Multi-faceted microarchitecture level reliability characterization for NVIDIA and AMD GPUs

Multi-faceted Microarchitecture Level Reliability Characterization for NVIDIA and AMD GPUs

Alessandro Vallero[§], Sotiris Tselonis, Dimitris Gizopoulos* and Stefano Di Carlo[§]
[§] Politecnico di Torino, {stefano.dicarlo | alessandro.vallero}@polito.it * University of Athens, dgizop@di.uoa.gr

***Abstract* – State-of-the-art GPU chips are designed to deliver extreme throughput for graphics as well as for data-parallel general purpose computing workloads (GPGPU computing). Unlike computing for graphics, GPGPU computing requires highly reliable operations. Since provisioning for high reliability may affect performance, the design of GPGPU systems requires the vulnerability of GPU workloads to soft-errors to be jointly evaluated with the performance of GPU chips. We present an extended study based on a consolidated workflow for the evaluation of the reliability in correlation with the performance of four GPU architectures and corresponding chips: AMD Southern Islands and NVIDIA G80/GT200/Fermi. We obtained reliability measurements (AVF and FIT) employing both fault injection and ACE-analysis based on microarchitecture-level simulators. Apart from the reliability-only and performance-only measurements, we propose combined metrics for performance and reliability that assist comparisons for the same application among GPU chips of different ISAs and vendors, as well as among benchmarks on the same GPU chip.**

Keywords – GPGPU; microarchitecture simulator; reliability; performance; fault injection; throughput.

I. Introduction

Graphics Processing Units (GPUs) are a powerful computing platform for both graphics and general-purpose, data-parallel and computing-intensive applications. GPUs are increasingly used in applications where reliability and performance are a top tier concern [2]. Trading-off reliability and performance is therefore a key aspect of GPU based systems [1]. Reliability analysis for these systems is challenging. It requires accurate and fast techniques able to carefully trade-off between analysis time and accuracy of the reported measurements. Moreover, it must produce results able to guide the system designers in the selection and development of efficient error protection mechanisms. Apart from the use of physical error injections [3][4], simulation-based techniques are the preferred solution to analyze the reliability of GPU systems. Two reliability estimation methodologies for GPUs have been established in this field similarly to the CPU domain: (i) fault injection [5][6][7][15][21] and (ii) Architectural Correct Execution (ACE) analysis [15][22].

This paper presents the results of an extended study aiming at characterizing the main factors (hardware and software) that influence the reliability of GPU chips in the presence of soft-errors. The study focuses on errors in the register file and in the local/shared memory[1]. These arrays are among the biggest arrays of the GPU and are therefore prone to be affected by radiation induced soft-errors. Even if some of these arrays are often delivered with hardware-protection techniques such as Error Correction Codes (ECC), these can incur performance and energy overheads and hence may not be enabled by users in selected applications [6]. Moreover, they may be protected with an ECC scheme that cannot cover the expected cardinality of faults. Therefore, evaluating the reliability of even protected arrays is an important task. The paper considers several important factors including: correlation between reliability and performance, size of the hardware structures, resource occupancy and execution scheduling. Different reliability assessment methodologies are used to identify trade-offs between analysis time and accuracy of results. GPUs from different vendors, architectures and programming models are compared: AMD Southern Islands and NVIDIA G80/GT200/Fermi. Reliability of all devices is analyzed running the same set of benchmarks written using the typical development framework for each architecture: OpenCL[2] for the AMD GPU and CUDA[3] for the NVIDIA GPUs. Simulations have been performed using a microarchitecture-level simulation. The framework includes tools to perform both soft-error fault injection campaigns and ACE-analysis. Microarchitecture-level simulators provide a good accuracy for the considered hardware structures as demonstrated for CPUs in [10] and, at the same time, they significantly improve the simulation throughput compared to RTL models that are often not publicly available.

To our knowledge, this is the first work that extensively compares the reliability correlated with the performance of some of most important GPU families with different microarchitecture, Instruction Set Architecture (ISA) and computational model using the same set of benchmarks and employing the two most prominent reliability evaluation methodologies. Preliminary results in this direction have been recently discussed in [11]. Such a multidimensional study delivers significant insights on: differences in GPU vulnerability estimations between fault injection experiments and ACE analysis; variations in the vulnerability of specific hardware components and benchmarks among different GPU architectures; joint evaluation of reliability and performance to support designers and programmers when evaluating different GPUs and workloads.

