Reliability Evaluation of Embedded GPGPUs for Safety Critical Applications

D. Sabena, *Student Member, IEEE*, L. Sterpone, *Member, IEEE*, L. Carro, *Senior Member, IEEE*, and P. Rech, *Member, IEEE*

***Abstract*—Thanks to the capability of efficiently executing massive computations in parallel, general purpose graphic processing units (GPGPUs) have begun to be preferred to CPUs for several parallel applications in different domains. Two are the most relevant fields in which, recently, GPGPUs have begun to be employed: high performance computing (HPC) and embedded systems. The reliability requirements are different in these two applications domain. In order to be employed in safety-critical applications, GPGPUs for embedded systems must be qualified as reliable. In this paper, we analyze through neutron irradiation typical parallel algorithms for embedded GPGPUs and we evaluate their reliability. We analyze how caches and threads distributions affect the GPGPU reliability. The data have been acquired through neutron test experiments, performed at the VESUVIO neutron facility at ISIS. The obtained experimental results show that, if the L1 cache of the considered GPGPU is disabled, the algorithm execution is most reliable. Moreover, it is demonstrated that during a FFT execution most errors appear in the stages in which the GPGPU is completely loaded as the number of instantiated parallel tasks is higher.**

***Index Terms*—General purpose graphic processing units (GPGPUs), radiation testing, single-event effects.**

I. Introduction

THE very high computational power of general purpose graphics processing units (GPGPUs) combined with low cost, reduced power consumption and flexible software development platforms, are pushing GPGPUs adoption not only for graphical applications and high performance computing (HPC) market but also for embedded devices (i.e., mobile applications). Moreover, low-power embedded GPGPUs are becoming increasingly interesting also for mission-critical applications, such as automotive, avionics, space, and biomedical, where the parallelism capabilities of GPGPUs would be very suitable [1]. As an example, the advanced driver assistance systems (ADAS), which are increasingly common in cars, make an extensive usage of images or radar signals coming from external cameras and sensors to detect possible obstacles triggering the breaking system. The European space agency (ESA) is employing low power GPGPUs for images compression on the future version of the COROT satellite [2] to reduce the bandwidth required to send data to ground, while Airbus S.A.S. is finalizing the ARAMIS project, aimed at the integration of all the electronics required to implement the collision avoidance system into a single board including a GPGPUs core [3]. However, up to now the European Aviation Security Agency (EASA) does not accept multicores chips with more than two cores, such as GPGPUs, on an aircraft. The main reason for this constraint on multicores usage from EASA is that a recognized evaluation protocol is required.

Given their high degree of parallelism, in the GPGPUs environment is not so difficult to implement (in software) traditional soft error mitigation techniques, i.e., duplication with comparison (DWC), and triple modular redundancy (TMR); however, their size, their complexity, and some of their components, could make them particularly sensible to soft errors [1], [4], [5], [6]. Moreover, while effective soft-error hardening techniques already exist for systems based on traditional CPUs, similar solutions for GPGPUs architectures are still under investigation and under development [4], [7].

Our paper moves on the direction of understanding the reliability of embedded GPGPUs, giving novel insight on their behaviors when exposed to ionizing radiation. The main contribution of this paper is the reliability analysis of a FFT algorithm designed for embedded GPGPUs through neutron beam radiation experiments. The analysis is performed giving particular attention to caches and thread scheduler corruption. More in particular, caches have been demonstrated to be a critical module in terms of reliability [4], since these memories have a very important role during the parallel algorithms execution. Caches memories have a key role on GPGPUs architectures as they significantly improve the performance allowing parallel tasks to share data. Consequently, caches represent a critical component of GPGPUs computing, since an error in a data induced by a radiation could affect the computation of all the cores, thus compromising the whole algorithm execution. Clearly, this behavior is not acceptable if GPGPUs are employed for safety critical applications. This represents the motivation of the work proposed in this paper: we present the soft error sensitiveness data of a typical parallel algorithm, executed with different GPGPUs cache configurations. Moreover, we present data about the same algorithm, but executed with different threads distribution.

In order to be fully compliant with the degree of parallelism (DOP) adopted by embedded GPGPUs, we considered the progressive reduction of the FFT parallel tasks from iteration to

Manuscript received July 11, 2014; revised September 05, 2014 and October 06, 2014; accepted October 07, 2014. Date of publication October 30, 2014; date of current version December 11, 2014.

D. Sabena and L. Sterpone are with the Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, Torino 10129, Italy (e-mail: davide.sabena@polito.it; luca.sterpone@polito.it).

L. Carro and P. Rech are with the Instituto de Informatica, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil (e-mail: carro@inf.ufrgs.br; prech@inf.ufrgs.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2014.2363358

iteration, depicting both overall GPGPU cross-section and the error rate for all the FFT stages. The progressive reduction of the parallel tasks is a common behavior of several GPGPUs algorithms [e.g., the breadth-first search (BFS) algorithm, where the 30% of the GPGPU execution time is managed by one thread], and it is called *underutilized parallelism in GPGPU computing* [8]. Consequently, even if in this paper the analysis is performed only on the FFT algorithm, the results and discussion are extendable to any other similar parallel algorithms for embedded GPGPUs.

