SIFI: AMD southern islands GPU microarchitectural level fault injector

SIFI: AMD Southern Islands GPU Microarchitectural Level Fault Injector

Alessandro Vallero*, Dimitris Gizopoulos†, and Stefano Di Carlo*
*Politecnico di Torino, Control and computer engineering department, 10129, Torino, Italy.
Email: stefano.dicarlo,alessandro.vallero@polito.it
†University of Athens, Greece Email: dgizop@di.uoa.gr

***Abstract*—General Purpose computing on Graphics Processing Unit offers a remarkable speedup for data parallel workloads, leveraging GPUs computational power. However, differently from graphic computing, it requires highly reliable operation in several application domains. In this paper we present SIFI a reliability evaluation framework for soft-errors on AMD GPUs built on top of Multi2Sim, a micro-architectural level simulator. SIFI is capable of computing different reliability metrics by means of two different techniques: fault injection and ACE analysis. Experiments performed on a set of 14 GPGPU applications targeting the AMD Southern Islands GPU architecture show the capability of the tool and the potential of its use to support decisions about the best architectural parameters for a given application.**

I. Introduction

Recent years have witnessed an increase of computational power demand in several application domains. General Purpose computing on Graphics Processing Units (GPGPU) has gained a primary role in the delivery of high computational power leveraging the inherent high parallel architecture of GPUs to accelerate complex tasks. In this scenario, GPUs are no longer employed just for graphics. They have increasingly found application in areas where reliability is a primary concern (i.e., advanced driver assistance systems, aviation, medicine, super computing, etc.). This trend is however threatened by the technology shrinking, which has a detrimental effect on the susceptibility to faults for new devices (especially for large storage arrays) [1]. Characterization of the reliability of GPGPU systems is therefore becoming a mandatory task.

One of the main open challenges in evaluating the reliability of GPGPU systems is the development of fast and accurate reliability assessment tools able to properly trade-off simulation time and accuracy and providing information able to guide the system designers in the choice of proper architectural parameters and error protection mechanisms to achieve the target reliability and performance requirements.

Tan et al. proposed [2] GPGPU-SODA, an Architectural Correct Execution (ACE) analysis evaluation framework for NVIDIA GPUs. Being based on the high level PTX assembly language it suffers from high inaccuracy. Fang et al. proposed GPU-Quin, a tool for evaluating the soft-error resilience of applications in NVIDIA GPU chips [3]. GPU-Quin targets transient faults in the functional units of the GPU processor. One of the main limitations of this tool is that it performs injection at the level of the assembly code instructions. It therefore misses the details of the micro-architecture of the GPU and their effect on the fault propagation and masking. Hari et al. proposed SASSIFI, a fault injection tool for NVIDIA chips working at the SASS assembly level [4]. Compared to GPU-Quin, SASSIFI also works performing injections on the assembly instructions but introduces a set of improvements to control predicate and condition code registers and to improve the performance of the injection process. Recently, Tselonis and Gizopoulos proposed a fault injection reliability analysis framework based on a micro-architectural model of the NVIDIA GPUs able to analyze the effect of soft-errors in these devices [5]. Being based on a very detailed micro-architectural model, it allows for improved capability of modeling faults in the real hardware, considering also large memory arrays. This framework represents a valuable instrument for GPGPU system designers as reported in [6] where it was exploited for the reliability analysis of complex heterogeneous systems. Similar tools able to analyze systems based on AMD GPU chips that together with NVIDIA cover almost the totality of the GPU market share are still not mature and available. Reliability studies using fault injection to analyze systems based on the old AMD Evergreen GPU architecture were presented in [7] and [8]. However, to the best of our knowledge, no tools for the analysis of the AMD Southern Islands architecture have been presented in the literature.

This paper presents SIFI (Southern Islands Fault Injector), a framework for the reliability analysis of systems based on the AMD Southern Islands GPU architecture in presence of soft-errors. Using SIFI, reliability can be assessed not only by means of fault injection but also using very fast ACE analysis [9]. In both cases soft-errors in the main memory arrays of the GPU are the considered fault model. In particular, SIFI is currently able to analyze the effect of soft-errors in the vector register file, scalar register file and local memory of the GPU. 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 ECC, these can incur performance and energy overheads and hence may not be enabled by users in selected applications, thus motivating their consideration in SIFI [3].

