Experimental Findings on the Sources of Detected Unrecoverable Errors in GPUs

Fernando Fernandes dos Santos, Sujit Malde, Carlo Cazzaniga, Christopher Frost, Luigi Carro, and Paolo Rech

***Abstract*—We investigate the sources of Detected Unrecoverable Errors (DUEs) in Graphics Processing Units (GPU) exposed to a neutron beam. Illegal memory accesses and interface errors are among the more likely sources of DUEs. ECC increases the launch failure events. Our test procedure has shown that ECC can reduce the DUEs caused by Illegal Address access up to 92% for Kepler and up to 98% for Volta. Additionally, we analyze if the compiler optimizations can impact the DUE sources distribution for the Matrix Multiplication. We found that the machine codes generated by the different optimization levels can change the DUE source by no more than 24% on average.**

***Index Terms*—Detected Unrecoverable Error, reliability, radiation experiments, Graphic Processing Units**

I. Introduction

Graphics Processing Units (GPUs) have evolved from graphics rendering to general-purpose accelerators extensively employed in HPC and safety-critical applications such as autonomous vehicles for the automotive and aerospace markets. The GPUs' highly parallel architecture fits most HPC codes' computational characteristics and machine learning applications. The most recent GPU architecture advances, such as tensor core and mixed-precision functional units, move toward improving the performances and software flexibility for HPC and deep learning applications [1].

Today, the reliability of parallel processors is a significant concern for both safety-critical applications and HPC systems. Unexpected errors in parallel devices' may lead to catastrophic accidents in a self-driving vehicle, lower HPC systems scientific productivity, lower operational efficiency, and even significant monetary loss.

Most recent studies target Silent Data Corruption (SDC) in their evaluation. SDCs, being undetectable, are, in fact, considered the main threat for modern computing device's reliability [2], [3]. Detected Unrecoverable Errors (DUEs), such as device hangs, application crashes, or functional interruptions, are considered less harmful as, being detectable by definition, they could be easily handled using solutions such as checkpoints and software/hardware watchdogs [4]–[6]. Nevertheless, the recovery from a DUE or the action taken to reach a fail-safe state requires a significant amount of time, which risks reducing supercomputers' productivity [7]. A small cluster with 32K cores would take almost an hour to restart after a crash [4], without considering the overhead of performing checkpointing time.

In safety-critical real-time systems, such as autonomous vehicles, the DUE risk is even higher, as it may compromise the system's ability to complete the task before the deadline. For instance, a GPU for autonomous vehicles must process between 20 to 80 frames-per-second [8], [9]. The recovery from a DUE must be sufficiently efficient not to miss any frame, which is highly challenging. In this scenario, tracing the DUEs software and hardware sources and quickly identifying a DUE occurrence are essential tools to create more tolerant applications against crashes and hangs. Additionally, as few works cover the DUE analysis for GPUs [10], [11], it is still unknown how much the compiler optimizations impact the DUEs rate of an application.

In this paper, we investigate the sources of DUE in two NVIDIA architectures: Kepler and Volta. We provide a novel and detailed analysis of DUE sources on GPUs based on neutron experimental data and system logs profile. We create an experimental setup that allows the tracing of the GPU crashes and hangs observed during radiation experiments. We select a set of eight algorithms and compare their DUE and SDC rates, considering both the case of ECC disabled and enabled. Each code has peculiar characteristics regarding memory utilization, computing power, control-flow operation, highlighting specific architecture behaviors that could be generalized to similar algorithms.

For a particular code, Matrix Multiplication, we also perform a deeper analysis of the DUEs by evaluating different compiler versions and three optimization levels. We tested two CUDA compilers, 10.2 and 11.3, to assess if the improvements of a newer compiler can modify the error sources of an application. The compiler optimizations analysis aims to find if the machine code generated impacts the DUEs events. Additionally, we report findings from recently completed (remotely controlled) neutron beam testing that represents a total of more than 2 million years of operation in a natural environment. Finally, we discuss how system log tracing can be used to improve the detection and recovery from DUEs. The main contributions of this work are as follows:

- A deep investigation on the DUE sources (i.e., manifestation at software level) based on experimental data.
- Eight benchmarks reliability evaluation. The benchmarks in this work come from broad domains, such as machine learning (YOLOv3) and particle simulation (FLAVA).
- Experimental data and findings on the impact in the DUE rate when Error Correction Code is enabled and disabled.

F. F. dos Santos and L. Carro are with Institute of Informatics of Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. E-mail: {ffsantos, carro}@inf.ufrgs.br

S. Malde, C. Cazzaniga, and C. Frost are with Science and Technology Facility Council (STFC), UKRI, United Kingdom. E-mail: {sujit.malde, carlo.cazzaniga, christopher.frost}@stfc.ac.uk

P. Rech is with Department of Control and Computer Engineering of Politecnico di Torino, Italy. E-mail: paolo.rech@polito.it

- Evaluation of the impact of optimization flags and compilers on the DUE sources on beam experiments.

The remainder of the paper is organized as follows. The next Section presents the Background on radiation-induced SDCs and DUEs. The DUE classification is presented in Section III. Section IV presents the tested devices, the selected parallel algorithms and describes the evaluation methodology. Section V shows the radiation experimental results, and Section VI present the DUE source analysis. Section VII concludes the paper.

