2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 978-1-6654-6605-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISVLSI54635.2022.00056

Predicting GPU Performance and System Parameter Configuration Using Machine Learning

Zhuren Liu, Trevor Exley, Austin Meek, Rachel Yang, Hui Zhao, Mark V. Albert
Department of Computer Science and Engineering, University of North Texas
{zhurenliu, trevorexley, austinmeek2, rachelyang}@my.unt.edu, {hui.zhao, mark.albert}@unt.edu,

***Abstract*—GPUs are widely used in accelerating computation-intensive applications. Performance models are important for designing high-performance and cost-efficient GPUs. In this work, we developed machine learning models that can accurately predict GPU system performance. Our model can identify important features that can provide insights to designers on the most important hardware system parameters when executing their applications. We also developed a model for predicting minimum system configuration parameters based on performance. Our model can provide system configuration recommendations for users to meet their performance requirements.**

***Index Terms*—GPU, performance prediction, system recommendation, system parameters, system configuration**

I. Introduction

General Purpose Graphics Processing Units (GPGPUs) have become a strong challenger to the conventional workhorse of computing, i.e., the CPUs. Nowadays, we can find GPUs in various systems, from mobile devices to high end servers [3], [6]–[8], [15], [16], [18]. GPUs can be either lightweight and power efficient, or performance-centric and power hungry. Studies have shown that changes in hardware configurations (such as the memory capacity and number of cores) can lead to significant changes in GPU performance. This presents challenges to customers who needs to find an optimal GPU that can fit their performance requirements within their budget. Therefore, it is important to develop some models for performance prediction and system configuration recommendation that can provide customers with some insight about the systems they want to select.

Machine learning techniques have been used to interpret patterns from training data and make decisions (such as classification and prediction) with reduced interaction from humans. A machine learning model can be used to predict the system performance and this can help with design space exploration. Prediction of accelerator performance has been explored by prior work [4], [12], [14], [20]. Such techniques include analytical model-based prediction [20], source code with idiom recognition [4], [14], and automatically constructed prediction models [12]. Most of these approaches use performance as the predictor output, that is, given a system configuration, they can predict the performance. However, they do not provide more insights about which system parameters have more significant impact on the performance. In addition, they require a user to have a knowledge of the system but cannot recommend a system parameter configuration given the desired performance from the user.

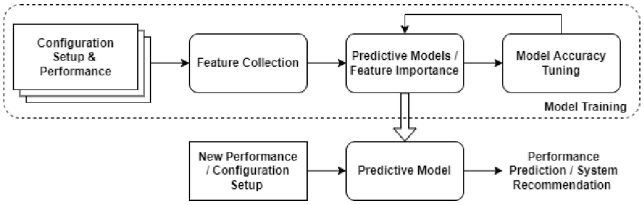
Configuration Setup & Performance
Feature Collection
Predictive Models / Feature Importance
Model Accuracy Tuning
Model Training
New Performance / Configuration Setup
Predictive Model
Performance Prediction / System Recommendation

Fig. 1. High level architecture of our proposed prediction model.

In this work, we developed machine learning methods that can predict GPU performance, provide feature importance analysis, and give system configuration recommendations. In particular, we first built prediction models using SVM for GPU performance prediction. Then, feature importance was derived from the prediction models using random forest to evaluate the impact of different features on system performance. A SVC machine learning model for reverse engineering is also built to recommend system parameter configuration in order to meet user requirements in performance. Our results show that our performance prediction model has a normalized root mean square error (RMSE) lower than 0.07, and the R2 score varies from 79% to 94%. Our model for system configuration recommendation also has high accuracy (up to 97%).

Our major contributions in this work include:

- We created machine learning models to predict GPU system performances with high accuracy.
- We provided a detailed feature importance analysis. Our analysis can provide insights on what impact different system parameters can have on the system performance.
- We developed a prediction model for system configuration recommendation. Our recommendation model can help users to find a system configuration that can reach their performance target.

