Demystifying GPU Reliability: Comparing and Combining Beam Experiments, Fault Simulation, and Profiling

Demystifying GPU Reliability: Comparing and Combining Beam Experiments, Fault Simulation, and Profiling

Fernando Fernandes dos Santos*, Siva Kumar Sastry Hari⋆, Pedro Martins Basso*, Luigi Carro*, and Paolo Rech†
*UFRGS, Brazil ⋆NVIDIA Corporation, United States †Politecnico di Torino, Italy

***Abstract*—Graphics Processing Units (GPUs) have moved from being dedicated devices for multimedia and gaming applications to general-purpose accelerators employed in High-Performance Computing (HPC) and safety-critical applications such as autonomous vehicles. This market shift led to a burst in the GPU's computing capabilities and efficiency, significant improvements in the programming frameworks and performance evaluation tools, and a concern about their hardware reliability.**

In this paper, we compare and combine high-energy neutron beam experiments that account for more than 13 million years of natural terrestrial exposure, extensive architectural-level fault simulations that required more than 350 GPU hours (using SASSIFI and NVBitFI), and detailed application-level profiling. Our main goal is to answer one of the fundamental open questions in GPU reliability evaluation: whether fault simulation provides representative results that can be used to predict the failure rates of workloads running on GPUs. We show that, in most cases, fault simulation-based prediction for silent data corruptions is sufficiently close (differences lower than 5×) to the experimentally measured rates. We also analyze the reliability of some of the main GPU functional units (including mixed-precision and tensor cores). We find that the way GPU resources are instantiated plays a critical role in the overall system reliability and that faults outside the functional units generate most detectable errors.

I. Introduction

GPUs have evolved from supporting hardware for user applications and graphics rendering to general-purpose accelerators extensively employed in HPC and safety-critical applications such as autonomous driving for the automotive and aerospace markets. The GPU architecture fits the computational characteristic of most HPC codes and is efficient in executing matrix multiplication, which is the computing core of Convolutional Neural Networks (CNNs) used to detect objects in autonomous vehicles. 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.

Focusing on HPC and safety-critical applications, GPU vendors have worked to improve the reliability of the memory cell [1]. They are working on platforms compliant with strict automotive reliability standards, such as the ISO26262 [2]. The research community has been carefully studying GPU reliability with both fault simulation [3], [4], and beam experiments [5]. Beam experiments provide a very realistic analysis but lack visibility as it is hard to associate observed behaviors with the source of the fault and identify the most vulnerable GPU resources. In contrast, fault simulation provides full visibility of the fault propagation, thus identifying the resources or code portions that, once corrupted, are more likely to affect the computation. However, faults usually can only be injected on a subset of resources, and the adopted fault-model risks to be unrealistic if not tuned with experiments.

The comparison between beam experiments and fault simulation is an essential missing piece in the GPU reliability evaluation puzzle that this paper intends to find. It is still mostly unclear whether a reliability evaluation based only on fault simulation is realistic. Evaluating the effectiveness of many error mitigation techniques requires fault injection. However, data based on fault simulations alone does not imply that the method will be useful in the field. To evaluate at which level and under which assumptions fault simulation can provide a realistic reliability evaluation for GPUs, we compare the codes' Failure In Time (FIT) rates measured through beam experiments with the FIT rates predicted using fault injections.

To characterize Kepler [6] and Volta [7] architectures' reliability, we measure, through beam experiments, the FIT rates of the main functional units (including mixed-precision and tensor core) and register file. We also measure the FIT rates of 15 representative codes for HPC and safety-critical applications, including two CNNs. Some codes have been executed using different data types (e.g., double-, single-, or half-precision floating point and integer) to understand the impact of mixed-precision on code's reliability. For most codes, we run experiments both with Error Correcting Code (ECC) enabled and disabled to evaluate the efficacy of GPUs' built-in reliability solutions and distinguish between the contribution of logic and memory faults to the codes error rate. Then, thanks to the fault injection analysis, we identify the most likely sources for the observed Silent Data Corruptions (SDCs) and Detected Unrecoverable Errors (DUEs). We find out that integer codes have higher Architectural Vulnerability Factor (AVF) than floating point codes while different floating point precisions do not seem to affect the code AVF.

To compare fault simulation with beam experiments, we predict the codes FIT rate by multiplying the experimentally measured FIT rates of the functional units and memory resources (the latter only when ECC is disabled) with the AVF of the respective hardware resource. To account for

GPU's parallelism management, in the FIT rate prediction we consider both the GPU occupancy and code IPC (i.e., how many threads are active and how many functional units are being used) obtained through kernel profiling. We show that our method can provide a reasonable prediction of the SDC FIT rate for most codes (differences lower than 5×). Thanks to our analysis, we can understand the impact of *hidden* GPU resources (parallelism management, scheduler, dispatcher, queues, etc.) and identify the code/architecture characteristics/metrics that have a significant impact on the GPUs error rate. Additionally, we show that the DUE FIT rate is dominated by faults in resources that are not accessible through architecture-level fault simulations. As a result, high-level fault simulation-based methods underestimate GPU's DUE FIT rate by orders of magnitude.

To make our results reproducible and to provide a reference for third party analysis, all reported data, including kernels profiling, fault injection, and beam experiments result, is made available in a GitHub repository[1]. The remainder of the paper is structured as follows. Section II presents the past works related to our research. The experiments and the tools that we used are described in Section III. In Section IV, we propose a simplification of the FIT rate prediction based on the error probability. Section V details how we measure the FIT rate of micro-instructions. The fault simulation and beam experiments results are discussed in Section VI. The prediction results using our model and the fault simulation-beam comparison are presented in Section VII. Section VIII concludes the paper.

II. Background and Related Work

Ionizing particles strike may generate a bit-flip in memory or produce current spikes in logic circuits that, if latched, lead to a fault [8]. A transient fault may propagate through the stack of system layers leading to a Silent Data Corruption (SDC, i.e., undetected output corruption) or Detected Unrecoverable Errors (DUEs, i.e., program or system crash) or no effect to the program output (i.e., the fault is masked).

The error rate of computing devices, including GPUs, running specific applications has already been measured through-beam experiments in previous work [9], [10]. The accelerated particle beam induces transient faults in the device hardware and, counting the manifestations of errors at the output, it is possible to measure the realistic error rate of the device running a code. Beam experiments, while providing the realistic error rate, jointly consider all the factors that influence the device error rate, impeding the distinction of each factor's contribution and making it very challenging to identify the most vulnerable parts of the system.

Fault simulation is used to understand the fault's probability to propagate, generating an error. Faults can be injected by the user at different levels of abstractions: from Register-Transfer Level (RTL) [11], [12] to microarchitecture [13], [14] and software [15], [16]. Fault simulation assumes that the fault had occurred and tracks its propagation without giving any information about the probability for the fault to originate. Fault simulation has two main limitations: (1) the fault model and fault injection probabilities are defined/modeled by the user and/or the simulator, thus the obtained results risk to be unrealistic. (2) faults can be injected only in that subset of available and accessible resources.