SIFI implements a set of techniques to reduce simulation time of fault injection campaigns, thus allowing the analysis

of complex systems executing realistic software applications. Moreover, SIFI offers the possibility to perform reliability analysis just considering the portion of the hardware resources actually used by the running software. This decouples the reliability assessment from the occupancy of the target architectural hardware components, focusing on the analysis of the resiliency of the executed software to the injected faults. Since SIFI is built on top of the Multi2Sim micro-architectural simulator [10], it can be easily extended to other architectures supported by this simulator, including the AMD Evergreen architecture. The optimized simulation environment makes SIFI a valuable tool to assist designers when taking decisions on specific GPU architectural parameters. Several system configurations can be analyzed and compared to identify the best configuration given the application constraints.

In order to show the potential of SIFI , a reliability study has been performed considering 14 GPGPU applications executed on different AMD Southern Islands GPUs. Moreover, results obtained using a preliminary version of SIFI have been also presented in [11] where first findings of a study to compare reliability between NVIDIA and AMD chips have been reported.

Since SIFI is built on top of an open micro-architectural simulator, its source code is also open and available to the research community. Information to download the tool is reported at the end of the paper.

The remainder of this paper is organized as follows: Section II shortly overviews the Southern Islands GPU architecture while SIFI is fully described in Section III. The results of the use of SIFI on different benchmarks are presented in Section IV. Finally, Section V concludes the paper.

II. The Southern Islands AMD architecture

From the hardware standpoint, the Southern Islands AMD architecture consists of several compute units (CUs) managed by a scheduler and sharing a global memory (Fig.1). Each CU is composed of a front-end that fetches instructions and dispatches them to the appropriate unit: a branch unit, a local data storage unit performing operations on local memory, a scalar unit executing scalar operations on scalar registers and several SIMD units. Each SIMD unit contains a vector ALU and the integer and floating point units operating on the vector register file in parallel. OpenCL is the programming model used by the Southern Islands AMD architecture [12].

In OpenCL, parallel portions of an application (kernels) are executed on the GPU as work-items. Work-items are grouped into work-groups. Communication among work-items is possible only for work-items belonging to the same work-group by means of the local memory. When a new kernel is executed a new ND-Range is created. The programmer specifies the number of work-groups and the number of work-items per work-group characterizing the ND-Range. Every time the GPU can execute a new work-group, the work-group is assigned to a CU. The number of work-groups a CU can concurrently execute ($\#wg$) depends on the maximum number of work-groups a CU can accommodate ($\#max\,wg$) and on

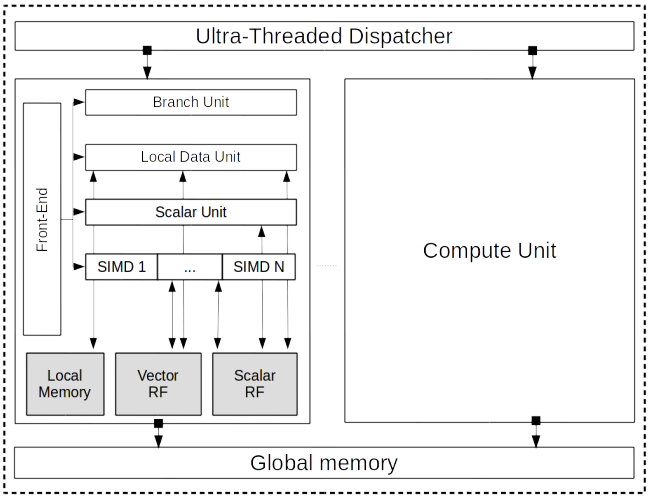

Fig. 1. Southern Islands AMD architecture

the amount of resources required by a single work-group. More specifically, each work-group requires a certain amount of resources: $\#v_{wg}$ vector registers, $\#s_{wg}$ scalar registers and $\#lm_{wg}$ local memory cells. These resources are taken from the pool of resources available in the CU: $\#V_{RF}$ vector registers, $\#S_{RF}$ scalar registers and $\#LM$ local memory cells. Interested readers may refer to [13] for details on how $\#wg$, $\#s_{wg}$, and $\#lm_{wg}$ are computed.

In order to be executed in parallel, work-items of the same work-group are grouped into wavefronts. The number of work-items per wavefront depends on the parallelism of the SIMD Units of the chip.

III. SIFI architecture and functionalities

SIFI is built on top of Multi2Sim v. 4.2, a micro-architectural simulator for heterogeneous systems accurately modeling the micro architecture of the AMD Southern Islands GPUs [10] [14]. The analysis carried out by SIFI focuses on soft-errors in the main memory arrays of the GPU (i.e., the vector register file, the scalar register file and the local memory). These errors, mainly caused by the effect of radiations, thermal cycling transistor variability and erratic fluctuations of voltage, are very relevant for large memory arrays such as the one considered by SIFI. Currently, the Single Event Upset (SEU) is the considered fault model, i.e., a bit-flip of a memory element. Analysis of multiple bit upsets can however be easily implemented.