II. Approach

A. Prediction Model Overview

Figure 1 shows the high level view of our proposed framework. The first stage is to select GPU system configuration parameters that can impact the performance. The next step is to configure the GPU systems using these parameters and collect the training data through benchmark execution. We selected several representative applications from the benchmark suit *ispass-2009* [9] in our experiments. Further details are discussed in the following subsections. After the training data is collected, it is fed into the prediction model for training. We

2159-3477/22/$31.00 ©2022 IEEE

DOI 10.1109/ISVLSI54635.2022.00056

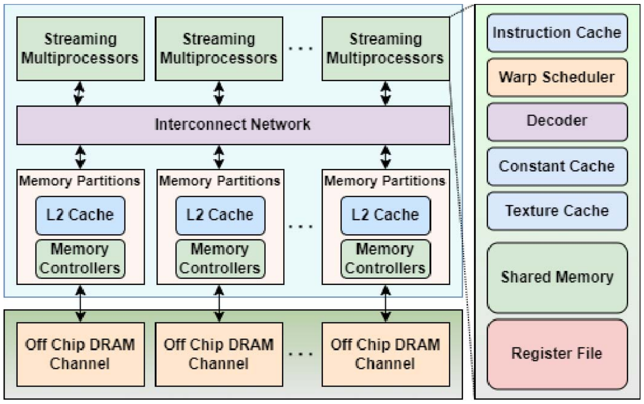

Fig. 2. GPGPU architecture

also go through steps to tune the model accuracy. After the model is trained, we can use it to either predict the system performance or generate system configuration recommendations. The model also generate the correlation between the inputs and output so that we can analyze the importance of each system parameter (which is also called feature in this paper).

B. Approach

We used a supervised machine learning model for performance prediction. A unique feature of our model is that we also provided feature importance information that can help users to understand the role of an input feature in determining the output, i.e. system performance in this case. While there have been several models proposed for performance prediction in CPUs and GPUs [4], [12], [14], [20], they do not provide the feature importance analysis. In fact, such information is very important to both system designers and users because this can help them to build or find an optimal product. Other than predict performance using system parameters, our model can also perform the *reverse prediction*. That is, given a target performance and part of the input features (system configuration parameters), we can predict the remaining parameters so that a system configured with these parameters can reach the performance target. This technique is in fact a kind of reverse engineering and it has use cases in reality. For instance, a customer wants to purchase a GPU and there are different categories of GPUs to select from with different costs. He has some requirements in execution time and can select a GPU with large memory but fewer Streaming Multiprocessors, or smaller memory with more Streaming Multiprocessors. Using our model, he can find a product that can meet his performance requirement and then select the most economic system.

Feature selection is a important part of machine learning, it is a way of trimming down the number of input variables when devising a prediction model. The reduction in the number of input variables can decrease the computational cost and improve the model accuracy. In our case, the features are from the configurations in GPU systems. Figure 2 shows the architecture of a modern GPGPU. It consists of multiple Streaming Multiprocessors (SMs), and several layers of memory partitions. Inside each Streaming Multiprocessor, there are control units, registers, execution pipelines, scratchpad memory, and caches. Through experiments, we found that the most important features for our prediction models are the number of Streaming Multiprocessors, number of Memory Controllers, Shader Register File size, number of Shader Cores, L1 Configuration, and L2 Configuration. We use Instruction per Cycle (IPC) as an indicator for performance.

TABLE I
CONFIGURATION SETTINGS

Configuration	Settings
SM	2,3,6-8,12-15,20-24,30-35,42-48,56-63,72-80,90-99
MM	1 - 10
Shared Memory	8KB, 16KB, 32KB, 64KB
Shader Register	12288, 24576, 49152
Shader Core	2048:32, 1024:32, 1536:32
L1	32KB / 64KB, 4-way set assoc, 64B lines, LRU
L2	64KB / 128KB, 8-way set assoc, 64B lines, LRU

Number of Streaming Multiprocessors (SM), Number of Memory Controllers (MM), Size of shared memory per SIMT core (Shared memory), Number of registers per shader core (Shader Register), Shader core pipeline configuration (Shader Core), L1 cache (L1), L2 cache (L2).