[1]https://github.com/UFRGS-CAROL/ipdps2021.git

TABLE I: Codes characteristics on Kepler and Volta GPUs.

Kepler						Volta					
		SHARED	RF	IPC	Occupancy			SHARED	RF	IPC	Occupancy
CCL	INT	123B	34	0.14	0.11	Lava	FP16	8KB	255	0.26	0.1
BFS	INT	0B	21	1.22	0.81		FP32	8KB	255	0.12	0.1
Lava	FP32	7KB	37	4.12	0.57		FP64	16KB	254	0.07	0.1
Hotspot	FP32	3KB	23	3.89	0.94	Hotspot	FP16	16KB	26	0.48	0.94
Gaussian	FP32	0B	14	0.51	0.34		FP32	32KB	27	0.32	0.95
LUD	FP32	8.6KB	27	0.58	0.37		FP64	64KB	30	0.18	0.96
NW	INT	8.2KB	32	0.2	0.08	MxM	FP16	0B	27	2.84	1
MXM	FP32	8KB	25	1.5	1		FP32	0B	25	2.62	1
GEMM	FP32	31KB	248	4.94	0.19		FP64	0B	29	2.3	1
Mergesort	INT	2.5KB	16	2.11	0.97	GEMM	FP16	64KB	127	2.34	0.25
Quicksort	INT	328B	27	1.97	0.96		FP32	64KB	134	2.36	0.13
Yolov2	FP32	8KB	97	2.84	0.59		FP64	64KB	234	1.22	0.13
Yolov3	FP32	9.1KB	100	3.11	0.65	Yolov3	FP16	21.5KB	55	0.06	0.7
–							FP32	34.2KB	39	0.09	0.7

A recent study [17] tried to predict application SDC rate using micro-architectural fault injection on ARM CPUs. However, the paper does not perform analysis on the application level. On GPUs, Hari *et. al* [18] tried to predict the FIT rate at implementation and application-level. The low-level implementation considered beam experiments, and application-level analysis employed fault injection. The results show that the SDC prediction is plausible. However, the paper did not provide insights into the impact of hidden GPU resources (parallelism management) on the SDC rate or identifying the code/architecture characteristics/metrics that significantly impact GPUs. We show that analyzing multiple GPU architectures and compiler versions are crucial for application failure rate analysis. We also investigated models with ECC and without ECC to reveal new insights.

To the best of our knowledge, this is the first paper that (1) combines beam experiments and fault simulation to deeply understand the reliability of GPUs and (2) provides essential information to ensure that fault simulation derives a realistic GPU reliability evaluation.

III. Metrics and Evaluation Methodology

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

A. Devices

Devices: We consider Kepler (Tesla K40c) and Volta (Titan V and Tesla V100) NVIDIA GPU architectures. The tested NVIDIA K40c is based on the $Kepler$ architecture and has 2,880 CUDA cores divided into 15 Streaming Multiprocessors (SMs). Single Error Correction Double Error Detection (SECDED) ECC protects the register file, shared memory, and caches. The Titan V and Tesla V100 GPUs are based on Volta architecture. Volta GPUs support three IEEE754 float point precisions: double, float, and half. Each of the 80 Volta SMs has 64 FP32 cores, 64 INT32 cores, 32 FP64 cores [19]. Volta also includes eight *tensor cores*, i.e., specific hardware that performs the Matrix Multiplication and Accumulate (MMA) operation. ECC can be disabled or enabled by the user on both K40c and V100.

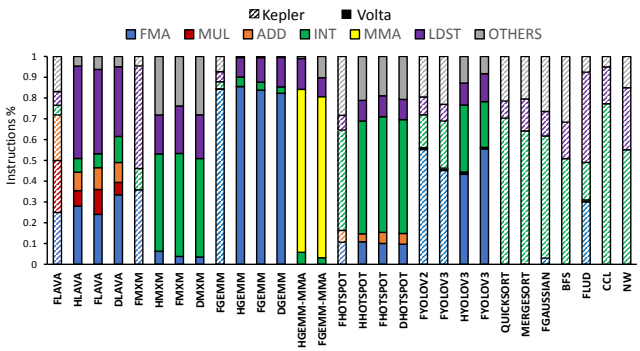


Fig. 1: Instruction type per code for Kepler and Volta GPUs.

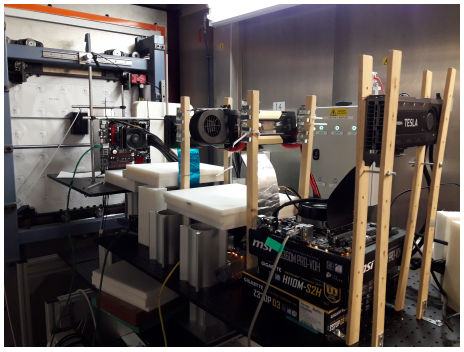


Fig. 2: Beam experiment setup at ChipIR.

B. Tested Codes

To increase our results' quality, we chose fifteen representative codes, listed in Table I, that come from broad domains, from HPC to deep learning. We pay particular attention to Matrix Multiplication because of its importance in CNNs and HPC. We test both the naive version (*MxM*) and optimized version that digests 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 selected input size, precision, and device configuration.

We identify the codes that are more vulnerable and discuss if the code reliability characteristic is due to the sensitivity of resources, the number of resources used for computation, and/or the fault propagation probability (AVF). Table I also lists the metrics that we use to consider the GPU parallel management in the FIT rate prediction: the amount of shared memory, the average number of registers used for computation, the IPC, and occupancy (details in Section IV-B).

For each code listed in Table I, we profile the kernel using NVPROF and NSIGHT-COMPUTE. Profiling the codes gives us insights in to the instructions that significantly contribute to the benchmark execution. Figure 1 shows, in percentage, the instructions that compose each code. Each floating-point code has the precision made explicit in the first letter of its name – D for double-precision (64 bits), F for single-precision (32 bits), and H for half-precision (16 bits). For example, *HHOTSPOT* is Hotspot executed using half-precision and *DGEMM* is GEMM using double-precision floating-point data type. INT32 based codes do not have their name modified.

Based on the profile, we divide the instructions into: (1) common arithmetic instructions (i.e., FMA, MUL, ADD, INT, MMA), (2) data movement instructions (LDST), (3) "OTHERS", which are the ones that have a minor contribution to the final benchmark (i.e., transcendental functions, branch, inter-thread communication, thread barrier, NOP, and atomic directives). As testing all the instructions would be unfeasible due to beam time restrictions (NVIDIA ISA has more than 20 different instructions classes), we measured through-beam experiments the FIT rate of only the former set of instructions, as they are the most likely to be corrupted and the most common in a wide range of codes. As we demonstrate in Section VII, even considering a (large but not exhaustive) subset of instructions allows a reasonable FIT rate estimation for the codes executed in the GPU.

