2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)

2022 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) | 978-1-6654-6958-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/PDP55904.2022.00025

Predicting the Soft Error Vulnerability of GPGPU Applications

Burak Topçu
Computer Engineering Department
Izmir Institute of Technology
Izmir, Turkey
Email:buraktopcu@iyte.edu.tr

Işıl Öz
Computer Engineering Department
Izmir Institute of Technology
Izmir, Turkey
Email:isiloz@iyte.edu.tr

***Abstract*—As Graphics Processing Units (GPUs) have evolved to deliver performance increases for general-purpose computations as well as graphics and multimedia applications, soft error reliability becomes an important concern. The soft error vulnerability of the applications is evaluated via fault injection experiments. Since performing fault injection takes impractical times to cover the fault locations in complex GPU hardware structures, prediction-based techniques have been proposed to evaluate the soft error vulnerability of General-Purpose GPU (GPGPU) programs based on the hardware performance characteristics.**

In this work, we propose ML-based prediction models for the soft error vulnerability evaluation of GPGPU programs. We consider both program characteristics and hardware performance metrics collected from either the simulation or the profiling tools. While we utilize regression models for the prediction of the masked fault rates, we build classification models to specify the vulnerability level of the programs based on their silent data corruption (SDC) and crash rates. Our prediction models achieve maximum prediction accuracy rates of 96.6%, 82.6%, and 87% for masked fault rates, SDCs, and crashes, respectively.

I. Introduction

Heterogeneous computing systems that bring together general-purpose multi-core processors (CPUs) and data-parallel graphic processing units (GPUs) offer high performance as well as energy efficiency in large-scale computing platforms. Due to their highly parallel computational power, GPU architectures have been largely utilized for general-purpose computations as well as graphics applications [1]. While the GPUs reduce the execution times significantly, they exhibit higher vulnerability to soft errors due to their complex structures. Hence, the soft error reliability becomes a critical concern for GPGPU applications.

To improve the reliability of the GPU architectures, various fault tolerance techniques like error correction codes, redundant multithreading have been employed [2], [3], [4]. Those hardware or software redundancy-based techniques induce additional cost and performance overhead. Therefore, evaluating the soft error vulnerability of the programs becomes quite important to make decisions about the fault tolerance techniques.

Fault injection is widely-used vulnerability evaluation technique based on controlled experiments which introduce faults into the system [5], [6]. This technique introduces faults in the hardware structures during the program execution, then observes the program execution to understand the effect of the injected fault. The fault injection yields SDC rates for the target execution. By utilizing SDC rates as a soft error vulnerability metric, one can decide whether the explicit fault tolerance techniques are essential for the program execution or we can count on the protection methods available in the target system.

While the fault injection is useful to quantify the vulnerability of the target execution, the large number of experiments may become impractical, especially for the long-running applications. Therefore, predicting the soft error vulnerability without performing fault injection experiments for each target program has become an attractive solution. In this work, we present a soft error vulnerability prediction study for GPGPU applications using machine learning. Our work aims to eliminate long fault injection times and predict the vulnerability of the programs by using performance metrics collected either from the simulation or the program profile. While there are several works based on machine learning models for CPU systems' reliability prediction [7], [8], [9], [10], [11], [12], the prediction approaches for GPU architectures are limited [13], [14]. PRISM [13] presents statistical models for predicting fault rates based on the instruction types of the GPU programs. The proposed regression models utilize linear regression and K-nearest neighbor algorithms, where SDC prediction does not perform well. Our work considers both program characteristics like instruction intensities and hardware performance metrics like SM throughput and occupancy. Moreover, we build regression models for predicting masked fault rates while we utilize a classification approach for SDC and crash conditions. Nie et al. [14] propose machine learning models for the prediction of GPU errors. The work does not focus on the soft error vulnerability, it aims to predict if an error occurs during the target program execution in a large-scale system with GPUs. The model includes both temporal and spatial features, like application-specific characteristics, temperature/power consumption, node location, and error frequency. Our work specifically targets to predict the soft error vulnerability of GPGPU programs by building machine learning models.

To the best of our knowledge, this is the first work to predict soft error vulnerability of GPGPU applications by utilizing

