Analyzing the Criticality of Transient Faults-Induced SDCs on GPU Applications*

Fernando Fernandes dos Santos
Institute of Informatics UFRGS
Porto Alegre, Brazil
ffsantos@inf.ufrgs.br

Paolo Rech
Institute of Informatics UFRGS
Porto Alegre, Brazil
pech@inf.ufrgs.br

ABSTRACT

In this paper we compare the soft-error sensitivity of parallel applications on modern Graphics Processing Units (GPUs) obtained through architectural-level fault injections and high-energy particle beam radiation experiments. Fault-injection and beam experiments provide different information and uses different transient-fault sensitivity metrics, which are hard to combine. In this paper we show how correlating beam and fault-injection data can provide a deeper understanding of the behavior of GPUs in the occurrence of transient faults. In particular, we demonstrate that commonly used architecture-level fault models (and fast injection tools) can be used to identify critical kernels and to associate some experimentally observed output errors with their causes. Additionally, we show how register file and instruction-level injections can be used to evaluate ECC efficiency in reducing the radiation-induced error rate.

CCS CONCEPTS

• **Computer systems organization → Reliability**; • **Hardware → Fault models and test metrics**; **Hardware reliability screening**;

KEYWORDS

Reliability, high performance computing, soft errors, fault injection

ACM Reference Format:
Fernando Fernandes dos Santos and Paolo Rech. 2017. Analyzing the Criticality of Transient Faults-Induced SDCs on GPU Applications. In *Proceedings of ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, Denver, CO, USA, November 12–17, 2017 (ScalA17),* 7 pages.
https://doi.org/10.1145/3148226.3148228

1 INTRODUCTION

Thanks to their high performance, increased energy efficiency, and flexible development platforms, Graphics Processing Units (GPUs) have enjoyed wide-spread adoption in various application domains, including Hight Performance Computing (HPC) and automotive industries. NVIDIA Kepler GPUs, for instance, act as accelerators in two out of the top 10 supercomputers, including Titan (today's third powerful supercomputer [12]). Applications running on such large-scale systems can be interrupted by radiation-induced crashes or experience silent output corruptions, also known as Silent Data Corruption (SDC). For example, Titan's radiation-induced Mean Time Between Failures (MTBF) is in the order of dozens of hours [35]. As we approach exascale, the resilience challenge will only exacerbate due to an increase in system scale [25, 31]. Since GPUs are traditionally designed for graphics applications, where reliability is not a primary concern, their resilience characteristics are not yet fully understood [19].

A transient fault in a low-level state (flip-flop or SRAM) can either generate a crash/hang at the higher levels, or propagate to the output generating an SDCs, or get masked with no impact at the application-level. SDCs are typically studied by comparing the code experimental output with the expected output [2, 27]. On parallel applications, a soft-error can either corrupt only thread/data value, corrupting the output in an insignificant manner, or propagate to several threads/data values, leading to a significant corruption in the output. Based on the application, certain insignificant output corruptions may be tolerated. For instance, errors affecting the least significant positions of the mantissa could be considered inside the intrinsic imprecision of floating-point operations. Additionally, wave simulations may accept misfits of about 4% [9] and imprecise computation can be generally applied to various HPC applications [4].

Hence, SDCs occurring in massively parallel applications running on GPUs should not be considered equal. Instead their criticality, both in terms of output error magnitude and patterns, should be taken into account. In this study, we focus on analyzing SDC criticality of different applications and evaluate tools/techniques to measure SDC criticality.

We perform architectural-level fault injections on eight parallel benchmarks and expose a subset of them to controlled neutron beams. We analyze and evaluate not only the probability of output corruptions, but also quantify how the corrupted output differs from the expected output. We first categorize the SDCs based on a metric that considers relative difference from the expected output and filter the ones that are lower than a threshold. Then we distinguish between spatial distributions of SDCs at the output to better understand errors propagation.

For our fault injection campaign, we use NVIDIA's SASSIFI tool [21] that analyzes transient error propagation from architectural-level to the application output. We inject failures separately on assembly instruction outputs and register file to measure both the

*Produces the permission block, and copyright information

ScalA17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5125-6/17/11…$15.00
https://doi.org/10.1145/3148226.3148228

Program Vulnerability Factor (PVF) [32] and the Architectural Vulnerability Factor (AVF) [27] of our benchmarks, respectively. Additionally, we perform injections separately with two different errors injection models (single bit-flip and random value corruption). We show that while the error model affects the AVF and PVF of most benchmarks, it does not significantly affect neither the fraction of SDCs that we consider critical nor the spatial distribution of corruption in the outputs.