C. Beam Experiment Setup

Beam experiments allows to measure the FIT rate of a computing device running a code by dividing the number of observed errors by the received particles fluence ($neutrons/cm^2$). Our experiments are performed at the ChipIR facility of the Rutherford Appleton Laboratory, UK, and at the LANSCE facility of the Los Alamos National Laboratory, USA. Figure 2 shows the setup mounted at ChipIR. Both facilities deliver a beam of neutrons with a spectrum of energies that resembles the atmospheric neutrons [20]. 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 $neutrons/(cm^2 \cdot h)$ [21]). 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. We test at least ten codes and seven microbenchmarks per device. Each code was tested for at least 72 hours, not including the setup, result check, initialization, and recovery from the crash time. When scaled to the natural exposure, the 1,224 accelerated beam hours account for more than 13 million years.

It is worth noting that the FIT rates do not depend on a code's execution time, but only on the number of resources used during computation, their sensitivity (probability for the fault to occur), and criticality (probability with which a fault in the resource propagates to affect the calculation). If the same amount of memory is exposed for a given time t or $2 \times t$, its FIT rate will not change. In fact, in $2 \times t$, we expect twice the errors and twice the neutrons (i.e., twice the fluence). Similarly, under the assumption that at most one fault can affect the GPU during code execution (because the natural flux is very low), executing x or $2 \times x$ *sequential* ADD instructions does not change the probability of having one ADD corrupted by neutrons. However, what can change is the probability with which an error in one of the ADDs propagates to the output of

the sequence of the operations (i.e., the AVF). If the additional x ADDs are executed in *parallel* with the original sequence, the FIT rate is expected to double (same execution time, same fluence, but doubled the error rate). We use these observations in Section IV-B to account for GPU parallelism management in the FIT rate prediction based on fault injection.

D. Fault Simulation Frameworks

Software fault injectors can help us understand how a fault propagates by providing the AVF (number of observed errors divided by the number of injected faults) [22], which is the probability of a fault leading to a failure and identifies which instruction or resource, once corrupted, is more likely to affect the GPU computation. We use SASSIFI and NVBitFI frameworks that can inject transient errors in the GPU's architecture state, which is also visible to the GPU's native instruction set, such as general-purpose registers, predicate registers, condition instructions, and arithmetic instructions [3], [23]. SASSIFI supports NVIDIA's Kepler and Maxwell architectures, whereas NVBitFI supports Kepler, Maxwell, Pascal, Volta, and Turing architectures. SASSIFI and NVBitFI are the most suitable fault injectors for this work since it is possible to instruct the kernels at the SASS level (SASS is the name of NVIDIA's native instruction set). Other fault injectors such as GPUQin, CAROL-FI, Kayotee, GPGPU-SIM [10], [24]–[26], either do not allow to inject faults at the SASS level or they offer support for Kepler and Volta architectures. We inject at least 4,000 single bit flip faults per code using NVBitIF and 10,000 single bit flip faults per application using SASSIFI (1,000 for each instruction kind), for a total of more than 50,000 faults per ISA that took 350 GPU hours, ensuring 95% confidence intervals to be lower than 5% [3].

SASSIFI can inject faults in the output of the floating-point, integer, and load instructions. SASSIFI also has the capability to injects faults in the predicate registers, general purpose registers, and instruction address. In contrast, the current NVBitFI version can inject faults only at double, float, load instructions, and instructions that write in the general-purpose registers. The differences between the fault injectors can lead to differences in the application AVF along with other factors (e.g., compiler version), as discussed in Section VI.

Unfortunately, neither SASSIFI nor NVBitFI (nor any other fault injector) supports fault injection on NVIDIA proprietary libraries on Kepler at the time of this publication. However, NVBitFI can inject faults in proprietary libraries on Volta. So we use the AVF measured with NVBitFI on Volta for applications using pre-compiled libraries as predictions for the AVF for the application running on Kepler. As shown in Section VI, on the codes that do not use proprietary libraries executed on Kepler, the AVF measured with SASSIFI is 18% smaller than NVBitFI, on average. As shown in Section VII, this still allows the accurate prediction of the SDC FIT rates.

IV. FIT RATE PREDICTION THROUGH FAULT SIMULATION

Based on the definition and observations of Section III, we predict the FIT rate of codes running on a GPU using the AVFs and FIT rates of the primary resources used for computation.

A. Contributions to codes FIT rate

We can assume that the cause of the observed code output error is a single neutron strike (that can produce a single or multiple bit-flips, as we evaluate in Section V) in a single resource. This is justified by the observation that, with the current technology and the low intensity of natural flux of particles, the probability for more than one neutron to generate faults during a code execution is negligible. The neutron-induced error rate of a bit of a 28nm SRAM cache memory, for instance, is in the order of of 10^{-15} - $10^{-16} errors/h$ [27]. Even on a hypothetical GPU with 1Gbit of these SRAM internal memories (caches, shared memory, etc.), we expect at most $1 \times 10^{-6} errors/h$, making it unlikely for more than one fault to occur during one application execution. Additionally, all neutron-induced events are transient and not cumulative: the resource's sensitivity does not depend on the number of faults it has undergone [9]. A code can then be affected only by a corruption during its execution, independently of previous computations. If the code uses previously computed corrupted data as input, the error must be attributed to the previous computation and not counted while deriving the FIT rate.

Neutron-induced errors are uncorrelated and stochastic events. Thus, a device's probability of being corrupted by a neutron is the sum of the probabilities of having a neutron-induced corruption in one of its resources. Consequently, the FIT rate of a code is the sum of the probabilities of having a neutron-induced fault in each of the resources used for its computation, multiplied by the probability for the fault in that resource to propagate to the output (the resource AVF).