II. Reliability Evaluation Framework

Simulations on the two GPU families have been carried out using two tools named GUFI and SIFI. GUFI, previously presented in [6], is a micro-architectural level fault injector based on GPGPU-Sim (v3.2.2) [19]. It has been developed to perform reliability analysis on NVIDIA GPUs. GUFI has been extended for the purposes of our study to perform ACE-based analysis. SIFI is a fault injection and ACE analysis tool developed to characterize AMD GPUs [15]. SIFI is built on top of Mult2Sim (v4.2) and supports the Southern Island assembly language [13]. For both tools, reliability is analyzed looking at the low-level assembly code running on the real hardware. Therefore, for NVIDIA GPUs, the SASS assembly is used instead of the intermediate and device-independent PTX assembly. We made this choice to study the

[1] Local memory is the AMD terminology while shared memory is the NVIDIA terminology for the same memory type.

[2] https://www.khronos.org/opencl/

[3] https://www.geforce.com/hardware/technology/cuda

vulnerability of the actual physical registers (instead of the virtual PTX registers), allowing for a fair comparison of NVIDIA and AMD chips. This study focuses on soft-errors in memory elements. Addressing the impact of these faults is extremely relevant since GPUs include a large number of big memory arrays.

Several reliability metrics are considered. The *architectural vulnerability factor* (AVF) of a hardware structure is the main considered reliability metric. The AVF is the probability that a bit-flip (soft-error) at the target structure manifests as a visible error in the computation (output corruption, crash). It jointly considers hardware and software masking effects. To evaluate the masking properties independently from the occupancy, the concept of AVF_{Util} is used [5]. The AVF_{Util} is the probability that a soft-error in hardware structure that is actually used by the program causes an error in the computation. The AVF can be expressed in terms of the AVF_{Util} and the occupancy, that is the percentage of the hardware structures actually used by the running program, as: $AVF = AVF_{Util} \times Occupancy$. By combining the AVF and the raw soft-error rate (λ_i) of every hardware structure, the GPU failures in time rate (FIT_{GPU}) can be computed as: $FIT_{GPU} = \sum_i AVF_i \times \lambda_i \times \#bit_i$, where $\#bit_i$ is the number of memory elements of the hardware structure *i*. Finally, a system designer or a programmer can be provided with a broader idea of the system performance and reliability for any given workload when combined metrics are used. Such a metric can be the *executions per failure* (EPF) defined as the number of executions of a benchmark before a failure manifests: $EPF = EIT/FIT_{GPU}$ [15], [18]. EIT (*executions in time*) is the number of complete correct executions of a benchmark in 10^9 hours of device operation. Another metric that can be defined to jointly evaluate reliability and performance is the *instructions per failure* (IPF). The IPF measures the instruction throughput of a benchmark before a failure manifests instead of the number of complete program executions per failure like in the EPF case. IPF is defined as: $IPF = IIT / FIT_{GPU}$ where IIT (*instructions in time*) is the number of instructions executed in 10^9 hours.

A. Fault injection campaigns

Both GUFI and SIFI, the tools exploited in this paper, perform fault injection experiments following the flow reported in Fig. 1.A, which is composed of three macro phases.

The *fault generation phase* creates the list of faults (fault pool) to inject. The application is profiled to identify the time intervals in which the GPU is active and to gather information about the executed kernels. This is used to speed up the simulation injecting faults only during active intervals. After that, the fault pool is generated randomly selecting for each fault its time (i.e., the clock cycle it manifests) and the specific memory element to corrupt. During the *fault injection phase*, each fault is injected, and the behavior of the system is observed. Fault injections are simulated in parallel to save time, exploiting concurrent executions. As soon as the simulation of a fault completes, the output of the computation is analyzed (*fault classification*). If the computation ends properly and its output is correct, the fault is classified as masked. Otherwise, it is classified as not masked. To improve the injection throughput, faults that target idle resources (i.e., resources not used in the context of the application) are not simulated since they can be directly classified as masked. These faults are named non-util faults since they do not affect the AVF_{Util} computation. When all faults are classified, the AVF and the AVF_{Util} can be computed as reported in Fig. 1.B.

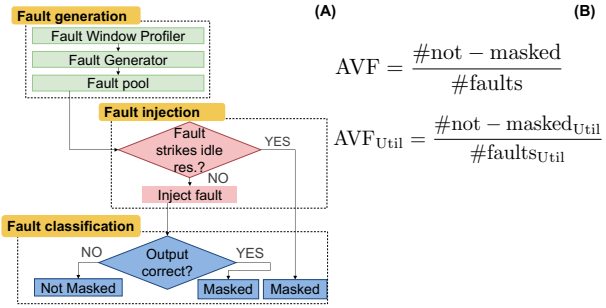

Fig. 1. Fault injection campaign workflow

B. ACE Analysis

ACE (Architecturally Correct Execution) analysis estimates the AVF simulating a single run of the program. It is therefore a very fast analysis that however has reduced accuracy with respect to fault injection. ACE analysis is based on the fact that not all entries of a hardware structure (e.g., registers of the register file) influence the output of the system. The AVF of the structure can therefore be estimated by determining which entry affects the system's output (ACE resources) and which does not (un-ACE resources). Fig. 2.A summarizes the implemented ACE analysis workflow for the register file that follows the approach presented in [8] and [9] for CPU memory arrays. A similar approach is implemented for the other hardware structures.