We also present and, whenever possible correlate the fault injection results with the beam radiation experiment results. Since neutron beam experiments induce faults in all the components of the GPU, including the scheduler, dispatcher, and control logic that are usually difficult to model at architecture-level, they provide deep insights about the resilience characteristics of parallel applications. Our results show that the fault injections and beam experiments can mutually benefit from each other. Specifically, we find that the fraction of SDCs that we consider critical obtained from fault injection campaigns and radiation experiments are similar. This indicates that SASSIFI can be used to identify the benchmarks that are likely to produce more critical SDCs. On the contrary, the output error distributions can differ between the fault injection campaigns and radiation experiments. This is expected as SASSIFI cannot inject faults in the microarchitecture-level resources. Finally, we compare beam data obtained with Error Correcting Code (ECC) enabled and disabled with the measured PVF and AVF. When ECC is enabled, data in register is to be considered protected so the majority of errors are expected from errors in instruction executions (i.e., PVF). On the contrary, when ECC is disabled, both register and instructions faults can affect the computation. As we show, AVF and PVF analysis can be used to have a first evaluation of the efficiency of ECC in reducing the error rate of parallel applications running on GPUs.

Considering these observations, we conclude that radiation experiments need to be performed only on those benchmarks with high PVF and/or AVF to better understand the nature and distribution of errors. Additionally, in some cases, SASSIFI increases the understanding of radiation experiments results by correlating the observed output errors with possible sites for the original radiation-induced event. We made available both our fault-injection and beam data on a publicly accessible repository to ease reproducibility and to allow third party analysis [1].

The reminder of the paper is organized as follow. Section 2 presents the background and related work on the impact of radiation in HPC and GPU fault injection frameworks. Section 3 describes our fault injection and radiation experiment methodologies. Section 4 presents the evaluation based on the proposed metrics for errors criticality. Finally, Section 5 concludes the paper and proposes future work.

2 BACKGROUND

In this section we introduce the proposed concept of SDCs criticality for HPC and give some insights and discuss related works on fault-injection and beam experiments on GPUs.

2.1 Transient Errors Induced SDCs Criticality in HPC

A transient error leads to one of the following outcomes: (1) no effect on the program output (the failure is masked, or corrupted data is not used), (2) SDC (incorrect program output), (3) application Crash or system Hang (the application or node has to be rebooted). Among these outcomes, (2) is harmful as it remains undetected and unpredictable while (3) leads to performance penalties and eventual data loss if a checkpoint was not performed. In this paper we concentrate on SDCs, as Crash or Hangs are at least detectable. Specifically, we will analyze the impact of incorrect program output.

A transient error may alter single or multiple bits in one or multiple words or may lead the executed instruction to produce a wrong output. Depending on how the device or algorithm uses the corrupted data or output, the outcome can vary widely. For instance, scientific applications with filter phases, like stencils, may mask or lower the magnitude of errors, but still spread them among several elements in the output due to repetitive operations with the corrupted data.

We claim that a corrupted output is not always critical in HPC applications. Floating-point operations' results have an intrinsic variance [38]. An error that affects the last digits of the mantissa of a float output element could then have little to no effect on the application correctness. Additionally, physical simulations accept as correct values in a given range [9, 20]. If the value of the corrupted output element is still inside the accepted margin, it may then still be considered as correct. Lately, various works have discussed the preference of not pursuing strict output correctness in HPC to improve performances and efficiency [4, 7]. In this paper, we aim at applying the concept of inexactness to transient errors-induced SDCs. Understanding how off an incorrect output could be from the correct one, and how errors may affect the output on different architectures is essential to precisely evaluate the effect of radiation in current and future HPC machines.

2.2 Reliability Evaluation

The most common strategies to measure devices reliability are fault-injection and beam radiation experiments.

By injecting fault at hardware or software level it is possible to measure the AVF, which is the probability for a low level corruption to propagate till the output [27], or the PVF, which is the probability for a fault at the instruction level to affect the program output [32]. Fault-injection could be performed at different levels of abstraction from RTL to software level [30]. As the RTL level descriptions of modern GPUs are not publicly available, RTL injections could be done only on simplified circuits and, thus, may be imprecise or unrealistic [16]. Faults should then be injected in user-accessible resources, like register files, as in GPUQin, a debugger-based fault-injector [15]. However, as GPUQin requires continuous context switches between GPU and host PC, it introduce a a non-negligible overhead. Lately, also microarchitectural level fault injectors were developed, like GUFI [36]. In this paper we choose SASSIFI NVIDIA fault-injector [21]. SASSIFI injects transient errors in Instruction Set Architecture (ISA) visible states such as general purpose registers, stored values, predicate registers, and condition registers [21]. SASSIFI is based on SASSI, which is an instrumentation tool that

operates at the final step of NVIDIA compilation flow [33]. SASSIFI then does not disrupt the perceived final instructions schedule or register usage. SASSIFI disposes of various errors model: single or double bit-flips (one or two bits error), random or zero value (a random value is assigned to the register or all register bits are reset). Since SASSIFI doesn't need to switch context to the host to inspect or modify GPU state it introduces a very small time overhead. On the average, SASSIFI induced overhead is lower than $5x$ the code execution time.