Knowing the AVF and FIT rate of every resource used for computation, in principle, would allow a perfect estimation of the FIT rate of a code. Unfortunately, even if each GPU resource were accessible by the user, it would be infeasible to measure each resource's FIT and AVF given that the architectures today integrate many resources on a single chip. So, we decided to limit our study to the contribution of the main functional units and memories of GPUs. We measure with beam experiments the FIT rates of most common functional units (arithmetical instructions) and of register file (details in Section V). We calculate, through fault injection, the probability for a fault in each instruction or used memory to affect the code output. Then, we estimate a code's FIT rate (†FIT) by adding the expected contribution of each instruction $P(E_{INST_i})$ and memory level $P(E_{MEM_i})$, as shown in Equation 1.

$$\dagger FIT = \sum_{i=1}^{n} P(E_{INST_i}) + \sum_{i=1}^{m} P(E_{MEM_i}) \qquad (1)$$

Consequently, the contributions to the FIT rate of the code, $P(E_{INST_i})$ and $P(E_{MEM_i})$, rely upon the number of resources used for computation, the probability of a fault to be generated (the resource cross-section or FIT), and the proba-

bility for the fault in that resource to affect the computation (AVF) as formalized in the following Equations 2 and 3.

$$P(E_{INST_i}) = f(INST_i) \cdot AVF_{INST_i} \cdot FIT_{INST_i} \quad (2)$$

$$P(E_{MEM_i}) = f(MEM_i) \cdot AVF_{MEM_i} \cdot FIT_{MEM_i} \quad (3)$$

Where $f(INST_i)$ and $f(MEM_i)$ are the probabilities of having one instance of an instruction $INST_i$ or a bit of memory level MEM_i used in the computation of the benchmark; AVF_{INST_i} and AVF_{MEM_i}, FIT_{INST_i}, and FIT_{MEM_i} are the average AVF and FIT of an instruction $INST_i$ and a bit of memory MEM_i, respectively.

$f(MEM_i)$ is the number of bits of memory level i instantiated for computation. When ECC is enabled in memory MEM_i, we can assume $AVF_{MEM_i} \approx 0$ and consequently $P(E_{MEM_i}) \approx 0$, simplifying Equation 1 to only its first summation. On a GPU, the FIT rate can vary significantly based on a code's degree of parallelism and how the GPU scheduler allocates the available functional units. Next subsection discusses a way to account for these using kernel profiling.

B. Profiling kernel dynamic instructions

On GPUs, the probability for a neutron to corrupt an operation in a specific thread depends on how many threads are active and how many parallel operations are being executed. The higher the number of instructions a thread is allowed to schedule, or the higher the number of active threads in an SM, the higher the number of functional units that become susceptible to a particle strike. To understand how many computing resources are exposed, and thus considered in Equation 2, we need to profile the codes on a target GPU.

In an NVIDIA GPU, the basic unit of execution is the warp, which is a collection of threads (e.g., 32 threads) that are executed simultaneously by an SM. The active warps in an SM can be in three different states: stalled, eligible (ready but waiting), and selected (executing). Each SM has four warp schedulers that pick the instructions from warps based on their state. Then, each scheduler determines the eligible warp to execute up to 2 instructions, limiting the instruction issue parallelism to 4 per SM [28]. The number of cycles a warp requires to be ready to execute the next instruction is the warp's latency. During these cycles, the active warp instructions are exposed and could be corrupted. Thus, it is necessary to consider the number of active warps (i.e., the *Achieved Occupancy* in NVIDIA profiling tools) to model the number of resources that could be corrupted.

The GPU's occupancy alone is not sufficient to model the number of used resources. In fact, the number of active threads could be limited by resource utilization (commonly, the amount of registers and shared memory). If the instructions in these active threads do not have dependencies, they could be scheduled in parallel, saturating the available functional units. This is the case of GEMM, for instance, that has a very low occupancy (see Table I) but imposes massive stress on the functional units. Other codes (e.g., sort) have high occupancy but suffer from long latencies. Then, to account for (in)efficient utilization of resources, we also consider the *Instructions Per Cycle (IPC)* of the code in our prediction model. A high IPC indicates that many instructions are executing and retiring per cycle, with implies that a high number of functional units are exercised. To consider the contribution of parallelism of GPUs, then, we multiply $P(E_{INST_i})$ in Equation 2 by a factor (φ_{INST}) defined as follows:

$$\varphi_{INST} = AchievedOccupancy_b * IPC_b \quad (4)$$

High occupancy and IPC indicate that many resources are employed for computation, which increases the probability of having a corruption.

V. Synthetic Micro benchmarks

To measure the FIT rate of Kepler and Volta architectures' functional units and main atomic instructions, we have designed seven classes of synthetic micro-benchmarks. Results collected based on these micro-benchmarks are used to predict the FIT rates of applications on the target device and are valuable to compare the reliability of the different functional units that compose the GPU.

A. Synthetic micro-benchmarks design

RF micro-benchmark measures the FIT rate of the Register File (RF) storage, which we consider is representative for other on-chip memory structures. In this micro-benchmark, each thread in an SM writes a known pattern in all accessible registers (255 registers per thread) and reads back the values after a pre-defined time to count bit-flips. We instantiate the lowest possible number of threads while fully utilizing the RF to reduce the probability of having errors in resources besides the targeted registers. The time between a write and read should be long enough to ensure that the setup/read-back time is negligible and short enough to prevent more than one neutron from generating faults. This latter constraint is necessary to detect eventual Multiple Bit Upsets (MBUs), more than one bit corrupted in a single word. We heuristically set the exposure time to $1s$. We anticipate the MBU rate is about 2% for the RF [1].

LDST performs a sequence of memory movements in global memory (Load followed by Store) with ECC enabled. The LDST kernel reads a memory region from global memory that contains a unique pattern and stores it in another location of the global memory. Each kernel consists of 4M threads, each performing 2^{10} memory movements. In total, this micro-benchmark allocates 2GB of memory. The host CPU setup compares if the expected pattern is correct on the output memory and counts the number of corruptions. CPU verification time is not considered for FIT calculation.

Each thread in **FMA** (Fused Multiply and Add), **ADD** (Addition), **MUL** (Multiplication), and **MAD** (Integer Multiply and Accumulate) micro-benchmarks executes 10^8 operations, while **MMA** performs 10^7 16x16 matrix multiplications (with FP16 on HMMA or FP32 casted to FP16 for FMMA). We choose a lower number of operations for MMA to keep the exposure time, and so the statistic, similar to the other

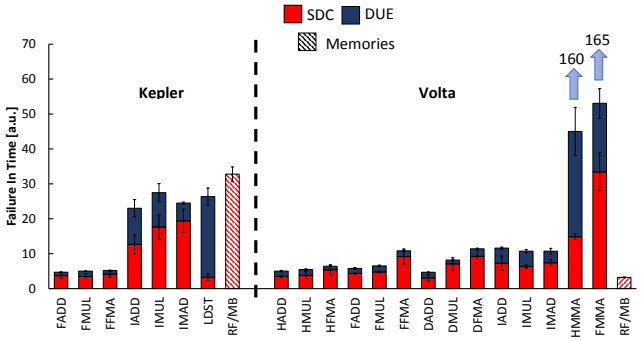


Fig. 3: Measured micro-benchmarks FIT rates, normalized to each device's lowest measured value – FADD's DUE on Kepler, HFMA's DUE on Volta. RF FIT rate is shown per megabyte. ECC was ON for all micro-benchmarks but RF.

micro-benchmarks. The inputs are pre-defined and have been randomly generated off-line. These inputs avoid overflow. We tested the integer and float versions of the micro-benchmarks on the Kepler and the integer (INT32), double (FP64), float (FP32), and half (FP16) versions of the micro-benchmarks on the Volta GPUs. The number of instantiated threads is tuned to occupy all the GPU's available functional units (3,840 threads for Kepler, 20,480 threads for Volta).