II. Radiation induced SDCs and DUEs in GPUs

Radiation-induced events are particularly critical, as they dominate error rates in commercial devices [2]. A transient fault may lead to one of the following outcomes: (1) No effect on the program output (i.e., the fault is masked, or the corrupted data is not used). (2) A Silent Data Corruption (SDC) (i.e., an incorrect program output). (3) A Detected Unrecoverable Error (DUE) (i.e., a program crash or device reboot).

Previous studies have stated that parallel architectures, particularly GPUs, have a high fault rate because of the high amount of available resources [7], [12], [13]. Recent works have identified some peculiar reliability weaknesses of GPUs architecture, suspecting that the corruption of the GPU hardware scheduler or shared memories can severely impact the computation of several parallel threads [3], [7], [12], [14], [15]. As a result, multiple GPU output elements can potentially be corrupted, effectively undermining several applications' reliability, including CNNs [16], [17].

Even if DUEs are detectable, they still can lead to monetary loss or harmful events. For instance, a self-driving car that relies on a GPU to perform object detection, if rebooted, can delay a response to a critical situation, thus putting human lives in danger. DUEs are generated by data-driven control-flow errors (the corruption of a memory address, an index, a jump instruction, etc.) or by faults in control logic (scheduler, memory controller, interfaces for synchronization, etc.). Additionally, the available Single Error Correction Double Error Detection (SECDED) ECC on GPUs crashes the application when a double-bit flip is detected.

Few works have studied the DUEs on GPUs on beam experiments [3], [10], [18]–[20]. However, most of the mentioned works do not present a detailed analysis of the events that cause DUEs on GPUs, not allowing a deep investigation of the weakness of the GPUs. Kojiro *et. Al.* [11] perform DUE analysis with neutron beam experiments and software fault injection manipulating the Program Counter (PC) The authors conducted experiments with two NVIDIA architectures and two benchmarks, Matrix Multiplication and Object Detection. The paper demonstrates that it is possible to trace the NVIDIA driver error messages to monitor the errors generated by the GPU in the radiation test. However, the article lacks generality as only two benchmarks are presented. Additionally, it is difficult to use the fault injection results to correlate with the DUEs obtained on radiation experiments, as the fault injection is limited to the application Program Counter. In contrast, on the beam experiments, all the device is exposed.

Our paper's goals are:

1) Understand the (software/hardware) causes of DUE in GPU. DUEs events are expected to be generated in other places than functional units [10], [11]. As we show, DUEs come from incorrect memory addresses, errors in the scheduler, errors in the stack, etc. (more details Section III).
2) Show how GPU system logs can be used to have prompt information about the DUE occurrence, triggering the action to take faster. We demonstrate that by showing the results for eight codes in two NVIDIA architectures.
3) Demonstrate how the compiler optimization levels (i.e., O0, O1, O3) and the compiler version (i.e., CUDA 10.2 and 11.1) can impact the DUE distribution of an application.

III. GPU DUE source tracing

While to detect SDC is sufficient to compare the experimental output with the pre-computed expected one, to identify the event that triggered the DUE, it is necessary to query the device or the operating system to understand what happened. GPUs feature the CUDA Runtime Application Programming Interface (CUDA Runtime API), a layer between the operating system and device hardware. With CUDA Runtime API, we can directly control the GPU using function calls. For instance, we can make a software reset on the GPU (i.e., delete all memory and kernels resources) by calling *cudaDeviceReset()* from a C++ code [21].

We used the CUDA Runtime API to trace and log the DUE events. After each GPU kernel execution, we query the device (cudaGetLastError followed by a cudaDeviceSynchronize) to check if any error happened. If an error code is returned, we finish the application and start a new one. All the possible errors that an NVIDIA GPU can generate are described in the CUDA Runtime API Guide, Section 6.3 [21].

It is worth noting that only the GPU is exposed to the beam, while the host CPU and DDR memories are outside of the beam. The GPU is connected to the motherboard with a 25cm PCI-e extension, and the motherboard is covered with boron plastic blocks to further reduce the probability of events induced by scattering neutrons in the host PC. Consequentially, the majority of the observed errors are caused by events on the GPU.

In the following, we list and describe the possible DUEs we have identified.

Devices Unavailable: the GPU is unavailable either in an Exclusive or Prohibited mode or the previous execution has not released the resources (the GPU is "falsely" full).

Illegal Instruction: an illegal instruction is executed, leaving the process in an inconsistent status.

Illegal or Misaligned Address: the address of a Load or Store does not point to a valid memory address or is not aligned.

Initialization Error: the CUDA driver fails to initialize, making it impossible for the API to continue.

Invalid Device or No Device: the GPU queried by the CPU is invalid, or the CUDA driver cannot detects a valid GPU device.