978-1-6654-6958-6/22/$31.00 ©2022 IEEE
DOI 10.1109/PDP55904.2022.00025

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

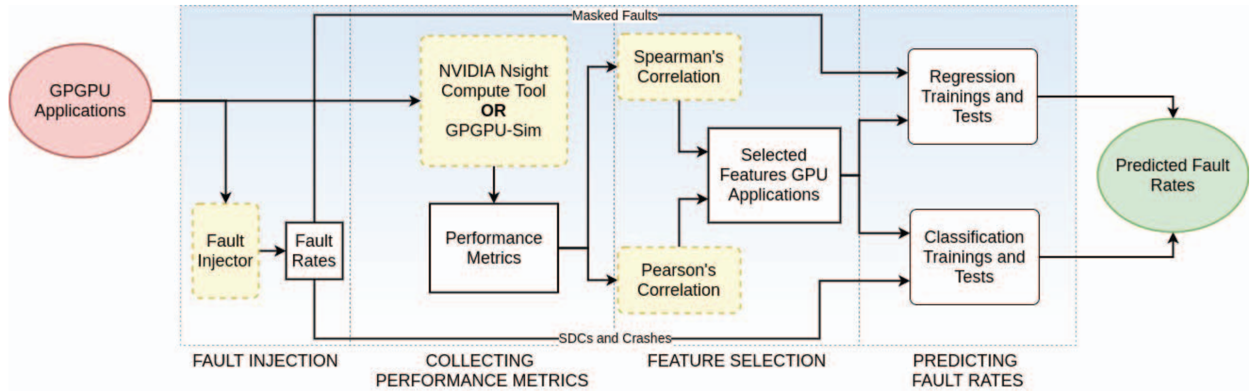

Fig. 1. General overview of our framework.

both regression and classification models based on program characteristics and hardware performance metrics. Our main contributions are as follows:

- We present ML-based soft error vulnerability prediction models for GPGPU applications. Our approach uses both program characteristics and hardware performance metrics. We collect the features from the simulator and the profiling tool, and build separate prediction models to compare both approaches.
- We utilize a fault injection tool that enables to perform regional fault injection experiments and obtain fault rates for kernel functions in a GPU code.
- We employ regression models to predict masked fault rates, while we use classification approach for predicting SDC and crash conditions, of which the rates are hard to predict.
- We obtain 96.59% prediction accuracy for our regression model with Gradient Boosting regression algorithm based on feature selection, and 87% and 80% accuracy for our 2-class and 3-class evaluations, respectively.

II. Background

A. GPU Architecture

Modern GPU architectures employ a single-instruction-multiple-thread (SIMT) execution in their Streaming Multiprocessor (SM) units, which contain many cores and high-bandwidth memory structure. While the cores inside the same SM can access the shared memory and L1 cache, all the cores can communicate via L2 cache structure. The global device memory maintains larger but relatively slower data access for all threads executing in the device. We choose the NVIDIA Pascal architecture [15] for our evaluation platform, where we execute our target programs both on a real hardware (Quadro P4000) and the simulation environment (GPGPU-Sim) with the specific configurations. The Pascal architecture has total 1792 CUDA cores. Since the architecture includes more SMs, it supports more active threads, warps, and thread blocks compared to prior GPU generations. We target the CUDA programs developed by CUDA programming model and compiled using NVIDIA CUDA compiler, nvcc [16].

B. Soft Error Vulnerability

In this paper, we consider soft errors that are transient errors causing bit flips in the register file [17]. We focus on single-bit errors since they are much more common. The most prevalent way to evaluate the soft error vulnerability of an application is to perform fault injection experiments. In a fault injection scenario, one bit of one register is flipped at a random time during the execution of the application, and the output result is examined to observe the effect of the fault. The outcome of the fault injection experiments can be classified into three categories: *1) Correct Execution (Masked)*: The faults have no effect on the output since the corrupted value is not used or overwritten in the remaining part of the program. *2) Silent Data Corruption (SDC)*: The program terminates but the program's output is not the expected output due to the corrupted data. *3) Crash*: The program fails by terminating with an error code.

III. Methodology

Our main focus is to predict fault rates by investigating the relationship between the hardware usage performance metrics of the GPU applications and the fault rates. In order to predict the occurrence of the faults, we utilize machine learning based regression and classification approaches. Figure 1 shows the general overview of our experimental framework, which consists of the following four main stages:

1) *Fault injection:* To get fault rates of the target GPGPU programs, we perform fault injection experiments by using a recently-proposed fault injector framework [18].
2) *Performance metric collection:* We both simulate and profile GPU applications by using GPGPU-Sim [19] and NVIDIA's Compute tool respectively and collect performance metrics from the target GPGPU programs.
3) *Feature selection:* Since there are many performance and hardware usage metrics that can degrade prediction performance, we select only the significant ones by applying Pearson and Spearman correlation methods to identify mostly correlated metrics for the faults.

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

4) *Fault rate prediction:* We predict the fault rates by using machine learning based regression and classification methods.

We explain the details of those steps in the following sections.

A. Fault Injection Framework

We utilize the fault injection tool that enables regional vulnerability analysis for the GPGPU programs [18]. The debugger-based fault injector targets specified kernel function execution as each fault injection point and enables us to evaluate SDC, crash, and masked fault rates for the target kernel functions. In this way, we can get the vulnerability values for each kernel function and obtain a set of data points for our prediction model. Figure 2 presents the fault rates for the kernel functions in our target applications.

