This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units

Daniel A. G. Oliveira, *Student Member, IEEE*, Laercio L. Pilla, *Member, IEEE*, Thiago Santini, *Student Member, IEEE*, and Paolo Rech, *Member, IEEE*

***Abstract*—Graphics Processing Units (GPUs) are increasingly attractive for both safety-critical and High-Performance Computing applications. GPU reliability is a primary concern for both the automotive and aerospace markets and is becoming an issue also for supercomputers. In fact, the high number of devices in large data centers makes the probability of having at least a device corrupted to be very high.**

In this paper, we aim at giving novel insights on GPU reliability by evaluating the neutron sensitivity of modern GPUs memory structures, highlighting pattern dependence and multiple errors occurrences. Additionally, a wide set of parallel codes are exposed to controlled neutron beams to measure GPUs operative error rates. From experimental data and algorithm analysis we derive general insights on parallel algorithms and programming approaches reliability.

Finally, Error-Correcting Code, Algorithm-Based Fault Tolerance, and Duplication With Comparison hardening strategies are presented and evaluated on GPUs through radiation experiments. We present and compare both the reliability improvement and imposed overhead of the selected hardening solutions.

***Index Terms*—GPU, fault-tolerance, neutron sensitivity, parallel processors, reliability**

I. Introduction

In recent years, Graphics Processing Units (GPUs) vendors have been expanding their markets to safety-critical and High-Performance Computing (HPC) applications. Each industry has its reasons for using GPUs, such as increasing computational speed, low device cost, increased energy efficiency, and flexible development platforms. Low-power GPUs are part of projects implementing the Advanced Driver Assistance Systems (ADAS), which analyzes camera or radar signals to detect obstacles and activate the car brakes to prevent collisions [1]. Efficient parallel processing is attractive to compress images in satellites and reduce the bandwidth necessary to send them to ground [2]. On aircrafts, GPUs are studied to integrate all the circuitry necessary to implement collision avoidance systems [3]. On the HPC scenario, the two currently most performing among the TOP500 supercomputers (Tianhe-2, National Super Computer Center, Guangzhou, China and Titan, Oak Ridge National Laboratory, TN, USA) are composed of several thousands of accelerators (Intel Xeon-Phi and NVIDIA K20, respectively) [4]. The reason for choosing accelerators is dictated by their highly parallel structure, which allows the execution of several tasks in parallel at high speed.

Nowadays reliability is an issue not only for safety-critical systems but also for HPC. GPUs for HPC are cutting-edge devices built with advanced technologies and, thus, may be particularly susceptible to radiation-induced corruption. Additionally, the pursuit of extreme performances may exacerbate the GPU error rate [5]. Large-scale scientific applications are often very long-running (a single simulation may take from a few hours to a couple of days) and involve a high number of devices (e.g., Titan is composed of more than $18,000$ GPUs). Due to the large-scale and the long duration, leadership scientific applications may encounter interruptions due to application crashes or system hangs as well as Silent Data Corruption (SDC) in the output.

As we approach exascale, the resilience challenge will become even more critical due to an increase in system-scale [6]. GPUs are predicted to be a part of the projected path to exascale due to their capability to offer more FLOPS/W compared to CPUs. Therefore, understanding the nature of GPU errors is critical for operating today's large-scale HPC centers as well as designing future extreme-scale systems.

While CPU reliability studies have produced efficient error detection and correction mechanisms implemented at architectural level, such as the Machine Check Architecture (MCA) [7], [8], GPU reliability is still in its infancy. This lack of reliability research is justified by the fact that GPUs have traditionally been used to accelerate graphics rendering in personal devices, and by the fact that graphics, gaming, and multimedia applications can tolerate a number of errors without issues [9]. Architectural research in GPUs has then been focused in increasing devices performance and energy efficiency over their reliability.

The architectural level reliability solutions available in high-end GPUs (but not in low-power GPUs) are parity and Single Error Correction Double Error Detection (SECDED) Error-Correcting Codes (ECC) applied to memory elements [10]. To improve reliability, a number of high-level solutions may be applied and engineered. Safety-critical applications usually adopt redundancy to guarantee high reliability [11] while scientific applications typically employ checkpoint/restart mechanisms [12]. Some recent research efforts have been carried out to implement software-based hardening solutions for parallel codes on GPUs [13], [14], [15]. These works propose efficient

This work was supported in part by CAPES foundation of the Ministry of Education, CNPq research council of the Ministry of Science and Technology, FAPERGS research agency of the State of Rio Grande do Sul, Brazil.

D. Oliveira, T. Santini, and P. Rech are with the Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500 - Campus do Vale - Bloco IV, Porto Alegre, RS, Brazil (phone : +55 (51) 3308-6806, fax : +55 (51) 3308-7308, email : dagoliveira, tcsantini, prech@inf.ufrgs.br).

L. L. Pilla is with the Departament of Informatics and Statistics, Federal University of Santa Catarina (UFSC), Campus Universtitário Reitor João David Ferreira Lima, Florianópolis, SC, Brazil (phone : +55 (48) 3721-7564, email : laercio.pilla@ufsc.br).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

solutions to correct errors following the Algorithm-Based Fault Tolerance (ABFT) philosophy [16], [17]. While extremely efficient, ABFT procedures have to be designed specifically for the algorithm to be hardened and are applicable to a limited set of problems. Duplication With Comparison (DWC) may represent a general solution that can be easily and automatically applied to any algorithm. DWC can be implemented duplicating the instantiated parallel processes or executing twice the operations in each parallel process, with different effects on the devices performance and errors detection capabilities.

The aim of this work is to provide a thorough analysis of GPUs radiation reliability based on analytical studies and a series of extensive accelerated beam tests. Section II details the internal structures of modern GPUs, focusing on how these are affected by radiation-induced errors, followed, in Section III, by a description of the adopted experimental methodology. We start our experimental analysis in Section IV by evaluating the error rate of registers and caches of two consecutive GPU generations to understand how vendors are reacting to the radiation threat from a technology design point of view. Details on pattern dependence and multiple error occurrences are also provided. Next, in Section V, we study a representative set of parallel algorithms from both safety-critical and HPC domains, correlating their characteristics and observed radiation sensitivity. Then, in Section VI, we present two philosophies of software-based fault tolerance that may be used to mitigate radiation effects: techniques specifically tailored for an algorithm and generic ones. Afterwards, in Section VII, we compare the efficiency and overhead of software-based fault tolerance with the available ECC mechanism. Finally, Section VIII presents related works while Section IX draws final remarks.

II. Background

This section provides a brief background on the architecture of modern GPUs, the available ECC mechanism, and sources of GPU errors and their impact on the application execution.

A. GPU Architecture

Figure 1 shows a simplified, representative GPU architecture for *Compute Unified Device Architecture* (CUDA) generations. The architecture is composed of an array of *Streaming Multiprocessors* (SMs), which are the basic building blocks of a GPU. The parallel code executed on a GPU, called a *kernel*, is divided into *blocks of threads* that a thread block scheduler dispatches to idle SMs. Each SM has multiple *CUDA cores*, each of which can execute only one thread in a clock cycle.

A group of threads (called a *warp*) is selected by the warp scheduler to be executed next on the CUDA cores in a given SM, and then instructions are dispatched by the instruction dispatch unit. GPUs with CUDA capabilities 2.0 or 3.5, as the vectors of this study, can execute up to 64 and 192 parallel threads in an SM in a computing cycle, respectively.

As depicted in Figure 1, each SM disposes of two schedulers. At every instruction issue time, the first scheduler issues one instruction for some warp with an odd ID and the second scheduler issues one instruction for those with an even ID [18].

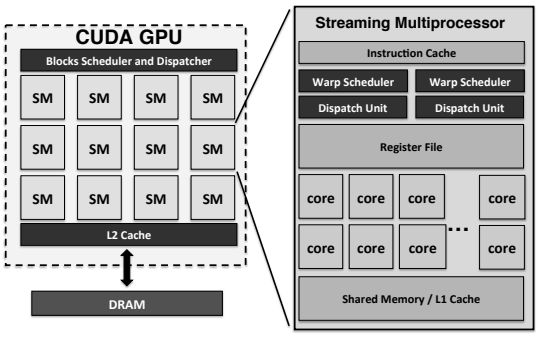


Fig. 1. A representative CUDA-based GPU architecture [10].

The second scheduler cannot issue instructions when double-precision floating-point instructions have to be executed. It is the programmer's task to divide the instantiated threads into a *grid* of *blocks* when designing a *kernel* to be executed on a GPU. The thread distribution can be easily modified, as the block size and the grid size are both parameters specified when launching a CUDA kernel.

Figure 1 also highlights the memory structures available on modern GPUs. An L2 cache is shared among all SMs. A thread executing on a SM has exclusive access to dedicated internal registers and shares L1 cache and shared memory with all threads in the same SM. A thread cannot access the L1 cache or shared memory of other SMs. Caches are particularly critical to resilience, as a radiation corruption in an L2 or L1 bit may affect the execution of many threads, leading to multiple output errors [15].

B. Radiation-Induced Effects on GPUs and Resilience Support

The impact of galactic cosmic rays with the upper level of the terrestrial atmosphere generates a cascade of particles. A great number of high-energy neutrons (i.e., neutrons with an energy higher than 10 MeV) is produced by the primary and secondary impacts of cosmic rays. A flux of about 13 $n/(cm^2 \times h)$ reaches sea level, and the amount of neutrons increases exponentially with altitude [19]. The interaction of a neutron may perturb a transistor state, generating bit flips in memory as well as a current spike in logic circuits that, if latched, leads to an error. The impact of radiation-induced errors in logic is expected to increase in future technology, eventually becoming more serious than errors in memory elements [20], [21].