Invalid PC: one of the kernel program counters is corrupted and becomes invalid.
Invalid Address Space: a kernel can operate in different memories spaces (i.e., global, shared, or local). This error happens when an instruction that belongs to a specific memory space tries to operate in a different one.
Memory Allocation: CUDA is not able to allocate memory on GPU.
ECC Uncorrectable: the ECC detects more than one bit flip. As this cannot be corrected, the ECC throws an exception to the driver.
Launch Failure: the CPU tries to launch a kernel and fails. There are many reasons it could happen, such as dereferencing an invalid device pointer or accessing out-of-bounds shared memory, etc.
Hardware Stack Error: an error in the call stack during kernel execution, generally due to stack corruption or exceeding the stack size limit.
Invalid Value: some kernel parameters are out of the acceptable range of values, i.e., more threads than the GPU supports.
System Crash: a DUE triggered by the watchdog in our setup. Then DUE source could not be traced or is generally unknown.

TABLE I: Benchmarks used for reliability evaluation

	Domain	Suite
Component Labeling - CCL	Graph	NUPAR [22]
Breadth First Search - BFS	Graph	
Lava	N-Body	Rodinia [23]
Gaussian	Linear Algebra	Rodinia [23]
LU Decomposition - LUD	Linear Algebra	Rodinia [23]
Matrix Multiplication - MXM	Linear Algebra	Rodinia [23]
Optimized MXM - GEMM	Linear Algebra	Rodinia [23]
Mergesort	Sorting	NVIDIA SAMPLES
Quicksort	Sorting	NVIDIA SAMPLES
YOLOv3	Object Detection	Darknet [24]

IV. Evaluation Methodology

This section first describes the devices and codes we characterize, the metrics adopted for the reliability evaluation of computing devices, and how we measure them for GPUs.

Devices: We consider Kepler (Tesla K40) and Volta (Titan V and Tesla V100) NVIDIA GPUs. The tested NVIDIA K40 (**Kepler**) is built with the Kepler ISA and fabricated in a $28nm$ TSMC standard CMOS technology [25]. This model has 2880 CUDA cores divided into 15 Streaming Multiprocessors (SMs). Each K40 SM has 64K registers, 64KB of L1/shared memory. The GPU has 1.5MB of L2 cache, and 6GB GDDR5 memory. Single Error Correction Double Error Detection (SECDED) Error Correcting Code (ECC) protects the register file, shared memory, and caches while read-only data cache is parity protected. It is worth noting that even when ECC is enabled, errors can still be generated in the address calculation, the memory buffers, or any unprotected memory resource, as we show in Section V.

The Titan V and Tesla V100 (**Volta**) are designed with the Volta micro-architecture and built with TSMC FinFET 12nm [26]. Volta GPUs feature hardware acceleration for three IEEE754 float point precisions: double, float, and half. Each of the 80 Volta SMs has 64 FP32 cores, 64 INT32 cores, and 32 FP64 cores [27], [28]. Each Volta SM has 64K registers and 96KB of configurable L1/shared memory. Titan V has 4.5MB of L2 cache, and V100 has 6MB of L2 cache. Volta also includes eight *tensor cores*, i.e., specific hardware that performs the Matrix Multiplication and Accumulate (MMA) operation on $16x16$ matrices. Only the V100 has SECDED ECC on the register file, shared memory, and caches. The difference between the Titan V and Tesla V100 is that while V100 operates at 1246MHz as a base clock, Titan V operates at 1200MHz. Tesla V100 has 16GB of HBM2 RAM. Titan V has 12GB HBM2 RAM. Finally, and more importantly, only the V100 has SECDED ECC on the main memories.

Our evaluation considers errors occurring only in the GPU core, not in the main memory. For Kepler, we chose a beam spot sufficiently small (2cm of diameter) not to hit the onboard DDR when ECC is disabled. As Volta's HBM2 memories are on top of the chip, when ECC is disabled, all the global memory accesses are made through Triple Modular Redundancy (TMR). For FMXM and FLAVA on Volta, when the ECC is disabled, the kernel starts loading the input from three different DDR positions, then compares the three memories and corrects the divergence. The correct value is then stored in the shared memory or in the register file to be used for computation. When computation finishes, the output registers are written in three different locations of the global output memory. The CPU checks and corrects if there is a divergence between the three outputs

Tested Codes: We chose the eight representative codes listed in Table I from HPC and deep learning domains. The choice of diverse codes increases this work's quality and can provide details for DUEs extendable to different applications [29]. Because of its importance in CNNs and HPC, we choose to pay particular attention to Matrix Multiplication and test both the naive version (*MxM*) and the optimized version that digest data in the most suitable way for GPUs as General Matrix Multiplication (*GEMM*) from the NVIDIA CUBLAS libraries. To be highly efficient, GEMM kernel is tuned for each input and device configuration. Float-based codes have their precision in their names. D for double, F for float, and H for half. INT32 based codes do not have their names modified.