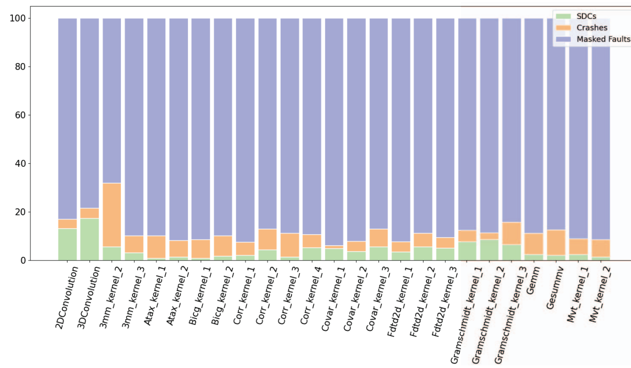

Fig. 2. SDC, crash and masked fault rates for the benchmark applications.

B. Performance Metric Collection

To characterize the GPGPU programs for our prediction evaluation, we collect a set of performance and hardware usage metrics. We extract those metrics by two prevalent methods including simulation and profiling. In this work, we perform both approaches and compare the results.

As the first approach, we execute our target programs in the GPGPU-Sim simulation environment [19]. The GPGPU-Sim (hereafter referred to as *the simulator*) enables to simulate the GPU applications by configuring the GPU hardware parameters such as bandwidth, DRAM, L1/L2 cache amounts, cores, and SMs. By utilizing the performance simulation model of the simulator, we simulate benchmark GPU applications and collect a set of metrics quantifying the execution.

As the other approach, we execute the programs using NVIDIA's Nsight Compute tool [20]. Nsight Compute tool (hereafter referred to as *the profiler*) profiles GPU applications at different levels like SASS and PTX level, and provides detailed hardware usage and performance metrics for the corresponding GPU program. Different from the simulator, the profiler requires real NVIDIA hardware.

To avoid the complexity in our dataset and increase the data quality for our fault prediction model, we collect only the metrics that can potentially represent the relationship between the benchmark applications and the GPU hardware. For instance, while the amount of load/store operations or streaming multiprocessor efficiency for an application is included due to their decisive characteristics, more specific metrics such as DRAM row utilization rate are eliminated. Table I and II present the metrics collected from the simulator and the profiler, respectively. Since the simulator and the profiler serve different purposes, it is not expected to collect exactly the same metrics from them. While the simulator mostly reports performance metrics, the profiler generates hardware usage and performance metrics statistically. However, some of the metrics such as IPC, the achieved occupancy in the SMs, and the total number of instructions executed for a kernel are common for both of them. Since they represent different executions, we consider the set of metrics separately, and build different prediction models for each set.

TABLE I
THE METRICS COLLECTED FROM THE SIMULATOR.

Performance Metric	Description
Load Instruction	# of load instructions
Store Instruction	# of store instructions
Param Mem Instruction	# of parameter memory instructions
Total Instruction	Total instructions for corresponding kernel
IPC	Instruction per cycle
Sim Rate	Simulation rate (total simulation per wall time)
Global Mem Read	Total global memory read
Global Mem Write	Total global memory write
Warp Occupancy	Average warp occupancy in the SMs
Control Flow Inst Intensity	# of control flow instructions per total instruction amount in ptx code
Data Mov Inst Intensity	# of data movement instructions per total instruction amount in ptx code
Float Point Inst Intensity	# of floating point instructions per total instruction amount in ptx code
Integer Arithmetic Inst Intensity	# of integer arithmetic instructions per total instruction amount in ptx code
Logic Inst Intensity	# of logical instructions per total instruction amount in ptx code
Load Inst Intensity	# of load instructions per total instruction amount in ptx code
Predicate Inst Intensity	# of predicate instructions per total instruction amount in ptx code

C. Feature Selection

To capture the significant metrics and potentially increase the prediction success rates of our prediction model, we investigate the correlations between the fault rates and the metrics. We use Spearman and Pearson methods to describe the correlation. Spearman correlation method describes the monotonic relationship between the features and the fault rates without considering if the relationship is linear or not. The results are scaled in the range [-1, 1], where the closeness to -1 and 1 describes a high relation between those two parameters. Spearman's rank correlation coefficient can be computed as follows:

$$\rho_s = 1 - \frac{\sum_{i=1}^{n} d_i^2}{n^3 - n}$$

where d is the difference between two rankings and n is the number of observations.

Pearson's correlation describes a linear relationship of the two inputs, and the results are scaled in between [-1, 1]

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

TABLE II
THE METRICS COLLECTED FROM THE PROFILER.

Performance Metric	Description
SOL SM	SM throughput
SOL MEM	Compute memory pipeline throughput
SOL L1 Tex Cache	L1 texture memory throughput
SOL L2 Cache	L2 cache throughput
SOL DRAM	GPU DRAM throughput
Duration	Total duration in msec
Elapsed Cycle	# of cycles where GPU is active
IPC	# of issued warp instructions per cycle
SM Busy	SM core instruction throughput calculated as percentage
Mem Throughput	# of byte accessed in DRAM (Gbyte/sec)
L1 Tex Hit Rate	# of sector hits per sector
L2 Hit Rate	Proportion of L2 sector lookups that hits
Mem Busy	Throughput of internal activity within caches and DRAM
Max Band	Throughput of interconnects between SMs, caches and DRAM
Active Warp Per Sch	# of active warps per scheduler
Warp Cyc Inst	Average # of cycles each warp is resident per instruction issued
Executed Inst	# of warp instructions executed
Reg Per Thread	Launched register per thread
Achieved Occupancy	Achieved occupancy in percentage
Achieved Active Warp	Achieved active warps per SM

as in Spearman's. The closeness to -1 or 1 describes that those two metrics are correlated, while closeness to 0 reveals the dissimilarity between those metrics. Pearson correlation coefficient can be computed as follows:

$$\rho_p = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

where E is the expectation, σ is the standard deviation, and μ is the mean.