There are a number of ways that neutron strikes perturb GPUs. A neutron may induce bit flips in memory elements as well as transient voltage spikes in logic computing resources or control circuitry. GPUs use large caches and complex task schedulers to orchestrate the parallelism on the system. While task scheduling is performed in software as part of the operating system tasks on CPUs, GPUs have dedicated, hardware-based task schedulers internally. The caches and schedulers are particularly critical for parallel processors and failures in these areas can lead to multiple output errors or functional interruptions [15], [22].

A radiation strike leads to one of the following outcomes: (1) no effect on the program output (the error is masked

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

or corrupted data is not used), (2) Silent Data Corruption (incorrect program output), (3) program crash, (4) system hang (the GPU has to be rebooted to restore its functionality). Out of these outcomes, (2) is harmful as it remains undetected and unpredictable, while (3) and (4) are to be strictly avoided in safety-critical applications and in HPC, as they lead to loss of functionality, performance penalties, and possible data loss.

From a radiation test point of view, the CUDA cores are isolated such that a single radiation-induced event in one of them will only corrupt the thread assigned to it. Threads that follow the corrupted one or assigned to CUDA cores near the struck one will not be affected. Nevertheless, errors in the L1 cache or shared memory are likely to affect several threads in the SM, as all threads can access that data. Similarly, errors in the L2 cache, shared among all SMs, are likely to affect several blocks of threads. A radiation strike in one of the schedulers may lead to wrong task assignments forcing threads to work on wrong data, to synchronization issues leading to incomplete results, or to conflicts or control flow errors that induce kernel panic or crashes. Instantiating a higher number of parallel threads typically reduces the code execution time but increases the scheduler strain required to manage execution and resource sharing. Imposing a higher scheduler strain (either on the warp or block scheduler) has the drawback of increasing the probability of having the scheduler affected by radiation [22].

Only the major storage structures of GPUs for HPC applications are protected with Single Error Correction Double Error Detection (SECDED) Error-Correcting Code (ECC) including device memory, L2 cache, instruction cache, register files, shared memory, and L1 cache. However, some resources are left uncovered, e.g., logic, queues, the thread block scheduler, warp schedulers, instruction dispatch units, and interconnect network. Unfortunately, the details of resilience support for these structures are considered business-sensitive by vendors and, hence, unavailable. A discussion on NVIDIA ECC efficiency and efficacy takes place in Section VII-A.

III. Methodology

In this section, we briefly describe the devices used in this study, the irradiation facilities and available particles beams characteristics, and our radiation sensitivity data collection methodology.

A. Devices Under Test

The devices considered in this study belong to two commercially available NVIDIA GPUs families: the Fermi-based C2050 and the Kepler-based K20, which were released approximately two years apart. In June 2011, 3 of the top 5 supercomputers used NVIDIA Fermi accelerators. Since 2013, the more advanced K20 has replaced the C2050 and acts as an accelerator in 2 of the top 10 supercomputers [4]. The purpose of testing C2050s and K20s is to measure the radiation sensitivity improvements from a GPU generation to the next one. The realistic applications and hardening solutions evaluation is performed only on the K20s.

Both devices feature a unified L2 cache and a GDDR5 main memory module. The C2050 features 14 SMs, each containing 32 CUDA cores while the K20 features 13 SMs, each containing 192 CUDA cores. Table I summarizes the main characteristics of C2050 and K20 GPUs. Register files and caches are SECDED ECC protected in both devices; K20's read-only data cache has parity protection support.

TABLE I
Characteristics of studied GPUs.

	C2050	K20
Architecture	Fermi	Kepler
CUDA Cores	448	2496
Process [*nm*]	40	28
Transistor Count [*billion*]	3.0	7.1
Core Clock [*MHz*]	1150	706
Total Register Files [*MB*]	1.75	3.25
Total L1 Caches [*KB*]	896	832
Total Read-Only Data Caches [*KB*]	-	624
Shared L2 Cache [*KB*]	768	1280
GDDR5 Memory [*GB*]	3	5
Single-Precision Performance [*TFLOPS*]	1.33	3.52
Double-Precision Performance [*TFLOPS*]	0.66	1.17

B. Neutron Beam Test Experimental Setup

Radiation experiments were performed at the Los Alamos National Laboratory (LANL) Los Alamos Neutron Science Center (LANSCE) Irradiation of Chips and Electronics House II, and in the VESUVIO beam line in ISIS, Rutherford Appleton Laboratories, Didcot, UK. The experiments were conducted at different times of years 2013 and 2014. As shown in Figure 2, both of these facilities provide a white neutron source that emulates the energy spectrum of the atmospheric neutron flux between 10 and 750 MeV. The ISIS beam has been empirically demonstrated to be suitable to mimic the LANSCE beam and the terrestrial radiation environment [23], despite its relatively lower component of high-energy neutrons. The ISIS beam includes a non-negligible component of thermal neutrons, which may increase the measured error rate if Boron-10 is present in the tested devices [5]. Therefore, we discard thermal neutrons contribution from the ISIS error rate and make a direct comparison only among experiments performed in the same facility and in the same time slot.

The neutron flux for energies above 10 MeV was about $1 \times 10^6 n/(cm^2 \times s)$ in LANSCE and $4 \times 10^4 n/(cm^2 \times s)$ in ISIS. The neutron fluxes used were higher than the neutron flux at sea level [19], but we have carefully designed the experiments to ensure that the probability of more than one neutron generating a failure in a single code execution remains practically negligible. The observed error rates were lower than 10^{-2} errors/execution. Since a much lower neutron flux hit a GPU in a realistic environment, it is highly unlikely to have more than one corruption during a single execution. We can, therefore, scale experimental data in the natural radioactive environment without introducing artificial behaviors.

The beam was focused on a spot with a diameter of 2 inches, which provided uniform irradiation of the GPU chip without directly affecting nearby board power control circuitry and DRAM chips. This implies that data stored in the main memory is not corrupted, allowing an analysis focused on just

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

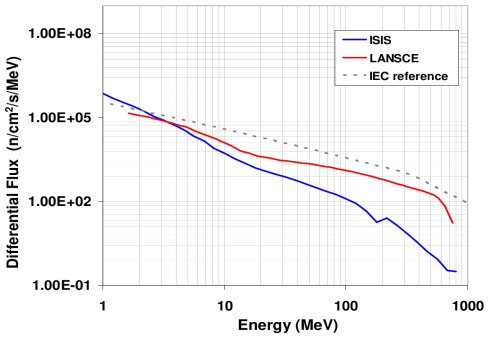


Fig. 2. LANSCE and ISIS neutrons spectra plotted against the reference of neutrons spectrum at the sea level [23].

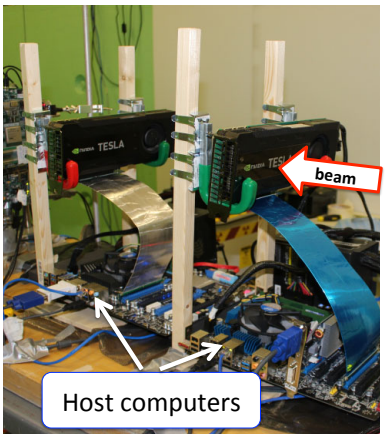


Fig. 3. Radiation test setup installed inside the ICE House II, Los Alamos Neutron Science Center (LANSCE), LANL.

the GPU core. Actually, having the DRAM exposed would have masked most of the GPU core behavior under radiation that we intend to highlight. Moreover, DRAM sensitivity to radiation has already been deeply analyzed and proved to decrease with the shrinking of technology nodes [5], and modern DRAM chips are provided with efficient ECC circuits that increase device reliability by several orders of magnitude [24]. It is worth noting that such a consideration does not apply to caches, for which technology shrinking, compact design, and performance requirements increase the probability of having failures as well as the efforts and penalties of adding ECC [25].

The GPU hardware setup includes connecting the GPU to a host computer through a PCIe extender (Figure 3). The PCIe extender is provided with a dedicated power line and fuses to decouple the GPU and motherboard power sources. Radiation may induce a Single Event Latchup (SEL) on the device under test, which is a short circuit that requires a prompt power cutoff to avoid permanent damages to the device. We never observed any SEL during our experimental campaign.

The role of the host computer is to initialize the test sending pre-selected input to the GPU and gather the computation results. When results are available, the host computer compares GPU output data with a pre-computed golden output. When a mismatched is detected, the execution is marked as affected by a *Silent Data Corruption (SDC)*. A software and a hardware watchdog were included in the setup. The software watchdog monitors a time-stamp written by the application running on the GPU. If the time-stamp is not updated in ten seconds the GPU application is killed and launched again. Such a watchdog is required to detect and manage radiation-induced program crashes or control flow errors that prevent the GPU from completing the assigned tasks (e.g., the GPU enters an infinite loop). The triggering of the software watchdog is counted as a *Functional Interruption*. The hardware watchdog is an Ethernet controlled switch that performs a power cycle of the host computer if the host computer itself does not acknowledge any ping requests in ten minutes. The hardware watchdog is necessary as radiation can corrupt the PCIe controller on the GPU board as well, possibly causing the host computer to hang.

As shown in Figure 3, two devices can be aligned with the beam (two K20s or C2050s). To reduce statistical error, we alternate the execution of code, benchmark, or configuration between the two GPUs, taking the distance to beam source derating factor into account.

IV. GPUs Memory Sensitivity Evaluation

As a first characterization, we measure, report, and discuss the raw sensitivity of the memory structures from different GPU architecture generations: Fermi architecture-based C2050 and Kepler architecture-based K20.

A. On-chip Memory Test Design

The main memory structures in the GPU chip are Shared Memory, L1 and L2 caches, and the register file. Shared memory and the L1 cache share the same physical on-chip storage, which is here referred simply as L1 cache. To measure cache and register file sensitivity, it is necessary to force the memory to a given value, expose the device to a given radiation fluence to allow errors to accumulate, and check whether all bits in memory are still holding the initial value.

