriate proton flux that allowed us to launch the LU algorithms. We found out that medium-intensity proton fluxes, (i.e. $2.0 \times 10^8 p/(cm^2 s)$), hung or rebooted the Linux operating system even if it was not executing any user application. Besides, in those cases, the system was unable to complete the rebooting process without hanging again. Therefore, we reduced the flux to a value that allowed us to launch at least an average of 10 consecutive LU decompositions without hanging the test or the operating system. The average flux used in all our subsequent experiments was $0.8 \times 10^7 p/(cm^2 s)$.

In previous experiments performed by other authors no Single Event latchup (SEL) were observed, even when the Tegra TK1 was irradiated with the equivalent of at least 21.0 years heavy ion fluence up to a Linear Energy Transfer of approximately $10 MeV cm^2/mg$ [10]. Our results, using lower energy protons, but with higher fluxes and fluence, confirm this behaviour. No SEL was observed in our experiments, where the device received a total fluence of $2.2 \times 10^{12} p/cm^2$ at $15.4 MeV$.

Once the average flux was fixed, we performed experiments to evaluate the behaviour of the DUT depending on the algorithm used to perform matrix decomposition, the CPU and GPU frequencies, and also on the size of the matrix. Single precision elements were used in all the experiments. Table II shows the main parameters of each of the tests with both LU decomposition algorithms. For each algorithm and problem size we launched the test using two different CPU and GPU frequencies called `f1` and `f2` in the names of the tests. Those frequencies were different for each algorithm and problem size. As radiation affects the whole SoC, we cannot evaluate separately the effects on the GPU and CPU cores. However, we can use the clock frequencies of both components to highlight the effects on the GPU, where we are running our benchmark. We should be cautious when comparing results obtained under different frequencies, since radiation behaviour of the device under test can be different. However, this method allows us to increase or decrease the percentage of the time of the test devoted to performing the LU decomposition in the GPU. As this computation is performed while the CPU is mainly idle, most radiation errors would be caused by the GPU activity.

For example, the first two rows of Table II show the parameters used in the algorithm `block` with matrices of size 1024 (1k). The test `blk1kf1` uses the minimum GPU frequency (72 MHz) and the LU kernel executed in the GPU takes 73.2% of the total time of the test. On the contrary, the test `blk1kf2` uses a higher GPU frequency (324 MHz), and in this case the LU kernel takes only 18.6% of the total time of the test under radiation. This way, if we increase the percentage of time spent by the LU decomposition, the results of the radiation mainly show how the kernels running in the GPU are affected by the proton flux. We can then compare the results with the ones obtained by using a higher GPU frequency, which reduces the percentage of time devoted to the LU decomposition and thus highlights the effect of the radiation on the tasks performed by the process running on the CPU. That process is in charge of generating the matrix to factorize, transferring the data to and from the GPU and verifying the result of the decomposition with respect to a previously stored golden result.

Our experiments show that the `block` algorithm is 8x faster than the `rc` algorithm with matrices of size 1024, and 10x faster with matrices of size 2048. The times shown in Figure 3 have been obtained using a frequency of 396 MHz for the GPU. The frequency of the CPU is not fixed as it does not affect the execution time of the kernel in the GPU. In order to submit the tests with both algorithms to the same radiation dose, we chose CPU and GPU frequencies that produce the same total execution time for the same problem size. The execution times are shown in the column `Total` of Table II.

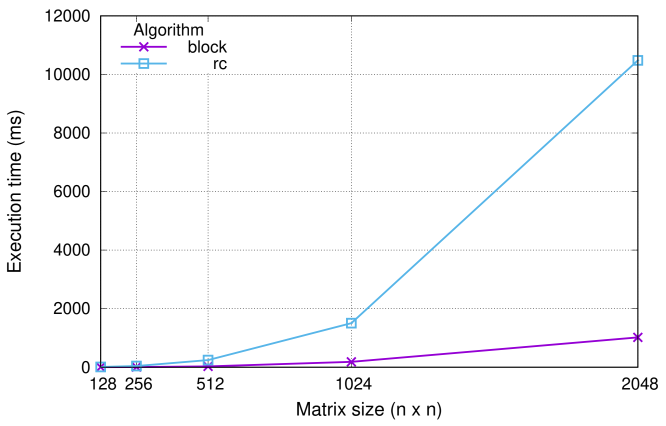

Fig. 3. Execution times of both LU algorithms.