Figure 3 and Figure 4 present the correlation results for the simulator and the profiler, respectively. The number of streaming multiprocessors (SMs) in GPUs, the number of cores in those SMs, the DRAM efficiency in terms of utilizing memory coalescing, the memory bandwidth, the hit/miss rates of L1/L2 caches, and the warp occupancy on those SMs are the metrics that can be used to characterize the relationship between a CUDA application and the GPU hardware. For example, *Active_Warp_Per_Sch* metric describes the active warp amount per scheduler. As demonstrated in Figure 4, it results in larger correlation values for Spearman method (0.24 for SDC, 0.19 for crash, 0.34 for masked) than for Pearson method (-0.17 for SDC, 0.18 for crash, -0.025 for masked). The amount of load/store instructions is crucial since the faults can be masked due to the store operations. The corresponding correlation results are -0.45 for load and -0.4 for store instructions with Spearman method. The warp occupancy, which describes the efficiency of the cores on the SMs, is another important feature in terms of describing the relationship between the faults and the metrics. For the soft errors injected into registers, the register amount per thread is another significant feature. Those registers can be used for indexing memory locations or temporal local registers, and the faults can be masked or affect the output depending on the register's mission in a program. While the Spearman correlation results between the number of registers per thread and faults are 0.072 for SDC, 0.32 for crash and -0.33 for masked, Pearson Correlation results are 0.18 for SDC, 0.35 for crash and -0.39 for masked. In addition, there are other selected features obtained by classifying instructions as data movement and control flow instructions, and scaling them according to their intensities. The intensities are calculated by classifying PTX instructions with the corresponding opcodes specified in [21]. As an example, *Predicate_Inst_Intensity* metric results in larger values for both Spearman (-0.38 for SDC, -0.16 for crash, 0.42 for masked) and Pearson Correlation (-0.38 for SDC, -0.035 for crash, 0.61 for masked) methods. The instruction intensity metrics in Table I also have a high correlation with the fault rates compared to the other metrics.

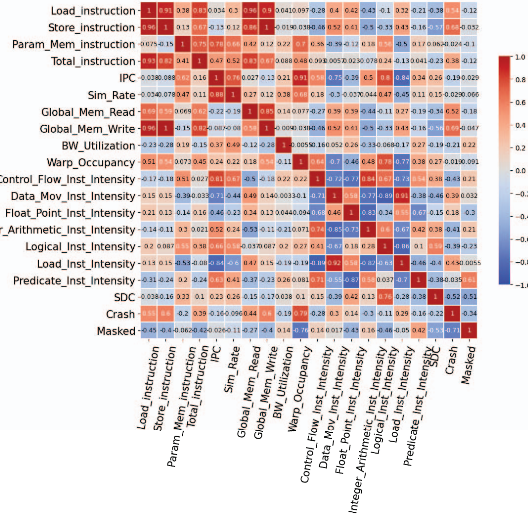

Fig. 3. The upper triangle shows Pearson correlation results, while the lower triangle shows the Spearman correlation results between the simulator metrics and the fault rates.

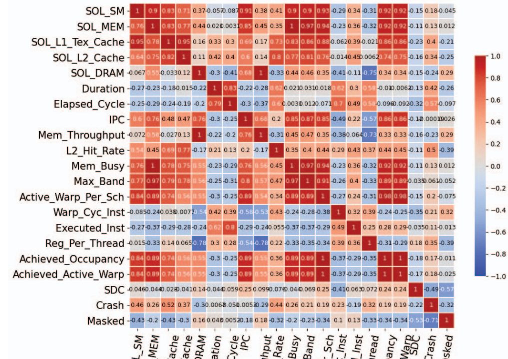

Fig. 4. The upper triangle shows Pearson correlation results, while the lower triangle shows the Spearman correlation results between the profiler metrics and the fault rates.

We determine the lowest correlation limit as 0.2 intuitively and select the features having the correlation values (shown

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

TABLE III
SELECTED FEATURES FROM THE COLLECTED METRICS.