The tests are written in the CUDA programming language and run natively on the device under test. For the SM register file test, each thread is responsible for testing its privately accessible registers. For the caches test, each thread is responsible for testing a pre-assigned portion of the cache. To precisely measure the exposure time, each thread records a timestamp into system memory at the beginning and end of its execution. The exposure time is parameterized and carefully engineered to make it unlikely for more than one neutron to generate a failure. This approach is essential to evaluate the occurrences of Multiple Bit Upsets (*MBUs*) and Multiple Cell Upsets (*MCUs*). If more than one error is detected in the same test, those errors are likely to be produced by the same impinging neutron, and a MCU is detected. A MCU in which the multiple errors belong to the same word is called a MBU. MBUs are particularly critical, as they may undermine the ECC mechanism error detection/correction capability.

The pattern to be written and checked in memory is also parameterized since the design of some memory cells may not be symmetric, meaning that the pull-up and pull-down transistor capacitances may differ [26]. All 0s ($0s$) and all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

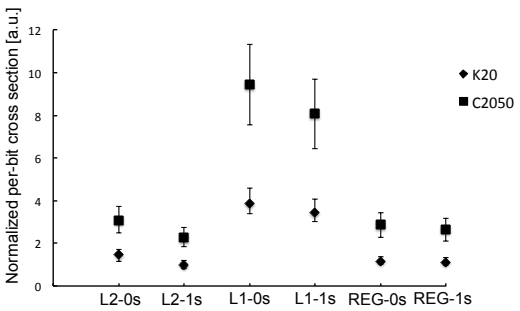


Fig. 4. Per-bit cross sections for the main memory structures of K20 and C2050 with two different test patterns: $0s$ (all 0s) and $1s$ (all 1s).

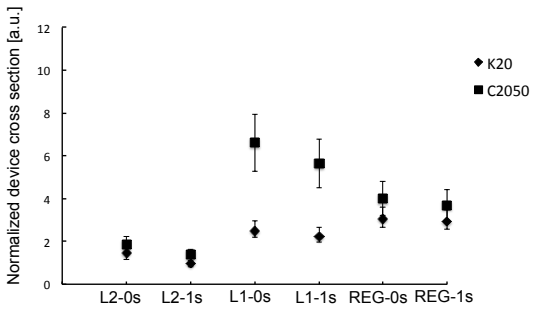


Fig. 5. Device cross sections for the main memory structures of K20 and C2050 with two different test patterns: $0s$ (all 0s) and $1s$ (all 1s).

1s ($1s$) patterns are checked to measure the probability of corrupting bits set to 0 or to 1, respectively.

Since our goal for this part of the study is to measure the raw sensitivity of the registers and SRAM structures of the GPUs, we have disabled the built-in ECC and parity mechanisms.

B. Experimental Results

We experimentally measured the *cross section*, a widely used metric to evaluate radiation sensitivity [5], for the register file cells and the L1 and L2 cache cells of both K20 and C2050 boards. The cross section is calculated by dividing the number of observed errors by the received neutron fluence ($neutrons/cm^2$). If the cross section of the memory structure is divided by the number of cells, it yields the average sensitive area of a single cell, i.e., the portion of the cell that, if hit by a neutron, will produce a failure. Higher cross sections imply higher probabilities for an impinging neutron to corrupt a bit.

A reduced transistor dimension lowers the transistor exposed area, reducing its probability of being hit by a particle. Nevertheless, reducing the feature size typically reduces the device node capacitance, thus a neutron hitting a bit cell has a higher probability of generating a failure in K20 with respect to C2050. The combination of reduced sensitive area and capacitance is expected to bring an overall benefit in reducing the cross section for future technologies [5]. Supply voltage also plays a key role in the radiation sensitivity of memory cells. Reducing the memory cells supply voltage has the drawback of increasing linearly the device radiation sensitivity [27]. For current and future devices, the voltage reduction per generation is limited to 5-10% [28]. Hence, no significant radiation sensitivity increase is expected in the K20 with respect to the C2050 due to a reduced supply voltage. As supply voltage reduction in future technology generations is expected to be small, it can be anticipated that increases in circuit sensitivity will be relatively small.

Figure 4 shows the *per-bit* cross section for L2 cache, L1 cache, and register file normalized to the cross section of K20 L2 cache obtained with the $1s$ pattern. In this calculation, MCUs (and, thus, MBUs) were considered as events generated by a single neutron. A discussion on MBU and MCU takes place later in the section. The 95% confidence intervals, which result from a combination of neutron counts uncertainty and statistical error, are also reported. It is clear from experimental data that K20 memory cells have an increased reliability with respect to those of the C2050 for all the tested memory structures and patterns. The C2050 shows an increase in the cross section with respect to the K20 that spans from $2.4\times$ for the L1 with the $0s$ pattern to $2.15\times$ in the L2 with the $0s$ pattern. The increased reliability is much more pronounced than what was observed for similar technology scales [5]. The better reliability gain observed is probably caused by a reliability-wise bit cell design in the K20 whose details are, unfortunately, not available.

Results highlight that bits set to zero are 40% more prone to corruption than bits set to one in the L2 caches for both generations of GPUs. However, this is not true for the register file or the L1 caches (Figure 4). This is due to the intrinsic asymmetries to the cache cell design: L2 caches must be dense and compact to minimize area while L1 and register files should maximize performance. L2 caches design may then have weaker pull-up transistors while L1 and the register file are symmetric [26]. It is worth noting that this specific result can be achieved only through radiation experiments and is fundamental to precisely evaluate the resilience of GPUs. Fault injection designed to evaluate GPU reliability can greatly benefit from the reported L2 caches sensitivity pattern dependence to tune injection probability functions.

The results reported in Figure 4 are normalized to the number of available bits. However, the K20 structures are significantly larger than the C2050 structures. Figure 5 shows the *total-area* cross section for the K20 and C2050, obtained by multiplying the *per-bit* cross section with the size of the respective memory structure. Figures 4 and 5 show that the Kepler generation has better reliability than the Fermi one per bit as well as for the whole caches and register area. This is an encouraging result that can be attributed to intrinsic technology evolution and better cell design factors. Even if the details of cell design are considered business-sensitive, our data indicate that architects have improved the cell design to combat the danger of increased GPU sensitivity that derives from the increased memory availability.

Figure 6 shows the percentage of events that were found to be MCU and MBU (the L1 test did not provide a statistically significant amount of multiple events on the C2050 and is not included). Whenever more than one bit was found corrupted during a test, a MCU was detected. If the corrupted bits belonged to the same word, a MBU was counted. K20's

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

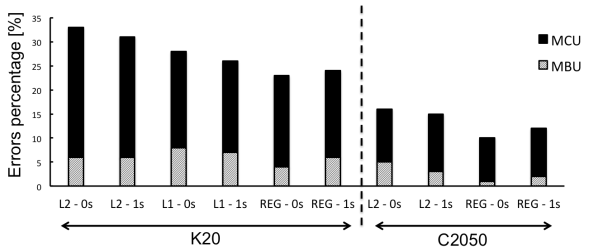


Fig. 6. Percentage of errors found to be MCU in the memory structures of K20 and C2050. MBUs are those MCUs with more than one bit corrupted in the same word. C2050 L1 test (not included in the picture) did not provide a statistically significant amount of failure.

memory structures are about $2\times$ more prone to experience multiple events than C2050's. These results are very reasonable given the small feature size, and many other microprocessor components have higher MCU and MBU rates [29]. For both generations of GPUs, the L2 cache is more likely to experience multiple events, probably because of its dense and compact design. There is no significant pattern dependence on multiple events probability. The reported results are of extreme importance for the tuning of fault injectors like the one presented in [30] as they give the correct probability for multiple events occurrences.

The distinction between MBUs and MCUs is fundamental as it categorizes whether a radiation-induced event could be corrected with the Single Error Correction Double Error Detect (SECDED) ECC mechanism included in the K20 and C2050 devices. MCUs could occur as multiple single errors that would be correctable with a SECDED ECC, whereas a MBU would be incorrigible. Then, even if about 33% of neutron-induced events are multiple events in the K20 L2 cache, only 6% are incorrigible MBUs. It is worth noting that no MBU with more than 2 bits corrupted was detected in the experimental campaign. For the current GPU generation it is reasonable to believe that the great majority of radiation-induced failures in the memory structures of GPUs can be detected or corrected by the included ECC mechanism. The SECDED ECC may become insufficient if the observed trend of increasing MBU occurrences from a GPU generation to the new one is maintained. A discussion on ECC efficiency obtained using benchmarks takes place in Section VII-A.

V. Parallel Applications Radiation Sensitivity

In this section, a reliability evaluation for a representative set of parallel algorithms for both safety-critical and HPC applications is presented. We discuss methods to improve the reliability of applications in Section VI.

Fault injection simulations are one of the possible ways to evaluate applications reliability by measuring their Architectural Vulnerability Factor (AVF) (i.e., the probability for a failure in a resource to be observed at the output [31]). Some recent works evaluate the AVF for various parallel algorithms executed on GPUs [30], [32], [33]. Nevertheless, only a limited set of GPU resources are accessible to the user and, thus, not all the possible sources of errors can be considered. In light of that, we take advantage of controlled neutron beams to gather the precise error rate of GPUs executing parallel algorithms. As described in Section III-B, the beam spot was chosen to ensure that all the GPU resources are uniformly exposed to radiation and could then be corrupted. Usually, when performing radiation experiments, the error propagation observability is limited in a way that is not possible to identify unequivocally the source of the observed failure. As a result, the evaluation may be limited to the tested configuration and be hardly generalizable. We aim at going a step further with our experiments by selecting a broad set of applications and configurations that cover a wide range of computational and data movement requirements. Additionally, we correlate the experimentally observed sensitivity trends with specific code characteristics and, when available, with the AVF measured in [32] and [33]. We claim that our experimental and analytical procedure not only allows a precise evaluation of the realistic error rate of GPUs exposed to the natural neutron flux but also allows to draw reliability conclusions applicable to other algorithms and future architectures.