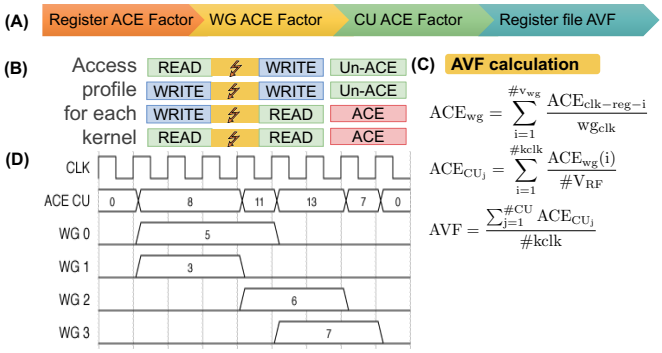

Fig. 2. Register File ACE analysis workflow

Each GPU kernel is analyzed separately and then results are aggregated. For each kernel, the number of registers assigned to each work-group ($\#v_{wg}$) is first computed as explained in [20]. All registers not assigned to any work-group are classified as *idle* and marked as un-ACE, while the others are profiled during the execution of the kernel. During the time intervals (i.e., clock cycles) between a read and a write operation (*read-to-write intervals*), and between two consecutive write operations (*write-to-write intervals*) a register can be safely considered un-ACE (Fig. 2.B). In all other cases, it is marked as ACE. The ACE factor of each work-group (ACE_{wg}), i.e., the work-group average number of ACE registers per clock cycle, can be computed as reported in Fig. 2.C where wg_{clk} is the number of clock cycles required to execute the work-group and $ACE_{clk-reg-i}$ is the number of clock cycles in which the register *i* is ACE. At this point, the ACE factors of every work-group of the kernel must be aggregated to obtain the ACE factor of the compute unit (ACE_{CU}). To perform this computation, we build a timing diagram representing the time window of every work-group executed by the compute unit (CU).

An example is shown in Fig. 2.D, where 4 work-groups (WG1 – WG3) are executed and the number of concurrent work-groups is 2. For each work-group and each clock cycle the diagram reports the related $ACE_{wg.}$ Based on the timing diagram of each work-group, the ACE_{cu} timing diagram is computed by summing up the ACE_{wg} of the scheduled blocks at each clock cycle divided by the number $\#V_{RF}$ of registers per compute unit (Fig. 2.C). The AVF of the entire register file running the selected application can be finally computed as the sum of the AVF of all CUs divided by the number #kclk of clock cycles required to execute the GPU kernel (Fig. 2.C). The AVF_{Util} of the register file can be computed in a similar way, but the ACE_{cu} timing diagram is divided by the number of utilized registers of the running work-groups at each clock cycle (#rf-used), instead of dividing it by the total number of registers of the register file. In the example of Fig. 2.D, considering #rf=32 and #kclk=7, the AVF can be computed as $AVF_{RF} = \frac{\frac{8}{32}+\frac{8}{32}+\frac{8}{32}+\frac{11}{32}+\frac{13}{32}+\frac{13}{32}+\frac{7}{32}}{7} = 0.3$ (30%). Assuming that each work-group consists of 3 work-items and each work-item utilizes 4 registers then a work-group utilizes 3x4=12 registers. Considering the timing of the work-groups the number of utilized registers at each clock cycle is always 24, apart for the last cycle where it is 12. Consequently, the AVF_{Util} can be computed as $AVF_{Util\_RF} = \frac{\frac{8}{24}+\frac{8}{24}+\frac{8}{24}+\frac{11}{24}+\frac{13}{24}+\frac{13}{24}+\frac{7}{12}}{7} = 0.45$ (45%).

The accuracy of the ACE analysis could be increased by taking into account the presence of data processed by dead instructions (i.e., not contributing to the output results) and logic masking (i.e., logical and arithmetic operations resulting in masking of the fault) [8] [9]. From our simulations, we noticed that dead data in the selected benchmarks represent a negligible portion of the application (less than 0.5%). Therefore, neglecting them does not introduce a significant loss of accuracy while reducing the analysis time. Logic masking instead may have an impact on the accuracy. However, its inclusion in the ACE computation significantly increases the complexity of the analysis and therefore has not been implemented in this study. ACE analysis is mainly exploited here for its fast simulation time and we are interested in understanding the difference in accuracy with respect to the fault injection.