For SDC	For Crash	For Masked
From the simulator		
Param Mem Instruction	Load Instruction	Load Instruction
IPC	Store Instruction	Store Instruction
Sim Rate	Param Mem Instruction	Total Instruction
Data Mov Inst Intensity	Total Instruction	Global Mem Read
Float Point Inst Intensity	Global Mem Read	Global Mem Write
Logical Inst Intensity	Global Mem Write	Warp Occupancy
Load Inst Intensity	BW Utilization	Float Point Inst Intensity
Predicate Inst Intensity	Warp Occupancy	Integer Arithmetic Inst Intensity
	Control Flow Inst Intensity	Logical Inst Intensity
	Data Mov Inst Intensity	Predicate Inst Intensity
	Integer Arithmetic Inst Intensity	
	Load Inst Intensity	
From the profiler		
SOL DRAM	SOL SM	SOL SM
IPC	SOL MEM	SOL MEM
Active Warp Per Sch	SOL L1 Tex Cache	SOL L1 Tex Cache
Warp Cyc Inst	SOL L2 Cache	SOL L2 Cache
Reg Per Thread	SOL DRAM	L2 Hit Rate
Achieved Occupancy	Mem Throughput	Max Band
Achieved Active Warp	L2 Hit Rate	Active Warp Per Sch
	Mem Busy	Reg Per Thread
	Warp Cyc Inst	Achieved Occupancy
	Reg Per Thread	Achieved Active Warp

in Figure 3 and Figure 4) higher than 0.2 with any of the fault type. Table III presents our selected features for both the simulator and the profiler.

D. Prediction Model Evaluation

In this work, we aim to predict the soft error vulnerabilities of the GPGPU applications. Since we have distinct feature and fault rate values per each GPU kernel, where some of those kernels may belong to the same GPU program, we employ kernels as our data samples. Scaling down the analysis from the GPU program to the kernel level provides a more detailed study opportunity to observe the possible scenarios and the reasons for the occurrence of the faults. We utilize regression and classification approaches for the fault prediction task.

In the regression approach, we apply the regression algorithms to predict all fault rates with the help of the selected features. Specifically, we use Random Forest (RF), Support Vector Machine (SVM), and Gradient Boosting (GB) algorithms. RF benefits from the multiple decision trees by selecting more accurate ones resultant from the training set. SVM starts with a curve type such as linear or parabolic, and a certain amount of error range, called epsilon. Then, it tries to fit the pre-specified curve with respect to the training set by calculating errors such that the absolute value of the difference between the expected fault rate and the actual fault rate is lower than the epsilon value. GB builds an additive curve model in a forward stage-wise fashion such that it allows for the optimization of arbitrary differentiable loss functions as in deep neural network algorithms. In each stage, a regression tree is fitted on the negative gradient of the given loss function, and the resultant stable curve is used for the prediction tests of the remaining samples of the dataset.

We build 24 prediction models for each fault type (i.e., SDC, masked, and crash). Specifically, we apply three algorithms, with four different hyper-parameter configurations, with all the features and only the selected features. We utilize the accuracy results calculated according to the following formula:

$$(1 - \frac{|error_{predicted} - error_{observed}|}{error_{observed}}) * 100$$

Even if we can reach reliable accurate prediction rates for the masked fault rates, which are in the range of [0.682, 0.939], we cannot observe similar accuracy results with the regression methods for the crash and the SDC rates, which are in the range of [0.012, 0.263] and [0.008, 0.173], respectively. Since the masked fault rates are quite large values compared to the SDC and the crash rates, the little oscillations around the fitted values for the test samples are acceptable. However, the similar small deviations around the fitted values are not acceptable for the SDC and the crash rates because the small deviations result in larger prediction errors. Unlike the regression approach for predicting masked fault rates, we use the classification approach to predict the SDC and crash rates.

For the classification approach, similar to the previous work [11], we define different classes by considering the SDC and crash rates, and predict the class of each kernel function in our evaluation. Specifically, in the two-class model for SDC prediction, we define two different classes by considering the SDC values of the target kernel functions. For example, the kernel functions with SDC rates between [0.010, 0.050] and [0.051, 0.200] are classified as Not Vulnerable (with lower SDC rates) and Vulnerable (with higher SDC rates), respectively. Accordingly, we can predict whether the functions are vulnerable or not vulnerable to soft errors. By specifying the threshold values based on the requirements of the application domain, one can build models for different ranges. In our evaluation, we create two different models by dividing our dataset into two and three classes such that each class has nearly the same amount of data.

In addition to RF and GB algorithms, we utilize an ensemble classification model, which is configured by cascading a standard scaler (SS) and Stochastic Gradient Descent (SGD) classifier. Our purpose is to investigate the effect of the scaled features on the classification success. SS transforms the given dataset such that its distribution has the mean value of zero and the standard deviation of one. SGD algorithm calculates the gradient loss estimated for each sample, and the algorithm parameters are updated along with the learning rate. Moreover, there are L1 and L2 regularizations used in the SGD algorithm to disturb algorithm parameters and prevent overfitting. For our classification model evaluation, we use the accuracy metric, which is simply the rate of correct classifications. Additionally, we utilize precision, recall, and F-score metrics. Precision

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

is the percentage of the relevant samples found among the recovered instances, whereas recall is the percentage of the relevant instances found. Hence, both precision and recall are based on relevance. F-score is the harmonic mean of precision and recall used for test accuracy results. The formula for each term is as follows:

$$precision = \frac{tp}{tp + fp} \qquad recall = \frac{tp}{tp + fn}$$

$$Fscore = 2 * \frac{precision * recall}{precision + recall}$$

where *tp* stands for true positive, *fp* stands for false positive, *fn* stands for false negative.