Radiation experiments do not restrain faults on a limited set of resources and are not biased on the error model. As a result, beam experiments realistically evaluate the device FIT rate [22]. However, radiation experiments offer limited visibility as it is not possible to correlate the observed errors with their causes. Additionally, beam time is typically limited and it is hard to have good statistic for a great number of applications.

It is worth noting that a direct comparison of the AVF/PVF measured through fault-injection and the FIT obtained with radiation experiments is unfeasible, for two main reasons: (1) the raw sensitivity of resources in which errors are injected are unknown, thus faults are injected with equal probabilities in all accessible resources, which is unrealistic; (2) not all the resources accessible, which makes fault-injection incomplete with respect to beam data. However, as we demonstrate, SASSIFI fault injector and radiation experiments can provide similar results in terms of transient faults criticality and are both necessary to have a deep understanding on the reliability of parallel applications.

3 METHODOLOGY

Our experiments are performed on K20 and K40 NVIDIA *Kepler* GPUs, based on the GK110b architecture [28]. Here we describe the selected codes characteristics and present both SASSIFI and radiation experiments setup. Finally, we present the metrics we use to evaluate the criticality of SDCs.

3.1 Selected Benchmarks

Aiming at considering different parallel programming styles and application domain, we chose eight representative and heterogeneous benchmarks from different benchmark suites. Hotspot, Lu Decomposition (LuD), Needleman-Wunsch (NW), and LavaMD are from Rodinia [6]. LULESH is a Department Of Energy (DOE) miniapp [24]. ACCL is from NUPAR [37]. Mergesort and Quicksort are from NVIDIA [8].

Hotspot estimates a processor temperature using an architectural floor plan and simulated power measurements [6]. Hotspot is representative of stencil solvers, used in applications from geophysics to molecular dynamics [9].

Lu Decomposition (LuD) is a linear algebra method that calculates solutions for a square system of linear equations. LuD is representative of highly CPU-bound codes [6].

Needleman-Wunsch (NW) is a bioinformatics algorithm that aligns DNA sequences. NW uses the solution of smaller problems to solve the original problem [6].

LavaMD simulates particle interactions in a large 3D space [6]. LavaMD is representative of Multi-physics Particle Dynamics Code applications which solve a series of ordinary differential equations using Finite Difference Methods [34].

LULESH is a proxy application for shock hydrodynamics developed at Lawrence Livermore National Laboratory (LLNL) as a representative DOE workload [24].

Accelerated Connected Component Labeling (ACCL) is a labeling algorithm that scans the image to find contiguous pixels using child threads through dynamic parallelism [37].

Mergesort is a divide and conquer sorting algorithm of O(n log n) complexity. CUDA toolkit Mergesort creates an array of indexes for each input element, if an element on input data is changed, its index is moved along [8].

Quicksort is one of the most known sorting algorithms. CUDA toolkit Quicksort uses Cuda Dynamic Parallelism, which allows the main kernel create child kernels reducing the host-GPU communications overhead [8].

Input values were pseudo-randomly generated to avoid the bias of results. We carefully balanced the number of zeros and ones in the input vectors and avoid values that could generate overflow during computation. Smaller input sizes were generally selected for SASSIFI than for radiation tests, to speed up fault-injection. This does not undermine the proposed comparison, since a bigger input size only requires the GPU to repeat the same instructions.

3.2 SASSIFI Fault Injection

We use SASSIFI to inject faults both at the instruction output (INST) and in the register file (RF). INST injections are interesting to measure the code PVF, which is how transient errors that modify the result of an instruction propagates at the architecture level till the output. RF are used to measure register file AVF and how an error in memory elements are digested by applications. For each injection site mode (INST or RF) we separately inject single bit-flips and random value to evaluate eventual reliability dependence on error model.

Using SASSIFI, we injected at least 5,000 faults per benchmark (more than 51,000 for all tests), which are sufficient to guarantee the worst case statistical error bars at 95% confidence level to be at most 1.96%.