A. Selected Parallel Algorithms

The algorithms chosen for the reliability evaluation are matrix multiplication, implemented both in memory-bound (MxM) and compute-bound ($dgemm$) approaches, a 512×512 64-point Fast Fourier Transform (FFT), $LavaMD$, $Hotspot$, Needleman-Wunsch (NW), and 1024 subsequent matrix transpositions ($MTrans$). The chosen codes are part of benchmarks suites specifically designed for parallel architectures studies [34], [35] or designed for test purposes [14], [15]. Matrix operations and FFT are common tools for linear algebra operations and signal processing, used in both safety-critical and HPC applications. The others are typical applications of HPC systems: many real-world workloads use the algorithms implemented in these benchmarks as their kernel [36].

To give an overview of the difference among the tested algorithms, Table II lists the amount of input and output data, intended as double-precision floating-point elements for all the algorithms but for NW, which is executed with integer data. Each complex element in FFT is counted as 2 double numbers. Please note that $dgemm$ requires an additional input matrix with respect to MxM that was filled with zeros in our experiments, which is why $dgemm$ and MxM are listed with the same input size.

It is reasonable to believe that the higher the amount of data to be elaborated, the higher the error rate. Nevertheless, the radiation sensitivity may not be linearly dependent to input size. In fact, computational power will also influence the radiation sensitivity of the code. A higher number of instructions, threads, and operations will increase the probability for a neutron to generate a control flow error, a scheduler failure, and a single transient event that propagates to the output [22]. MxM, $dgemm$, and NW were tested with different input sizes to study the radiation sensitivity dependencies on problem size. When increasing the input size, only the number of instantiated blocks was increased, while the number of threads per block was kept constant. This allows to maintain the SM caches and resources efficiency throughout the different configurations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

TABLE II
PARALLEL APPLICATIONS DETAILS AND EXPERIMENTAL RESULTS. †TESTED IN LANSCE, § TESTED IN ISIS

	Domain	Input (≈)	Output (≈)	Instr. Exec. (≈)	Ex. time $[s]$	SDC FIT	FI FIT
$MxM(2^{10})^{\dagger}$	Linear algebra	$2.10M$	$1.05M$	$1.07G$	0.05	$(4.63 \pm 0.80) \times 10^2$	$(3.97 \pm 0.52) \times 10^2$
$MxM(2^{11})^{\dagger}$	Linear algebra	$8.39M$	$4.19M$	$8.59G$	0.37	$(5.79 \pm 1.02) \times 10^2$	$(2.64 \pm 0.63) \times 10^2$
$MxM(2^{12})^{\dagger}$	Linear algebra	$33.55M$	$16.78M$	$68.72G$	2.90	$(6.03 \pm 0.64) \times 10^2$	$(2.61 \pm 0.72) \times 10^2$
$dgemm(2^{10})^{\dagger}$	Linear algebra	$2.10M$	$1.05M$	$1.07G$	0.01	$(7.43 \pm 0.97) \times 10^2$	$(2.79 \pm 0.39) \times 10^2$
$dgemm(2^{11})^{\dagger}$	Linear algebra	$8.39M$	$4.19M$	$8.60G$	0.02	$(7.83 \pm 0.92) \times 10^2$	$(1.96 \pm 0.42) \times 10^2$
$dgemm(2^{12})^{\dagger}$	Linear algebra	$33.55M$	$16.78M$	$68.75G$	0.14	$(1.04 \pm 0.32) \times 10^3$	$(2.28 \pm 0.55) \times 10^2$
$MTrans^{\S}$	Memory	$4.19M$	$4.19M$	$88.08M$	3.46	$(1.80 \pm 0.39) \times 10^1$	$(1.60 \pm 0.35) \times 10^1$
$FFT^{\dagger}$	Signal processing	$33.55M$	$33.55M$	$402.65M$	0.07	$(2.88 \pm 0.19) \times 10^3$	$(7.02 \pm 0.05) \times 10^2$
$LavaMD^{\S}$	Molecular dynamics	$1.69M$	$1.69M$	$63.49G$	0.18	$(3.44 \pm 0.52) \times 10^3$	$(1.52 \pm 0.23) \times 10^3$
$Hotspot^{\S}$	Physics simulation	$2.10M$	$1.05M$	$14.68G$	7.49	$(2.04 \pm 0.31) \times 10^2$	$(1.12 \pm 0.17) \times 10^2$
$NW(2^{12})^{\dagger}$	Bioinformatics	$8.19K$	$16.78M$	$65.56M$	0.06	$(6.33 \pm 1.26) \times 10^1$	$(5.48 \pm 1.87) \times 10^1$
$NW(2^{13})^{\dagger}$	Bioinformatics	$16.38K$	$67.10M$	$261.73M$	0.22	$(3.06 \pm 0.45) \times 10^2$	$(2.17 \pm 0.33) \times 10^2$
$NW(2^{14})^{\dagger}$	Bioinformatics	$32.77K$	$268.44M$	$1.05G$	0.83	$(9.00 \pm 1.36) \times 10^2$	$(3.60 \pm 0.56) \times 10^2$

Table II also lists the number of CUDA instructions each benchmark executes and the kernel execution time. $LavaMD$ is the benchmarks with the highest computational demand as it needs one order of magnitude more $instructions-per-data$ (obtained dividing column 5 by column 3 in Table II) than the others. $MTrans$, the opposite case, is considered to evaluate the radiation effects on a GPU when rapid data movements are performed with little computation.

The last two columns of Table II show the results of experimental evaluation for all the tested algorithms. $LavaMD$, $Hotspot$, and $MTrans$ were tested at ISIS in December 2013 and May 2014 while the others where tested at LANSCE in September 2013 and December 2014. The ECC and parity mechanisms were disable for these experiments to access the raw dynamic GPU behavior under radiation. ECC, in fact, can mask errors in a uncorrelated way with the tested code and could also mislead the Silent Data Corruption and Functional Interruption distinction. When ECC is off, a MBU may be masked during computation, without appearing at the output. Nevertheless, that same MBU would have been detected by the ECC as an incorrigible error, triggering the application crash. A discussion on ECC efficiency takes place in Section VII-A.

The experimental analysis starts by measuring the application cross section dividing observed error rate ($errors/s$, separately for SDC and FI) by the average particles flux ($neutrons/(cm^2 \times s)$), yielding an area. The execution time is normalized when measuring the cross section, so the cross section has no dependence on the computational time, but only on the amount (and criticality) of resources required for computation. As ISIS and LANSCE reasonably mimic the atmospheric neutron flux, the probability of having a neutron corrupting the GPU during our experiments or at sea level during normal operation are very similar [23]. The experimentally observed cross section is then an intrinsic characteristic of the device and code, independent on the neutron source. Multiplying the cross section (cm^2) with the expected neutron flux on the GPU ($13n/(cm^2 \times h)$ at sea level), one can estimate the GPU error rate or Failure In Time (FIT), expressed as $errors/10^9 h$. FIT are reported in Table II with a 95% confidence interval that includes both statistical error and neutron counts uncertainty.

B. Silent Data Corruption

A SDC is typically produced when radiation corrupts memory elements storing variables or data used for computation, or when the logic executing an operation experiences a Single Event Transient (SET) [5], [27]. Additionally, SDCs can occur when the scheduler fails in synchronizing threads, assigning a thread to a wrong CUDA core, or presents results that are still incomplete [22]. SDCs are among the most harmful failures in computing devices, as they are unpredictable and undetectable and may leave the system in an unknown state. SDCs undermine the confidence on the safety-critical system correct functionality and result correctness in HPC applications.

As reported from data in the SDC FIT column of Table II, the SDC rate ranges from 1.80×10^1 FIT for $MTrans$ to 3.44×10^3 for $LavaMD$, varying by almost two orders of magnitude between tested applications. The underlying reason is differences in the GPU resources utilization and intrinsic code characteristics (i.e., the AVF [31]), amount of data elaborated, and the number of executions performed. For instance, the AVF of register files for the $Hotspot$ and MxM benchmarks has been estimated to be approximately 20% while for the $MTrans$ benchmark the estimation is one order of magnitude lower (about 2%) [33]. As register files represent more than 50% of memory resources in the K20, it is reasonable to assume that registers are among the main sources of errors. Our experimental observations agree with the AVF estimations as the difference of one order of magnitude is preserved for both SDC and FI rates. Fast Wavelet Transform (FWT) also shows an AVF of 20% for register files [33]. As FWT is comparable with FFT, we can conclude that a similar AVF applies to FFT register files, which agrees with our FFT FIT rate trend. Applications that heavily use register files are then expected to be more prone to be corrupted.