III. Reliability evaluation

A. Experimental setup

We analyzed 4 GPU chips with different architectures: AMD HD Radeon™ 7970 (Southern Islands architecture), NVIDIA Quadro™ FX 5600 (G80 architecture), NVIDIA Quadro™ FX5800 (GT200 architecture) and NVIDIA Geforce™ GTX 480 (Fermi architecture). We used 10 benchmarks: 7 available both in the CUDA SDK[4] and AMD-APP SDK[5] and 3 from the Rodinia benchmarks [16]. All benchmarks have equivalent OpenCL and CUDA implementations. Table 1 summarizes the characteristics of the CUs of the considered GPU chips. Since manufacturing data are not public, the raw bit soft-error rate per cell (λ) has been estimated from literature data [17]. The error rate is normalized to 1E-3 for the 90nm SRAM cell. For all chips, we measured the AVF of the general-purpose register file (RF) and the local memory (LM) using both Fault Injection (FI) and ACE Analysis (ACE). For the FI experiments, we applied statistical fault sampling [12]. Considering the size of the targeted structures, to reach a 2.88% error margin with 99% confidence level, 2,000 fault injection experiments were conducted for each hardware component of each GPU architecture. Every benchmark was executed with the same input data-set for all considered GPU devices. The data-sets were chosen to maximize and stress the use of the CUs. However, since this significantly increases the simulation time when considering multiple CUs, following the approach used by Farazmand et al. [5], we scaled the analysis considering a single CU.

Table 1: CU details of the target GPU architecture

	Chip name	Quadro™ FX 5600	Quadro™ FX5800	Geforce™ GTX 480	HD Radeon™ 7970
	Technology	90 nm	55 nm	40 nm	28 nm
	λ FIT/bit	1E-3	0.72E-3	0.52E-3	0.32E-3
	Vendor	NVIDIA	NVIDIA	NVIDIA	AMD
	Architecture	G80	GT 200	Fermi	S. Islands
Register file		32KB	64KB	128KB	256KB
Local memory		16KB	16KB	48KB	64KB
SIMD Units		1	1	2	4
Max	#wg	8	8	8	40
	#wavefronts	24	32	48	40
	#work-items	768	1024	1536	1840

B. Experimental results

The two charts in Fig. 3 report the AVF for RF and LM computed using both FI and ACE analysis. By analyzing RF (Fig. 3.A), we observe significant differences in the AVF depending both on the hardware platform and on the executed benchmarks. Overall, for all benchmarks the HD Radeon 7970 is the chip with the lowest RF vulnerability while the Quadro FX 5600 is the one with the highest vulnerability. Looking at a single hardware platform, the difference in the AVF for the different benchmarks is the result of the way the software stresses the resource and is able to be resilient to the injected faults. Instead, when looking at the same benchmark executed on the different chips we can note a strong correlation of the AVF with the occupancy (red bullets). Considering RF, based on their average occupancy, the architectures can be ordered as follows: HD Radeon 7970 (12.89%), GTX480 (37%), Quadro FX5800 (50%), Quadro FX 5600 (57%). This trend holds for all benchmarks with the exception of histogram in which the RF occupancy of the GTX480 is 35% while the one of Quadro FX5800 is only 26%. The occupancy of a resource is therefore a strong indicator to predict the AVF trend for a single application executed on different chips. However, it does not provide information on the actual AVF values. In fact, benchmarks with similar occupancy have significantly different AVFs (e.g., backprop and scan in Fig. 3.A). This is an important finding obtained from the analysis of our simulations. It confirms that occupancy is not the only contribution to AVF but the particularities of the access patterns from each benchmark are also important. When considering the AVF of LM (Fig. 3.B), while the correlation between AVF and the occupancy of the resource still holds, a clear AVF variation trend between the four chips valid for all benchmarks cannot be identified. This suggests that the way this resource is used strongly depends on the executed application code. In particular, the AVF of the HD Radeon 7970 is significantly higher than the one of the other architectures for histogram, where occupancy is 100%. In the scan and reduction benchmarks the vulnerability is almost equal for all

[4] https://developer.nvidia.com/cuda-toolkit-42-archive

[5] http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

architectures, while Quadro FX 5800 has a slightly higher AVF for backprop and dwt where its occupancy is 22% and 53%, respectively. Not all benchmarks actually use the local memory, this is the case of gaussian, kmeans and vactoadd.

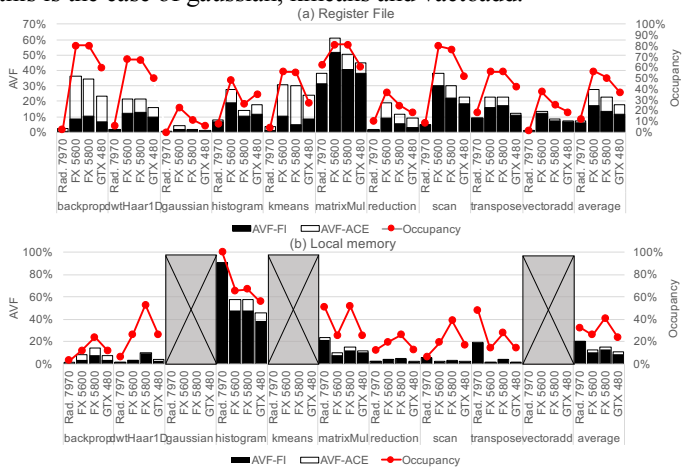

Fig. 3. AVF measured by fault injection (FI) and ACE.