Errors are identified by comparing the result with the pre-computed fault-free output after completion of each thread's operations. We do not check for errors at each operation to avoid excessive overhead and to make the probability for a fault in the comparison negligible. However, if two different neutrons corrupt two different instructions during the 10^8 operations, we would count it as one error, thus underestimating the functional unit FIT rate. Nevertheless, in each hour, we observe $<$10 events. Considering that an average of 2 seconds is necessary to execute 10^8 operations, the probability of having two corruptions is a single execution is lower than 1%. Moreover, we found more than one thread being corrupted (more than one sequence of operations with incorrect data) in a few cases. If we checked at each instruction, we would catch all the errors but, as we check only after a sequence of operations, some errors might be logically masked, reducing the FIT rate. To consider the masking effect of subsequent operations, we run fault injection on the micro-benchmarks and found that the AVF is always higher than 70% (being 100% for the integer versions). To have the most accurate prediction, in Section VII, we multiply the micro-benchmark's FIT by the AVF measured from simulations.

B. Micro-benchmarks' profile and error rates

Figure 3 shows the SDC and DUE normalized FIT rate for all micro-benchmarks on Kepler and Volta GPUs. FIT rates are normalized and shown in arbitrary units (a.u.) to allow comparison without revealing business-sensitive data. We test FMA, ADD, MUL, and MAD both with ECC ON and OFF and found that the error rates are comparable (differences lower than 20%). These mico-benchmarks use very little memory (few registers). Figure 3 shows that for all the tested float instructions (FADD, FMUL, FFMA) on Kepler, both SDC and DUE rates are similar. The measured FIT rates for INT32 micro-benchmarks are 4$\times$ higher on average compared to the respective FP32 ones. This is probably because, on Kepler, the integer operations are executed in the same hardware as the FP32 operations with evident lower efficiency that can increase the vulnerability. The error rates for INT32 micro-benchmarks vary based on the type of the instruction, following the operation's complexity. For example, IMUL's FIT rate is approximately 30% higher than that of IADD. These results suggest that an integer addition is less complex and demands less hardware than integer multiplication. Since IMAD performs integer multiplication and addition, its FIT rate is higher than both IMUL (10% higher) and IADD.

LDST on Kepler was run only with ECC enabled. LDST is the only micro-benchmark for which the DUE rate is higher (7.1$\times$) than the SDC rate, which is expected because the critical operand in the LDST micro-benchmark is a memory address. An incorrect address can either be valid or invalid. The likelihood of a corrupted address being valid is low if the total memory allocated is small, which is our case. Hence, the chances of invalid addresses is higher and such accesses trigger a device or CUDA API exception.

For the Volta GPU, we focus on mixed-precision cores reliability as this is a novelty in the newer architectures. These results are also shown in Figure 3. The differences in the FIT rates between int, double, float, and half precision operations in Figure 3 rely on the different Volta mixed-precision cores' complexities. Since a multiplication requires more resources than a sum, its FIT rate is expected to be higher, and FMA (fused multiply and addition) is expected to have FIT rate higher than ADD and MUL, which is in accordance with our results. Additionally, the higher the operation precision, the higher the FIT rate (again, higher precision implies higher resource utilization). Contrary to the Kepler GPU, integer operations on the Volta GPU are executed on dedicated cores [6].

Figure 3 also shows the FIT rate of micro-benchmark focussing on Matrix Multiplication and Addition (MMA), also known as *Tensor Core*, for Volta architecture (MMA is not available for Kepler). Hardware MMA operations can be used via CUDA on $16x16$ input matrices. The introduction of a specific (sophisticated) hardware to execute matrix multiplications in mixed-precision was driven by the importance of this operation in DNN training and interference.

As the complexity of MMA is higher than those of other functional units and so is the utilization, its FIT rate is expected to be higher than all other micro-benchmarks on Volta. As shown in Figure 3, Half and Float MMA have FIT rates that are 12$\times$ higher than that of DFMA, which has the highest FIT rate among the others micro-benchmarks. It is worth noting that, as one and only one neutron can generate an error in the 10^7 or 10^8 operations each thread performs (details in Section V-A), the number of operations each thread executes does not impact the FIT rate. A higher

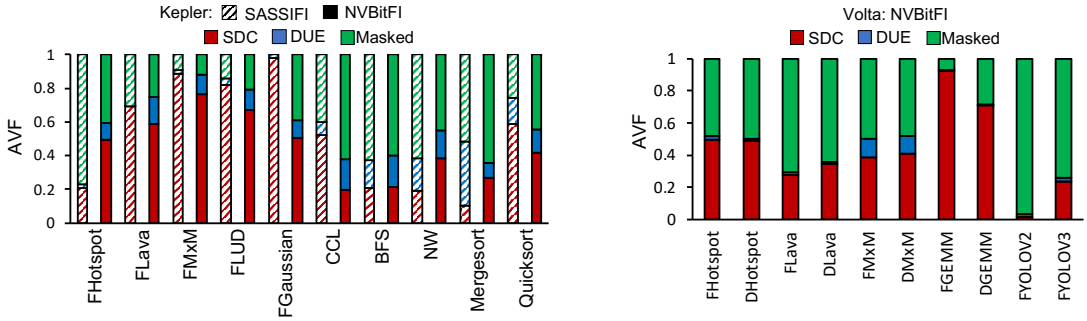


Fig. 4: AVF for the Kepler GPUs on the left (using SASSIFI and NVBitFI) and Volta GPUs on the right (using NVBitFI).

number of sequential operations increases the number of errors because it increases the execution (i.e., exposure) time and, thus, the neutron fluence, not because the hardware is more vulnerable. If more operations are executed in parallel, the FIT rate is expected to increase as we perform more operations but roughly the same execution time.

One interesting observation is that, while being more sensitive, the MMA core performs, in one operation, the equivalent of $4x4$ FMA or $4x4$ ADD, MUL, and the loop control variables needed to implement MxM in software (these latter instructions can economize with a loop-unrolling). From our data, we know that the FIT of each HMMA and FMMA micro-benchmarks is $9\times$ and $12\times$ higher than an FMA micro-benchmarks (we recall that FMMA uses the HMMA core after a cast). As 64 MMA instructions are required to multiply two $16x16$ matrices, and for each warp-wide MMA instruction we could instead execute a warp of 32 FMAs, we can deduce that the use of MMA is $2\times$ (64/32, where 32 is the number of threads in a warp) more reliable than the combination of operations needed to execute a software MxM. The use of MMA eliminates repeated fetches of the multiply-and-add operations and reduces activity in instruction memory and pipelines. As the size of the matrix multiplication supported by the MMA increases the reliability (and performance) benefit will also increase. Figure 3 shows the error rate of a megabyte stored in the Register File (RF). The result indicates RF (and memory in general) as a critical GPU resource, when ECC is OFF. This result confirms previously published research [5]. While we do not show the Kepler vs. Volta FIT rates (we use different normalization for the two boards in Figure 3), we find that the fabrication process plays a significant role. Kepler RF (28nm planar CMOS) has an approximately an order of magnitude higher error rate than Volta RF (16nm FinFET), in accordance with previous work [29].