Since our dataset consists of 14 GPU applications created with 26 different GPU kernels, we prefer to use the K-fold cross-validation method for the prediction work. In this method, training model parameters are obtained with K-1 amount of GPU kernels, and the remaining kernel is used to test for prediction. In our study, K is equal to 24 for the experiments carried out on GPGPU-Sim environment, while K is equal to 26 for the experiments on Nsight Compute tool.

IV. Experimental Study

In this section, we explain the experimental setup details and the results of our experiments.

A. Experimental Setup

We select twelve CUDA applications from Polybench [22] benchmark suite including 2DConvolution, 3DConvolution, 3mm, Atax, Bicg, Corr, Covar, Fdtd-2D, Gramschmidt, Gemm, Gesummv, Mvt. For our prediction framework, we perform fault injection experiments by targeting each kernel function in those programs and collect the metrics for the kernel functions separately.

We run the fault injection experiments in an Intel-based workstation with an NVIDIA Quadro P4000 GPU. We use 1000 fault injections per each kernel function by using a statistical approach [23] with the confidence level of 95% and an error margin of 3%. We run the target programs for our profiler-based metric collection in the same environment with NVIDIA's Nsight Compute tool, version 1.0.0. For the simulator part, we execute the same programs in GPGPU-Sim simulator, version 4.0. We configure Quadro P4000 device based on the configuration provided in the simulator by specifying hardware parameters such as SM amount, warp scheduler, and DRAM bandwidth. Then, we collect the mentioned parameters for this configuration shared in our Github repository [1].

As mentioned in Section III, we employ three different machine learning algorithms for the regression (SVM, RF and GB) and the classification (ensemble, RF and GB) experiments, where each algorithm has four different configurations with changing hyper-parameters. We utilize the algorithms implemented in Sci-kit library [24]. The learning rates used for the GB regression algorithm are 0.001, 0.01, 0.1 and 0.25. For the RF regression algorithm, we configure the effect of randomness as 20 and 50, the number of estimators as 100, 1000 and 10000 and keep other parameters as default. For the SVM regression, we assign the kernel function as polynomial curves with two and three degrees, sigmoid curve and radial curve. For the classification, the maximum depth of trees is two, the total iteration amount is 1000 for each algorithm. In addition, the epsilon error is 0.001 and the number of parallel workers is 20. With those hyper-parameters, we evaluate prediction models for GB, RF and ensemble learning classification algorithms.

B. Experimental Results

1) Feature Selection: After collecting the hardware performance metrics from both the simulator and the profiler, we compute Spearman and Pearson correlation coefficient values to understand which features affect the fault rates of the target code. As we explain the detailed correlation results among all the values including the features and the fault rates in Section III-C, in this section, we focus only on the correlation results between the features and the fault rates including masked faults, crash, and SDC. Figure 5 and 6 present the Spearman

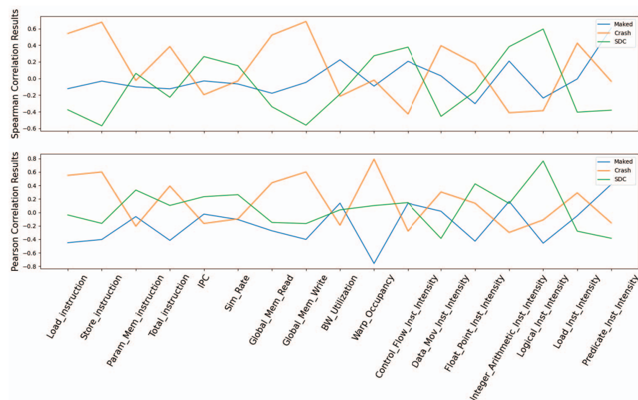

Fig. 5. Spearman and Pearson correlation results between the simulator features and the fault rates.

and Pearson correlation results between the faults and the metrics, respectively. In line with those correlation results, we select the features as shown in Table III from both the simulator and the profiler to eliminate irrelevant features and reach out more accurate prediction rates.

2) Regression Results: As mentioned in Section III-D, we use the regression model to predict the masked fault rates. Table IV shows the accuracy results obtained by the regression algorithms, based on the simulator and the profiler metrics. While all ML algorithms yield prediction accuracy values that are larger than 90%, GB algorithm results in the highest prediction accuracy that is 96.59%. We observe that the simulator results are better than the profiler results except for one case (SVM with selected features). Since the accuracy values are not significantly different for the simulator and

[1]https://github.com/topcuburak/FaultPredictionOnGPGPUs

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

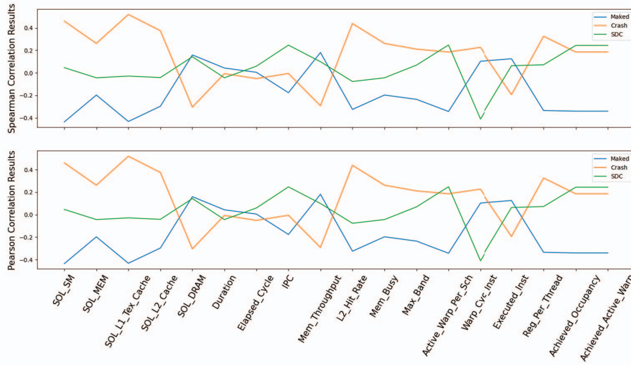

Fig. 6. Spearman and Pearson correlation results between the profiler features and the fault rates.

the profiler, deciding on the platform to be used for metric collection is not reliable at this moment.