To further investigate the operative behaviors of GPUs under radiation, we note that amount of data elaborated alone is not sufficient to indicate the program neutron-induced error rate. For example, $MTrans$ elaborates $4\times$ more data than $LavaMD$ but has a two orders of magnitude lower SDC FIT. Similarly, FFT elaborates about $20\times$ more data than $LavaMD$, but has a comparable SDC FIT. The SDC sensi-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

tivity of a code actually increases with input dimensions, as demonstrated for MxM, $dgemm$, and NW executed with different input sizes, but the increase is not linear. In MxM and $dgemm$, the GPU caches are fully loaded even with the smallest tested size, so the actual exposed sensible area does not changes linearly with the input size. If a cache region is used twice for computation, the data it holds changes but the exposed area remains the same. The same occurs for logic computing resources. Parallel codes for GPUs are typically extremely regular [34], [35]. When the input size is increased, there is no significant modification in the number and kind of per-data-operations to be performed. If the GPU is fully loaded, the increased input size requires the reuse of some logic gates without a significant increase in the GPU exposed area. As a result, the cross section and FIT increase with input size is caused by the additional resources required to manage a higher amount of blocks (the number of threads per block is kept constant) and synchronize a more complex execution [22]. In NW, the FIT increase with input dimension is more remarkable, as the GPU memories are not fully filled. The input integer vectors, in fact, largely fit in the K20 register files, 1,280KB L2 and 832KB L1 caches. Thus, increasing the input dimensions has the side effect of increasing the GPU used (and exposed) area. As a general result, increasing the amount of data elaborated increases the GPU SDC FIT. If the data fills the GPU memory, the increase is sub-linear and caused by the additional scheduling resources required.

Finally, the difference between MxM and $dgemm$ is of particular interest as they solve the same problem with different implementations, modifying the resources criticality (mainly memory). Data in the GPU memory is exposed to radiation and susceptible to be corrupted. No effect is expected if the corrupted data has already been digested by the GPU, is obsolete, or will be overwritten. Data that still has to be used or needs to be written back is critical and, if corrupted, leads to an observable failure which is counted in the FIT measure. In other words, a high caches hit rate brings better performances but a higher SDC cross section and FIT.

In MxM, the data required by a thread does not fit in the caches, so the execution time is dominated by memory latencies to move data from the DDR. While waiting for new data, possible neutron-induced failures in logic are not critical as the thread is in idle state, and data in cache is likely to be removed to accommodate new requests, so radiation corrupts obsolete data. Meanwhile, a thread in $dgemm$ digests all data in the SM caches before requiring new elements, and the whole optimization aims at maximizing the hit rate in cache and reducing accesses to the device memory. Under radiation, this turns into a higher criticality for $dgemm$ caches, which increases the code FIT. The resource utilization efficiency, then, affects the radiation sensitivity of the GPU. State registers, on the contrary, are critical during memory exchange latencies, which is why MxM has a higher FI rate than $dgemm$, as described in the following subsection.

C. Functional Interruption

A Functional Interruption occurs when radiation corrupts the GPU instruction caches, inducing a control flow error or generating a kernel panic that forces the GPU to close the application without returning an output, when a scheduler failure hangs the GPU kernel, or even when the PCI-Express bus controller is corrupted. FIs increase the cost of computation in HPC applications, as a node becomes unavailable and its work needs to be recovered (possibly with a checkpoint-rollback strategy) and assigned to a new node. In safety-critical applications, FIs should be strictly avoided.

The FI rate depends on application, and it ranges from between 20% and 30% of the SDC rate for $dgemm$ to 55% for $Hotspot$, and almost 90% for $MTrans$ and NW(see the last two columns of Table II). The higher occurrences of FI vs. SDC in $MTrans$ are explained noting that FIs are directly related to the number of instructions executed by a thread that, if corrupted, lead to a control flow error. Additionally, it depends on the number of parallel threads instantiated (i.e., the scheduler strain required for computation). In $MTrans$, it is unlikely for radiation to corrupt data, as it sits on internal memory for a very short period of time. $MTrans$ does not perform arithmetical operations on data but only re-allocations. A failure in such an operation may have a high probability of leading to a control flow error and, thus, to a FI. NW has little data to work on, but a great number of operations are performed on each element. NW will then require a higher amount of instruction cache than data cache with respect to other algorithms [34]. A failure in the instruction cache is likely to generate a control flow error, explaining why for NW the ratio of FI vs. SDC is higher than other codes.

When the problem size is increased, the number of instructions in the instruction cache is not significantly increased, as the code is compiled only once and executed with different inputs [37]. It is unlikely for CUDA to introduce input specific optimizations at kernel launch or run time. This is in accordance with experimental data reported in the last column of Table II that shows an almost constant FI FIT for MxM, $dgemm$, and NW when the problem size is increased (variations are inside statistical error tolerance). The combination of SDC increase and almost constant FI makes the ratio of FI vs. SDC to be significantly reduced with increasing problem size. In fact, FIs are 85% of SDC for $MxM(2^{10})$ and 43% for $MxM(2^{12})$. A similar trend is observed for $dgemm$ and NW. As a general result, we can conclude that increasing the problem size will not significantly increase the FI FIT of the application.

D. Mean Workload Between Failure

The code radiation sensitivity discussed in the previous subsection indicates the probability for a neutron that strikes the GPU during the code execution to generate an observable failure. When analyzing processors reliability, the execution time has also to be taken into account. The execution time, in fact, determines the number of neutrons that hits the processor during computation. The higher the execution time, the more neutrons will impinge the processor before completing computation. To take execution time into account, the *Mean Executions Between Failures* (MEBF), defined as the number of executions correctly completed between failures,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

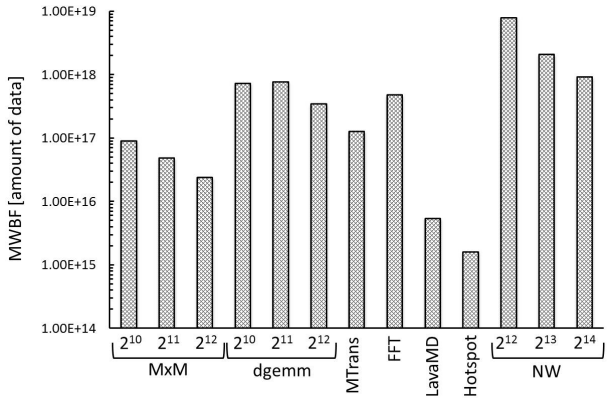


Fig. 7. SDC and FI combined Mean Workload Between Failure.

was introduced [38]. The MEBF is calculated dividing the Mean Time Between Failures (MTBF) by the code execution time.

On a GPU, and parallel processors in general, all the design and programming efforts aim at increasing performance. Not taking execution time into account would then result in an unfair and imprecise GPU reliability evaluation. Moreover, the GPU throughput depends on the input size, so the amount of data elaborated is not linearly dependent on the execution time. We then evaluate the codes *Mean Workload Between Failure* (MWBF), measured multiplying the MEBF by the amount of data elaborated in each execution [22] (double-precision floating-point elements for all the codes except NW, which works with integers). The MWBF depends on the used resources reliability and the resource efficiency like the MEBF, but also on the throughput gain the resources bring to the code. Basically, the MWBF will also consider the fact that GPU throughput varies with input size.

Figure 7 shows the MWBF for the tested algorithms and versions, considering either SDC or FI as failures. From the figure, it is clear that less data can be correctly computed in $LavaMD$ and $Hotspot$ with respect to other codes. This is expected, as those are the codes with the highest instructions-per-data and the lowest throughput.

Even though $dgemm$ has a higher FIT than MxM, Figure 7 shows that much more data can be correctly elaborated by the former than the latter. For matrix multiplications, the compute-bound implementation brings a reduction in the execution time which is enough to compensate the increase in radiation sensitivity. $dgemm$ is then the implementation to be chosen, as it combines the best performance with the highest reliability. Even if this result is not generalizable, we claim that the use of MWBF allows a more precise evaluation of GPU reliability.

MxM and NW MWBF decreases with input size. For MxM, this is caused by the increased scheduling resources required (and thus the increased FIT) and its reduced efficiency. When the input size is increased, each thread will execute more operations that require more data, thus increasing waiting time for memory and reducing the GPU throughput. NW throughput does not decrease with input size [34]. However, the excessive increase in the cross section documented in the previous subsection undermines the benefit that the throughput possibly brings. Choosing $dgemm$ with size $2^{11} \times 2^{11}$ seems to be the most reliable solution for matrix multiplication. $dgemm$ MWBF increases slightly from 2^{10} to 2^{11} indicating that $dgemm$ throughput efficiency and reliability is improved when increasing the input size. At $2^{11} \times 2^{11}$, the maximum throughput gain is reached as the caches and register utilization saturates and does not improve the throughput for bigger input sizes. As a general programming advice, increasing the workload of a parallel code increases reliability if the throughput gain is sufficient to balance the higher FIT rate caused by the higher use of scheduling resources.

VI. Software-Based Hardening Strategies

This section examines the available software-based hardening strategies that can be applied to mitigate the applications sensitivity to radiation effects on GPUs explored in the previous section. First, we discuss the use of Algorithm-Based Fault Tolerance and apply it to two applications. Then, we reason the use of Duplication With Comparison for applications for which ABFT strategies have not been developed. Section VII compares the performance and resiliency of different applications hardened with software-based strategies and ECC. It is worth noting that ECC corrects only radiation-induced failures that directly affect a memory bit, discarding errors in logic resources or in the scheduler. On the contrary, software-based hardening strategies operate with errors that appear at the output, regardless of origin. We can anticipate that software-based hardening strategies are then likely to detect or correct a higher number of radiation-induced failures.

A. Algorithm-Based Fault Tolerance

The *Algorithm-Based Fault Tolerance* (ABFT) hardening philosophy comprises the analysis of an algorithm basic structure in order to find an efficient and clever way to harden it. ABFT strategies are usually based on the encoding of input data and algorithm redesign to work with this data. At the end of the algorithm, the output is checked taking advantage of the encoding. When an error is detected, it can be corrected if the ABFT strategy has enough information to do so, or a re-computation is triggered.

Since ABFT strategies are algorithm-specific, we exemplify its use with two different applications: a matrix multiplication and a Fast Fourier Transform.