3.3 Beam experiments

Experiments were performed at the LANSCE facility, Los Alamos, NM and at the ChipIR facility, RAL, Didcot, UK, both suitable to mimic the terrestrial neutron flux effects on electronic devices. To evaluate ECC efficacy and compare beam data with both INST and RF injections we performed radiation experiments with both ECC enabled and disabled.

The available neutron flux was higher than $1 \times 10^5 n/(cm^2 \times s)$, about 6 orders of magnitude higher than the atmospheric neutron flux at sea level ($13n/(cm^2 \times h)$ [23]). Tests were conducted for more than 200 hours of beam time. As we test multiple boards in parallel (two K20 and one K40), we report here results obtained in 600 hours of effective test. If scaled to the natural environment, our results cover at least 8×10^8 hours of normal operations (about 68,000 years).

In a realistic environment, it is highly likely to have no more than one corruption during a single execution. To maintain this behavior,

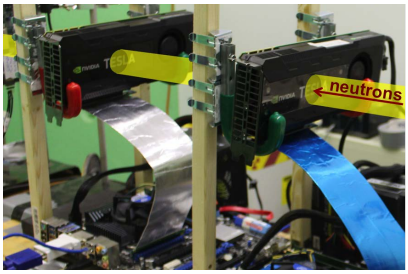

Figure 1: Part of the experimental setup at LANSCE.

experiments were tuned to guarantee observed output error rates lower than 10^{-3} errors/execution, ensuring that the probability of more than one neutron generating a failure in a single code execution remains negligible.

Figure 1 shows part of the experimental setup mounted at LANSCE. A de-rating factor was applied to consider distance from the source attenuation. After the de-rating the device radiation sensitivity seemed independent on the position.

3.4 Metrics to Evaluate SDCs Criticality

Most of HPC simulations or benchmarks work on matrices of various dimensions. Multiple output errors patterns for these benchmarks can be characterized as line, square, cubic, or random (if errors share 1, 2, 3 or none of the dimensions, respectively). It is worth noting that in a realistic environment (and so in our experiments) it is very likely to have at most one transient fault generating an error per execution. Multiple corrupted elements appear as the single transient fault propagates and spreads affecting multiple processes or values. SDCs distribution, then, correlates well with the algorithm sensitivity and is especially significant for parallel architectures.

We also consider sorting algorithms, that have peculiar types of errors: Out Of Order elements (OOO) when at least a pair of elements are not in the correct order, random when in the sorted array there is at least an element that was not present in the input array, Link if there is a wrong link between an element and its index (Merge Sort only). Under radiation we also observed synchronization errors (Sync): GPU and host or GPU blocks or warps lost synchronization (typically Sync results in a multiple of 32 corrupted elements, all with 0 value).

As discussed in Section 2.1, some applications may accept as correct results that slightly wander off the precise value. For instance, a seismic wave application accepts misfits of about 4% [9]. Additionally, the magnitude of the error becomes fundamental for imprecise computations [4, 13]. In this work, being conservative, we chose to consider only mismatches that have a difference with the expected value greater than 2%. When we apply the $> 2\%$ filter we ignore all incorrect elements whose relative error is lower than 2%. We remove faulty executions where there are no mismatches left after the filter. We are aware that the relative error allowed by scientific simulation or imprecise computations depends on the application. Our results are available on a publicly accessible repository [1] also to allow users to apply different filters.

4 RESULTS

In this section we present both our SASSIFI fault-injection and beam radiation experiment result. Presented data is available on a publicly accessible repository [1] to ease reproducibility and to allow third party analysis.

4.1 SASSIFI Criticality evaluation

Following the methodology described in Section 3 we have injected single bit-flips and random value errors in 8 benchmarks. Figures 2a and 2b show the SDCs PVF and AVF for each benchmark measured injecting errors at the instruction output and on register file, respectively. For each error model and error site we show the PVF and AVF measured considering all mismatches and considering only errors with a difference with the expected value greater than 2% (*ALL* and $> 2\%$ in Figures 2a and 2b). For sorting algorithms we did not apply the $> 2\%$ filter as they are not arithmetic codes. Reported AVFs are divided into the observed output error patterns.