TABLE II
MAIN PARAMETERS OF THE RADIATION TESTS. "BLK" IN THE NAME OF THE TESTS REFERS TO `BLOCK` ALGORITHM, 1K=1024 AND 2K=2048.

	CPUFreq (MHz)	GPUFreq (MHz)	Total (ms)	Kernel (% Total)
blk1kf1	696	72	1003	73.2%
blk1kf2	204	324	1145	18.6%
rc1kf1	2065	612	1013	90.4%
rc1kf2	564	804	1055	68.6%
blk2kf1	2065	180	2008	92.1%
blk2kf2	696	252	2050	74.8%
rc2kf1	2065	852	5043	96.8%

B. Radiation results

Due to the total available time to perform radiation tests, we run around 1000 LU decompositions for each combination of algorithm and frequencies for the problems with size 1k=1024, and around 500 LU decomposition for the problems with size 2k=2048. Once the parameters defining each batch of tests were established, the radiation was started and the beam was kept active during the 1000 LU decompositions. Each decomposition involved not only the factorization of the matrix, but also the transfer of its initial values from CPU to GPU, the transfer of the result matrix from GPU to CPU and also the verification of the result using the golden copy. After each batch of LU decompositions the beam was stopped and a few minutes were devoted to configure the next batch. Columns in Table III show the results of the radiation for both algorithms, with both problem sizes and frequencies. Column `Correct` contains the number of LU decompositions that finished with the correct result, while column `SDC` contains the number of LU decompositions affected by a Silent Data Corruption and finished with a wrong result. Regarding the column `HangTest`, it contains the number of hangs produced during one of the LU decompositions due to the kernel running on the GPU or the processes running on the CPU. Those hangs forced us to reboot the Jetson board, which took more than 20 seconds. Finally, column `HangBoot` contains the number of hangs produced by the radiation during the reboot of the board,

TABLE III
RESULTS OF THE IRRADIATION CAMPAIGN.

	Correct	SDC	HangTest	HangBoot
blk1kf1	1011	18	21	11
blk1kf2	1017	15	39	19
rc1kf1	1023	3	27	9
rc1kf2	1014	4	32	12
blk2kf1	500	18	21	1
blk2kf2	478	22	17	8
rc2kf1	478	19	55	21

which involved restarting the slow reboot process. The table shows that most of the LU decompositions are not affected by the radiation or mask its effects. Only a few SDCs were detected for each algorithm and problem size, and the number of times the test or the reboot hung was larger than the number of decompositions finished with wrong results.

In order to compare the radiation sensitivity of both LU decomposition algorithms we computed their cross-section with the two problem sizes and show the results in Figure 4. Cross-section is calculated by dividing the number of errors by the radiation flux ($protons/cm^2$). Higher cross sections imply higher probabilities for a particle that hits the GPU to produce an error. The figure includes the confidence intervals computed with a confidence of 95%. Firstly, we can see that increasing the size of the problem increases the cross-section of both algorithms. Besides, a more intensive use of the resources of the SoC involves a higher probability of radiation induced errors. The `block` algorithm gets much higher performances than the `rc` algorithm by leveraging the shared memory of the GPU. However, stressing the internal memory and other components of the GPU also involves a higher cross-section. Specifically, Table II shows how the `block` algorithm achieves a larger occupancy by having more eligible warps per cycle. That algorithm also executes more instructions per cycle and uses the shared memory of the GPU.