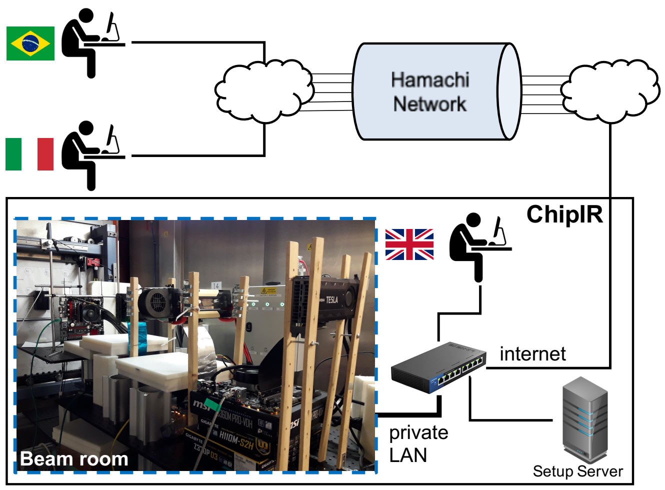

Fig. 1: Remotely controlled beam experiment setup

Beam Experiment Setup: Beam experiments are the most effective way to measure the FIT rate of code running on a computing device, which is obtained by dividing the number of observed errors by the received particles fluence ($neutrons/cm^2$). Our experiments were performed at the ChipIR facility of the Rutherford Appleton Laboratory, UK. Figure 1 shows the setup mounted in the ChipIR facility. The facility delivers a beam of neutrons with a spectrum of energies that resembles the atmospheric neutron one [30]. The available neutron flux was about $3.5 \times 10^6 n/(cm^2/s)$, $\sim$8 orders of magnitude higher than the terrestrial flux (13 $neutrons/(cm^2 \cdot h)$ at sea level [31]).

Since the terrestrial neutron flux is low, it is improbable to see more than a single corruption during program execution in a realistic application. We have carefully designed the experiments to maintain this property (observed error rates were lower than 1 error per 1,000 executions). Experimental data can then be scaled to the natural terrestrial environment without introducing artifacts. Each code was tested for at least 72 hours, not including the setup, result check, initialization, and recovery from the crash time.

Most experiments were performed during the *COVID-19 pandemic*. Consequentially, the setup was adapted for remote control as not all the scientists involved in this research were physically present at the experiments. Figure 1 shows the adapted setup for mobility restrictions. A Virtual Private Network was created using Hamachi LogMeIn to connect researchers in Brazil, Italy, and United Kingdom (ChipIR). A researcher was in charge of assembling the hardware setup on the ChipIR facility and configuring the network. The other researchers were then able to monitor and adjust the irradiated GPUs' applications remotely.

We added the setup software and hardware watchdogs to monitor and perform recovery from crashes. The software watchdog controls the application under test, and if it stops responding in a predefined time interval, the kernel is killed and relaunched. This watchdog detects kernel crashes or software hangs, i.e., application crashes or control flow errors that prevent the GPU from completing assigned tasks (e.g., an infinite loop). Before finishing the process related to the GPU's kernel, our setup makes a CUDA API call to get the last error that happens inside the GPU. Then, the return of the API call is logged in a structured file. The hardware watchdog is an Ethernet-controlled switch that performs a host computer's power cycle if the host computer itself does not acknowledge any ping requests in a predefined time interval. The hardware watchdog is necessary to detect when the operating system hangs.

V. Neutron Beam Experiments Results

In this section, we present the results from radiation experiments for the eight evaluated benchmarks. We show the SDC and DUE rates for the codes when ECC is ON and when ECC is OFF.

Figure 2 shows the experimentally measured SDC and DUE *normalized FIT* rates for the GPUs executing the codes with ECC disabled and enabled. The reported data is the measured FIT rate divided by Mergesort SDC rate on Kepler when ECC is ON. We report arbitrary units (a.u.) not to reveal business-sensitive data. Values are reported with 95% confidence intervals considering a Poisson distribution.

Not surprisingly, both the SDC and DUE FIT rates of Kepler and Volta change significantly depending on the code. It is interesting to notice that, on the *Average*, the SDC FIT rate is higher than the DUE rate when ECC is OFF (1.2× for Kepler, 4.1× for Volta), while when the ECC is ON, the DUE rate is (significantly) higher than the SDC rate (2.2× for Kepler, 2.7× for Volta). This is because the ECC reduces (significantly) SDCs and increases DUEs, primarily because of ECC Uncorrectable errors (details in Section III).

The average SDC FIT rate with ECC OFF is up to 3.5× higher than with ECC ON for Kepler and 2.6× higher for Volta. When ECC is ON, the DUE FIT rate is almost the same as ECC OFF for Kepler (20% difference) and 4.2× higher than ECC OFF for Volta. The DUE increase caused by ECC is exacerbated in Volta GPUs because the main memory is stacked on top of the chip, consequentially being irradiated as the GPU core. It is then likely for neutrons to corrupt also data in the DDR and, in the event of a double bit flip, the ECC triggers a DUE. It is worth noting that, when the ECC is OFF, we triplicate data in the DDR to avoid its corruption to bias our SDC evaluation.

SDCs, being *silent*, are typically considered much more critical than crashes and hangs. The most used techniques to detect and correct SDCs require modular redundancy or algorithm-based fault tolerance. On the other hand, DUEs are more treatable as the event is, by definition, *detected*. Nevertheless, even if a plethora of methods exist to mitigate the impacts of DUEs (such as checkpoint-restart), the overhead of the recovery from a DUE can be extremely high, as the execution must be restored or the device (or even the whole server) must be rebooted.