VI. AVF and Neutron Beam Experiments Results

AVF: Figure 4 shows the AVF for all the codes we considered. On the Kepler GPU, we inject faults with both SASSIFI and NVBitFI, while on the Volta GPU faults can be injected only with NVBitFI. We recall that a higher AVF (probability for a fault to affect computation) does not necessarily imply a higher error rate (as shown in Equations 2 and 3).

We discover, from Kepler data in Figure 4, that SASSIFI and NVBitFI AVF results are different. For most of the benchmarks, the AVF is higher while using NVBitFI than SASSIFI. On average, the AVF was 18% higher for NVBitFI. The differences between SASSIFI and NVBitFI on the Kepler GPU is particularly high for CCL ($2.6\times$), Gaussian ($1.9\times$), Quicksort ($1.4\times$), and Hotspot ($0.4\times$). SASSIFI is relatively older than NVBitFI and supports CUDA 7.0, while NVBitFI supports CUDA 10.1+. As NVIDIA compiler has been improved throughout the versions, the generated SASS code changes with the versions of the compiler when compiled with default options even for the older architectures (e.g., Kepler). NVIDIA compiler has two main parts. The front-end compiler takes the code written in a high-level language (e.g., CUDA) and generates intermediate code in a virtual ISA called parallel thread execution (PTX). The back-end compiler takes the target-independent PTX code and applies many code optimizations (e.g., unrolling, loop-invariant code motion, dead code elimination) before generating SASS code that can run on the target GPU. Significant updates are often made to the front-end and back-end compiler infrastructure to support new features and future target SASS versions. As a result, while both fault injectors support similar fault models and similarly instrument the SASS code, the generated code can itself be different due to the compiler version and have a significant impact on the code's AVF.

Gaussian, LUD, MxM, and Lava have the highest AVF on the Kepler GPU, for both SASSIFI and NVBitFI. All these benchmarks are floating point-based applications (see Figure 1). The smaller AVFs comes from integer applications: Quicksort, Mergesort, CCL, and NW. The resource utilization for the benchmarks that use only integer data and operations can be more efficient than for floating point-based benchmarks. CUDA compiler can use optimizations such as code reordering on integer arithmetic, but may have limited flexibility for floating point arithmetic. The optimizations can hence be impacting the error propagation when using SASSIFI versus NVBitFI, and it seems that a more optimized code increases the AVF. This is in accordance with previously published observations [30]–[32]. The main reason that has been identified for optimized code to have a higher AVF is the reduction of latencies and inefficient instructions. In optimized codes, more instructions are expected to contribute to the output calculation and once corrupted may also increase the chance of corrupting the output. For example, the reduction in dead code

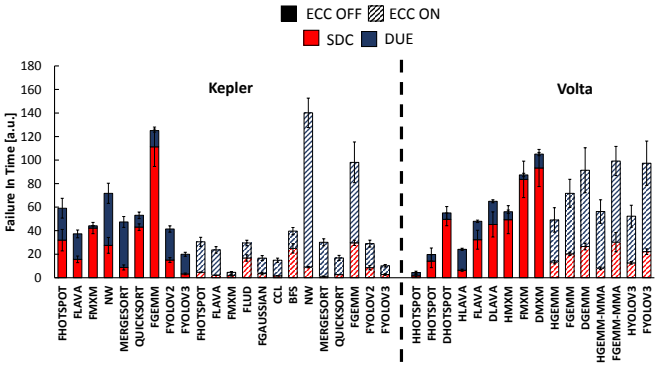


Fig. 5: FIT rates for the Kepler and Volta GPUs, normalized to the DUE rates of FADD and HFMA micro-benchmarks, respectively.

(with aggressive dead-code elimination) and increase in reuse (with aggressive loop invariant code motion) can increase the likelihood of an error propagating to the output.

Figure 4 shows the AVF for the Volta GPU obtained with NVBitFI, focusing on mixed-precision hardware. Half precision fault injection is not shown because NVBitFI tool does not support injections into half instructions as of the date of this submission (October 2020). As Hotspot, Lava, and MxM execute the same kernel for all precisions, their SDC AVF is independent of data precision (the variation between double and float is lower than 4%).

GEMM (and thus YOLO that relies on GEMM for convolution) executes a different kernel for each input and precision configuration. For each GEMM configuration, the data is organized to fit each CUDA group (block, warp, and threads). For each group level, there is a different memory tile configuration that will try to adapt to better use the caches, maximizing and balancing the computational and memory throughputs. Consequently, different kernels will generate different instrumentation, which will impact the AVF. As shown in Figure 4, FGEMM has 30% higher AVF than DGEMM. YOLOv3 is an improved version of YOLOv2, being more accurate and complex (YOLOv3 has 113 layers, YOLOv2 has only 32). On CNNs, some faults that propagate to the output are not considered errors since they do not modify the classification result [33]. Thus, the framework's complexity will impact the AVF, as a less precise CNN can tolerate more incorrect results than a more precise one. Faults in YOLOv2 are then less likely to affect the output as it is less accurate than YOLOv3.

FIT: Figure 5 shows the experimentally measured SDC and DUE FIT rates for the GPUs executing the codes with ECC disabled and enabled. We report normalized values not to reveal business-sensitive data. Values are reported with 95% confidence intervals considering a Poisson distribution.