The proposed analysis has been implemented on a device by NVIDIA with Fermi architecture and the data has been acquired in an extensive radiation campaign at the ISIS facility in the Rutherford Appleton Laboratories (RAL) in Didcot, U.K. The gathered experimental results provide interesting data about the sensitivity to radiation of GPGPUs device when different configurations are considered.

The obtained results show that, if the L1 cache of the considered GPGPU is disabled, the FFT algorithm execution has the lowest cross section. Instead, when the number of parallel threads managing the same algorithm is reduced, the execution is less reliable. We correlate this effect to a different usage of the GPGPU caches due to a different number of threads running in the GPGPU itself. Finally, we experienced a greater number of errors when the algorithm exploits all the parallel resources of the considered GPGPU, i.e., in the first stages of the FFT algorithm, when the number of parallel tasks is higher.

The rest of the paper is organized as follow: Section II overviews some previous works in the area; Section III outlines the GPGPU device we address, while in the Section IV the used FFT algorithm is explained; Section V presents the experimental setup we used to retrieve the data, and Section VI explains and comments experimental data. Finally, Section VII draws some conclusions and outlines our current work.

II. Related Work

Radiation effects are a concern for the reliability of electronic devices not only in harsh radiation environments such as space or avionic, but also at ground level [4]. Today, device technology shrinking has led to a drastic reduction of critical charges in logic gates and memory cells that results in a higher sensitivity to soft-errors induced by ionizing radiation. In the last years, an increasing research interest has been devoted to the soft-error sensitiveness evaluation of GPGPUs [1]. A first method for the evaluation of their radiation sensitivity has been proposed in [9], where authors present an analytical model for the evaluation of single event upset (SEU) occurrences on GPGPUs depending on the memory and register usage of the running application. Recently, GPGPUs have been evaluated using spallation neutron sources that provide the user with an atmospheric-like spectrum. A preliminary experimental setup for the execution of neutron radiation test of a GPGPU has been proposed in [10]. The authors describe a low-cost but effective setup providing some guidelines on how to test GPGPUs, focusing on the constraints imposed by the radiation source and the device connections with the host computer controlling the experiment. The first radiation test results demonstrate that both memory and logic resources of a GPGPU may be corrupted by atmospheric neutrons. Being characterized by high-performance computational units such as fixed and floating point units, a further evaluation of the probabilities that radiation-induced errors may affect the mantissa, the exponents or the sign has been evaluated in [7]. A strong variation on the output error rate has been observed when different types of data are elaborated showing a higher sensitivity for the resources used by the mantissa. Moreover, radiation evaluations also addressed the distribution of the computational workload on GPGPUs such as the thread distribution and their relation to multiple output errors [11], [12].

Aside from testing approaches, software-based hardening solutions have also tentatively addressed in [7] where authors developed an optimized software-based hardening strategy exclusively oriented to matrix multiplication applications. Researches have also investigated the possibility of using software-based self-test (SBST) programs in order to detect and localize permanent faults in GPGPUs: a preliminary work proposing a possible SBST solution has been presented in [13].

Considering the characterization of the GPGPUs memories with respect to radiation, in [4] we provide a set of methods based on monitoring programs capable to initialize and observe the soft-error effects on the different memory levels belonging to GPGPUs. The main scientific contribution provided by this method is the possibility of effectively evaluating the impact of soft-errors occurring into the arrays and control circuitry of GPGPUs cache memories through monitoring programs. This allows evaluating the expected soft-error rate of memory cells belonging to shared memories and caches of L1 and L2 levels, independently from the running application and without any intrusiveness into the radiation test data obtained [4].

In [1] we propose the first work aimed at the evaluation of the traditional soft-error mitigation techniques developed for traditional CPU architectures when applied on the GPGPUs environment. In particular, the addressed techniques are time and thread redundancy applied to a common parallel algorithm. The results have been acquired with a campaign of radiation experiments, and they demonstrate that reach a good degree of fault tolerance, with respect to the minimization of the introduced overhead, in GPUPUs environments is not a simple task.

Recent previous works have also been done to evaluate the reliability of HPC-oriented GPGPUs [5], [7]. Embedded low-power GPGPUs have a similar architecture than HPC devices, but have a different programming paradigm. If on one side, HPC applications have performances as a major concern, so the GPGPU is likely to be always fully loaded with parallel processes; on the contrary, embedded GPGPUs have to take care on the power consumption and must respect tight constraints on area and device resources. For this reason GPGPU devices should have limited area usage and, in order to save energy, optimized computational cores usage (e.g., a temporarily not used core is switched off in idle state).

III. GPGPU Internal Structure

The GPGPU architecture we address in this paper is the NVIDIA *Fermi* Architecture [14]. The Fermi GPGPU family is composed of an array of *streaming multiprocessors* (SMs), as shown in Fig. 1, each of which has the ability of execute

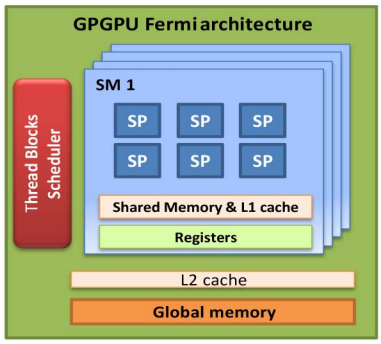

Fig. 1. GPGPU Fermi Architecture is composed of several SMs.