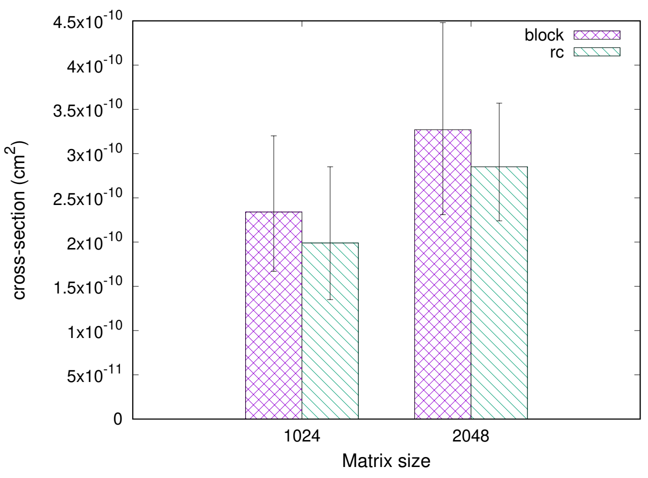

Fig. 4. Cross-section of both LU decomposition algorithms.

We can also highlight the effect of the radiation on the

GPU by reducing its frequency and analyzing the cross-section obtained for both algorithms and problem sizes. Recall that with lower frequencies the percentage of radiation time that affects the LU decomposition on the GPU increases. Figure 5 shows that the cross-section increases in all cases when the percentage of the total time running the LU in the GPU decreases. This percentage is shown at the top of each bar of the figure. In those cases we reduce the effect of the radiation on the GPU while keeping its effect on the processes running on the CPU. Therefore, it seems that the processes running on the cores of the CPU are more sensitive to the radiation than the kernels running on the cores of the GPU.

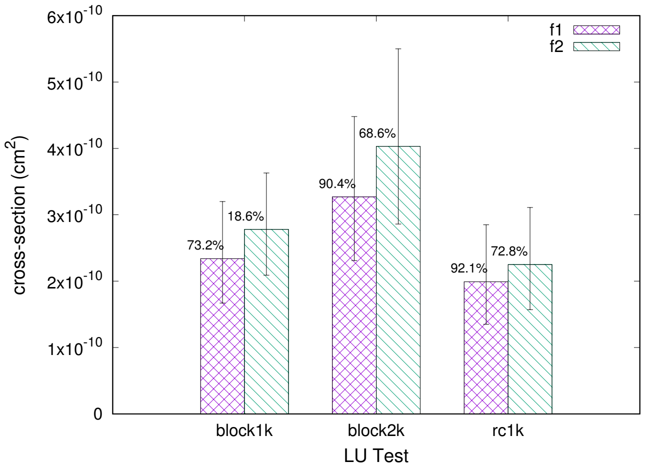

Fig. 5. Effect of the GPU frequency in the cross-section of both LU decomposition algorithms. Problem size: 1k=1024 and 2k=2048. Labels in bars contain the percentage of time of the test spent running the LU kernels on the GPU. Frequencies `f1` and `f2` are different for each algorithm and problem size, and their values can be found in Table II

Finally we wanted to isolate the effect on the cross-section of the very long time taken by the reboots of the operating system whenever the test hung. Figure 6 shows that including the `HangBoot` errors (see Table III) in the computation of the cross-section reduces it. The number of errors taken into account increases, but the total time of the test increases even more, due to the long time taken to reboot the board.