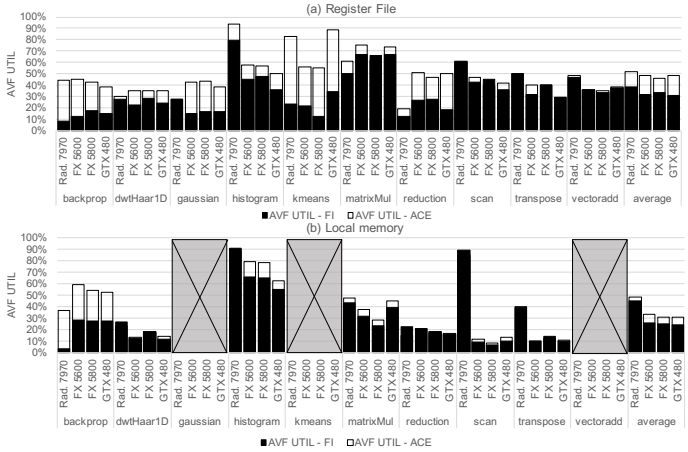

Fig. 4. AVF Util measured by fault injection (FI) and ACE.

To discuss vulnerability decoupled from the occupancy of the resources, we can use the AVF_{Util} measurements reported in Fig. 4. Again, both diagrams present AVF_{Util} based on both FI and ACE analysis for all considered GPU models. The AVF_{Util} is mainly influenced by the software logic masking and the different ISAs of the GPU chips. Therefore, it is hard to observe a clear trend or correlation in its value between NVIDIA and AMD architectures. However, it is interesting to note that the three NVIDIA chips, which implement similar native ISAs and use the same programming model, have very similar AVF_{Util} for each benchmark. Apart from the different LM and RF size, which in turn influence the number of vulnerable resources leading to different vulnerabilities (Fig. 6), the NVIDIA GPUs employ different wavefront scheduling mechanisms (warps in NVIDIA/CUDA terminology). Each SIMD unit can accommodate a different number of wavefronts. Changing the number of resident wavefronts changes the scheduling process. This leads to a variation of the vulnerability timing windows for both registers and memory words. An increment of the time a wavefront has to wait before being scheduled leads to a longer exposure of a critical resource to a fault. Moreover, the number of wavefronts that a single CU can concurrently execute is another factor that influences the wavefront scheduling. The CUs of NVIDIA Quadro FX 5600 and NVIDIA Quadro FX 5800 can schedule a single wavefront at a time while the CUs of NVIDIA GeForce GTX 480 process two wavefronts in parallel. This difference significantly influences the ACE analysis results for the AVF_{Util} of RF (Fig. 4.A), which measures reliability on the basis of the vulnerable timing windows of the used memory elements. This does not apply to LM since it is shared among all wavefronts of the same CU. In particular, for some benchmarks, the two NVIDIA Quadro chips show very close values while the NVIDIA GeForce chip features a different value in benchmarks gaussian, histogram, kmeans and transpose.

Fig. 3 and Fig. 4 also allow us to compare AVF estimations obtained with FI and ACE analysis. Such a comparison must consider two main aspects: the measurement accuracy and the time required to perform the analysis. Regarding the accuracy, it is well-known from the literature that the error margin and the confidence interval of statistical fault injection are determined by the number of injected fault. Differently form FI, the accuracy of the ACE analysis cannot be quantified even if it is well known that it delivers pessimistic evaluations. This trend is confirmed for RF, where FI estimates lower AVF compared to ACE analysis (Fig. 3.A and Fig. 4.A). The error is strongly benchmark dependent. In particular, in our ACE analysis, to be very fast, we do not consider the program logical masking of faults. Fig. 3.B shows that ACE analysis AVF overestimation is lower for LM than for RF. This can be explained since RF and LM are used in a different way by the work-items of an application. LM is significantly slower than RF. For this reason, LM is mainly used to move data from/to RF, where logic and arithmetic operations take place. Consequently, logic masking effects not detected by ACE analysis are more relevant in RF than in LM. The only exceptions to this trend are histogram and backprop. For these two benchmarks, the ACE analysis introduces a higher error. This can be explained looking at the AVF_{Util} (Fig. 4). Their AVF_{Util} significantly changes depending on the evaluation method (FI or ACE analysis). Although the error between AVF_{Util} based on FI and AVF_{Util} based on ACE analysis is higher for backprop than histogram, histogram occupies more intensely the local memory compared to backprop. Thus, even if histogram features the second highest difference in LM AVF_{Util} (for different methods of evaluation), its high local memory occupancy leads to the highest difference in AVF depending on the evaluation method. Among the different benchmarks, backprop, is the one presenting the highest AVF overestimation between FI and ACE analysis for both RF and LM, respectively 3.3 and 1.1 times higher (Fig. 3). On average, the overestimation of the AVF and AVF_{Util} made by ACE analysis with respect to FI is respectively 95% and 80% for RF, while 15.9% and 17.4% for LM. It is also interesting to remark that, for some combinations of benchmarks and architectures, we observe that ACE analysis slightly underestimates AVF. This applies to HD Radeon for dwtHaar1D (0.35 p.u. – percentile units), histogram (0.38 p.u.) and reduction (0.93 p.u.) and to Quadro FX 5800 for reduction (0.31 p.u.). Finally, in case of the scan benchmark we observe a singularity: ACE analysis underestimates the AVF_{Util} (88.9% for FI against 85.2% of ACE analysis). Nevertheless, this difference is very close to the 2.88% error margin of fault injection.