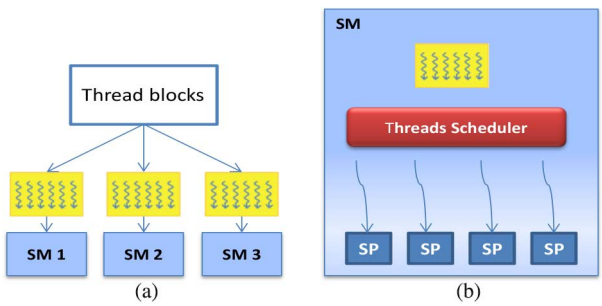

Fig. 2. Block of thread is forwarded to each SM by the ThreadBlock scheduler (a); each SP receives one thread at a time (b).

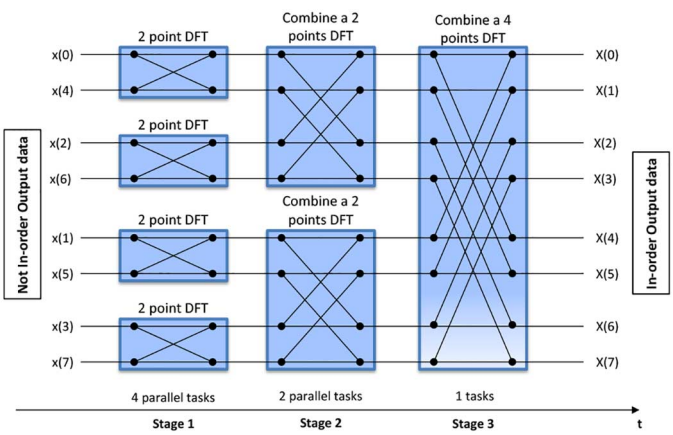

Fig. 3. Representation of the butterflies for each stage of the FFT algorithm, independently from the GPGPU configuration. The number of parallel tasks at each stage (highlighted with boxes in the figure) decreases exponentially from a stage to the following one.

several threads in parallel. The SMs are composed of several computational units, called *NVIDIA CUDA cores* or *streaming processors* (SPs), where each core manages a thread at a time (0). The Thread Blocks Scheduler assigns the thread blocks to the SMs while the Thread Scheduler inside a SM assigns a thread to a SP [see Fig. 2 (a) and (b)].

Each SM accesses to a global memory and both reading and writing operations are cached in the L2 cache. The maximum L2 cache size is 768 KB. Each SP can access the global memory, but only the read operations are cached in L1; moreover, all the shared memory locations are accessible by all and only the CUDA cores embedded in the same SM and these accesses are

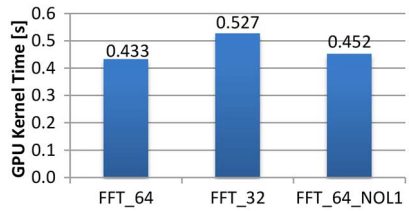

Fig. 4. Total GPGPU Kernel Time increases in the cases of 32 thread (FFT_32) and disabling the L1 cache (FFT_64_NOL1).

not cached. The maximum L1 cache size can be 16 or 48 KB, depending on the user configuration [4].

From the software point of view, CUDA extends C language by allowing the programmer to define C-based functions, called *kernels*, that, when called, are executed in parallel by N different CUDA threads; the NVIDIA CUDA programming model assumes that the threads execute on a physical separate *device* (i.e., the GPGPU) that operates as a coprocessor of the *host* (i.e., the CPU) running a controlling program. CUDA is particularly suitable for embedded GPGPUs as the user can easily define directly on the code the portion of the application to be executed on the host and the portion that requires GPGPU acceleration [14].

IV. The FFT Algorithm Implementation

The main goal of our study is to evaluate how parallel algorithms behave when executed in embedded GPGPUs. Additionally, we investigate how the radiation-induced errors propagate in the parallel algorithm till reaching the output. Such a study is of great interest as it highlights the weaker parts of the code that should be hardened to increase the device reliability. Additionally, we analyze the effects of different threads and caches distributions on the GPGPUs output error rate. In this way it is possible to understand which GPGPU configuration ensures the highest level of reliability under several constraints.

We chose as a benchmark the Cooley–Tukey algorithm [17], the most common FFT implementation for embedded applications. Cooley–Tukey algorithm allows to reduce the computations complexity from $\mathrm{O}(\mathrm{N}^2)$ to $\mathrm{O}(\mathrm{N} * \log_2 \mathrm{N})$, where N is the number of the input data. The input data of the tested algorithm is composed of 65 536 complex numbers, each of which represented with 2 float values.

First of all, a CUDA version of that algorithm has been implemented, taking into account the parallel resources, the memory usage, and the programming paradigm provided by the NVIDIA environment.