The *ALL* PVF and AVF measured through SASSIFI varies of 3 orders of magnitude across codes. For most of the codes, but for sorting, the PVF and AVF are only slightly dependent on the errors model (single bit flip vs random value). Mergesort has a 60% and Quicksort a 40% lower PVF (and similarly for AVF) when random value are injected compared to single bit flips. Crashes for sorting, in fact, are more likely for random value than for single bit-flip (data not shown). For the other codes, on the average the difference between single bit-flip and random value injection is lower than 10%. On the contrary, there are significant differences between the PVF measured injecting errors at the instruction output (Figure 2a) and the AVF of the register file (Figure 2b). Hotspot, for instance, shows a $4.5x$ higher PVF than RF AVF. For all tested benchmarks but Mergesort INST errors are more likely to propagate than RF errors. As RF injects errors only in registers that are actually used (and not on all instantiated registers), differences are observed because the way the error (injected in INST or RF) is consumed by the application is different. For arithmetic codes, errors at the instruction level are more likely to propagate to the output than errors in the RF. For Mergesort the opposite trend is observed as an RF error mainly affects elements to be compared and is likely to modify the control flow and propagate to the output.

While providing different sensitivities, INST and RF injections are still consistent in identifying the codes with higher sensitivity. Both the PVF and AVF of Hotspot, lavaMD, NW, and sorting algorithms is at least one order of magnitude higher than others, independently on the error model. This means that a simplistic error model and one injection site can still be sufficient to identify the most sensitive kernels. ACCL, LuD, and Lulesh PVF and AVF are so low that they are not to be considered critical (this is confirmed by low FIT for ACCL measured through radiation experiments in Section 4.2).

The codes criticality can be further analyzed considering the difference between the corrupted value and the expect value. As shown in Figures 2a and 2b comparing *ALL* and $> 2\%$, the portion of SDCs with a difference from the expected value greater than 2% strongly depends on the code. A portion of SDCs of about 85% for Hotspot, 50% for Lulesh, and 30% for NW only slightly differs from the expected value. A key finding of our fault-injection campaign

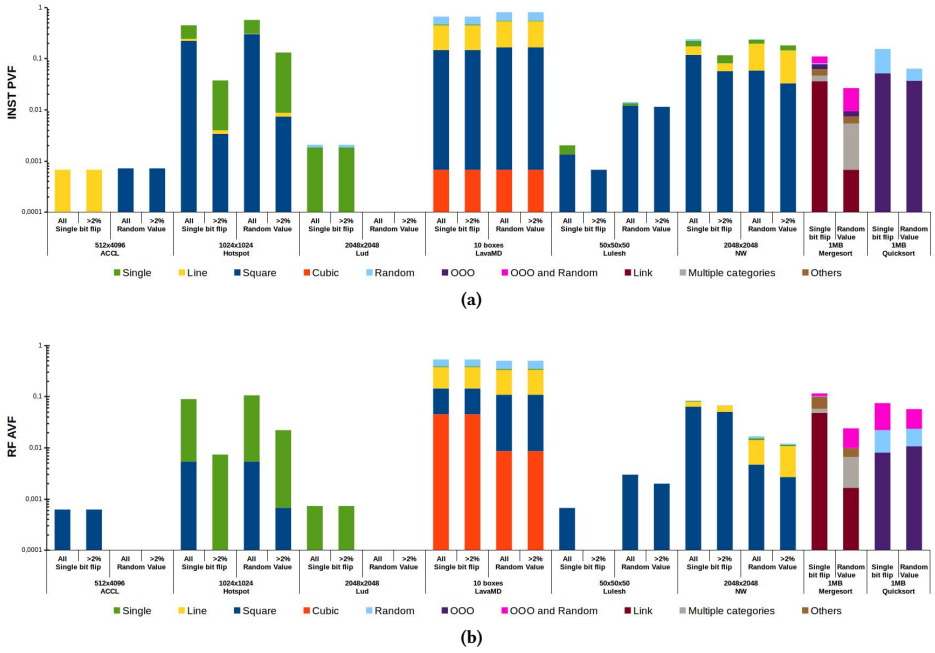

Figure 2: SASSIFI results for INST (a) and RF error sites (b). For each code we inject single bit flips and random value. We report the PVF and AVF considering all SDCs (ALL) and only SDCs with a relative difference with the expected value greater than 2% ($> 2\%$). PVF and AVF are divided into the observed output erros patterns. A blank space means that no SDC was observed during fault-injection campaign.

is that neither the error model (single bit-flip vs. random value) nor the injection site (INST or RF) significantly impact the portion of SDCs that are critical (i.e., $> 2\%$). This means that the criticality of SDCs is dependent on the algorithm more than the error model.