C. Error propagation

Error propagation is a very important issue in LU decomposition. In most cases the corruption of one element of the matrix due to a soft error propagates to a large number of elements of the final matrix. This is caused by the dependence in the computations performed during the decomposition, where the values of the elements depend on others previously computed. Therefore, this problem has been previously analyzed in the literature. For example, in [17], [18], [31] the authors studied the pattern of the error propagation of block versions of LU decomposition. In all three cases the authors studied the behaviour of the main steps of the algorithm and concluded that the rate and pattern of error propagation depend on the step where the soft-error occurs. Therefore, when designing error mitigation methods or ABFT versions of LU decomposition, different detection and correction schemes should be applied to the different steps of the algorithm. Usually, the authors perform a theoretical analysis or modelling of the error propagation based on the operations performed in each step of the algorithm and the dependence of the computations affecting the different elements of the matrix. Fault injection is used in [17], [18] in order to evaluate the coverage and overhead of the detection and correction schemes proposed. To the best of our knowledge no experimental evaluation of the error propagation of LU decomposition has been performed under radiation.

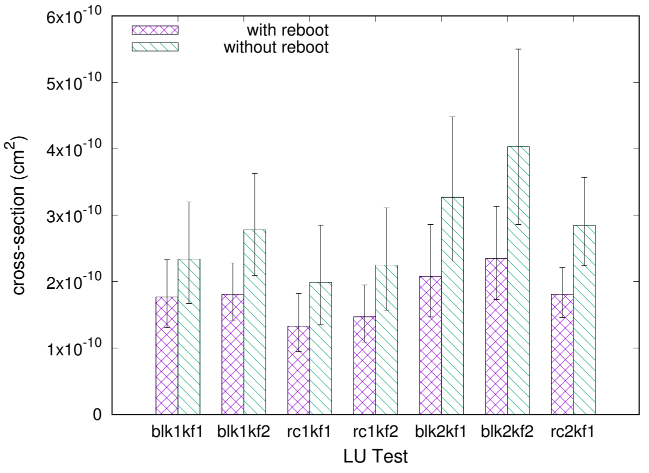

Fig. 6. Cross-section of the tests with and without including the effect of the hangs during the Jetson board reboot.

During our experiments we logged not only if each test got the correct result or hung, but also the number of wrong elements when an SDC occurred. Nevertheless, as the number of SDCs is small, we cannot perform a significant statistical analysis or detailed comparison of the behaviour of the algorithms.

Figure 7 shows the percentage of wrong elements in the 99 SDCs that were detected during the 5831 LU decompositions launched under radiation. The first bar shows the number of tests with a percentage of wrong elements in the result matrix between 0% and 1%, the second bar shows the number of tests with a percentage between 1% and 2%, and so on. We can see that in most cases, when an SDC occurs, the error propagates to less than 1% of the elements of the matrix. In 8 tests the error did not propagate at all and affected only one element of the result. In almost all cases, less than 20% of the elements are affected by the data corruption. Nevertheless, in 4 cases almost all the elements of the decomposition were different from the golden result.

Both algorithms show a similar behaviour regarding their error propagation. That is, in most cases the data corruption propagates to less than 10% of the elements.

V. Conclusions

In this work we have evaluated the reliability of a low-power GPU-accelerated SoC under proton irradiation. We have used as benchmark two very different CUDA parallel algorithms that perform LU decomposition on the GPU. By leveraging the frequencies of both CPU and GPU we have been able to

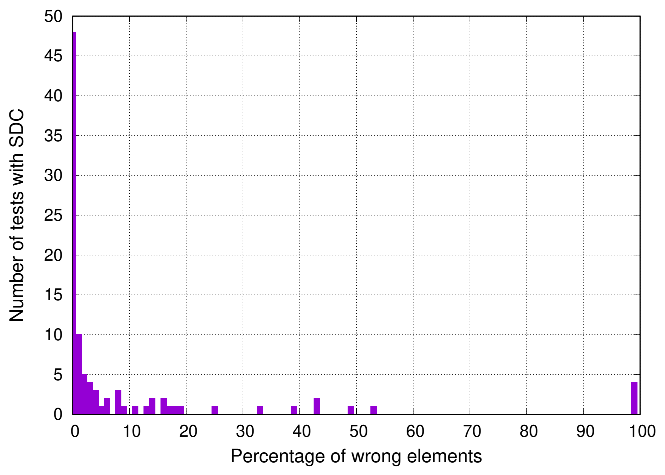

Fig. 7. Percentage of wrong elements in the tests with SDC.

compare the behaviour of both algorithms with problems of the same size and submitting the tests to the same radiation dose.