C. Data Collection

Table I lists our selected system configuration parameters as the feature inputs for our machine learning model. It also shows the parameters' possible values and ranges. In our experiments, we created system settings using different combinations of these parameters. Different configuration parameters can lead to different IPC for a benchmark. Note that some of the configuration options are not continuous, such as the number of Streaming Multiprocessors. These are constraints from the GPGPU-sim simulator. There also exists some interrelations between certain parameters. For example, the number of Streaming Multiprocessors and Memory controllers is related. This is because the number of Memory controllers and Streaming Multiprocessors is determined by the mesh interconnection network size. All our training data is collected using the benchmarks suite of *ispass-2009* [9]. This benchmark suite is widely used in GPU performance evaluation. Among all the benchmarks in the suite, we select five that are representative of different types of GPU applications. The amount of training data collected for each benchmark is determined by the possible configurations' variations. Instead of real GPUs, we used GPGPU-sim [2] to simulate the benchmark executions because it is not feasible to find hardware systems that match all the configurations we want to test.

Data collected from each benchmark are extracted directly from the configuration file. Streaming Multiprocessors, Memory Controllers, Shader Register, and IPC are stored as integers. Shader Core, L1 Configuration, and L2 Configuration data are stored as strings. Ideally, we would want them to be of the same type so that we changed the strings to an integer representation. Shader core, L1 Configuration, and L2 Configuration are classified as different classes then saved.

III. MACHINE LEARNING MODEL DESIGN

In this section, we provide design details of our machine learning models.

A. Support Vector Machines and Support Vector Regression

A very popular classification and regression technique used in machine learning is the support vector machines (SVM). The SVM algorithm creates a line or a hyperplane which tries to separate the data into classes, often after the data has been transformed using nonlinear kernels, allowing the borders between classes to be nonlinear in the original feature space. The data is separated so that the maximum margin is the optimal hyperplane.

However, SVM can only deal with binary data. In order to predict a discrete value such as the performance, we choose to use support vector regression (SVR). Support vector regression is based on the same principles as SVM. As described in prior work [1], the regression problem can be seen as a generalization of the classification problem. Therefore, instead of returning a finite set in classification, the model returns a continuous-valued output. Unlike other simple regression models, SVR tries to find the best fit hyperplane with the maximum number of points.

B. Feature Importance with Random Forest

To design a prediction model, it is important to have an accurate and interpretable model in many cases. A good way of understanding what causes the model to perform the way it does is feature importance. Feature importance can help understand a problem by providing which features are relevant. As a supervised learning algorithm, random forest takes advantage of ensemble learning for classification and regression. There is a built-in feature importance computation based on Gini importance (or mean decrease impurity) and mean decrease accuracy in random forests. Gini importance is derived from the random forest structure. Based on the average of all trees in the forest, we can then determine the importance of each feature. Sklearn's implementation for both random forest regressor and random forest classifier is based on this idea [17].

We used Scikit-learn's (sklearn) implementation of SVM, SVR, and random forest for our prediction model. Sklearn [17] is a machine learning library in Python. It features a collection of classification, regression, and clustering algorithms. It also incorporates widely used numerical Python libraries such as NumPy. We experimented with several different classification (Lasso, Ridge Classification, K-Nearest Neighbors Classifier) and regression (Bayesian Regression, Naive Bayes) algorithms and settled on SVM and SVR for the prediction model. Compared to the other algorithms that we have tried, SVM and SVR were selected because they fit best with our input data structure and have the best accuracy. The random forest algorithm in sklearn was selected for our feature importance portion. The algorithm provided by sklearn has a *feature_importances_* property that can be used to retrieve the relative importance scores of each input feature.