In Fig. 3, the degree of parallelism (DOP) of each stage is outlined. At each stage a FFT *butterfly unit* [18] combines the results of two smaller *discrete Fourier transforms (DFTs)* into a large DFT [19]. The butterfly units on a stage can be executed in parallel, while to start the next stage computation it is necessary to wait for the current stage computation to be completed. The number of parallel tasks required for computation, then, decreases exponentially from a stage to the following one.

In the tested code the number of stages required to solve the problem is 16, since the input data are 65 536 complex numbers. The stages management is a task of the host device that

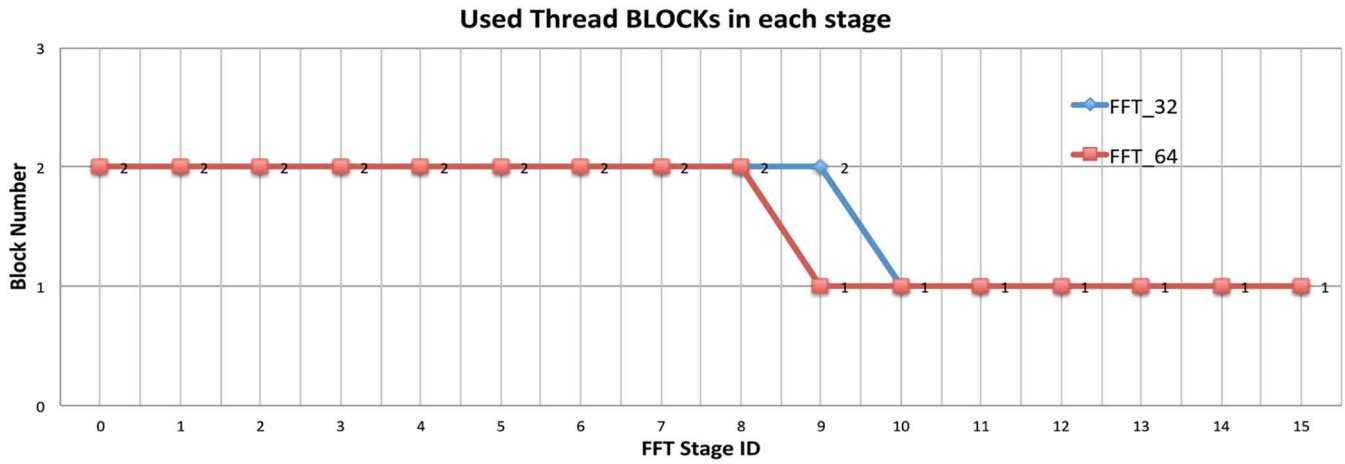

Fig. 5. Number of the used Thread Blocks per each FFT Stage.

controls the GPGPU, which computes, per each stage, the optimal number of threads and of thread blocks to be instantiated (considering the addressed GPGPU configuration). In the ideal solution for performances each thread should compute only one butterfly unit of the Cooley–Tukey algorithm [18]. Nevertheless, since the required number of threads would be too high (almost 32 K in the case of 65 536 input numbers), in the initial stages of the computation each thread is asked to compute one or more butterfly units. In the initial stages of the algorithm the GPGPU is then fully loaded, while in the final stages the number of the parallel processes needed decreases, becoming lower than the number of computing units available in the NVIDIA Quadro GPGPU. In the last stage, for instance, a single process is instantiated (see Fig. 3). It is worth noting that such a behavior of having the GPGPU fully loaded only in the early stage of computation is typical of embedded GPGPUs. On the contrary, GPGPUs used in high performance computing usually are continuously loaded to take full advantage of their computational capabilities.

In the FFT algorithm execution that we address in this paper, the L1 caches are always fully exploited (when they are enabled), since the number of input data is big (65 536 complex numbers); consequently, the reliability impact of the L1 caches in the different steps of the algorithm execution is the same.

To characterize the behavior of embedded GPGPUs we implement the FFT algorithm instantiating a maximum of 2 thread blocks and 64 threads per block (*FFT_64*), and then 2 thread blocks and 32 threads per block (*FFT_32*). As 2 SMs are available in the tested GPGPU, the block scheduler is not activated. *FFT_32* does not require thread scheduling, since the 32 instantiated threads can run at the same time in parallel given that 48 are available per SM in the Quadro 1000M GPGPU. Vice versa, for the *FFT_64* configuration the thread scheduler is mandatory since the parallel threads instantiated exceed the maximum threads that can be executed in parallel. Moreover, we tested the FFT algorithm with 2 thread blocks, 64 threads per block and the L1 cache of each SM disabled (*FFT_64_NOL1*). Disabling the L1 may sensibly reduce the number of errors observed at the output. Even if performances are likely to be affected when L1 is disabled (see Figs. 4 and 7), on safety critical applications the execution time is not as critical as reliability.