SIFI measures the reliability of a GPU based system by computing the Architectural Vulnerability Factor (AVF) of its hardware structures [9]. The AVF quantifies the probability of a soft-error to manifest as a failure of the system jointly considering masking properties of the hardware architecture as well as of the executed software. One of the main drawbacks of the AVF is that it is influenced by the actual occupancy of the hardware structures. To understand the contribution of the instruction flow on the error masking, SIFI is able to compute

the AVF Util metric introduce by Farazmand et al. in [7]. The AVF Util is the probability that a soft-error in a *used* hardware structure causes a system failure. The relation between AVF and AVF Util can be expressed as:

$$AVF = AVF_{Util} \times Occupancy \tag{1}$$

The AVF Util is different from high level software metrics such as the Program Vulnerability Factor (PVF) defined in [15]. PVF is a pure software vulnerability metric defined to be micro-architecture independent, while AVF Util still takes into account the micro-architecture of the device.

Once the AVF is estimated for each GPU hardware structure, the Failure In Time (FIT) rate of the system (λ_S) can be computed combining size and vulnerability of every hardware structure of the GPU:

$$\lambda_S = \sum_{i \in \{vRF, sRF, LM\}} AVF_i \times \lambda \times \#bit_i \tag{2}$$

where $\#bit_i$ is the number of memory elements of the hardware structure *i* and λ is the raw error rate per bit of the target technology node.

Since the FIT rate is a pure reliability metric and does not provide any information about the system performance, SIFI is also able to compute a new reliability metric named *executions per failure* (EPF). EPF is the number of times an application must be executed before observing a system failure. It is computed as:

$$EPF = EIT/\lambda_S \tag{3}$$

where EIT (Executions in Time) is the number of executions of an application in 10^9 hours of device operation. The EPF enables to jointly analyze performance and reliability into a single metric.

SIFI can compute the presented metrics using different simulation engines described in the following subsections.

A. Fault injection engine

The fault injection (FI) engine is the most accurate simulation engine available in SIFI. It performs reliability analysis by simulating the occurrence of faults in the GPU hardware structures considering a statistically significant number of program executions (one fault per execution) [16]. The impact of a fault on the system is evaluated by comparing the output of the computation with the one of a golden execution. At a high-level, the impact of the fault is classified as masked or non-masked. Fine grained classification of non-masked faults into Silent Data Corruption (SDC) and Detectable Unrecoverable Error (DUE) is also possible.

Since FI is a computational intensive task, SIFI is designed to speedup the FI campaign trying to reduce the required number of simulations without loosing accuracy. A FI campaign consists of several steps. At first, the application is profiled in order to identify the time intervals in which the GPU is active and to collect information about the executed kernels. The faults to be injected are then randomly generated and another simulation is run to profile whether these faults affect at least one hardware structure assigned to a work-groups. In case a fault hits a non-assigned hardware structure, it is marked as masked without performing any simulation. Otherwise, it is marked as *Util*. Eventually, all faults marked as *Util* are simulated and classified. Using the results of FI simulations the AVF and AVF Util of an hardware structure can be computed as:

$$AVF = \frac{\#\ inj_{not-masked}}{\#\ inj.} \tag{4}$$

$$AVF_{Util} = \frac{\#\ util - inj_{not-masked}}{\#\ util - inj.} \tag{5}$$

The speedup obtained by skipping *non-Util* simulations depends on the application and mainly on the occupancy of the hardware structures. It can be computed as:

$$S = \#\ inj./\#\ util - inj \tag{6}$$

B. ACE analysis engine

Unlike FI, the ACE analysis engine requires just a single simulation of the application to perform AVF estimations. It is therefore a very fast analysis that however has reduced accuracy with respect to FI. SIFI ACE analysis engine is based on the techniques presented in [9] and [17] for CPU memory arrays.

Let us consider the computation of the AVF for the vector register file (a similar procedure can be applied to the other hardware structures). The ACE analysis is based on the principle that not all registers continuously contribute to the computation. Therefore, the AVF of the hardware structure can be estimated by determining which registers affect the final system output (ACE registers) and which do not (un-ACE registers).

When performing ACE analysis, each kernel is analyzed separately and then results are recombined together. For each kernel, the amount of registers assigned to each work-group ($\#v_{wg}$) is first computed (see Section II). All registers not assigned to any work-group are classified as *idle* and directly 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. In all other cases it is marked as ACE. To reduce the complexity of the computation and to implement a very fast reliability analysis workflow, SIFI does not implement more sophisticated techniques, which also take into account dead data (i.e., data not contributing to the output results) and logic masking (i.e., logical and arithmetic operations resulting in masked results).