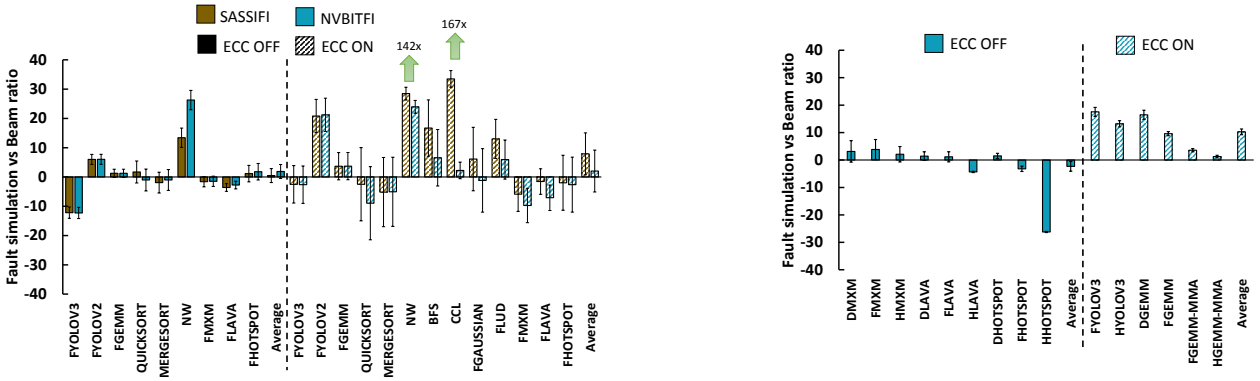
Not surprisingly, the ECC reduces the SDC FIT rate significantly. For K40c, the average SDC FIT rate with ECC OFF is up to 21× higher than with ECC ON. For V100, we were not able to test the same codes with ECC enabled and disabled due to beam time restrictions. The DUE FIT rate increased up to 5× when ECC was enabled. The DUE increase is exacerbated for NW and FGEMM, because of the frequent accesses to the main memory. NW is a component labeling algorithm that processes different parts of the input simultaneously and FGEMM uses lots of global memory and highly utilizes the memory bandwidth, which may result in more ECC detections (or DUEs).

Matrix multiplication (either MxM or GEMM) is the code with the highest SDC FIT rate on K40c or V100. The SDC FIT rate of matrix multiplication is particularly significant when ECC is OFF (2 to 3× higher than the other codes). Matrix multiplication heavily relies on FMA operation which, according to the data in Section V-A, is among the most vulnerable functional units. Moreover, as the code is easily parallelizable, most GPU functional units are used for computation, which exacerbates the probability of faults being activated. Additionally, as shown in Figure 4, matrix multiplication has the highest AVF (i.e., higher likelihood of propagating the activated faults to outputs). In CNNs YOLOV2 and YOLOV3, more than 75% of the operations are similar to matrix multiplications [34]. CNNs share with matrix multiplication the problem of using a high amount of the most sensitive functional units. However, as shown in Figure 4, CNN's AVF is extremely low, reducing the probability of the (likely) faults to propagate to the output.

For the Volta GPU, we focus our analysis on comparing the FIT rates of codes executed with different precisions (double, single, and half). For all the codes, independent of the ECC status, increasing the precision increases the code FIT rate. While the probability of a fault propagation through an application does not significantly depend on the operation's precision (see Figure 3), a higher precision functional unit has a higher area and, thus, a higher probability of being hit by a neutron. According to Equation 2, increasing precision, then, increases FIT_{INST_i} keeping AVF_{INST_i} and $f(INST_i)$ almost constant, resulting in a higher FIT rate. When ECC is disabled, the trend is exacerbated as higher precision also implies a higher number of bits to be stored in memory and this has a linear dependence with the FIT rate (Equation 3).

VII. Beam vs Fault injection

After having detailed the reliability characteristics of Kepler and Volta architectures and several codes, we now compare the codes' FIT rates measured with beam experiments against the ones predicted with fault simulation and profiling. Following the methodology described in Section IV to predict the codes' FIT rate on a target device, we combine fault injection results (detailed in Section VI), application profiling, and results from beam experiments on functional units (detailed in Section V-A). This comparison's main scope is to evaluate whether the reliability analysis based on fault simulation can be considered realistic. Recall that we use both SASSIFI and NVBitFI on Kepler, but only NVBitFI on Volta as SASSIFI is not supported. Moreover, on Kepler, neither SASSIFI nor NVBitFI supports fault injection on NVIDIA proprietary libraries such as cuDNN and CUBLAS. For the codes based on NVIDIA libraries (GEMM and YOLO), we chose to use the AVF measured with NVBitFI on Volta as AVF predictions for



(a) K40c (Kepler)

(b) V100 (Volta)

Fig. 6: Comparison between the SDC FIT rates measured with the beam and predicted with fault injection.

Kepler. As discussed in Section VI, NVBitFI and SASSIFI provide SDC AVFs that differ, on average, by ∼18%. This difference slightly reduces the accuracy of the prediction for GEMM and YOLO on Kepler.

A. SDC

Figure 6 compares different codes' SDC FIT rate as measured with beam experiments and predictions from fault injections and profiling. For visualization and analysis, we divide the measured FIT rate using beam experiments with the FIT rate prediction using our fault injection based method. We plot the ratio whenever the measured FIT rate is higher than the predicted value. If the measured FIT rate is lower than the predicted value, we plot the negative of the inverse (i.e., -1×predicted FIT/measured FIT). For example, on K40c with ECC disabled, beam experiments for FYOLOv2 report a 12× higher FIT rates than the predicted value from fault injections. For FYOLOv3, fault simulation predicts a FIT rate that is 7× higher than that measured with beam experiments.

A promising result that emerges from Figure 6 is the SDC FIT rate prediction is reasonably close to the measured rate in most cases, despite the simplifications used in the fault simulation based method and the two methods being significantly different. The *average* difference between fault simulation and beam experiments, on K40c, is 0.5× for SASSIFI and 1.8× for NVBitFI, with ECC disabled. When ECC is enabled, the average difference is 7.9× for SASSIFI and 2.7× for NVBitFI. For the V100, the average difference is -2.2× when ECC is disabled and 10.2× when ECC is enabled.

For 25 out of 38 predictions, the fault injection underestimates the SDC FIT rate. One limitation of our model is that not all resources are accessible for architecture-level fault simulation, preventing us from considering all the possible sites for faults. As discussed in Section V, we contemplate only the most common instructions as testing all 20 instructions classes is infeasible. While the considered instructions cover more than 70% of instructions that compose the codes (see Figure 1), it is still possible that some faults in the unconsidered instructions generate an error, which the beam experiments inherently account for. When ECC is disabled, the prediction model will consider the memory error rate (Eq. 1) that has already been shown to dominate GPUs' FIT rate [35]. On the average, when ECC is disabled, fault injection can better predict the beam SDC FIT rate, as the contribution to the FIT rate of the not modeled functional units and instructions is much smaller than the memory contribution.

For some applications, e.g., NW and CCL on K40c (Kepler) and HHotspot on V100 (Volta), the fault injection based prediction is significantly different from the measured rates. The kernels used in NW and CCL are not optimized well for GPUs. They under-utilize the available resources and have poor memory access patterns (Table I). We suspect that these inefficiencies reduce the possibilities of having corruptions in functional units (as they are not stressed) and increase the error rate due to other sources of errors, like threads and memory management. Our model does not consider these sources of errors yet, resulting in an underestimation for not well-parallelized codes. HHotspot on V100 (Volta) overestimation (fault injection FIT rate is 27× higher than beam FIT rate) relies on the impossibility to inject faults in half-precision functional units (intrinsic limitation of NVBitFI). We use the float functional units AVF also for the half precision. This simplification is acceptable for most codes (HGEMM and HLava prediction is sufficient) but not for HHostspot, probably because of its intrinsic characteristic of iterating the computation that can smooth the faulty value [36].