The considered configurations have different execution times. In Fig. 4, we reported the total GPGPU Kernel time for the different GPGPU configurations: *FFT_32* kernel time increases of about 21.8% with respect to *FFT_64*, while disabling the L1 cache increases execution time of only about 4.5%. The performances degradation is not remarked as the L1 cache, although completely filled during computation (the tested FFT uses about 1 MB of data), is used only for reading operations. Consequently, the writing operations (about 65 536 for each stage) performed by the FFT algorithm don't involve the L1 cache. As reported in Fig. 7, the FFT_32 and FFT_64 have the same kernel time after the stage 9, but in the stages 1 to 9, the kernel time of the FFT_32 is almost the double if compared with the FFT_64 configurations.

In Fig. 5 and Fig. 6 we present the profile of the used Thread Blocks and of the number of threads used in each block, considering the FFT stages. As the reader can notice, the number of the parallel tasks decreases after the stage 9, in case of the FFT_32 configuration is used, and after the stage 10, in case the used configuration is FFT_64. In these figures we have not reported the profile of the FFT_64_NOL1 configuration since this configuration is equal, in terms of used threads and blocks, to the FFT_64.

Considering the GPGPU utilization shown in Figs. 5 and 6, and 7, we expect an higher number of errors in the first stages of the FFT algorithm execution, i.e., when all the computational resources of the GPGPU under test are fully exploited.

Finally, considering the FFT behavior, the analysis we present in this paper is valid for all the GPGPU algorithms with a similar usage of the L1 cache, i.e., the algorithms that initially acquire the required data from the L1 cache, and then make a long computation (in terms of clock cycles) without interacting more with the cache itself. It is clear that the data reported in this paper are not completely suitable for the algorithms that continuously access the data stored in the considered cache.

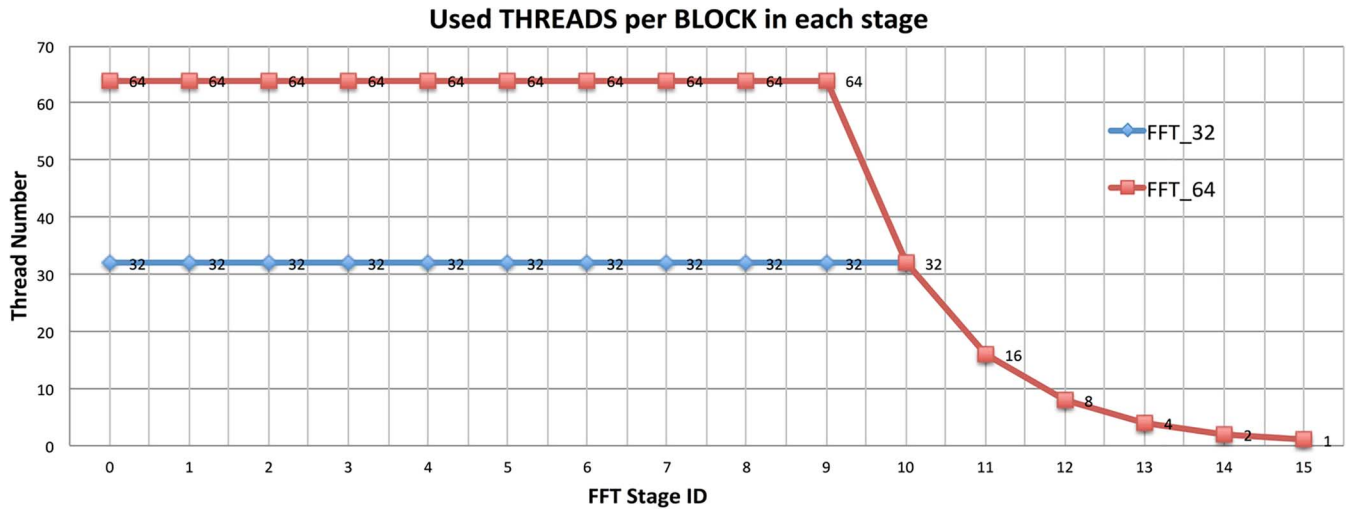

Fig. 6. Number of the used Threads per Block in each Stage of the FFT algorithm.