Hotspot is particularly significant. If all SDCs were considered, Hotspot would be one of the codes with the highest PVF and AVF. However, if a 2% of difference is tolerated, Hotspot sensitivity is reduced of about one order of magnitude. We believe this result to be common for stencil applications that iteratively update the solution based on the current state. Similar behavior and explanation applies to Lulesh. For NW, an error in smaller problems can be averaged with correctly computed solution to other problems when the final solution is reconstructed. For these applications a transient fault could modify the current simulation state but, in the following iteration, the error is smoothed and filtered. On the contrary, for ACCL, LuD, and lavaMD 95% to 100% of observed errors where significantly different from the expected value. LavaMD performs dot products between particles which prevent injection to be smooth with other correctly calculated particles. Moreover, products use exponentiation operation which can turn small value variations into large differences. LuD has many interdependencies of data, which makes the single transient error to be used to compute various output elements. Most of injections for ACCL are filtered, which is expected for an image processing algorithm. Only major corruptions that alter the code control flow affect the output.

To better understand injections propagation to the output, we divide the AVFs in Figures 2a and 2b in the observed patterns. As described in Section 3.4, for 6 codes we use spatial error distributions while sorting has specific output errors patterns. The output error distribution is almost unaltered by the error model for all the codes but for sorting. Sorting algorithms control flow is dependent on data value, which justifies the error model dependence.

Output errors distribution is significantly dependent on injection site (INST or RF) only for lavaMD and Hotspot. LavaMD has 60% more cubic errors (and less square errors) when injection are on RF. Cubic errors are likely to be caused by errors on shared registers (eventually corrupted by INST). For Hotspot about 50% of errors are single when injections are made on INST and 90% on RF (Figures are in log scale). Errors in memory elements are then less likely to spread than INST errors. Our output error distribution analysis can be used to further understand errors propagation. Knowing how transient faults appear at the output is essential to evaluate the potential harm of errors and eventually to create wise SDCs detection strategies.

4.2 SASSIFI vs Radiation Experiments

Figure 3 shows the normalized FIT obtained irradiating a subset of 5 benchmarks with controlled neutron beam. The probabilities obtained with SASSIFI are different from the FIT measured with radiation experiments. For INST injections, Hotspot and lavaMD have similar PVF while under radiation Hotspot has 7x lower FIT than lavaMD. As detailed in Section 2.2, it is not possible to directly compare AVF or PVF and FIT as the formers do not include information on the amount of resources used for computation nor

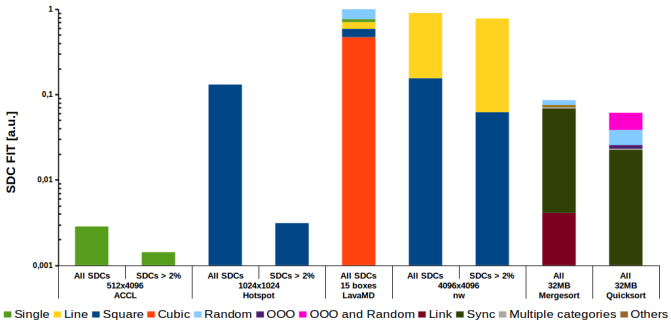

Figure 3: Normalized FIT from beam test, expressed in a.u.. Experiments were performed with ECC OFF.

their sensitivity. However, the comparison between Figures 2 and 3 allows to draw additional insights on SDCs criticality.

The codes with very low PVF/AVF (e.g ACCL) were also very reliable under radiation as only few errors when exposed to the beam. On the contrary, LavaMD, Hotspot, and NW which have high application-level propagation have also a high FIT rate. SASSIFI can the be used to select those codes which are more likely to have a high FIT rate. Radiation experiments should be performed only on those code, as it is likely to gather a statistically significant amount of data in reasonable time.

A key observation is that SASSIFI can be used to measure the portion of SDCs which are critical, i.e., that have a high difference from the expected value. In fact, more than 99% of injections caused a critical error for lavaMD, 80% for NW, and only 15% for Hotspot. This is in good accordance with our radiation experiment results shown in Figure 3. This enforce out insight that the criticality of SDCs depends on the algorithm and not on the error model or error site.

Output errors patterns is significantly different between SASSIFI and radiation experiments. We attribute this difference to error model and injection site differences. Radiation experiments induce bit-flips in the low-level state such as flip-flops and SRAM buffers at the microarchitecture-level, while SASSIFI injects errors at the architecture-level. The low-level transient fault can potentially propagate in ways that are different than the chosen error models. For instance, low-level bit-flips (e.g. bit-flip in a decoder) can corrupt one instruction in multiple threads in the warp.