While we measured the FIT rates of functional units through beam experiments, an estimation of the FIT rate based on micro-architectural models can also be sufficiently precise as demonstrated in [17] for ARM CPUs. Our model can then be applied to predict the fault rate of codes executed in future GPUs once the micro-architectural model is available.

B. DUE

Our analysis can be used to derive exciting insights on the origins of DUEs as well. DUEs can be caused by several factors, which include interrupts triggered by ECC, a corruption during device-host synchronizations, illegal memory accesses, corruption in the hardware scheduler, changes in the program flow (such as corruption in the instruction cache or the jump

destination address), or faults in hardware resources that stuck the device. Most of these causes for DUEs are independent of the arithmetical operation executed and are not modeled with our prediction strategy.

We characterized the arithmetical functional units, memories, and Load/Store instructions of GPUs with beam experiments. In our prediction model, we include only a subset of the causes for DUEs, and a significant underestimation of the code DUE FIT rate is to be expected. We find that the beam DUE FIT rate is 120× higher, on average, than the predicted DUE FIT rate for K40c ECC OFF and 629× higher with ECC ON. For V100, the measured rates are 60× and 46,700× higher with ECC OFF and on, respectively. This high divergence attests that a large portion of DUEs does not originate from errors in arithmetic instructions. Thus, modeling instructions and memories are not sufficient to predict the GPU DUE rate.

VIII. Conclusions

We compared the FIT rates obtained from beam experiments and fault simulation prediction for 15 codes, using Kepler and Volta based NVIDIA GPUs and two fault injection frameworks (SASSIFI and NVBitFI). If we consider the GPU parallelism management (GPU occupancy and IPC), fault simulation provides SDC FIT rates that are comparable with the beam test results. The result holds for two GPUs for a broad set of codes. Unfortunately, fault simulation alone is not enough to evaluate the probability of DUEs, as faults in inaccessible resources probably are the leading cause of these events. We also investigated the source of the SDC FIT rates of codes executed on GPUs. We combined beam and fault simulation data to understand if the FIT rate of a code is due to the high resource usage, the high criticality of resources (AVF), or a combination of the two. Finally, we compared the main GPUs' functional units sensitivity, including mixed-precision and tensor cores. This data can be used to tune future fault simulation frameworks.

IX. Acknowledgments

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 (Finance Code 001). Neutron beam time was provided by ChipIR (DOI: 10.5286/ISIS.E.RB2000161) thanks to S. Malde, C. Cazzaniga, and C. Frost and by LANSCE thanks to S. Wender and G. Sinnis. Authors also thank P. Racunas, NVIDIA, for donating the V100 GPU.

References

[1] P. Rech *et al.*, "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.

[2] NVIDIA, "NVIDIA Announces World's First Functionally Safe AI Self-Driving Platform." "https://nvidianews.nvidia.com/news/nvidia-announces-worlds-first-functionally-safe-ai-self-driving-platform".

[3] S. K. S. Hari *et al.*, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[4] J. Wei *et al.*, "Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014.

[5] D. A. G. Goncalves de Oliveira *et al.*, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, 2016.

[6] NVIDIA, "Whitepaper: Nvidia's next generation cuda compute architecture:kepler gk110/210,"

[7] NVIDIA, "Whitepaper:nvidia tesla v100 gpu architecturethe world's most advanced data center gpu,"

[8] N. Mahatme *et al.*, "Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process," *IEEE T. on Nuclear Science*.

[9] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, 2005.

[10] D. Oliveira *et al.*, "Experimental and analytical study of xeon phi reliability," in *2017 SC*.

[11] X. Iturbe *et al.*, "Soft error vulnerability assessment of the real-time safety-related ARM cortex-r5 CPU," in *2016 International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*.

[12] H. Cho *et al.*, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th DAC*, 2013.

[13] A. Chatzidimitriou *et al.*, "RT level vs. microarchitecture-level reliability assessment: Case study on ARM(r) cortex(r)-a9 CPU," in *2017 DSN-W*.

[14] C. Constantinescu *et al.*, "Error injection-based study of soft error propagation in amd bulldozer microprocessor module," in *DSN*, 2012.

[15] D. Ferraretto *et al.*, "Simulation-based fault injection with QEMU for speeding-up dependability analysis of embedded software," *Journal of Electronic Testing*.

[16] Z. Chen *et al.*, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *ACM Supercomputing Conference*, 2019.

[17] A. Chatzidimitriou *et al.*, "Demystifying soft error assessment strategies on arm cpus: Microarchitectural fault injection vs. neutron beam experiments," in *Annual IEEE/IFIP DSN*, 2019.

[18] S. K. S. Hari *et al.*, "Estimating silent data corruption rates using a two-level model," 2020.

[19] N. Ho *et al.*, "Exploiting half precision arithmetic in nvidia gpus," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*.

[20] C. Cazzaniga *et al.*, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," *Journal of Physics: Conf. Series*.

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

[22] S. S. Mukherjee *et al.*, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," IEEE Computer Society, 2003.

[23] NVLABS, "Nvbitfi: An architecture-level fault injection tool for gpu application resilience evaluations." https://github.com/NVlabs/nvbitfi.

[24] B. Fang *et al.*, "Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications," in *IEEE ISPASS*, 2014.

[25] S. Jha *et al.*, "Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors."

[26] M. Khairy *et al.*, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ISCA*.

[27] J. Baggio *et al.*, "Analysis of proton/neutron SEU sensitivity of commercial SRAMs-application to the terrestrial environment test method," *IEEE Transactions on Nuclear Science*, 2004.

[28] NVIDIA, "GameWorks documentation - Issue Efficiency." https://docs.nvidia.com/gameworks/index.html.

[29] J. Noh *et al.*, "Study of neutron soft error rate (ser) sensitivity: Investigation of upset mechanisms by comparative simulation of finfet and planar mosfet srams," *Nuclear Science, IEEE Transactions on*.

[30] R. A. Ashraf *et al.*, "Exploring the effect of compiler optimizations on the reliability of hpc applications," in *2017 IEEE IPDPSW*.

[31] F. M. Lins *et al.*, "Register file criticality and compiler optimization effects on embedded microprocessor reliability," *IEEE TNS*, 2017.

[32] L. L. Pilla *et al.*, "Memory access time and input size effects on parallel processors reliability," *IEEE Transactions on Nuclear Science*.

[33] F. F. d. Santos *et al.*, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*.

[34] J. Redmon *et al.*, "Yolo9000: Better, faster, stronger," *arXiv:1612.08242*.

[35] I. S. Haque *et al.*, "Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu," in *2010 10th CCGRID*.

[36] D. A. G. Oliveira *et al.*, "Radiation-induced error criticality in modern hpc parallel accelerators," in *2017 IEEE HPCA*.