The ACE factor of each work-group, i.e., the work-group average number of ACE registers per clock cycle, can be computed as:

$$ACE_{wg} = \sum_{i}^{\#v_{wg}} \frac{ACE_{clk-vreg-i}}{wg_{clk}} \tag{7}$$

where wg_{clk} is the number of clock cycles required to execute the work-group and $ACE_{clk-vreg-i}$ is the number of clock cycles in which the register i is classified as ACE.

The ACE factors of each work-group can be combined together to compute the ACE factor of the CU (ACE_{CU}). To perform this computation, the time window of every work-group executed by the CUs is analyzed to build a time diagram of the execution. Fig. 2 is an example of timing diagram for a single CU assigned to 4 work-groups and able to execute 2 work-groups concurrently.

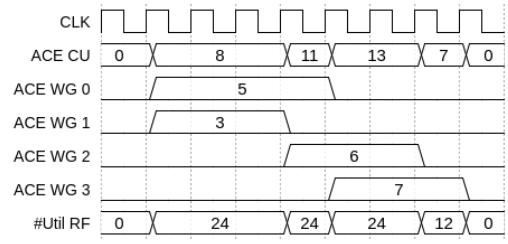

Fig. 2. An example of ACE_{CU} timing diagram for a single CU.

The ACE factor of the CU at clock cycle i ($ACE_{CU}(i)$) is computed by summing the ACE_{wg} of its active work-groups on that clock cycle (Fig.2). Finally, the AVF of the entire vector register file is computed as:

$$AVF = \frac{\sum_{j}^{\#CU} \sum_{i}^{\#kclk} \frac{ACE_{CU_j}(j)}{\#V_{RF}}}{\#kclk} \quad (8)$$

where $\#kclk$ is the number of clock cycles required to execute the GPU kernel, $\#CU$ is the number of available compute units and $\#V_{RF}$ is the number of registers per compute unit. The computation of the AVF takes into account the ratio between the number of ACE registers and the total number of registers available in the vector register files of the GPU.

In the example of Fig.2, considering $\#V_{RF} = 32$ and $\#kclk = 7$, the AVF is equal to:

$$AVF = \frac{\frac{8+8+8+11+13+13+7}{32}}{7} = 0.3 \quad (9)$$

Similarly to the AVF, by considering the average number of used vector registers of the active work-groups of each CU ($\#util\ RF_j$ in Fig.2) instead of the total available registers ($\#V_{RF}$), the AVF Util can be computed as:

$$AVF_{Util} = \frac{\sum_{j}^{\#CU} \sum_{i}^{\#kclk} \frac{ACE_{CU_j}(j)}{\#util_{RF_j}(i)}}{\#kclk} \quad (10)$$

In the example of Fig.2, if $\#v_{wg} = 12$ then the AVF Util can be computed as:

$$AVF_{Util} = \frac{\frac{8}{24} + \frac{8}{24} + \frac{8}{24} + \frac{11}{24} + \frac{13}{24} + \frac{13}{24} + \frac{7}{12}}{7} = 0.45 \quad (11)$$

IV. Experimental results

This section shows the capability of SIFI when analyzing the reliability of a set of benchmark systems.

A. Experimental Setup

For our evaluation we considered 14 software benchmarks with SIFI configured to resemble the architecture of the AMD HD Radeon 7970 GPU device[1]. A CU of this GPU consists of 4 SIMD Units. The scalar register file is composed of 2K 32-bit registers while the local memory size is 56KB. The scalar register file and the local memory are shared among all SIMD Units. Moreover, each SIMD Unit implements a vector register file of 56K 32-bit registers.

Starting from this basic configuration we also performed experiments considering CUs with different number of SIMD units, thus demonstrating the capability of SIFI to compare different GPU architectures.

The data workload of each benchmark has been chosen to maximize and stress the use of the CU memory arrays considered during the analysis. Since this significantly increases the simulation time when considering multiple CUs, following the approach proposed by Farazmand et al. in a similar GPU study [7], we scaled the analysis considering a single CU. Results have been then extended to the case of multiple CUs without loosing accuracy.

The benchmarks considered in our experiments have been selected from the AMD-APP-SDK benchmarks[2]: (1) Binary Search (BinS), (2) Bitonic Sort (BitS), (3) DCT, (4) DwtHaar1D (DWT), (5) FastWalshTransform (FWT), (6) FloydWarshall (FW), (7) Histogram (HIS), (8) MatrixMultiplication (MM), (9) MatrixTranspose (MT), (10) QuasiRandomSequence (QRS), (11) RecursiveGaussian (RG), (12) Reduction (RED), (13) SimpleConvolution (SC) and (14) URNG.