Sorting output error patterns difference is particularly significant. Only few *OOO* errors were observed under the beam while they dominate Quicksort injection results. This is because under radiation a great portion of errors are *Sync*, which are host-GPU communications errors not reproducible with SASSIFI, yet. Comparing INST and RF injection sites can help understanding the reasons for some observed errors under radiation. About 60% of LavaMD radiation errors are cubic. Cubic errors hardly appear when injecting in INST but they are common in RF injection. We can conclude that observed cubic errors under radiation could be caused by errors in a shared memory location.

5 CONCLUSIONS AND FUTURE WORK

A terrestrial neutron strike may perturb a transistor's state, generating bit-flips in memory or current spikes in logic circuits that, if latched, lead to an error [5, 26]. As GPUs were originally designed for graphics rendering, a task where high reliability is not a concern [3], their architecture has some inherent reliability weaknesses [10, 11, 19, 39]. A single corrupted thread could feed thousands of future parallel threads with erroneous data, leading to multiple errors in the output. A vast majority of radiation-induced errors affect multiple output elements in GPUs [29]. This is an issue particularly for arithmetic-intensive kernels such as General Matrix Multiplication (GEMM), Object detection, Physics simulation, Computer graphics, and others.

Low-power embedded GPUs have enjoyed widespread adoption in various application domains, including the automotive one. Embedded GPUs, in fact, are part of projects implementing the Advanced Driver Assistance Systems (ADAS), which analyzes camera or radar signals to detect pedestrians or obstacles and activate the car brakes to prevent collisions [14]. Additionally, autonomous driving systems, which is the new trend in the automotive market, rely on humans or obstacles computer-aided detection.

We are the first research group which experimentally evaluated embedded GPUs under a neutrons source. With radiation tests, we were able to propose hardening techniques for Object detection algorithms which are commonly used on ADAS [17, 18].

REFERENCES

[1] 2016. experimental data; anonymous github repository. https://github.com/reliabilitylogdata/date_2017. (2016).
[2] R.C. Baumann. 2005. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on* 5, 3 (Sept 2005), 305–316. https://doi.org/10.1109/TDMR.2005.853449
[3] M.A. Breuer et al. 2004. Defect and error tolerance in the presence of massive numbers of defects. *Design Test of Computers, IEEE* 21, 3 (May 2004), 216–227. https://doi.org/10.1109/MDT.2004.8
[4] Melvin A Breuer. 2005. Multi-media applications and imprecise computation. In *Digital System Design, 2005. Proceedings. 8th Euromicro Conference on*. IEEE, 2–7.
[5] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger. 1997. Comparison of error rates in combinational and sequential logic. *Nuclear Science, IEEE Transactions on* 44, 6 (1997), 2209–2216.
[6] Shuai Che et al. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 44–54.
[7] Wei-Fan Chiang et al. 2013. Determinism and reproducibility in large-scale HPC systems. In *Workshop on Determinism and Correctness in Parallel Programming (WoDet)*.
[8] NVIDIA Corporation. 2015. *CUDA toolkit documentation*. Technical Report. http://docs.nvidia.com/cuda/cuda-samples/#axzz4JVHU2IKO
[9] Josep de la Puente et al. 2014. Mimetic seismic wave modeling including topography on deformed staggered grids. *GEOPHYSICS* 79, 3 (2014), T125–T141. https://doi.org/10.1190/geo2013-0371.1 arXiv:http://dx.doi.org/10.1190/geo2013-0371.1
[10] D. A. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech. 2016. Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units. *IEEE Trans. Comput.* 65, 3 (March 2016), 791–804. https://doi.org/10.1109/TC.2015.2444855
[11] Nathan DeBardeleben, Sean Blanchard, Laura Monroe, Phil Romero, Daryl Grunau, Craig Idler, and Cornwell Wright. 2013. GPU Behavior on a Large HPC Cluster. *6th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids in conjunction with the 19th International European Conference on Parallel and Distributed Computing (Euro-Par 2013), Aachen, Germany*, (August 26-30 2013).
[12] J.J. Dongarra et al. 2015. TOP500 Supercomputer Sites: November 2015. (2015). http://www.top500.org
[13] D. Ernst et al. 2004. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro* 24, 6 (Nov 2004), 10–20. https://doi.org/10.1109/MM.2004.85

[14] European New Car Assessment Programme. 2012. Euro NCAP Rating Review, Report from the Ratings Group. (June 2012). http://www.euroncap.com

[15] B. Fang et al. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *ISPASS, 2014*. 221–230. https://doi.org/10.1109/ISPASS.2014.6844486

[16] N Farazmand et al. 2012. Statistical fault injection-based AVF analysis of a GPU architecture. *Proceedings of SELSE* 12 (2012).