Figure 2 shows, for some codes, the DUEs FIT can be higher than SDC FIT. Constant interruptions on complex systems such as supercomputers and embedded safety-critical systems can lead to performance degradation, causing significant impact as much as SDCs. For Mergesort and YOLOv3 (object detection CNN), on Kepler, even with ECC OFF, the DUE rate is 4.4× and 5.1× higher than the SDC rate, respectively. Mergesort continuously requires the GPU to exchange data with the CPU as the host performs the input array split. Similarly, YOLOv3, which is a neural network of more than 100 layers, launches several kernels per layer and requires a high amount of data transfer through the system bus between the host processor and the GPU (each layer output is used as input in the downstream layer), which increases the probability of invalid device pointers to be corrupted leading to a DUE. The synchronizations between CPU and GPU and heavy memory exchange are also highly susceptible, and their corruption is likely to cause a DUE. As a counter-proof, Quicksort, despite solving the same sorting problem (with the same input) as Mergesort, uses *Dynamic Parallelism*, which allows launching kernel inside the GPU without requiring data exchange with the CPU. As a result, Quicksort has a much higher SDC rate than the DUE rate *when ECC is OFF*. Quick-

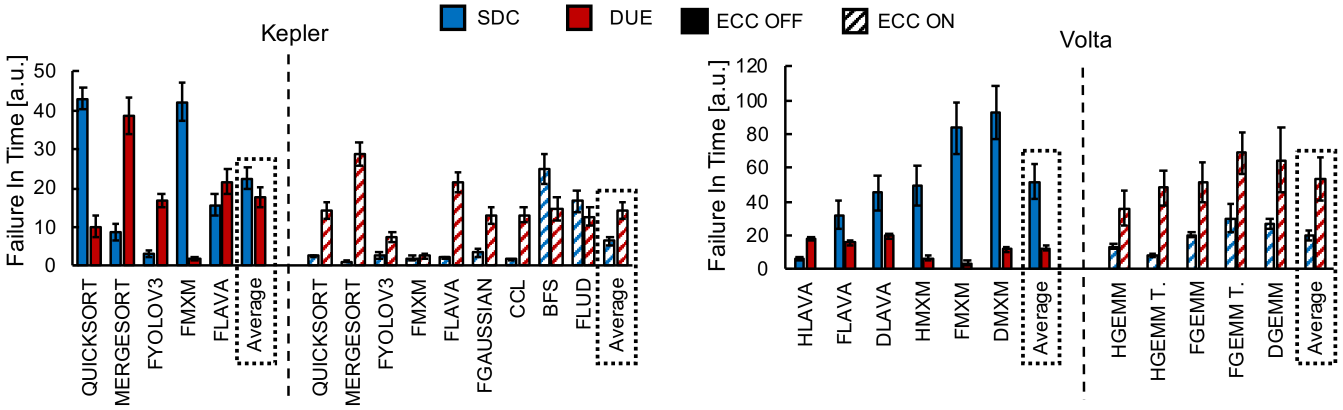

Fig. 2: Normalized FIT rates for Kepler and Volta.

sort uses the GPU resources better than Mergesort, avoiding synchronization with Dynamic Parallelism. Consequentially, increasing Quicksort SDC rate when ECC is OFF (that is 4.9× higher than Mergesort SDC rate) but, when ECC is ON, most SDCs are corrected, making Quicksort more reliable than Mergesort.

For Volta GPU, we also perform the analysis for FIT rates of codes executed with different precisions: Double 64bit (D), Float 32bit (F), and Half 16bit (H). A higher precision functional unit has a higher area and a higher probability of being hit by a neutron since the circuit area is bigger [1], [32]. Thus, higher precision implies more bits to store data, which is linearly dependent on the FIT rate. The SDC FIT rates for Volta in Figure 2 confirm this trend.

When ECC is OFF, the DUE rate is almost constant for the different precisions (while it changes between Lava and MxM). This is because the probability of faults in the control circuit, interfaces, host-device communications (likely to generate a DUE) does not depend on the data precision. Other sources of DUEs, such as host-device synchronizations, data-driven control-flow errors (corrupted indexes, addresses, or jumps), depend on the application. So, when ECC is off, the DUE will have other sources than ECC Uncorrectable errors. Interestingly, when ECC is ON, the DUE FIT rate increases linearly with data precision. This is again due to the DDR being stacked over the GPU chip. In fact, the higher the amount of data, and consequentially, the higher the probability of having ECC uncorrectable errors. This is confirmed by the analysis of the DUE sources we present in the next Section, which shows that, for Volta with ECC ON, most DUEs come from uncorrectable errors.

VI. DUE source

In this Section, we present our findings on the analysis of DUE sources. First, we start discussing how the particularities of each application, discussed in Section V, can change the occurrences of DUEs. Then we select the FMXM app to perform a deeper analysis of the compiler optimizations impact on the final application DUE causes.

Thanks to the setup described in Section III, we can trace the cause of the observed DUEs. Figure 3 shows, in percentage, the sources of DUEs we have identified in our beam experiments for Kepler and Volta. In Figure 3 we have grouped Device Unavailable, Invalid Value, No Device, Initialization Error, Hardware Stack Error, and Invalid Device events under the category ”Other” as, on average, the combination of these events caused less than 3% of DUEs in our experiments.