It is worth pointing out that the results of most of the LU decompositions are not affected by the radiation, and SDCs occur in less than 2% of the tests. However, the experiments show a substantial number of application hangs (close to 4%), and most of them forced a very slow reboot of the system in order to launch the next LU decomposition, which would be unacceptable in safety-critical applications. Besides radiation also hung more than one third of the booting processes, which forced a hardware reset of the board from the host controller. In order to reduce the very negative effect of the rebooting process a compact Operating System with a very fast booting should be used.

Results show that a better use of the resources of the GPU, such as its shared memory, can greatly improve the computational performance of LU decomposition. However, it also increases by about 15% its cross-section for all problem sizes. That is, the `block` algorithm is much faster than the `rc` algorithm, but more prone to radiation-induced errors. Our experiments also show that the processes running on the cores of the CPU are more sensitive to radiation-induced errors than the kernels running on the cores of the GPU. However, different tests and more extensive experiments are required to confirm this behaviour.

References

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

[2] M. Díaz, R. Guerra, P. Horstrand, E. Martel, S. López, J. F. López, and R. Sarmiento, “Real-time hyperspectral image compression onto embedded GPUs,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 8, pp. 2792–2809, 2019.

[3] L. Kosmidis, I. Rodriguez, Á. Jover, S. Alcaide, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in space-Latest project updates,” *Microprocessors and Microsystems*, vol. 77, p. 103143, 2020.

[4] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, “High-performance embedded computing in space: Evaluation of platforms for vision-based navigation,” *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, 2018.

[5] G. Furano and A. Menicucci, “Roadmap for on-board processing and data handling systems in space,” in *Dependable Multicore Architectures at Nanoscale*, pp. 253–281, Springer, 2018.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in *2009 IEEE international symposium on workload characterization (IISWC)*, pp. 44–54, IEEE, 2009.

[7] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, “SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation,” in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 249–258, IEEE, 2017.

[8] D. Oliveira, F. F. dos Santos, G. P. Dávila, C. Cazzaniga, C. Frost, R. C. Baumann, and P. Rech, “High-energy versus thermal neutron contribution to processor and memory error rates,” *IEEE Transactions on Nuclear Science*, vol. 67, no. 6, pp. 1161–1168, 2020.

[9] NVIDIA, “NVIDIA Tegra K1. A new era in mobile computing. NVIDIA Whitepaper,” January 2014.

[10] H. Wang, Q. Chen, L. Chen, D. M. Hiemstra, and V. Kirischian, “Single event upset characterization of the Tegra K1 mobile processor using proton irradiation,” in *2017 IEEE Radiation Effects Data Workshop (REDW)*, pp. 127–130, IEEE, 2017.

[11] 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 Transactions on Computers*, vol. 65, pp. 791–804, March 2016.

[12] D. A. G. De Oliveira, L. L. Pilla, M. Hanzich, V. Fratin, F. Fernandes, C. Lunardi, J. M. Cela, P. O. A. Navaux, L. Carro, and P. Rech, “Radiation-induced error criticality in modern HPC parallel accelerators,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 577–588, IEEE, 2017.

[13] F. F. dos 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 *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 289–298, IEEE, 2021.

[14] S. K. S. Hari, P. Rech, T. Tsai, M. Stephenson, A. Zulfiqar, M. Sullivan, P. Shirvani, P. Racunas, J. Emer, and S. W. Keckler, “Estimating silent data corruption rates using a two-level model,” *arXiv preprint arXiv:2005.01445*, 2020.

[15] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, “Analyzing and increasing the reliability of convolutional neural networks on GPUs,” *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.