TABLE IV
REGRESSION ACCURACY RESULTS FOR MASKED FAULTS ON EACH MACHINE LEARNING ALGORITHM.

Algorithm	The simulator metrics		The profiler metrics	
	All	**Selected**	**All**	**Selected**
RF	96.127	96.209	95.209	95.509
GB	96.463	**96.592**	95.671	95.812
SVM	92.111	92.105	91.747	94.814

3) Classification Results: For the SDC and crash rate predictions, instead of predicting the absolute rates, we build classification models by defining the problem as classifying the kernel functions according to their SDC and crash rates. We present our results for both 2-class and 3-class model, where we define different classes by considering the fault rates of the kernel functions. Besides the prediction accuracy values considering all class predictions, we include precision, recall, and F-score values with respect to the prediction of the class with the highest rates. Since we believe that the predicting a vulnerable function (with higher SDC or crash rate) as not vulnerable is more problematic than the case where the model tells more vulnerable for a function with lower vulnerability, we choose to evaluate the precision/recall values for the first case.

Table V and Table VI present our 2-class and 3-class evaluations, respectively. Since we observe the similar or higher accuracy results by using the all the features and the selected features, we only include the results with the selected features (given in Table III), and omit the values obtained from the experiments where all the features are used for training.

The maximum classification accuracy achievable in 2-class SDC prediction experiments is 82.6%, and we observe this rate by using the simulator metrics and the GB algorithm. Similarly, the GB algorithm using the simulator metrics yield the highest accuracy values for crash conditions. While we achieve 80.0% classification accuracy with the GB algorithm using the profiler metrics, which is the highest accuracy rate in 3-class SDC prediction experiments, the ensemble algorithm trained with the simulator metrics yield similar accuracy value (78.3%) and perfect precision/recall values. We can figure out that the algorithms are able to predict SDC and crash conditions more accurately with the simulator metrics compared to the profiler. The implicit reason behind this is that the profiler provides statistical metrics for the program's execution on the hardware and does not provide any explicit metrics describing the instruction intensities of the applications. Therefore, as we use the metrics related to the execution of any GPU application in hardware, we can say that we move away from its relationship with fault injection that injects faults onto specific locations on the hardware such as registers or memory locations and observes the fault effects. However, the simulator metrics are more representative for the application's structure such that they characterize application's behaviour by the instruction intensities in addition to the hardware usage metrics. Although the profiler seems to provide those metrics through the cache hit/miss rates implicitly, the register usage per thread, row utilization and memory coalescing for DRAM, there are many dependent configurations that determine these metrics without the applications behaviour such as the limitation on the hardware resources. Furthermore, our fault injection experiments target the registers for our benchmark applications. Thus, rather than other hardware metrics in the GPU, the application's use of registers and what it uses these registers (i.e pointing for memory locations) is more beneficial for predicting fault rates as confirmed by our experimental results.

While the classification accuracy achieved in the 2-class classification of SDCs and crashes are 82.6% and 87.0%, the corresponding values for 3-class classification are 80.0% and 60.9%, respectively. Furthermore, the GB algorithm results in higher accuracy values in 2-class case, while the RF and ensemble algorithms yield more successful results in 3-class evaluation with relatively lower accuracy values. We can see that GB algorithm is able to maintain successful results due to its ability to deal with complex datasets [25].

The classification results reveal that we can utilize the classifiers for vulnerability prediction of GPGPU programs or kernel functions. When we formulate the problem as a classification problem to obtain the vulnerability level of the target program, we can predict in which range we expect to see SDC or crash conditions. This evaluation enables us to understand how vulnerable the program is, even if we are not able to predict the absolute SDC or crash rates. Based on the classification outcome, we can decide whether to perform any fault tolerance techniques.

V. CONCLUSION

To conclude, we try to predict occurrence rates of SDCs, crashes and masked faults with the help of the simulator and the profiler metrics. Since SDCs and crashes are observed dramatically less compared to the masked faults, we use a classification method to determine the error vulnerable conditions instead of the regression approach. For both the regression and the classification approaches, the experiments with the

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.

TABLE V
CLASSIFICATION RESULTS FOR 2-CLASS EVALUATION.