In terms of simulation time, the single-run ACE analysis offers significantly better performance compared to FI. Table 2

quantifies this benefit comparing the simulation time of ACE analysis with the number of fault Injections Per Hour (IPH) that we were able to simulate for each benchmark employing both GUFI and SIFI. However, it is important to remember that this benefit must be traded-off with the reduced accuracy delivered by ACE analysis and with the capability of FI to precisely quantify the error margin of the computed metrics. Nevertheless, looking at the results provided in Fig. 3 and Fig. 4, it is clear that, despite its lower accuracy, ACE analysis is very efficient in providing a rough idea about the vulnerability of a hardware component or the differences between benchmarks in a very short simulation time. Overall, from our analysis we can conclude that ACE analysis represents a good characterization technique for LM, since it combines good accuracy and low computation time, whereas it is less suitable for RF given the higher inaccuracy. In the remaining of this section discussions will focus on results obtained resorting to fault-injection experiments.

Table 2. Simulation time required to perform the reliability analysis

Benchmark	**SIFI**		**GUFI**	
	ACE time (s)	**IPH**	**ACE time (s)**	**IPH**
backprop	3	1200	13	277
dwt	9	400	1	3600
gaussian	29	124	37	97
histogram	173	21	44	82
kmeans	24	150	90	40
matrixMul	21	171	20	180
reduction	4	900	4	900
scan	5	720	2	1800
transpose	2	1800	6	600
vectoradd	39	92	5	720
AVERAGE	**30.9**	**557.8**	**22.2**	**829.6**

Fig. 5 combines the AVF of the different hardware structures with the raw bit soft-error rate of the technology (λ) used to build the different chips (see Table 1). The result is the global FIT_{GPU}. The figure breaks down the contribution of RF and LM to the global FIT_{GPU}. Interestingly, the contribution of LM is significantly lower than the one of RF for most benchmarks in which this memory array is used. Understanding the motivations for the variation of the FIT rate is not simple since it depends on the fabrication technology, the size of the hardware structures and their vulnerability for each benchmark. In general, the combination of the technology node and the size of the structure seems to be the predominant factors. In our experimental setup, the size of RF is bigger than the one of LM and we observe a higher number of failures caused by faults in RF for all benchmarks and architectures except for histogram. In the histogram benchmark, which has an intense use of LM, the LM AVF is much higher than the one of RF and represents the main contribution to the FIT. In histogram, LM is used as a read-only memory, so all the Util resources are always ACE. Moreover, the GPU architectures whose hardware components are larger have the potential to better exploit their intrinsic parallelism, since the number of parallel work-items is also influenced by the availability of resources for the execute kernels. Executing more work-items concurrently increases the number of potentially vulnerable resources. More specifically, this trend can be evicted from Fig. 6 for NVIDIA architectures. However, Radeon 7970, featuring the largest components, shows an opposite behavior in some cases, highlighting the influence of compilers on vulnerable resources.

The bigger size of a GPU hardware component naturally makes it more vulnerable to soft-errors. However, it increases the execution parallelism and thus improves performance. Therefore, as discussed in Section II, to combine the reliability evaluation with the performance profile of each benchmark and GPU chip we analyzed the EPF (Fig. 7) and the IPF (Fig. 8) metrics because FIT alone (Fig. 5) does not take into account the amount of work carried out by the GPUs before a failure arises. EPF incorporates the execution time and FIT for a program, while IPF also includes information about the instruction throughput of GPUs when executing an application.

Table 3 shows for each benchmark (rows) and architecture (cols) the execution time (cycles) as well as the number of executed instructions required to compute EPF and IPF.