C. Validation of Machine Learning Models

Cross-validation is a validation technique for assessing how well the results generalize on an independent data set by rotating which portion of a data set is used for training and which is used for testing. It is a commonly used statistical method in machine learning to evaluate the accuracy of the given prediction model. However, when using cross-validation to tune hyperparameters or select among many models, the same data are used to tune and evaluate a model when using Cross-validation. This can lead to inflated accuracy of the prediction model due to potential overfitting. To avoid this, one can keep a hold-out test set with cross-validation used for hyperparameter tuning. However, the hold-out test sets needs to be large enough to be reliable, which is difficult with limited data. One approach to overcoming this problem is to use nested cross-validation. Nested cross-validation is a variant of cross-validation which also rotates the test set while cross-validation is performed on the training and validations sets used for hyperparameter tuning. In other words, nested cross-validation 'nests' the optimization of the hyperparameter into the model selection. Nested cross-validation separates the data into a series of test, train, and validation data. It runs a grid search on each of the training sets to get an approximately maximized score by fitting the model. Then this score is maximized by selecting the hyperparameters in the validation set. Finally, by averaging the test set scores over the dataset splits, it'll get the estimate of the generalization error.

In this study, we use nested k-fold cross-validation for the performance prediction model and nested stratified k-fold cross-validation for the system recommendation prediction model. k-fold cross-validation is a validation technique that separates the data into k consecutive folds. Each fold is used only once as the test/validation set, while the remaining k-1 folds are used as the training sets. Stratified k-folds cross-validation is a variation of the k-fold cross-validation by returning stratified folds. Unlike traditional k-fold, stratified k-fold preserves the percentage of each group for each fold. This ensures that each fold of the dataset has the same proportion of observations with a given label, which can be helpful with imbalanced class distributions. The reason that we choose nested k-folds for the performance prediction model is due to both our model type and data. For our system recommendation prediction model, we used nested stratified k-fold. Some of the configurations in our data are not equally distributed. This can result from either the configuration not being executed on the benchmark or the distribution among the configuration setting is not equal. For example, MM has ten different values, with 1 appearing over 1000 times and 10 appearing only 144 times. Using nested stratified k-fold can eliminate the inaccuracy caused by imbalanced class distribution.

D. Feature Importance Analysis

Feature importance is a technique that calculates a score for each feature representing its importance. We used the random forest algorithm provided by sklearn to implement this function. We ran the feature importance when predicting the IPC with our benchmarks, which turns out to be highly representative and can better serve the purpose of our model.

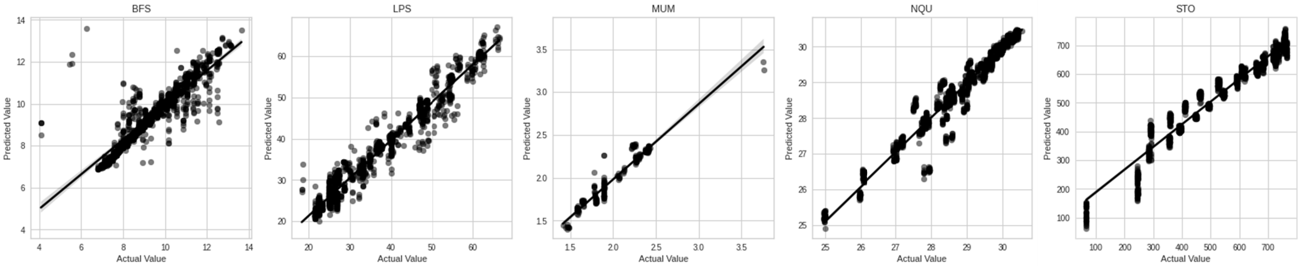

Fig. 3. Predicted value vs. Actual value scatter plot with 95% confidence interval, X axis is Actual Value, Y axis is Predicted Value