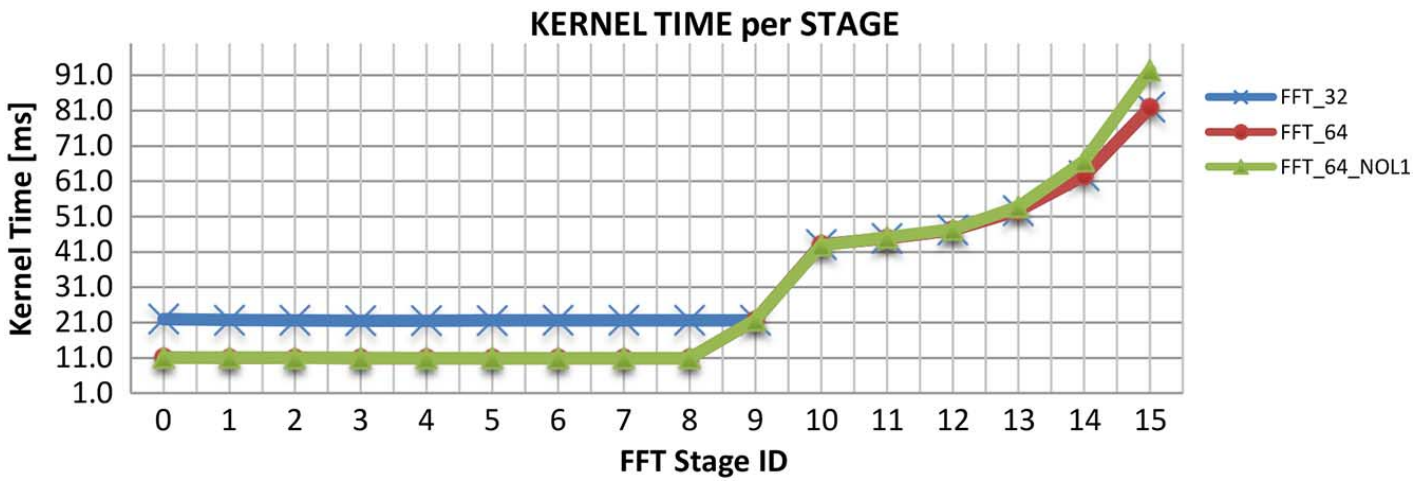

Fig. 7. GPU Kernel time per each FFT Stage.

V. Experiment Setup

The evaluation board we used for the radiation test campaigns is the CARMA Development Kit [15]. The CARMA DevKit features a Qseven NVIDIA Tegra 3 Quad-core ARM A9 CPU and the NVIDIA Quadro 1000M GPGPU with 2 SMs of 48 SPs each for a total of 96 CUDA cores [16]. The NVIDIA Quadro 1000M has been selected for our experiments since the restricted size combined with low power consumption make this GPGPU a fully compliant test case for embedded and mobile systems.

Radiation experiments were performed at the VESUVIO neutron facility at ISIS, Rutherford Appleton Laboratories (RAL), Didcot, UK. We irradiated the device with the available spectrum that has been already demonstrated to be suitable for emulating the atmospheric neutron flux [20]. The available flux was of about $3.89 \cdot 10^4$ n/(cm$^2 \cdot$ s). Irradiation was performed at room temperature with normal angle of incidence and the beam was focused on a spot with a diameter of 2 cm plus 1 cm of penumbra. Spot size is sufficient to uniformly irradiate the GPGPU chip, leaving the ARM processor, the control circuitry, and critical peripherals out of the beam. This is essential for preventing neutron-induced errors on power switches, which may compromise the experiment, and on the PCI express protocol that manages the data communications between the CPU and the GPGPU.

VI. Experimental Results

Fig. 8 shows a detailed analysis of the observed errors propagation in the FFT algorithm reporting the error rate of each stage of the FFT algorithm (i.e., the number of errors generated in a stage divided by the time required to compute the stage itself). As it can be noticed, in the initial stages of the algorithm (from stage 1 to stage 10) the error rate is much greater than in the later stages. In stages from 0 to 10 the GPGPU is fully loaded since the instantiated parallel processes exceeds the GPGPU parallel resources and the thread schedulers is active. From stage 11 to 15 the number of threads instantiated is lower than the number of available SPs, till the extreme case of stage 15 in which a single thread is active (see Fig. 3). The area exposed to radiation is then the whole GPGPU in the first 11 stages, and then decreases making it more likely for the GPGPU to be corrupted when fully loaded. Moreover, in some stages of the FFT algorithm, we haven't detected any error during the execution of

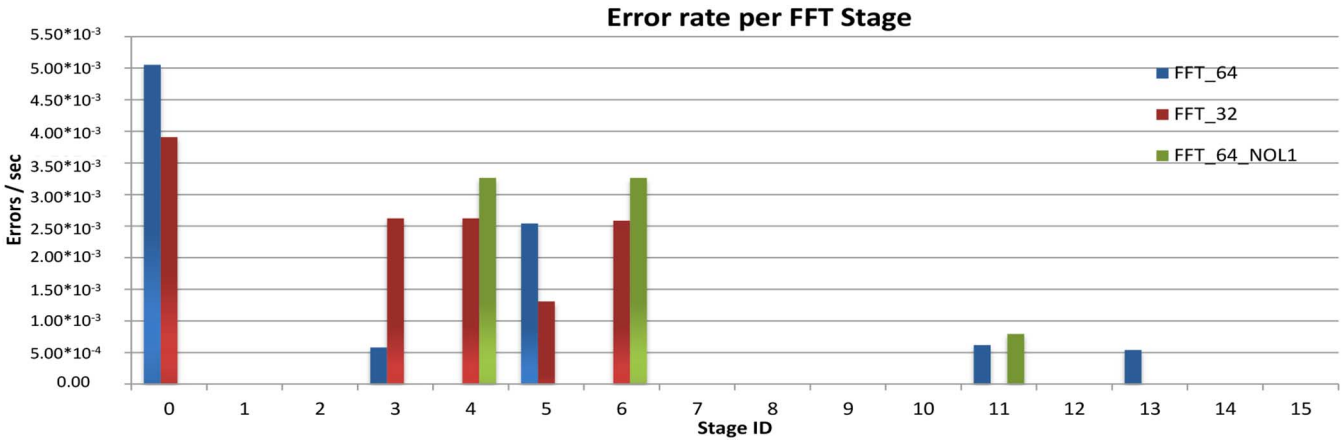

Fig. 8. Error rate of the different stages of the FFT algorithm, for each GPGPU configuration.