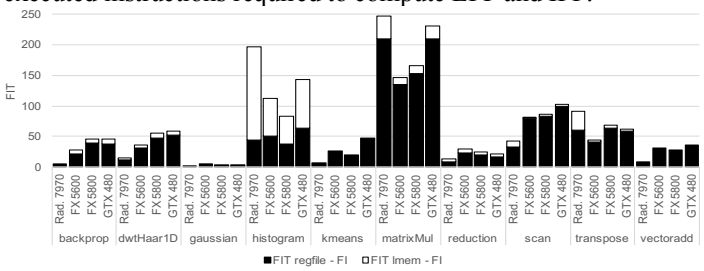

Fig. 5. Breakdown of Failures in Time (FIT) rate using the AVF measurements from Fault Injection.

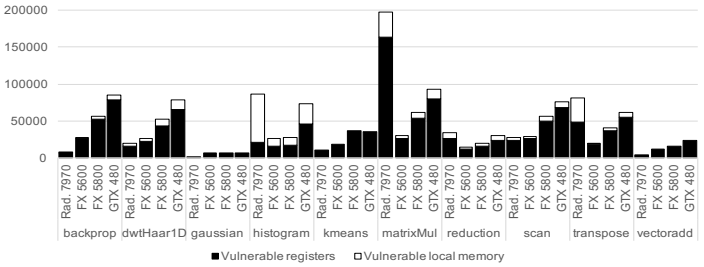

Fig. 6. Vulnerable resources in bits.

The IPF for a particular benchmark is proportional to the EPF and to the instruction throughput. Since this throughput strongly depends on the target execution device, to fairly compare different GPU architectures we must look at the EPF instead of IPF. The IPF is instead useful for evaluating the reliability of different programs on the same GPU chip. On the one hand, the EPF metric is useful to the architects who can quantify the effectiveness of a hardware-based error protection technique which can be applied to their designs (if needed) along with a performance cost at the early design stages. Larger EPF numbers show a larger number of executions before a failure and different protection mechanisms can deliver different improvements in the FIT rates and can also have different impact on performance. Combining performance and reliability measurements in the EPF metric delivers a broader view for decision-making. This could be for instance important when evaluating real-time applications that are not continuously executed, but they are scheduled once every time period. On the other hand, IPF is useful to the programmers who want to quantify the effectiveness of software redundancy-based protection techniques which can be applied to their programs running on the same architecture, thereby correlating the error resilience of their applications at a performance cost. IPF summarizes both the performance cost and the resilience improvement.

In Fig. 7, using the EPF to compare the different architectures, we can identify that the HD Radeon is in general the best choice for the selected benchmarks with the exception of histogram, scan and transpose. Using the IPF to compare different benchmarks on

the same architecture (Fig. 8), we can instead notice that for HD Radeon, backprop and gaussian have higher IPF than the other benchmarks while gaussian has the higher IPF when executed on Quadro FX5600, Quadro FX 5800 and GTX 480. Metrics that combine reliability and performance have also the potential to help comparing CPUs and GPUs. In particular, while the IPF is a raw throughput of work (instructions per failure occurrence) the EPF is a complete execution rate per failure occurrence, which is very useful if one wants to compare different processing elements like CPUs and GPUs [18].

Table 3. Execution time and instructions

Benchmark		HD Radeon 7970	Quadro FX 5600	Quadro FX 5800	GeForce GTX 480
	Freq (MHz)	925	337.5	325	700
backprop	cycles/	94376/	423594	369855	206834
	inst.	2108160	10312032		
dwt	cycles	41072	44998	39412	25859
	inst.	1839075	1180042		
gaussian	cycles	5862543	561060	555732	541687
	inst.	7308189	5488224		
histogram	cycles	3198537	1031394	1029746	885491
	inst.	20029440	21784328		
kmeans	cycles	913526	1278216	1267144	1397604
	inst.	31930960	35984844		
matrixMul	cycles	269591	439591	400594	299346
	inst.	10924032	15007744		
reduction	cycles	27836	47377	47086	27231
	inst.	312736	854719		
scan	cycles	123763	18707	16572	19721
	inst.	3025801	468720		
transpose	cycles	50862	98911	82821	49942
	inst.	733184	2818048		
vectoradd	cycles	31687	29219	21603	30225
	inst.	1523712	638976		

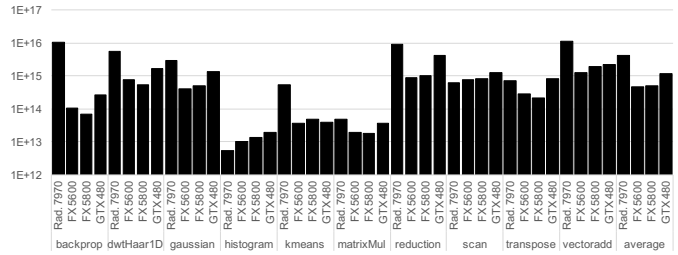

Fig. 7. Executions per Failure (EPF).