For each benchmark we used SIFI to compute the AVF and AVF Util of the target system resorting both to the FI and ACE analysis engines described in Section III. According to [16], for each fault injection campaign (i.e., for each benchmark and for each hardware structure), we applied statistical fault sampling injecting a number n of faults equal to:

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p \times (1-p))}} \quad (12)$$

where N is the population size[3], p is the estimated probability of a fault to generate a failure[4], e is the accepted error margin[5] and t is the cut-off point that defines the confidence level[6]. Considering the simulated benchmarks, and and the target GPU architecture, to characterize a hardware structure for a given benchmarks about 10K fault injections are required.

[1]Any chip belonging to the AMD Southern Islands family can be modeled

[2]AMD-APP-SDK v.2.7 available at: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-pro cessing-app-sdk/

[3]The size of the targeted memory array multiplied by the number of clock cycles of the application. When very large populations such as the ones considered in this paper this paper are considered, small differences have a minor influence on the sample size.

[4]since it is unknown, a typical value of 0.5 is used to maximize the sample size

[5]1% in our case

[6]95% in our case

B. Experimental results

Fig.3 reports the AVF of the vector register computed using both FI and ACE analysis. Results show how different software applications can significantly influence the AVF, thus confirming the need to carefully perform this type of analysis.

The benchmark with highest vulnerability is MM (21%), while some benchmarks are characterized by very low AVF (i.e., BinS, BitS, DWT, FW, RG, RED and URNG). As expected, ACE analysis provides a rough estimation of the AVF. In most of the cases, it overestimates more than two times the vulnerability. This difference can be attributed to the fact that the implemented ACE analysis does not take into account dead instructions and software logic masking. It is however worth to report that, from our simulations, we noticed that dead data in the selected benchmarks represent a negligible portion of the application (less than 0.5%). Therefore, the ACE analysis inaccuracy is probably mostly due to the effect of the software logic masking that is not taken into account in our implementation.

Interestingly, when considering the AVF of the local memory results obtained using the ACE analysis are quite accurate (Fig.4) and in general fall within the error margin of the estimations obtained using FI with the only exception of MT that is 8 percentile points higher. HIS has the highest local memory vulnerability (91%). This is due to the fact that it requires a large amount of memory basically used as a read-only buffer.

Finally, Fig. 5 reports the AVF for the scalar register file. It ranges between 0.2% (BinS) and 16% (MM) with an average value of 6.3%. ACE analysis estimation is close to the one obtained by FI just for DCT, while, in the other cases, the AVF is always overestimated (almost 2x compared to FI).

Looking at the results, it is not easy to identify one component with higher criticality (i.e., contribution to the global AVF of the system) across all benchmarks. This strongly depends on the application and on the use of the resources. This further motivate the need of tools such as SIFI able to quantify this contribution on a application base.

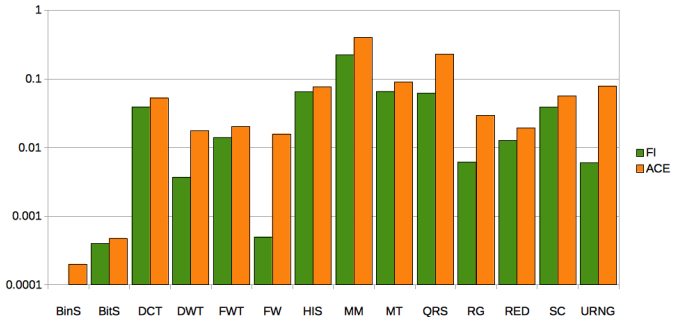

Fig. 3. Vector register file AVF (log scale) computed by FI and ACE analysis.

According to Section III, the AVF is influenced by two factors: the occupancy of the memory arrays and the AVF Util. SIFI allows us to analyze these contributions separately as reported in Fig. 6 for the vector register file. The results show

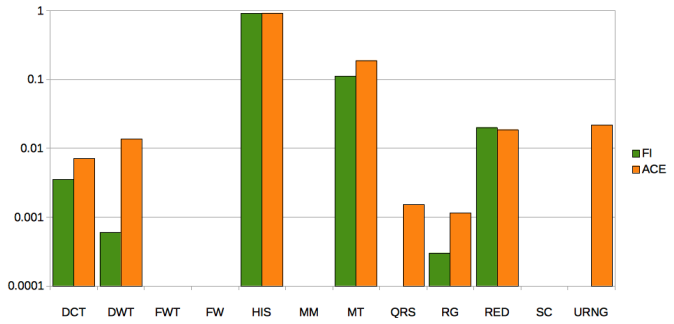

Fig. 4. Local memory AVF (log scale) computed by FI and ACE analysis. The AVF is reported just for benchmarks using local memory.