Figure 3 shows that DUEs' source is code and architecture dependent. When ECC is disabled, most DUEs are originated by *Illegal Address* (i.e., the code tries to access an incorrect memory address), which is to be expected since the Register File (RF), shared, and cache memories are unprotected, including the locations that store the memory addresses for Load/Store instructions. A corruption in the memory address is likely to violate the memory policy leading to an Illegal Address. When ECC protects memories, there is a drastic reduction of Illegal Address DUEs (less than 8% on Kepler and less than 2% on Volta) and an expected increase in the probability of *ECC Uncorrectable* errors (which are obviously absent when ECC is off). As mentioned, the DUE rate is exacerbated on Volta when ECC is enabled as DDR is on top of the GPU. We have experienced that the combination of transient, permanent, and intermittent errors in the stacked DDR makes double-bit errors probability very high.

Launch Failures are also frequent, mainly for Kepler with ECC on and for Volta. These DUEs happen when the GPU is in an inconsistent state due to corrupted parameters or an error at the kernel launch. For instance, an error invalidates the memory pointers passed to the kernel as parameters, making the GPU unable to launch the execution.

Generally, a *System Crash* source is hard to identify as this DUE is generated by exceptions from the host operating system or hanging kernels inside the device, preventing logging API data. Our watchdogs kill the application, reset the GPU, or perform a system power cycle in a System Crash event. As the System Crash percentage is not negligible, it would be necessary to investigate their causes deeper. As future work, we plan to consider the operating system logs related to the PCI bus and the GPU for an in-depth study of System Crashes.

Finally, BFS is the only code experiencing a large number of *Memory Allocation* DUEs. BFS is a code that manages the GPU memory inefficiently. BFS allocates a graph with

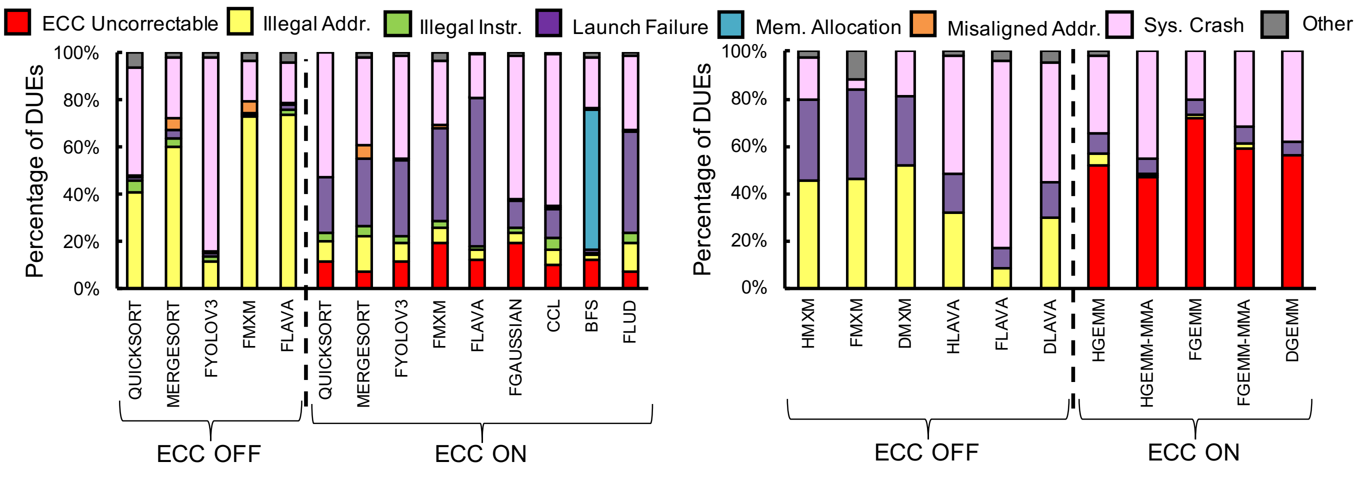

Fig. 3: Detailed DUE sources for Kepler and Volta GPUs. Other Sources includes Devices Unavailable, Invalid Value, No Device, Initialization Error, Hardware Stack Error, and Invalid Device.

1 million vertices for each CUDA parallel stream (i.e., a CUDA stream is an instance of a BFS kernel that runs parallel with other kernels). 50 CUDA streams will be launched. Consequentially, there will be a high memory demand and irregular memory accesses for each stream. Thus, it is expected that BFS will have the majority of the DUEs coming from memory errors.

In all the DUE sources we have seen but System Crash, in which the only solution is a power cycle or a device reset, the existing tools allow us to get data from the device to have prompt information about the DUE occurrence. Tracing logs can be used to design codes that can self-detect and self-recover from DUEs. For instance, NVIDIA Management Library gives details about ECC Uncorrectable errors. This information could be used not to crash the application and trigger a new data fetch or roll-back. We conclude that GPU system logs can be used as DUE detection as already done in other techniques that provide certification and code changes to improve the software reliability [33]. The data presented in this paper can be valuable to propose code certifications for GPUs kernels to improve code reliability.

A software engineering methodology for safety-critical applications can define actions to be performed soon after a traceable DUE occurs but before the watchdog triggers the device or system reboot, saving precious time. We show that for most of the benchmarks, most DUEs the source are traceable and allow a soft GPU reboot (a driver reset) without the need of a reboot of the whole system. If each each DUE was considered simply as "System Crash" most of the recovery time would be spent in the power cycling, with an evident waste of resources.