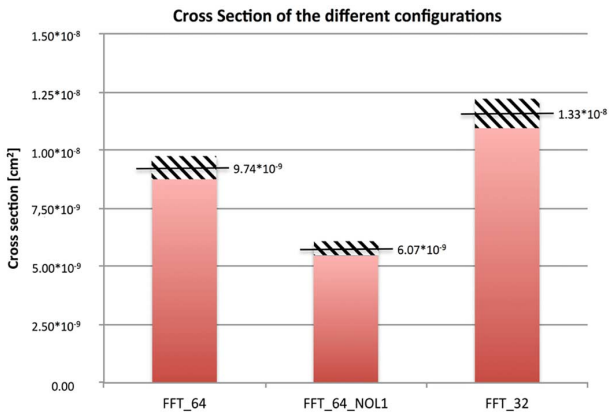

Fig. 9. Cross section of the different configurations of the FFT algorithm. The confidence interval (drawn with oblique lines) has been calculated considering a statistical error of 10% in the worst case.

our experiments (in all the considered configurations). This is mainly due to the fact that the error rate was low if correlated to the area exposed to neutron radiation.

Fig. 9 shows the experimentally obtained cross sections for the three different tested GPGPU configurations (FFT_64, FFT_64_NOL1, and FFT_32). The cross section was measured dividing the number of observed error per second by the average neutron flux. For each cross section value we reported also the confidence interval (drawn with oblique lines) calculated considering a statistical error of 10% in the worst case. As it can be noticed, even if the algorithm and thus the workload is the same, the three configurations show different cross sections. Such a different sensitivity to radiation relies on the amount of resources required to compute the solution more than in the workload that, again, is constant in the three tested cases.

It is clear from Fig. 9 that, as expected, disabling the L1 caches reduces the cross section of the algorithm of 38%, while the performance overhead is not so relevant (about 4.5%, as we explained in the previous section and in Fig. 4). As said, the significant decrease of the cross section is mainly due to the fact that the L1 cache is a very sensitive memory [4]. The contribution of L1 caches corruption in the overall SM radiation sensitivity is not negligible, especially when the algorithms perform a large amount of memory accesses. Obviously, when disabling L1 caches the execution time is increased (see Fig. 4). Nevertheless, in safety critical applications, performance may be sacrificed to gain higher reliability. It is worth noting that even if performances are not a major concern, having a longer execution time will force the GPGPU to be irradiated for a longer time before solving the assigned task. As demonstrated in [22], it is essential to carefully evaluate if disabling caches actually improve the device reliability.

FFT_32 has a 25% higher cross section than *FFT_64*. Limiting the number of parallel threads per block seems to increase the GPGPU sensitivity. We believe this behavior to be mainly due to a different data distribution in the GPGPU caches. The 32 instantiated threads per block in *FFT_32* are always active in a SM (each SM is composed of 48 SP); consequently, the data used by the threads in *FFT_32* is present in the L1 cache for all computation. On the contrary, in *FFT_64*, the number of instantiated thread (i.e., 64) imposes the thread scheduler to continuously swap the active threads. The data in the L1 cache is then refreshed frequently. If data is refreshed frequently, the *time vulnerability factor* [12] (TVF, i.e., the portion of time during which data is critical and a corruption propagates to the output) becomes smaller, and possible error caused by a radiation is more likely to corrupt obsolete data. When threads scheduling is enabled, as in *FFT_64*, a refresh of the data stored in the L1 cache is expected and, thus, the cross section is lower.

As we already highlight in the previous paragraph of this section, comparing the FFT_32 and FFT_64 configurations, it is possible to notice that FFT_32 has an higher cross section and an higher kernel time. It must be underlined that these two behaviors are not correlated, since the cross section values have been calculated dividing the number of observed errors per second by the average neutron flux (about $3.89 \cdot 10^4$ n/(cm$^2 \cdot$s)) present during our experiments. Consequently, the higher execution time of the FFT_32 configuration is not the reason of the higher cross section. As we have explained in the previous paragraph, the reason is the different time vulnerability factor of the data present in the L1 caches due to the different threads number executing the same FFT algorithm.

The most interesting point highlighted by the reported data is that, when the reliability of a GPGPU is a concern, all the stages

of the algorithms should be carefully designed. The designers of safety critical applications to be executed on GPGPUs should, in fact, take into account that the reliability of the GPGPU applications varies according to the number of parallel tasks running at the same time in the GPGPU itself, as shown in Fig. 3. Moreover, hardening techniques should be tuned taking into account that most errors occurs in the early stages of the algorithm, when the GPGPU is fully loaded.