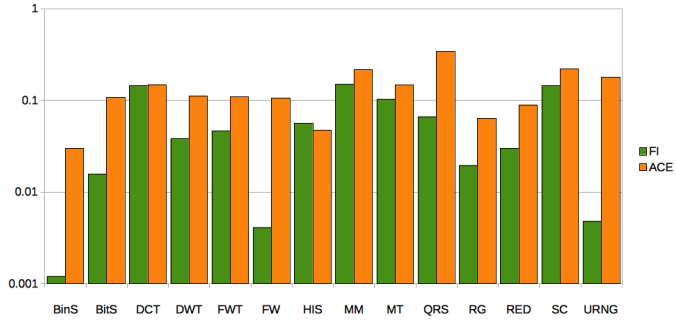

Fig. 5. Scalar register file AVF (log scale) computed by FI and ACE analysis.

that the occupancy is one of the most relevant contributions to the AVF. Let us consider results for HIS. The AVF Util of this benchmark is equal to 80.1%. This means that this application is potentially highly vulnerable to faults affecting active resources. However, this high vulnerability is compensated by a low occupancy of the register file that leads to a total AVF of only 6.7%. For benchmarks with higher occupancy (e.g., MM), the difference between AVF and AVF Util is reduced. Being able carefully analyze the relationship between AVF, AVF Util and Occupancy is an important instrument to carefully plan how to introduce fault-tolerance mechanisms in the system. Results similar to the one reported in Fig. 6 have been computed also for the local memory and for the scalar register file with similar trends. However, due to the limited space they have not been reported in the paper.

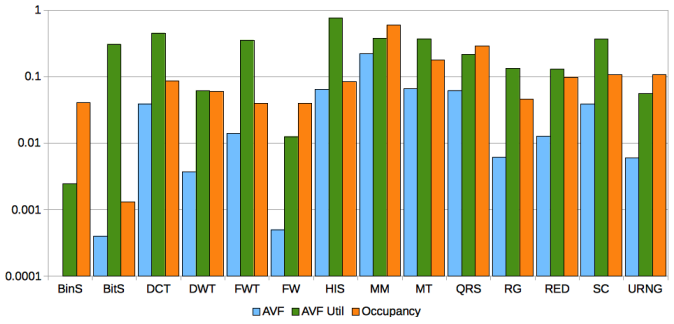

Fig. 6. The correlation between occupancy and AVF of the vector register file. The AVF Util is computed in order to decouple vulnerability and occupancy

One of the main benefits of SIFI is the possibility to evaluate the reliability of a system exploring different architectural parameters. To stress this capability we show how the number of SIMD units per CU affects the AVF of the vector register file (Fig.7), local memory (Fig.8) and scalar register file (Fig.9). Changing the number of SIMD units leads to AVF variations. More specifically, decreasing the number of SIMD units increases the AVF of the vector register file while decreasing the vulnerability of the local memory and the scalar register file. This is an interesting behavior that requires further investigation. However, some considerations can be drawn. Concerning the local memory and the scalar register file, AVF variations can be justified since increasing the number of SIMD units increments the required bandwidth and consequently the latency to main memory. This leads to a larger vulnerability time-window for a single memory element that in turns leads to an increment of the AVF. The trend for the vector register file is instead unexpected and requires further investigations.

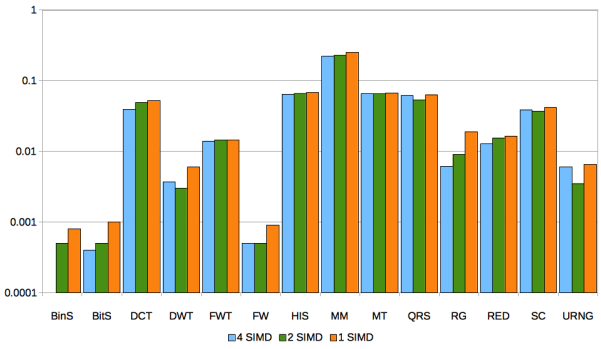

Fig. 7. Vulnerability comparison (log scale) of vector register file changing the number of SIMD Units per CU