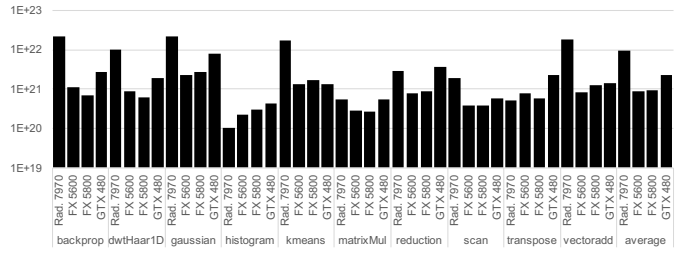

Fig. 8. Instructions per Failure.

IV. CONCLUSIONS

We have presented a multi-faceted comprehensive reliability assessment framework for state-of-the-art AMD and NVIDIA GPUs. Reliability measurements have been performed using both fault injection and ACE analysis to reveal the differences between the two approaches. We used 10 benchmarks to compare the vulnerability of the AMD/OpenCL versions and the NVIDIA/CUDA versions. We also proposed two combined performance/vulnerability metrics (EPF and IPF) that report the throughput of complete executions per failure or the throughput of individual instructions per failure. These metrics provide a wider picture of the GPU quality of execution and can be employed to compare different GPU chips for the same application or different programs on the same GPU chip. The proposed framework can be flexibly to jointly assess reliability and performance of further GPU configurations, any OpenCL and CUDA workload and, of course, to assist designers in making decisions for hardware- based or software-based error protections techniques in GPUs.

ACKNOWLEDGMENT

This research has been supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.

REFERENCES

[1] G. H. Asadi et al. "Balancing Performance and Reliability in the Memory Hierarchy," *ISPASS 2005*, pp. 269-279

[2] P. Rech et al. "Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability," DSN 2014, pp. 455-466.

[3] P. Rech *et al.*, "Neutron-Induced Soft Errors in Graphic Processing Units," *REDW 2012,* pp. 1-6.

[4] I. S. Haque et al. "Hard Data on Soft Errors: A Large-Scale Assessment of Real-World Error Rates in GPGPU," *CCGrid 2012*, pp. 691-696.

[5] N. Farazmand et al. "Statistical fault injection-based AVF analysis of a GPU architecture," *SELSE, 2012*.

[6] B. Fang et al. "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," *ISPASS 2014*, pp. 221-230.

[7] S. Tselonis et al. "GUFI: A framework for GPUs reliability assessment," *ISPASS 2016*, pp. 90-100.

[8] S. S. Mukherjee et al. "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," *MICRO-36,* pp. 29–40.

[9] A. Biswas et al. "Computing architectural vulnerability factors for address-based structures," *ISCA'05*, pp. 543.

[10] A. Chatzidimitriou et al. "RT Level vs. Microarchitecture Level Reliability Assessment: Case Study on ARM Cortex-A9 CPU", *DSN 2017.*

[11] A. Vallero et al. "Microarchitecture Level Reliability Comparison of Modern GPU Designs: First Findings", *ISPASS 2017*.

[12] R. Leveugle et al. " *Design, Automation & Test in Europe Conference & Exhibition. DATE 2009*, pp. 502-506.

[13] R. Ubal et al. "Multi2Sim: A simulation framework for CPU-GPU computing," *PACT 2012*, pp. 335-344.

[14] N. J. Wang et al., "Examining ACE analysis reliability estimates using fault-injection," *ISCA 2007,* pp. 460-469.

[15] A. Vallero et al., "SIFI: AMD Southern Island GPU Microarchitectural Level Fault Injector". *IOLTS 2017.*

[16] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," *IISWC 2009*, pp. 44-54.

[17] E. Ibe et al. "Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527-1538, July 2010.

[18] A. Chatzidimitriou et al. "Performance-aware reliability assessment of heterogeneous chips," *IEEE 35th VLSI Test Symposium 2017. VTS 2017*, Las Vegas, NV, USA, pp. 1-6.

[19] A. Bakhoda et al. "Analyzing CUDA workloads using a detailed GPU simulator," *ISPASS 2009*, pp. 163-174.

[20] AMD accelerated parallel processing opencl optimization guide available. [Online]. Available: http://amd- dev.wpengine.netdna-cdn.com/wordpress/media/ 2013/12/AMD OpenCL Programming Optimization Guide.pdf

[21] S. K. S. Hari et al., "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," *ISPASS 2017*, pp. 249–258.

[22] J.Tan et al. "Analyzing soft-error vulnerability on GPGPU microarchitecture," *IISWC 2011*, pp. 226-23.