1) Matrix Multiplication: We hardened the matrix multiplication application using the ABFT strategy proposed by Huang and Abraham [16], which is based on the result checking approach of Freivalds [39]. Input matrices A and B are coded before computation, adding column and row checksum vectors (A_c and B_r in Figure 8) by summing all the elements in the correspondent column or row.

The result of the multiplication of the expanded matrices is a fully-checksum matrix M, where the $(n+1)$-th row and the $(n+1)$-th column contain the column (M_c) and row (M_r) checksum vectors of M, respectively [39]. When the multiplication is finished, M column and row checksum vectors are re-calculated summing the first n columns and n rows of M, resulting on M'_c and M'_r, respectively. Output

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

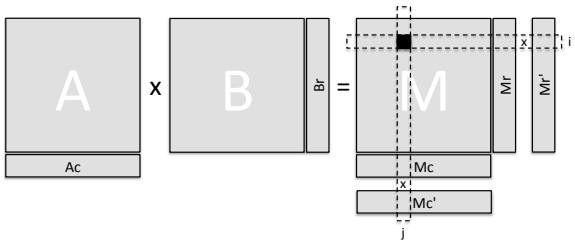


Fig. 8. Algorithm-Based Fault Tolerant for matrix multiplication [16].

verification is done by comparing the checksum vectors from the multiplication and the newly computed ones.

If a mismatch is detected between $M_r[i]$ and $M'_r[i]$, it means that at least one error is present in the i-th row of M, and respectively for columns. If $M[i,j]$ is identified as the only error in M, it can be corrected quickly using either the row or column checksum vectors following Equation 1 or Equation 2 [16].

$$M_{correct}[i,j] = M[i,j] - (M'_r[i] - M_r[i]) \quad (1)$$

$$M_{correct}[i,j] = M[i,j] - (M'_c[j] - M_c[j]) \quad (2)$$

On a GPU, the operations required to compute the checksums and detect errors can be done in $O(n)$, while error correction takes constant time [15]. The technique proposed by Huang and Abraham is only capable of correcting single output errors [16], which we have experimentally demonstrated to correspond to less than 43% of the cases [15]. Thanks to radiation experimental data, the ABFT strategy was extended to correct multiple errors on the same row or column of M in constant time and randomly distributed errors in $O(e_r \times e_c)$, where e_r and e_c are the number of mismatches between M row and column checksums, respectively (details in [15]). Please note that the ABFT implementation and the reliability evaluation of Section VII are neither dependent on implementation (MxM or $dgemm$) nor on input size.

2) Fast Fourier Transform: We hardened the Fast Fourier Transform application with the ABFT strategy proposed by Jou and Abraham [17] for fault-free N-point FFT networks of N processors [14]. This strategy is based on the superposition principle of linear systems and the circular shift property of the FFT. Its basic idea is to detect errors arising in any processor or connection with the use of input coding and a checksum comparison at the output. Figure 9 illustrates this process. The ABFT was implemented on 512×512 64-point FFTs. Nevertheless, the reliability evaluation presented in Section VII are easily extendable to other input sizes.

A thread in the hardened algorithm starts by taking its input sequence of N complex elements X and encoding it, resulting in the sequence X', as represented in Equation 3. The original vector X is kept unmodified for the situation where the FFT has to be re-computed due to the detection of an internal error. After the encoding phase, vector X' is given as input to the original FFT algorithm. After the algorithm computation is completed, the FFT output stored in X' is decoded to vector Y following Equation 4, where w_N^{-k} is the N^{th} root of unity. The N decoded results are then summed,

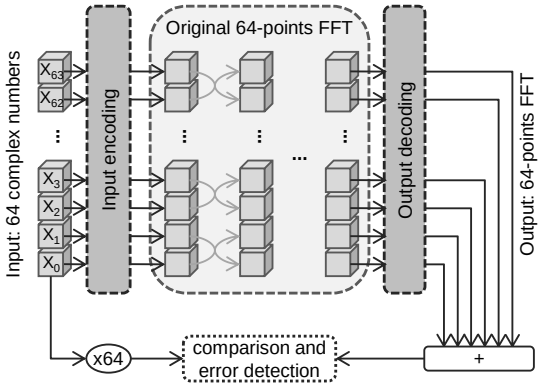


Fig. 9. Algorithm-Based Fault Tolerant FFT. The 64 complex elements of the input are coded, then the 64-point FFT is performed with the original algorithm, and its output is decoded. Errors are detected comparing the checksum generated summing the output values with $X_0 \times N$.

generating a checksum. As formally demonstrated by Jou and Abraham [17], this encoding and decoding scheme gives each output a nontrivial weighted contribution to the checksum such that any error will cause a detectable error syndrome, which is not the case with the original FFT algorithm. After computation, the checksum is compared to $N \times X[0]$. Any mismatches will identify the FFT as faulty, and will signalize the necessity of a re-computation. It is important to notice that both encoding and decoding phases could be processed by more than one thread for each 1D FFT, increasing parallelism.

$$X'[i] = \begin{cases} 2X[i] + X[i+1] & 0 \leq i < N-1 \\ 2X[i] + X[0] & i = N-1 \end{cases} \quad (3)$$

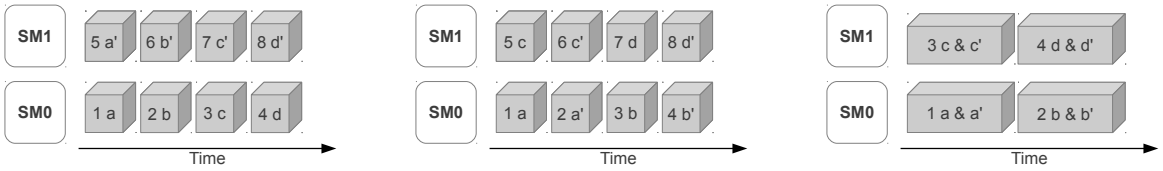
$$Y[k] = \left\{ \frac{1}{2+w_N^{-k}} X'[k] \quad 0 \leq k < N \right. , \quad (4)$$

In addition to encoding and decoding data, our ABFT strategy also includes two vectors to signalize any FFT re-computations needed, and to detect any failures from SM interruptions and scheduler failures that could prevent threads from completing their execution. Overall, this hardening mechanism increases the GPU memory usage with the addition of the encoded matrix X' and these two vectors (which are much smaller than the former). Lastly, the proposed ABFT strategy keeps the computational complexity of the original algorithm, as the functions for input encoding, output decoding, and error detection are linear. In addition, only the y n-point FFTs with errors are re-computed, and their overhead ends hidden by the parallel execution of the other $512 \times 512 - y$ FFTs [14].

B. Duplication With Comparison

For most applications, an ABFT solution is not available, and its design and implementation would require a great effort. *Duplication With Comparison* (DWC), on the other hand, is mostly algorithm-independent, as so are its implementation and performance. DWC is based in the re-execution of the algorithm with the same input and the comparison of outputs, and is, thus, able to only detect errors, requiring a redundant execution of the application to correct them.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers



(a) *Spatial* DWC. (b) *E-O Spatial* DWC. (c) *Time* DWC.

Fig. 10. Duplication With Comparison strategies. Each block is represented by a box and have its identification index (i.e., *block ID* in CUDA) and a letter corresponding to the task assigned to it. The apostrophe after a letter means that the task is a duplicated one.

DWC can be implemented in several ways on a GPU, such as by duplicating blocks, threads, or by executing operations twice in a thread. It is worth noting that duplicated threads must be carefully distributed as the corruption of shared resources (like caches) or critical resources (like the scheduler) may propagate to both copies, undermining DWC detection capabilities.

We designed three DWC strategies with different duplication and distribution philosophies: *Spatial*, *E-O Spatial*, and *Time*. In the spatial duplication strategies (*Spatial* and *E-O Spatial*), the number of blocks needed to compute the solution and, thus, the number of instantiated threads is doubled. The thread blocks are divided into two domains, each executing the unhardened code. The result of a block is then compared to the result of the duplicated block, detecting possible mismatches. We choose to duplicate blocks instead of threads in a block to avoid having both instances using the same cache, as a radiation-induced failure in it would be likely to propagate to both copies undetected.

In *Spatial*, a domain is formed by the first half of the blocks and the other domain by the second half. The redundant blocks are not necessarily executed in parallel with the first half of the blocks when the number of blocks instantiated in the unhardened code already exceeds the GPU parallel capabilities. The second spatial duplication, *E-O Spatial*, alternates original blocks and duplicated blocks in an even-odd fashion. In other words, blocks i and $i+1$ execute on the same data. *Spatial* and *E-O Spatial* are illustrated in Figure 10(a) and Figure 10(b).

Spatial and *E-O Spatial* detection capabilities may significantly differ. The redundant calculations are more likely to be processed in the same SMs in *E-O Spatial*, sharing resources like caches. Even if more efficient, as data does not have to be moved in two different regions of the GPU, *E-O Spatial* may experience errors in the SM caches that propagate to both the duplicated blocks and thus remain undetected. On the other hand, the same data is going to be duplicated in two different locations of the GPU in *Spatial*, reducing performance and increasing the exposed area of the device.

In the last DWC strategy implemented, named *Time* and depicted in Figure 10(c), each thread performs a redundant calculation after completing the original one. In *Time*, about the same amount of parallel resources (i.e., number of threads, scheduling strain, caches) as the original implementation are employed. However, threads complexity is increased as each thread executes twice the operations and compares results.

When an error is detected, the faulty threads are re-executed in the GPU until the results of the execution and the redundant execution match, or a maximum number of re-executions is reached resulting in a failed execution. In our experiments, we allow each block or thread to be re-executed a maximum of 5 times when an error is detected. Nevertheless, we anticipate that all errors detected by our strategies were corrected in the first re-execution.

TABLE III
EFFICIENCY AND RESILIENCY OF THE AVAILABLE HARDENING SOLUTIONS FOR GPUS.