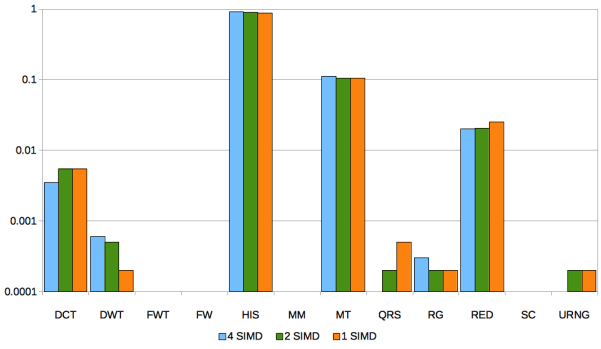

Fig. 8. Vulnerability comparison (log scale) of local memory changing the number of SIMD Units per CU

To conclude the proposed reliability analysis Fig.10 reports the FIT rate, the EIT and the EPF computed by SIFI for the analyzed systems. These metrics allow us to introduce the contribution of the technology and performance into the analysis. The FIT rate has been computed considering a raw failure rate per bit of λ = 1mFIT/bit.

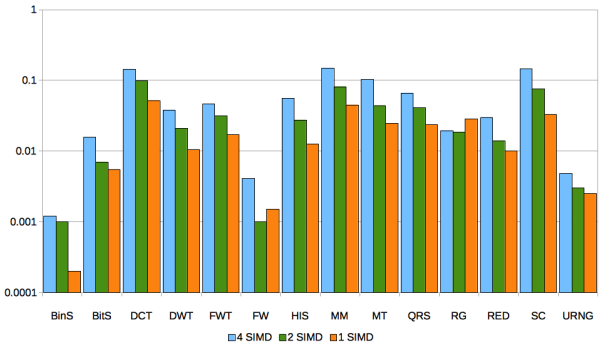

Fig. 9. Vulnerability comparison (log scale) of scalar register file changing the number of SIMD Units per CU

From the performance prospective (EIT), decreasing the number of SIMD units always translates into longer execution time. However in some cases, as BinS, HIS and RED, this overhead is negligible. Concerning reliability, the FIT is improved in all cases when the number of SIMD units is reduced, apart from QRS which has an opposite trend. Finally, thanks to this analysis it is possible to combine both performance and reliability, taking into account the EPF, that is the expected value of application executions before a failure manifests. In this case, both the EIT and FIT decrease for smaller numbers of SIMD units. As a consequence, the parameter that decreases faster dominates. On average, FIT prevails leading to higher EPF for a single SIMD unit and lower EPF in case 4 SIMD units. However the trend is not the same for all the analyzed benchmarks. In fact, BitS, DCT and MM show the opposite behavior. This strongly suggests that EPF should be evaluated separately for each application to avoid errors in the reliability assessment.

Finally, results illustrating the performance of SIFI as well as the capability of executing complex benchmarks in reasonable time are presented in Fig.11. In particular, the advantage introduced by the speedup techniques of SIFI fault injector can be appreciated. In fact, the time required by a fault injection campaign is often reduced by almost one order of magnitude. In addition, for each benchmark, the number of simulated GPU instructions is reported too.

V. Conclusion

In this paper we have presented SIFI, a full reliability assessment framework for systems based on the AMD Southern Islands GPU architecture. SIFI enables to compute reliability metrics using both accurate fault injection experiments or less accurate but very fast ACE analysis, allowing the reliability engineer to easily trade-off accuracy and simulation time. Both simulation engines are highly optimized in order to allow the analysis of realistic systems. We demonstrated the capability of SIFI by analyzing 14 OpenCL software applications executed on an AMD HD Radeon 7970 GPU device. Differences in the computed metrics demonstrate the value of carefully assessing the reliability based on the target application to carefully optimize the protection mechanisms to be applied. Since SIFI