A. *Code optimization impact on the DUE*

Past works have shown that the compiler optimizations can change the SDC rate on beam experiments and fault injection [34]–[37]. As the compiler optimizations can also change the DUE causes, we decided to analyze FMXM running on Kepler to show the impact of different optimization levels in the DUEs sources. We select three configurations, 00, 01, and O3, for two versions of the CUDA compiler 10.2 and 11.3. The O2 optimization was not tested as the machine code generated for O2 is identical to the O1. Many optimization and bug corrections were implemented after CUDA 11, including memory allocation bugs and CUDA API performance improvements [38]. Consequentially, we decide to evaluate the DUE sources of two different compilers under radiation.

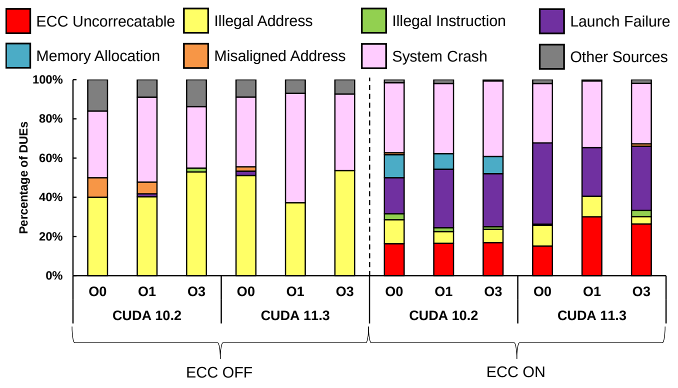

Fig. 4: Different optmizations levels on the MxM kernel impact on the distribution of the DUE sources

Figure 4 shows the FMXM DUEs source distribution for three optimization levels, i.e., O0, O1, and O3, for two CUDA compilers, 10.2 and 11.3. On average, the different optimization levels do not significantly change the DUE distribution, and this holds for all configurations. The observed differences are up to 24%. The most significant variation can be observed between the compilers (10.2 vs 11.3) when ECC is ON. Interestingly, we can see that Memory Allocation errors happen on CUDA 10.2 and not in CUDA 11.3. The memory allocation DUEs reduction on CUDA 11.3 may be related to memory allocation bugs correction for CUDA versions higher than 11, reducing these errors to the minimum at CUDA 11.3, reducing these errors to the minimum at CUDA 11.3. A bug in the

CUDA driver can change the device's behavior in the presence of an error. Consequentially, when the application tries to allocate memory, and the driver is in an inconsistent status, an exception can be raised, generating *Memory Allocation* error.

When comparing the DUEs sources for ECC OFF and ON, the distributions follow the same pattern presented for the other codes. With the *Illegal Address* dominating the ECC OFF, and *ECC Uncorrectable errors* and *Launch Failures* governing when ECC ON. As shown in Figure 4, the code optimizations are not enough to change the DUEs distribution drastically. Even if we consider two different compiler versions, the variations are not significantly different. Consequentially, we can deduce that the DUEs are majorly governed by issues generated in the hardware and the driver. The final machine code generated by the compiler can only produce minor variations at the DUE causes.

Even for a simple and naive code like MXM, the compiler can still optimize the final machine code (removing dead code and unnecessary memory movements) and change the application's outcome for some configurations when a fault happens. For instance, while the O0 code size is up to 6× larger than the O3 code, the O1 is almost the same as the O3 code. Since different instructions have different error rates and error probability, unnecessary instructions can generate differences in the DUE rate and the DUE sources, as shown in Figure 4.

It is worth noting that the performance is directly proportional to the optimization flags. As the optimization level increases, the execution time decreases. At the same time, as the performance increases, the resources utilization increases and so does the FIT rate. Nevertheless, the performance gain is such that the useful output produced before experiencing a failure is higher for the more optimized codes.

VII. Conclusions

We have discussed the causes of DUEs for various benchmarks to precisely evaluate the GPU behavior under radiation. The ECC, in fact, changed the outcome of the DUEs for all the tested codes. We experimentally show that ECC is a powerful technique to reduce the SDC rate for errors in the GPU memories, but it can change how the GPU manifests the DUE events.

Additionally, we show that the optimization performed by the compiler can change the DUE source. However, it is more dependent on the hardware and the driver than on the final machine code. In future research, we plan to deeply study the DUEs cause by ECC and find the causes of System Crashes. We also plan to investigate more the impacts of other CUDA optimization flags on the reliability of a code.

VIII. Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 886202 and from The Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001. Neutron beam time was provided by ChipIR (DOI: 10.5286/ISIS.E.RB2000161).

References

[1] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU architecture via microbenchmarking," *CoRR*, vol. abs/1804.06826, 2018. [Online]. Available: http://arxiv.org/abs/1804.06826

[2] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[3] G. León, J. M. Badía, J. A. Belloch, A. Lindoso, and L. Entrena, "Evaluating the soft error sensitivity of a GPU-based SoC for matrix multiplication," *Microelectronics Reliability*, vol. 114, pp. 856 – 861, 2020, 31st European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ESREF 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271420304558

[4] J. Cao, K. Arya, R. Garg, S. Matott, D. K. Panda, H. Subramoni, J. Vienne, and G. Cooperman, "System-level scalable checkpoint-restart for petascale computing," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, 2016, pp. 932–941.