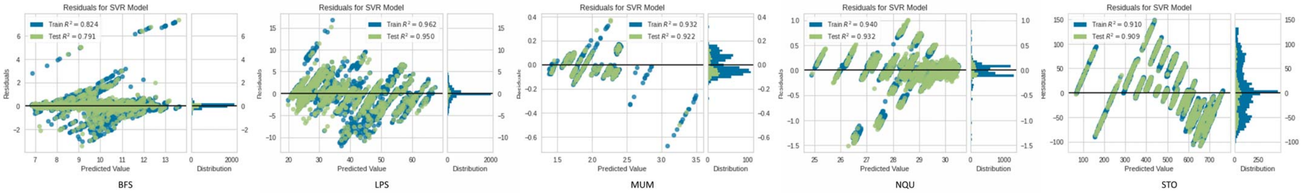

Fig. 4. Residual plot, X axis is Predicted Value, Y axis is Residuals

IV. Experimental Evaluation

This section consists of three parts: performance prediction, feature importance analysis and system configuration recommendation. As mentioned in the previous sections, the training data we collected is not equally distributed, and the configurations for each of the benchmarks also differ. However, our evaluation shows this does not affect the accuracy of our model. All of our experiments were run on the computing nodes on the Lonestar6 system in TACC's high-performance computing systems. Each compute node has two AMD EPYC 7763 processors and 256 GB of DDR4 memory. Each processor has 64 with 128 cores in total. All the cores are running at 2.45 GHz, which can be boosted up to 3.5 GHz. We ran our benchmarks with CUDA 9.1 and Ubuntu version 16.04 on GPGPU-sim 4.0 [2].

A. Performance Prediction

The purpose of our model is to predict a performance value (IPC) with the given configurations. IPC values are collected as we ran each benchmark with different configuration settings. We used a regression model as the IPC values we collected are continuous. A model was constructed for every benchmark. The root mean square error (RMSE) was used to optimize the regressor and assess the performance of our regression model. Coefficient of determination (R2 scores) was used as an alternative measure of how well our model works. R2 scores are used to determine the variance of the regression model. For instance, a 0.8 R2 score indicates that the variance of the independent variable explains 80% of the variance of the dependent variable.

As shown in Table II, all of the benchmarks have a normalized RMSE of 0.077 or lower. normalization of the RMSE is calculated by using the average RMSE/(Max value - Min value). This produces a value between 0 and 1, with values near 0 being better fitting values. BFS has an R2 score of 0.798, while all other benchmarks are over 0.9. The reason for BFS's R2 score being low is due to the data gathered. Some of the features show no relevance to the performance. For example, the L2 configuration has no influence on the performance. This is because many variations of the L2 configurations are not important to this benchmark's performance. In other words, the L2 configuration is not the bottleneck for the performance of BFS.

TABLE II
Performance prediction

Benchmarks	Avg. RMSE	Normalized RMSE	Avg. R2	Best R2
BFS	0.684	0.071	0.798	0.820
LPS	2.862	0.058	0.943	0.947
NQU	0.335	0.059	0.928	0.936
STO	54.173	0.077	0.905	0.908
MUM	0.109	0.046	0.911	0.916

Avg. RMSE: The average Root Mean Square Error for each benchmark (range differs); Normalized RMSE: Normalized Root Mean Square Error with 1 being worst and 0 being best; Avg. R2 score: Average Coefficient of determination with 1 being best and 0 being worst; Best R2 score: Best Coefficient of determination with 1 being best and 0 being worst.