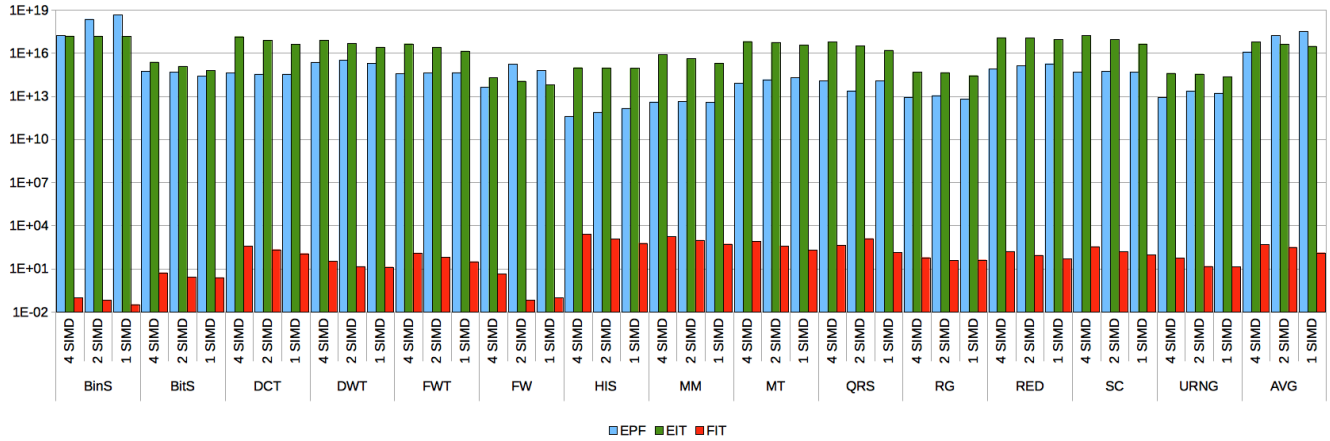

Fig. 10. Joint comparison of reliability and performance (log scale): the EPF changing the number of SIMD Units per CU.

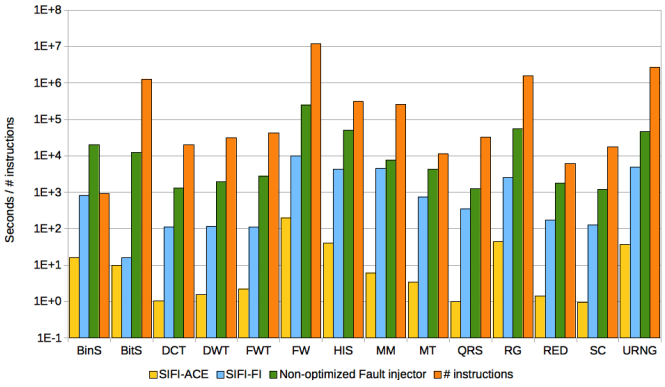

Fig. 11. SIFI timing performance (log scale). For each benchmark the figure reports the time required to estimate the vector register file AVF using ACE analysis, FI and non-optimized FI (10,000 injections using 8 cores) alongside the number of GPU instructions per simulation.

is built on top of an open micro-architectural simulator, it can be easily extended to other GPU architectures supported by this simulator.

ACKNOWLEDGMENTS

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

SOFTWARE AVAILABILITY AND REQUIREMENTS

SIFI runs under the Linux operating system. Its source code is open and available on-line at https://www.testgroup.polito.it/sifi/

REFERENCES

[1] P. Rech, L. L. Pilla, P. O. A. Navaux, and L. Carro, "Impact of gpus parallelism management on safety-critical and hpc applications reliability," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 455–466.

[2] J. Tan, Y. Yi, F. Shen, and X. Fu, "Modeling and characterizing {GPGPU} reliability in the presence of soft errors," *Parallel Computing*, vol. 39, no. 9, pp. 520 – 532, 2013.

[3] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 221–230.

[4] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258.

[5] S. Tselonis and D. Gizopoulos, "Gufi: A framework for gpus reliability assessment," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 90–100.

[6] A. Chatzidimitriou, M. Kaliorakis, S. Tselonis, and D. Gizopoulos, "Performance-aware reliability assessment of heterogeneous chips," in *Proceedings of the IEEE VLSI Test Symposium*, 2017.

[7] N. Farazmand, R. Ubal, and D. Kaeli, "Statistical fault injection-based avf analysis of a gpu architecture," in *IEEE workshop on silicon errors in logic*, 2012.

[8] R. Shah, M. Choi, and B. Jang, "Workload-dependent relative fault sensitivity and error contribution factor of gpu onchip memory structures," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2013, pp. 271–278.

[9] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE, 2003, pp. 29–40.

[10] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: a simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 335–344.

[11] A. Vallero, S. Di Carlo, S. Tselonis, and D. Gizopoulos, "Microarchitecture level reliability comparison of modern gpu designs: First findings," in *IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 129–130.

[12] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[13] 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

[14] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2sim: A simulation framework to evaluate multicore-multithreaded processors," in *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, Oct 2007, pp. 62–68.

[15] V. Sridharan and D. R. Kaeli, "Quantifying software vulnerability," in *Proceedings of the 2008 Workshop on Radiation Effects and Fault Tolerance in Nanometer Technologies*, ser. WREFT '08. New York, NY, USA: ACM, 2008, pp. 323–328.

[16] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 502–506.

[17] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ser. ISCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 532–543.