From data depicted in Fig. 3 and Fig. 9, it is clear that disabling the L1 caches (contained in the SMs embedded in the considered GPU), the impact in terms of performance is low, but the reliability of the GPU device increases significantly. This is a very important aspect when the design of safety critical applications is addressed: the tradeoff between the reliability improvement and the introduced overhead is one of the most important parameter that has to be considered. With the analysis presented in this paper, it is possible to understand that, if the algorithm has to be used in safety critical GPGPU applications, disabling the L1 caches represent a good solution in terms of reliability and performance overhead.

VII. Conclusion and Future Work

In this paper we analyzed the reliability of embedded GPGPUs and experimentally determine the cross section of different embedded GPGPU configurations running the FFT parallel algorithm based on the Cooley–Tukey method. The results show that the most reliable configuration is obtained disabling the L1 cache memory. Moreover, on embedded GPGPUs oriented to safety critical applications the number of parallel processes to instantiate should be carefully design to avoid undesired behaviors.

As future work, we plan to extend the present work to other algorithms where the GPGPU is used in a similar way (e.g., the BFS algorithm) in order to confirm the data proposed in this paper.

References

[1] D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, and L. Carro, "On the evaluation of soft-errors detection techniques for GPGPUs," in *Proc. 8th Design Test Symp. (IDT)*, Dec. 2013, pp. 1–6.

[2] "ESA corot mission documentation," [Online]. Available: www.esa.int/Our_Activities/Space_Science/COROT

[3] O. Bender, "HiPEAC," 2014 [Online]. Available: http://www.across-project.eu/workshop2013/121108_ARAMIS_Introduction_HiPEAC_WS_V3.pdf

[4] D. Sabena, M. Sonza Reorda, L. Sterpone, P. Rech, and L. Carro, "Evaluating the radiation sensitivity of GPGPU caches: New algorithms and experimental results," *Elsevier Microelectron. Rel.*, 2014, to be published.

[5] 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. (DSN)*, Atlanta, GA, USA, 2014, pp. 455–466.

[6] L. Battista Gomez, F. Capello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. Sonza Reorda, "GPGPUs: How to combine high computational power with high reliability," in *Proc. IEEE Design, Autom., Test Eur. (DATE)*, 2014, pp. 1–9.

[7] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on GPUs," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 4, pt. 1, pp. 2797–2804, Aug. 2013.

[8] H. Jeon and M. Annavaram, "Warped-DMR: Light-weight error detection for GPGPU," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2012, pp. 37–47.

[9] J. Tan, N. Goswami, T. Li, and X. Fu, "Analyzing soft-error vulnerability on GPGPU microarchitecture," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Nov. 2011, pp. 226–235.

[10] P. Rech, C. Aguiar, R. Ferreira, C. Frost, and L. Carro, "Neutron radiation test of graphic processing units," in *Proc. IEEE 18th Int. On-Line Testing Symp. (IOLTS)*, 2012, pp. 55–60.

[11] P. Rech, C. Aguiar, C. Frost, and L. Carro, "Experimental evaluation of thread distribution effects on multiple output errors in GPUs," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2013, pp. 27–32.

[12] N. Seifert and N. Tam, "Timing vulnerability factors of sequentials," *IEEE Trans. Device Mater.*, vol. 4, no. 3, pp. 516–522, Sep. 2004.

[13] S. Di Carlo, G. Gambardella, M. Indaco, I. Martella, P. Prinetto, D. Rolfo, and P. Trotta, "A software-based self-test of CUDA fermi GPUs," in *Proc. IEEE Eur. Test Symp.*, May 2013, pp. 33–38.

[14] :Fermi architecture documentation," [Online]. Available: www.nvidia.com/object/fermi-architecture.html

[15] "DEVKIT features," [Online]. Available: shop.seco.com/carma-devkit.html

[16] "Nvidia Quadro features," [Online]. Available: http://www.nvidia.com/object/quadro-mobile-features-benefits.html

[17] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comp.*, vol. 19, pp. 297–301, 1965.

[18] Y. Lee *et al.*, "An efficient FFT algorithm based on building on-line butterfly sub-structure," in *Proc. 4th Int. Conf. Signal Process. Proceedings (ICSP)*, 1998, pp. 97–100.

[19] M. Fallahpour, C. Lin, M. Lin, and C. Chang, "Parallel one- and two-dimensional FFTs on GPGPUs," in *Proc. Int. Conf. Anti-Counterfeiting, Security Identification (ASID)*, Aug. 2012, pp. 1–5.

[20] M. Violante *et al.*, "A new hardware/software platform and a new 1/E neutron source for soft error studies: Testing FPGAs at the ISIS facility," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pt. 2, pp. 1184–1189, Aug. 2009.

[21] P. Rech, T. D. Fairbanks, H. M. Quinn, and L. Carro, "Threads distribution effects on graphics processing units neutron sensitivity," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 6, pt. 1, pp. 4220–4225, Dec. 2013.

[22] T. Santini, P. Rech, G. L. Nazar, L. Carro, and F. R. Wagner, "Reducing embedded software radiation-induced failures through cache memories," in *Proc. IEEE Eur. Test Symp.*, 2014, pp. 1–6.