The predicted vs. the actual value scatter plots for each benchmark, as shown in Figure 3, were created to show the prediction ability of our model. The dots are created as transparent for each point for easier viewing or overlapping dots. The figures show that our points are close to the fitted line, with a confident band to narrow to show. The dots are plotted as transparent. The range of values for each benchmark is different, as seen in the figure, with STO having a wide range and MUM having the least variation in range. While other benchmarks are distributed quite equally, MUM clusters in the range of 1.4 to 2.0 with a few outliers, hence the figure skewing to the left.

A residual plot is shown in Figure 4. A residual plot is a good way of showing if our data is suited for a regression model. The data shows a fairly random, uniform distribution

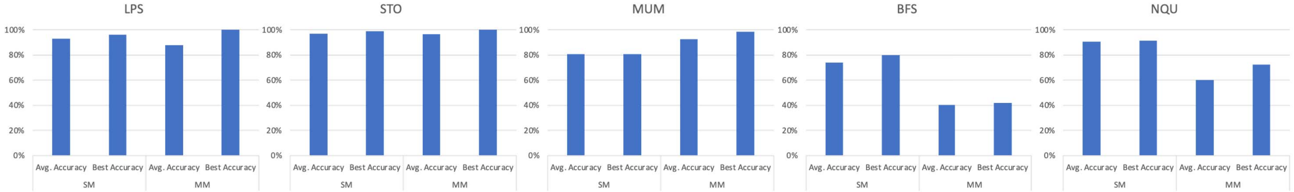

Fig. 5. System recommendation prediction results, Streaming Multiprocessors (SM), Memory Controllers (MM).

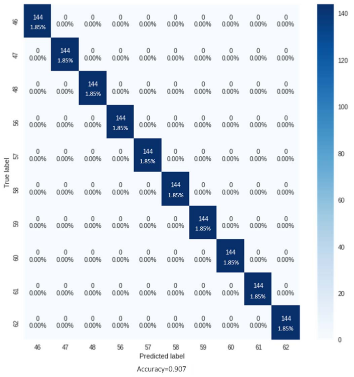

Fig. 6. SM Confusion matrix for the benchmark NQU

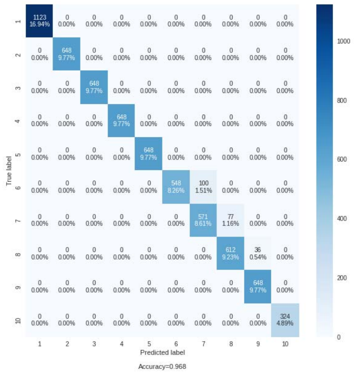

Fig. 7. MM Confusion matrix for the benchmark STO

TABLE III
RANDOM FOREST FEATURE IMPORTANCE RANKINGS

Rank	BFS	**LPS**	**MUM**	**NQU**	**STO**
1	L1	MM	MM	SM	MM
2	SM	SM	Registers	MM	SM
3	Registers	Registers	SM	L1	L1
4	MM.	L2	L1	Registers	L2

The top four important features, with rank 1 being the most important, are shown with the random forest feature importance ranking. We used short names for system parameters, including the number of Streaming Multiprocessor (SM), the number of Memory Controllers (MM), L1 Configuration (L1), L2 Configuration (L2), the number of registers per shader core (Registers).

around the horizontal axis (target) for all the benchmarks, suggesting that our choice of the model is an appropriate one. MUM has a few outliers, skewing the graph toward the top. The distribution of the residuals of each benchmark can be seen in the histogram left of the residual plot. The histogram shows that our data for all benchmarks are normally distributed around 0, indicating that our model fits well with the data. The R2 scores for both training and test sets are calculated with the residual plot. We observed a slightly higher R2 for training sets and high R2 scores for training and testing sets. For the benchmarks LPS, MUM, NQU, and STO, both R2 scores for training and testing sets are over 0.9, indicating a very good fit for our model. Although for BFS, we have a lower R2 score of 0.824 (training set) and 0.791(test set), it still shows that our model performs well. This observation shows that our model generalizes well with all of the tested benchmarks.