[16] E. Yao, J. Zhang, M. Chen, G. Tan, and N. Sun, “Detection of soft errors in LU decomposition with partial pivoting using algorithm-based fault tolerance,” *The International Journal of High Performance Computing Applications*, vol. 29, no. 4, pp. 422–436, 2015.

[17] P. Wu, N. DeBardeleben, Q. Guan, S. Blanchard, J. Chen, D. Tao, X. Liang, K. Ouyang, and Z. Chen, “Silent data corruption resilient two-sided matrix factorizations,” in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 415–427, 2017.

[18] J. Chen, H. Li, S. Li, X. Liang, P. Wu, D. Tao, K. Ouyang, Y. Liu, K. Zhao, Q. Guan, *et al.*, “Fault tolerant one-sided matrix decompositions on heterogeneous systems with GPUs,” in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 854–865, IEEE, 2018.

[19] M. V. O'Bryan, K. A. LaBel, E. P. Wilcox, D. Chen, M. J. Campola, M. C. Casey, J.-M. Lauenstein, E. J. Wyrwas, S. M. Guertin, J. A. Pellish, *et al.*, “Compendium of Current Single Event Effects Results from NASA Goddard Space Flight Center and NASA Electronic Parts and Packaging Program,” in *2017 IEEE Radiation Effects Data Workshop (REDW)*, pp. 37–47, IEEE, 2017.

[20] S. M. Guertin and M. Cui, “SEE test results for the snapdragon 820,” in *2017 IEEE Radiation Effects Data Workshop (REDW)*, pp. 155–160, IEEE, 2017.

[21] S. M. Guertin, W. P. Parker, A. C. Daniel, and P. Adell, “Recent SEE Results for Snapdragon Processors,” in *2019 IEEE Radiation Effects Data Workshop*, pp. 157–161, IEEE, 2019.

[22] E. J. Wyrwas, “Proton Testing of nVidia Jetson TX1,” Tech. Rep. 20170009004, NASA Goddard Space Flight Center, october 2016.

[23] E. J. Wyrwas, “Proton Testing of nVidia Jetson TX2,” Tech. Rep. 20190031856, NASA Goddard Space Flight Center, July 2019.

[24] E. Wyrwas, K. A. LaBel, M. Campola, and M. O'Bryan, "Guidance on Standardizing GPU Test Approaches," in *Nuclear and Space Radiation Effects Conference (NSREC)*, pp. 116–119, 2018.

[25] W. S. Slater, N. P. Tiwari, T. M. Lovelly, and J. K. Mee, "Total Ionizing Dose Radiation Testing of NVIDIA Jetson Nano GPUs," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 639–641, IEEE, 2020.

[26] D. Hiemstra, C. Jin, Z. Li, R. Chen, S. Shi, and L. Chen, "Single Event Effect Evaluation of the Jetson AGX Xavier Module Using Proton Irradiation," in *2020 IEEE Radiation Effects Data Workshop (in conjunction with 2020 NSREC)*, pp. 31–34, IEEE, 2020.

[27] Y. Morilla, P. Martín-Holgado, A. Romero, J. Labrador, B. Fernández, J. Praena, A. Lindoso, M. García-Valderas, M. Peña-Fernández, and L. Entrena, "Progress of CNA to become the Spanish facility for combined irradiation testing in aerospace," in *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pp. 250–254, IEEE, 2018.

[28] H. Kettunen, V. Ferlet-Cavrois, P. Roche, M. Rossi, A. Bosser, G. Gasiot, F.-X. Guerre, J. Jaatinen, A. Javanainen, F. Lochon, *et al.*, "Low energy protons at RADEF-application to advanced eSRAMs," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, pp. 147–150, IEEE, 2014.

[29] NVIDIA, *CUDA C++ Programming Guide. PG-02829-001_v11.2. Design Guide*, November 2021.

[30] NVIDIA, *Profiler. DU-05982-001_v11.4. User's Guide*, June 2021.

[31] T. Davies and Z. Chen, "Correcting soft errors online in LU factorization," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pp. 167–178, 2013.