	Algorithm	The simulator metrics				The profiler metrics			
		Precision	Recall	F-Score	Accuracy	Precision	Recall	F-Score	Accuracy
SDC	RF	66.7	50.0	57.1	73.9	50.0	40.0	44.4	60.0
	GB	**75.0**	**75.0**	**75.0**	**82.6**	54.5	60.0	57.1	64.0
	SS+SGD	50.0	50.0	50.0	56.5	100.0	50.0	66.7	60.0
Crash	RF	81.2	86.7	83.9	78.3	77.8	82.4	80.0	72.0
	GB	**87.5**	**93.3**	**90.3**	**87.0**	68.4	76.5	72.2	60.0
	SS+SGD	68.8	73.3	71.0	69.6	75.0	70.6	72.7	68.0

TABLE VI
CLASSIFICATION RESULTS FOR 3-CLASS EVALUATION.

	Algorithm	The simulator metrics				The profiler metrics			
		Precision	Recall	F-Score	Accuracy	Precision	Recall	F-Score	Accuracy
SDC	RF	50.0	25.0	33.3	60.9	x	x	x	68.0
	GB	75.0	75.0	75.0	69.6	**100.0**	**75.0**	**85.8**	**80.0**
	SS+SGD	**100.0**	**100.0**	**100.0**	**78.3**	33.3	25.0	28.7	60.0
Crash	RF	**69.2**	**81.8**	**75.0**	**60.9**	58.3	58.3	58.3	44.0
	GB	72.7	72.7	72.7	52.2	63.6	58.3	60.9	48.0
	SS+SGD	50.0	45.5	47.6	34.8	55.6	41.7	47.6	48.0

simulator metrics result in more reliable prediction accuracy results compared to the profiler metrics. Furthermore, we share the accuracy quantities such as precision and recall scores for the largest vulnerable fault classes of SDCs and crashes. This work can be expanded such that fault injection experiments can be enhanced to inject faults into the different hardware regions in different physical condition such as space condition. In this way, vendors generate their GPUs with different precautions by depending on the usage conditions.

ACKNOWLEDGEMENT

This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), Grant No: 119E011.

REFERENCES

[1] T. M. Aamodt, W. W. L. Fung, T. G. Rogers, and M. Martonosi, *General-Purpose Graphics Processor Architecture*, 2018.

[2] M. Dimitrov, M. Mantor, and H. Zhou, "Understanding software approaches for gpgpu reliability," in *Workshop on General Purpose Processing on Graphics Processing Units*, 2009.

[3] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1226–1238, 2016.

[4] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing software-directed instruction replication for gpu error detection," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, 2018.

[5] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "A systematic methodology for evaluating the error resilience of gpgpu applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3397–3411, 2016.

[6] "Nvbitfi: An architecture-level fault injection tool for gpu application resilience evaluations," 2020. [Online]. Available: https://github.com/NVlabs/nvbitfi

[7] L. Guo, D. Li, and I. Laguna, "Paris: Predicting application resilience using machine learning," *Journal of Parallel and Distributed Computing*, vol. 152, pp. 111–124, 2021.

[8] D. Oliveira, F. B. Moreira, P. Rech, and P. Navaux, "Predicting the reliability behavior of hpc applications," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018.

[9] I. Laguna, M. Schulz, D. F. Richards, J. Calhoun, and L. Olson, "Ipas: Intelligent protection against silent output corruption in scientific applications," in *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2016.

[10] Q. Lu, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, "Sdctune: A model for predicting the sdc proneness of an application for configurable protection," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2014.

[11] I. Öz and S. Arslan, "Predicting the soft error vulnerability of parallel applications using machine learning," *Int. J. Parallel Program.*, vol. 49, no. 3, pp. 410–439, 2021.

[12] D. Jauk, D. Yang, and M. Schulz, "Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, 2019.

[13] C. Kalra, F. Previlon, X. Li, N. Rubin, and D. Kaeli, "Prism: Predicting resilience of gpu applications using statistical methods," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18, 2018.

[14] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[15] "Nvidia, pascal architecture whitepaper." [Online]. Available: https://www.nvidia.com/en-us/data-center/resources/pascal-architecture-whitepaper

[16] "Nvidia, cuda llvm compiler." [Online]. Available: https://developer.nvidia.com/cuda-llvm-compiler

[17] S. Mukherjee, *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[18] I. Oz and O. F. Karadas, "Regional soft error vulnerability and error propagation analysis for gpgpu applications," *Journal of Supercomputing*, pp. 1–1, 2021.

[19] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[20] "Nvidia nsight compute." [Online]. Available: https://developer.nvidia.com/nsight-compute

[21] "Nvidia parallel thread execution isa version 7.4," 2021. [Online]. Available: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[22] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," ser. 2012 Innovative Parallel Computing (InPar), 2012.

[23] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," ser. Proceedings of the Conference on Design, Automation and Test in Europe (DATE), 2009.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[25] S. Lundberg, G. Erion, H. Chen, A. DeGrave, J. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable ai for trees," *Nature Machine Intelligence*, vol. 2, 01 2020.

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on July 28,2022 at 08:20:27 UTC from IEEE Xplore. Restrictions apply.