[17] Fernando Fernandes, Lucas Weigel, Claudio Jung, Philippe Navaux, Luigi Carro, and Paolo Rech. 2016. Evaluation of Histogram of Oriented Gradients Soft Errors Criticality for Automotive Applications. *ACM Trans. Archit. Code Optim.* 13, 4, Article 38 (Nov. 2016), 25 pages. https://doi.org/10.1145/2998573

[18] Fernando Fernandes, Lucas Weigel, Philippe Navaux, Luigi Carro, and Paolo Rech. 2017. Evaluation and Mitigation of Neural Network-based Object Detection in Three GPU Architectures. *SELSE* (2017).

[19] L. A. Bautista Gomez et al. 2014. GPGPUs: How to Combine High Computational Power with High Reliability. In *DATE*. Dresden, Germany.

[20] Qiang Guan et al. 2015. Towards Building Resilient Scientific Applications: Resilience Analysis on the Impact of Soft Error and Transient Error Tolerance with the CLAMR Hydrodynamics Mini-App. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. 176–179. https://doi.org/10.1109/CLUSTER.2015.35

[21] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. 2017. SASSIFI: An Architecture-level Fault Injection Tool for GPU Application Resilience Evaluation. *International Symposium on Performance Analysis of Systems and Software* (Oct 2017).

[22] I. Herrera-Alzu et al. 2014. System Design Framework and Methodology for Xilinx Virtex FPGA Configuration Scrubbers. *Nuclear Science, IEEE Transactions on* 61, 1 (Feb 2014), 619–629. https://doi.org/10.1109/TNS.2013.2292816

[23] JEDEC. 2006. *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*. Technical Report JESD89A. JEDEC Standard.

[24] Ian Karlin et al. 2013. *LULESH 2.0 Updates and Changes*. Technical Report LLNL-TR-641973. 1–9 pages.

[25] Robert Lucas. 2014. Top Ten Exascale Research Challenges. In *DOE ASCAC Subcommittee Report*.

[26] N.N. Mahatme, T.D. Jagannathan, L.W. Massengill, B.L. Bhuva, S.-J. Wen, and R. Wong. 2011. Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process. *Nuclear Science, IEEE Transactions on* 58, 6 (2011), 2719–2725.

[27] Shubhendu S. Mukherjee et al. 2003. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *IEEE/ACM Micro*. IEEE Computer Society, Washington, DC, USA, 29–.

[28] NVIDIA. 2015. NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110. (2015). http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[29] D. Oliveira, L. Pilla, M. Hanzich, V. Fratin, F Fernandes, C. Lunardi, J. Cela, P. Navaux, L. Carro, and P. Rech. 2017. Radiation-Induced Error Criticality in Modern HPC Parallel Accelerators. In *Proceedings of 21st IEEE Symp. on High Performance Computer Architecture (HPCA)*. ACM.

[30] G. P. Saggese et al. 2005. An experimental study of soft errors in microprocessors. *IEEE Micro* 25, 6 (Nov 2005), 30–39. https://doi.org/10.1109/MM.2005.104

[31] Marc Snir et al. 2014. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications* (2014), 1–45.

[32] V. Sridharan and D. R. Kaeli. 2009. The effect of input data on program vulnerability. In *Proceedings of the 2009 Workshop on Silicon Errors in Logic and System Effects (SELSE '09)*.

[33] Mark Stephenson et al. 2015. Flexible Software Profiling of GPU Architectures. In *Proceedings of the International Symposium on Computer Architecture (ISCA '15)*. 185–197.

[34] Lukasz G. Szafaryn et al. 2010. Experiences with Achieving Portability across Heterogeneous Architectures. (2010).

[35] Devesh Tiwari et al. 2015. Understanding GPU Errors on Large-scale HPC Systems and the Implications for System Design and Operation. In *Proceedings of 21st IEEE Symp. on High Performance Computer Architecture (HPCA)*. ACM.

[36] S. Tselonis and D. Gizopoulos. 2016. GUFI: A framework for GPUs reliability assessment. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 90–100. https://doi.org/10.1109/ISPASS.2016.7482077

[37] Yash Ukidave et al. 2015. NUPAR: A Benchmark Suite for Modern GPU Architectures. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ACM, 253–264.

[38] James H. Wilkinson. 1994. *Rounding Errors in Algebraic Processes*. Dover Publications, Incorporated.

[39] H. J. Wunderlich, C. Braun, and S. Halder. 2013. Efficacy and efficiency of algorithm-based fault-tolerance on GPUs. In *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*. 240–243. https://doi.org/10.1109/IOLTS.2013.6604090