		SDC		FI		Perf.
		FIT	a. u.	FIT	a. u.	
MxM	Unhardened	5.79×10^2	1.000	2.64×10^2	1.000	1.00
	ECC	5.62×10^1	0.097	4.02×10^2	1.523	1.01
	ABFT	1.04×10^1	0.018	2.97×10^2	1.127	1.14
FFT	Unhardened	2.88×10^3	1.000	7.02×10^2	1.000	1.00
	ECC	4.14×10^2	0.144	1.01×10^3	1.435	1.50
	ABFT	5.18×10^1	0.018	8.04×10^2	1.145	1.18
$Hotspot$	Unhardened	2.04×10^2	1.000	1.12×10^2	1.000	1.00
	ECC	1.81×10^1	0.089	1.61×10^2	1.439	1.00
	Spatial	0.00×10^0	0.000	1.05×10^2	0.937	2.51
	E-O Spatial	3.26×10^0	0.016	8.40×10^1	0.750	2.45
	Time	2.45×10^0	0.012	8.85×10^0	0.079	1.90

VII. HARDENING SOLUTIONS EVALUATION

In this section we evaluate the efficiency and efficacy of the ECC mechanism included in modern high-end GPUs. It is worth noticing that current low-power or embedded GPUs do not have any ECC or architectural-level protections. Moreover, we compare the ECC overhead and correction/detection capabilities with the ones of the software-based hardening solutions described in Section VI. Results for ABFT were obtained using 2048×2048 matrices for MxM and 512×512 64-point FFTs, while DWC strategies were applied to $Hostpot$. Nevertheless, similar error correction/detection capabilities are expected for different input sizes and algorithms. Whenever applicable, we discuss the generalization of the reported results to other algorithms or input sizes.

A. Error-Correcting Code Capabilities and Overhead

As explained in Section II-B, the ECC mechanisms included in advanced GPUs are able to correct single bit errors and detect double-bit errors on the main memory structures [10]. Data in Figure 6 demonstrates that a SECDED ECC detects

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

all the radiation-induced errors in memory elements and is sufficient to correct most of them.

Table III reports the reliability improvement and overhead when the whole ECC mechanism is enabled, relative to the unhardened version. Please mind that, when enabled, the ECC also reduces the DDR availability of about 15% [10]. The results show that enabling ECC reduces the SDC FIT by about one order of magnitude for all the tested algorithms. Some SDCs still occur even if ECC is enabled as flip-flops, queues, logic resources, and schedulers are left undetected. The former resources details are considered business-sensible, and are not available, while the latter resources have been demonstrated to contribute significantly to the GPU SDC rate [22].

The main issue with ECC is that even if the instruction cache and registers are protected [10], the GPU becomes more prone to experience a FI, possibly preventing the GPU from meeting reliability requirements in most safety-critical applications and imposing a higher checkpoint frequency in HPC applications. When ECC is off, an incorrigible MBU may be masked during computation, without appearing at the output [30]. Nevertheless, that same MBU would be detected by the ECC as an incorrigible error, triggering the application crash and, then, a FI. Additionally, the ECC will not reduce the FI caused by scheduler corruptions as those resources are left unprotected. When ECC is enabled, a 55% increase in FI is observed for MxM, 44% for FFT, and 43% for $Hotspot$ (refer to Table III). SDC and FI results for ECC were similar for the tested algorithms, and are expected to be comparable also for other codes.

The last column of Table III reports the execution time of the hardened codes normalized to their unhardened versions. Enabling ECC affects the execution time of FFT significantly, while it barely affects MxM and $Hotspot$. This is explained noting that FFT requires continuous accesses to the GPU DDR. When ECC is enabled, the data transfer bandwidth from and to the DDR is reduced as code data has to be transferred as well. If the unhardened version of the code saturates the bandwidth, as in the FFT case, the execution time is then likely to be higher when ECC is enabled.

B. ECC vs Software-Based Hardening Solutions

As can be seen in Table III, ABFT is extremely efficient in reducing the occurrence of SDCs. The FIT of both MxM and FFT protected with ABFT is less than 2% of the unhardened version. The ABFT FFT shows a SDC FIT that is almost one order of magnitude lower than the ECC protected version, while the ABFT MxM has a SDC FIT which is less than 18% the ECC protected one. The ABFT mechanisms are then more effective than ECC in increasing the GPUs resilience, since they are designed to detect mismatches in the output independently of their causes while, as said, ECC only corrects errors on memory elements directly affected by radiation.

The ABFT procedures to detect or correct errors are executed directly on the GPU, increasing the instruction caches requirements and scheduler strain. The higher number of operations needed to implement the ABFT procedure has the drawback of increasing the probability of having a FI, which is in accordance with experimental results. Nevertheless, the FI increase is not remarkable, being 12% for MxM and 14% for FFT, definitely lower than the 43% to 55% increment imposed by ECC.

ABFT correction capabilities were analytically calculated independently on input size [16], [17]. It is then reasonable to extend the reported resilience results to other input sizes. When implemented on GPUs, the complexity of ABFT procedures is linear with input data [14], [15]. The imposed ABFT overhead, both in terms of performance and FI caused by ABFT procedure corruption, is then expected to have a lower impact with increasing input size.

DWC strategies were also efficient, reducing the occurrence and even eliminating SDC for $Hotspot$. As DWC design is independent of the code, the imposed overhead is also likely to be independent on the code, and so the detection capabilities. The results and discussion that derive from $Hotspot$ analysis can then be applied to other parallel codes without significant variations. All errors where actually corrected with *Spatial*, while in *E-O Spatial* few errors were left undetected as the redundant block can be executed in the same SM and a shared resource error propagates to both copies. DWC strategies reduce the SDC rate by one order of magnitude with respect to ECC.

The FI rate of *Spatial* and *E-O Spatial* are comparable with the unhardened version and almost 50% lower than ECC-protected version. Only *Time* succeeds in reducing FI occurrences significantly. In both spatial DWC, the number of blocks to be dispatched is duplicated, increasing significantly the scheduler strain required for computation (which has been demonstrated to be particularly critical [22]). Even if some FIs are detected with spatial DWC, the higher block scheduler strain exacerbates the possibility of having a block stuck and the consequent GPU kernel hang. On the contrary, *Time* instantiates exactly the same number of blocks and threads of the unhardened version of $Hotspot$. Additionally, critical operations for FI, like blocks/threads dispatch, are interleaved with double operations in *Time* w.r.t. the other versions. This means that it is more likely for a neutron to generate a correctable SDC than a critical FI for *Time*.

Still, DWC comes with a non-negligible computational overhead, which ranges from 90% for *Time* to 151% for *Spatial*. *Time* performs better as there is no scheduler overhead, while in *Spatial* and *E-O Spatial* more blocks need to be created and scheduled. DWC pays its generality and ease of implementation with poor performance. DWC offers high detection capabilities, and may be particularly suitable for safety-critical application. In HPC, DWC usage should be carefully engineered to avoid excessive performance degradation.

The combination of ECC and ABFT or ECC and DWC is likely to provide an even higher reliability. Nevertheless, the resulting error rate would be extremely low even with the accelerated particle beams used in this evaluation, preventing the observation of a statistically significant amount of failures.

VIII. Related Work

Recently, there have been extensive scientific efforts studying and improving GPU reliability, motivated by the spread

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

of GPUs in both HPC and safety-critical areas [36], [40]. Some initial work in the HPC domain have started to look at reliability evaluation of GPUs [18] in general or exposed to terrestrial radiation [41], [42]. At the same time, fault injection experiments have been performed to track error propagation towards the GPU outputs and evaluate the Architectural Vulnerability Factor of several parallel codes [30], [32], [33], [43].

In this study, we complete the cited efforts by comparing the radiation sensitivity of all the major memory structures of modern GPUs, providing the probability of failure for register file, L1, and L2. Additionally, we highlight a 40% higher sensitivity of bits set to 0 than to 1 in the L2 caches. If these insights are taken into account, they can make fault injection tools/models more realistic and accurate. The occurrence of multiple bits upset in the GPU memory structures is also reported, validating the choice of a SECDED ECC as at most double bit errors were detected. The fault-injector presented in [30] will benefit from the reported multiple bit probabilities of occurrence. Unlike fault-injection, radiation experiments are normally performed on a limited set of applications. The wide set of benchmarks tested by our work is unprecedented for beam experiments and allows a better study of GPU operative behaviors under radiation. Moreover, MxM, $dgemm$, and NW were tested with different input sizes to evaluate the reliability dependence on different problem sizes and GPU throughput, both aspects that were never considered before.

Some works have aimed at enhancing the GPU reliability by evaluating the parallel code robustness to soft errors [44]. Experimentally-tuned and optimized hardening strategies for a limited set of algorithms have been proposed in [13], [14], [15]. A first comparison among ABFT, redundancy and ECC is presented in [45]. To complete these efforts, this study extend the hardening solutions evaluation to the considered set of applications and discusses the implication of their employment in safety-critical and HPC domains.

IX. Conclusions

We present an in-depth analysis of GPU radiation sensitivity both at low-level – by accessing the raw memory structures error rate – and at operative-level – by measuring the Silent Data Corruption and Functional Interruption rate of a representative set of parallel applications. Moreover, the available hardening solution for GPUs, including ECC, ABFT, and duplication were analyzed and their efficiency and efficacy experimentally and analytically compared.

The presented data serves as a pragmatic and precise estimation of the realistic error rate of modern GPUs exposed to the natural neutron beam and for the evaluation of the efficiency and effectiveness of architectural and software hardening solutions. Such an evaluation is useful for HPC centers and safety-critical applications designers to predict the GPU output error rate and engineer the selection and tuning of hardening solutions. Insights derived from our extensive radiation experiment data analysis and hardening solutions evaluation carry significant implications for future generations of GPU architectures, current and future HPC computing facilities, researchers focusing on GPU resilience, and end users.