[5] X. Chen, J. Feng, M. Hiller, and V. Lauer, "Application of software watchdog as a dependability software service for automotive safety relevant systems," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 596–600.

[6] A. Mahmood and E. McCluskey, "Concurrent error detection using watchdog processors-a survey," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 160–174, 1988.

[7] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[9] N. Tijtgat, W. Van Ranst, B. Volckaert, T. Goedemé, and F. De Turck, "Embedded real-time object detection for a UAV warning system," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 2110–2118.

[10] C. Lunardi, F. Previlon, D. Kaeli, and P. Rech, "On the efficacy of ECC and the benefits of FinFET transistor layout for GPU reliability," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1843–1850, 2018.

[11] K. Ito, Y. Zhang, H. Itsuji, T. Uezono, T. Toba, and M. Hashimoto, "Analyzing due errors on GPUs with neutron irradiation test and fault injection to control flow," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1668–1674, 2021.

[12] N. DeBardeleben, S. Blanchard, L. Monroe, P. Romero, D. Grunau, C. Idler, and C. Wright, "GPU behavior on a large HPC cluster," in *Euro-Par 2013: Parallel Processing Workshops*, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 680–689.

[13] L. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. S. Reorda, "GPGPUs: How to combine high computational power with high reliability," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1733–1741.

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

[15] D. A. G. Gonçalves de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2016.

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

[17] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19 490–19 503, 2020.

[18] S. K. S. Hari, P. Rech, T. Tsai, M. Stephenson, A. Zulfiqar, M. B. Sullivan, P. P. Shirvani, P. Racunas, J. S. Emer, and S. W. Keckler, "Estimating silent data corruption rates using a two-

level model," *CoRR*, vol. abs/2005.01445, 2020. [Online]. Available: https://arxiv.org/abs/2005.01445

[19] F. F. d. Santos, S. K. S. Hari, P. M. Basso, L. Carro, and P. Rech, "Demystifying GPU reliability: Comparing and combining beam experiments, fault simulation, and profiling," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 289–298.

[20] F. F. d. Santos and P. Rech, "Analyzing the criticality of transient faults-induced SDCs on GPU applications," in *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. ScalA '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 6–13. [Online]. Available: https://doi.org/10.1145/3148226.3148228

[21] NVIDIA. (2021) CUDA runtime API. [Online]. Available: https://docs.nvidia.com/cuda/cuda-runtime-api/index.html

[22] Y. Ukidave, F. N. Paravecino, L. Yu, C. Kalra, A. Momeni, Z. Chen, N. Materise, B. Daley, P. Mistry, and D. Kaeli, "NUPAR: A benchmark suite for modern GPU architectures," ser. ICPE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 253–264. [Online]. Available: https://doi.org/10.1145/2668930.2688046

[23] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 42–52.

[24] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[25] NVIDIA. (2012) Kepler GK110/210 whitepaper. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[26] NVIDIA. (2017) NVIDIA Tesla V100 GPU architecture. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[27] N.-M. Ho and W.-F. Wong, "Exploiting half precision arithmetic in NVIDIA GPUs," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 2017, pp. 334–341.

[28] NVIDIA. Floating point and IEEE 754 compliance for NVIDIA GPUs. [Online]. Available: https://docs.nvidia.com/cuda/floating-point/index.html

[29] H. Quinn, "Challenges in testing complex systems," *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, 2014.

[30] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," vol. 1021. 22nd Meeting of the International Collaboration on Advanced Neutron Sources (ICANS XXII), may 2017, pp. 159–164. [Online]. Available: https://doi.org/10.1088/1742-6596/1021/1/012037

[31] C. Slayman, *JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors*. Boston, MA: Springer US, 2011, pp. 55–76. [Online]. Available: https://doi.org/10.1007/978-1-4419-6993-4_3

[32] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability evaluation of mixed-precision architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 238–249.

[33] P. A. Abdulla, J. Deneux, G. Stålmarck, H. Ågren, and O. Åkerlund, "Designing safe, reliable systems using scade," in *Leveraging Applications of Formal Methods*, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 115–129.

[34] P. Rech, L. Pilla, P. Navaux, and L. Carro, "Impact of GPUs parallelism management on safety-critical and HPC applications reliability," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 455–466.

[35] A. Serrano-Cases, J. Isaza-González, S. Cuenca-Asensi, and A. Martínez-Álvarez, "On the influence of compiler optimizations in the fault tolerance of embedded systems," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016, pp. 207–208.

[36] F. M. Lins, L. A. Tambara, F. L. Kastensmidt, and P. Rech, "Register file criticality and compiler optimization effects on embedded microprocessor reliability," *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2179–2187, 2017.

[37] M. Demertzi, M. Annavaram, and M. Hall, "Analyzing the effects of compiler optimizations on application reliability," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 184–193.

[38] NVIDIA. (2021) NVIDIA CUDA toolkit release notes. [Online]. Available: https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html\#title-new-cuda-libraries