X. Acknowledgments

The authors thank Luigi Carro, Philippe O. A. Navaux, and Nathan DeBardeleben for useful discussions, Damiano Zanchetta for designing the GPUs supports, Heather Quinn, Thomas Fairbanks, Steve Wender, and Tanya Herrera from Los Alamos National Laboratory and Chris Frost from Rutherford Appleton Labs for help in scheduling and performing the experiments. The K20 devices tested were donated thanks to Steve Keckler, Timothy Tsai, and Siva Hari from NVIDIA.

References

[1] E. N. C. A. Programme, "Euro NCAP Rating Review, Report from the Ratings Group," June 2012. [Online]. Available: http://www.euroncap.com

[2] E. S. Agency, "ESA COROT Mission Documentation," 2014. [Online]. Available: www.esa.int/Our_Activities/Space_Science/COROT

[3] J. Becker and O. Sander, "ARAMiS: Project Overview," 2013. [Online]. Available: http://www.across-project.eu/workshop2013/121108_ARAMIS_Introduction_HiPEAC_WS_V3.pdf

[4] J. Dongarra, H. Meuer, and E. Strohmaier, "TOP500 Supercomputer Sites: Nov. 2014," 2014. [Online]. Available: http://www.top500.org

[5] R. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 305–316, Sept 2005.

[6] R. Lucas, "Top Ten Exascale Research Challenges," in *DOE ASCAC Subcommittee Report*, 2014.

[7] S. Shazli, M. Abdul-Aziz, M. Tahoori, and D. Kaeli, "A Field Analysis of System-level Effects of Soft Errors Occurring in Microprocessors used in Information Systems," in *IEEE International Test Symposium 2008*, Oct 2008, pp. 1–10.

[8] N. Pandit, Z. Kalbarczyk, and R. Iyer, "Effectiveness of machine checks for error diagnostics," in *IEEE International Conference on Dependable Systems and Networks 2009*, June 2009, pp. 578–583.

[9] M. Breuer, S. Gupta, and T. M. Mak, "Defect and Error Tolerance in the Presence of Massive Numbers of Defects," *Design Test of Computers, IEEE*, vol. 21, no. 3, pp. 216–227, May 2004.

[10] NVIDIA, "NVIDIA Kepler K20 GPU Datasheet," 2012.

[11] S. Mitra, "System-Level Single-Event Effects," IEEE Nuclear and Space Radiation Effects Conference, NSREC 2012 Short Course, 2010.

[12] I. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[13] C. Braun, S. Halder, and H. Wunderlich, "A-abft: Autonomous algorithm-based fault tolerance for matrix multiplications on graphics processing units," in *Dependable Systems and Networks (DSN), 2014 44th International Conference on*, June 2014, pp. 443–454.

[14] L. Pilla, P. Rech, F. Silvestri, C. Frost, P. Navaux, M. Sonza Reorda, and L. Carro, "Software-based hardening strategies for neutron sensitive fft algorithms on gpus," *Nuclear Science, IEEE Transactions on*, vol. 61, no. 4, pp. 1874–1880, Aug 2014.

[15] P. Rech, C. Aguiar, C. Frost, and L. Carro, "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 4, pp. 2797–2804, 2013.

[16] K.-H. Huang and J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *Computers, IEEE Transactions on*, vol. C-33, no. 6, pp. 518–528, June 1984.

[17] J.-Y. Jou and J. Abraham, "Fault-Tolerant FFT Networks," *Computers, IEEE Transactions on*, vol. 37, no. 5, pp. 548–561, 1988.

[18] S. Di Carlo, G. Gambardella, I. Martella, P. Prinetto, D. Rolfo, and P. Trotta, "Fault mitigation strategies for CUDA GPUs," in *Test Conference (ITC), 2013 IEEE International*, Sept 2013, pp. 1–8.

[19] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," JEDEC Standard, Tech. Rep. JESD89A, 2006.

[20] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger, "Comparison of error rates in combinational and sequential logic," *Nucl. Sci., IEEE Transactions on*, vol. 44, no. 6, pp. 2209–2216, 1997.

[21] N. Mahatme, T. Jagannathan, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, "Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 6, pp. 2719–2725, 2011.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2444855, IEEE Transactions on Computers

[22] 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 *IEEE International Conference on Dependable Systems and Networks (DSN 2014)*, Atlanta, USA, 2014.

[23] M. Violante, L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, C. Andreani, G. Gorini, A. Pietropaolo, G. Cardarilli, S. Pontarelli, and C. Frost, "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 1184–1189, 2007.

[24] S.-H. Kim, W.-O. Lee, J.-H. Kim, S.-S. Lee, S.-Y. Hwang, C.-I. Kim, T.-W. Kwon, B.-S. Han, S.-K. Cho, D.-H. Kim, J.-K. Hong, M.-Y. Lee, S.-W. Yin, H.-G. Kim, J.-H. Ahn, Y.-T. Kim, Y.-H. Koh, and J.-S. Kih, "A low power and highly reliable 400Mbps mobile DDR SDRAM with on-chip distributed ECC," in *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, Nov 2007, pp. 34–37.

[25] G.-H. Asadi *et al.*, "Balancing Performance and Reliability in the Memory Hierarchy," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, ser. ISPASS '05. Washington, DC, USA: IEEE Computer Society, 2005.

[26] M. Bajura, Y. Boulghassoul, R. Naseer, S. DasGupta, A. Witulski, J. Sondeen, S. Stansberry, J. Draper, L. Massengill, and J. Damoulakis, "Models and algorithmic limits for an ecc-based approach to hardening sub-100-nm srams," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 935–945, Aug 2007.

[27] J. F. Ziegler and H. Puchner, *SER–history, Trends and Challenges: A Guide for Designing with Memory ICs*. Cypress, 2010.

[28] M. White, "Scaled CMOS Technology Reliability User Guide," Jet Propulsion Laboratory, NASA, Tech. Rep., 2010.

[29] T. Fairbanks, H. Quinn, J. Tripp, J. Michel, A. Warniment, and N. Dallmann, "Compendium of TID, Neutron, Proton and Heavy Ion Testing of Satellite Electronics for Los Alamos National Laboratory," in *Radiation Effects Data Workshop (REDW), 2013 IEEE*. IEEE, 2013, pp. 1–6.

[30] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. Kaeli, "Calculating Architectural Vulnerability Factors for Spatial Multi-bit Transient Faults," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[31] 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 *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 29–.

[32] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A Methodology for Evaluating the Error Resilience of GPGPU Applications," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

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

[34] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmark Suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[35] NVIDIA, "CUBLAS Library User Guide," http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf, 2014.

[36] T. Devesh, G. Saurabh, R. Jim, M. Don, R. Paolo, V. Sudharshan, O. Daniel, L. Dave, D. Nathan, N. Philippe, C. Luigi, and B. Buddy, "Understanding GPU Errors on Large-scale HPC Systems and the Implications for System Design and Operation," in *21st IEEE Symp. on High Performance Computer Architecture*, February 2015.

[37] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann, 2012.

[38] C. Weaver *et al.*, "Techniques to reduce the soft error rate of a high-performance microprocessor," in *International Symposium on Computer Architecture (ISCA'04)*. IEEE Press, 2004, pp. 264–275.

[39] R. Freivalds, "Fast probabilistic algorithms," in *Mathematical Foundations of Computer Science 1979*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1979, vol. 74, pp. 57–69.

[40] L. A. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, S. Keckler, K. Pattabiraman, R. Rech, and M. S. Reorda, "GPGPUs: How to Combine High Computational Power with High Reliability," in *2014 Design Automation and Test in Europe Conference and Exhibition*, Dresden, Germany, 2014.

[41] 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 *IEEE 10th Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2014.

[42] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, "GPU Behavior on a Large HPC Cluster," *6th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids, Aachen, Germany,*, August 26-30 2013.

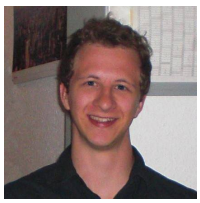
[43] I. Haque and V. Pande, "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rates in GPGPU," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 691–696.

[44] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen, "Matrix Multiplication on GPUs with On-Line Fault Tolerance," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, May 2011, pp. 311–317.

[45] D. A. G. Oliveira, P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro, "GPGPUs ECC Efficiency and Efficacy," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2014.

Daniel Alfonso Gonçalves de Oliveira is a Ph. D. student at Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil. He obtained his master's degree in 2013 from the Federal University of Rio Grande do Sul, Brazil. His research interests include radiation tests of parallel architectures and software-based hardening techniques for HPC algorithms.

Laércio Lima Pilla holds an associate professor position in the Federal University of Santa Catarina, Brazil. He obtained his Ph.D. degree in Computer Science in 2014 in a joint doctorate between the Federal University of Rio Grande do Sul, Brazil, and the University of Grenoble, France. His research interests include topology-aware scheduling and software-based fault-tolerance on GPUs.



Thiago Santini is a Ph. D. student and research assistant at the University of Tübingen, Germany. He obtained his master's degree (with honors) in 2015 from the Federal University of Rio Grande do Sul, Brazil. His research interests include the evaluation and mitigation of radiation-induced soft errors in applications and operating systems.

Paolo Rech is an associate professor at Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil. He received his master and Ph.D. degrees from Padova University, Padova, Italy, in 2006 and 2010, respectively. His main research interests include radiation tests of FPGAs, SoC, parallel and heterogeneous systems; the design of efficient hardening techniques for parallel algorithms, and the evaluation and mitigation of radiation-induced effects in devices designed for large-scale HPC centers.