B. Feature Importance

The feature importance, also known as variable importance, helps to improve the understanding of a problem by indicating relevant features. This can direct us on where to improve the model during feature selection and provide the relevance information to users. Random Forest feature importance was used in our design to describe the importance of each feature when predicting the performance. We first used cross-validation to get the best parameter for our estimator (random forest regressor). This is done with the GridSearch() function provided by sklearn. We then ran the property *feature_importances_* in RandomForestRegressor() 20 times with the best parameters for our estimator. Finally, we calculate the average for all the runs and gather the results. Table III shows the top four features we gathered from running Random Forest feature importance for each benchmark. As seen in Table III, Streaming Multiprocessors (SM) and Memory Controllers (MM) appear in the top 4 features on every benchmark. This observation somewhat aligns with what we were expecting. The increase or decrease in SMs or MMs affects the performance drastically. The number of registers per shader core (Register) and the L1 Configuration (L1) are the following most commonly important features. However, our results show that different parameters show different important to these benchmarks.

C. System Configuration Parameter Recommendation

The previous sections evaluated the prediction models for performance and the significant features when predicting the performance (feature importance). In this section, we evaluate the prediction model for system recommendation. This is a very useful feature of our model and it can have useful applications in the real world. This section presents the accuracy for two of our configurations with our model trained on each of the benchmarks with SVM. In our reverse prediction model, given a target performance and part of the system parameters, we can predict the remaining parameter so that the system can achieve the performance goal. As mentioned in the previous section, we found that SM and MM are the two most important features when predicting performance. Therefore, we develop our model to predict the SMs and MMs, given the performance and some of the other configurations.

We use the support vector classifier (SVC) provided by sklearn [17] as the supervised learning method with nested cross-validation to train our model. Our training data shows data imbalances in the configurations SM and MM, making it unsuitable to use KFold for the nested cross-validation. Stratified K-Fold is chosen here as it is the improved version of the K-fold by ensuring each fold of the dataset has the same proportion of observations with a given label. A confusion matrix was plotted for each benchmark to help us understand how well our model is performing in predicting both SM and MM. However, a confusion matrix is very large and we can only show part of it. Figure 6 is an example of a portion of SM's confusion matrix for benchmark NQU. Figure 7 shows the entirety MM's confusion matrix for benchmark STO. Figure 5 shows the results we collected from our training. The average accuracy when predicting SM varies between 73-97%, with the highest average being the benchmark STO and the lowest being BFS. The highest accuracy can reach 99% accuracy when predicting SM. The average accuracy varies more when predicting MM, with LPS, MUM, and STO having higher accuracy and BFS having a less accuracy. The highest accuracy can go up to 100% when predicting MM. The low accuracy of BFS is due to our training data because BFS needs relatively lower resource compared with other benchmarks. Therefore, some features do not affect the performance are still collected to make our training process consistent. However, BFS has many different configurations generating the same performance and this compromised the accuracy of our model. In general, our results match with the observation we have with feature importance, indicating that we have a reliable prediction model.

V. Related Work

Machine learning approaches have been used to analyze microarchitectural designs in design space exploration [5], [13]. Simulation tools such as GPGPU-Sim [2] are widely used to estimate GPU performance. By constructing power and performance regression models from a limited number of training points, techniques proposed in Stargazer [10] and Starchart [11] can decrease the number of simulation points required to explore a GPU's design space. Baldini et al. demonstrated that machine learning approaches can be used to create accurate GPU acceleration prediction models. [3] Wu et al. used performance counter values from hardware configurations to estimate a GPGPU kernel's performance and power on various hardware configurations [19]. Our approach differs from these techniques because our machine learning model can not only predict GPU performances but also support parameter importance analysis and system recommendation to meet performance requirements.

VI. ACKNOWLEDGMENTS

This work is supported in part by NSF Grants 1828105, 2046186, 2008911.

References

[1] Mariette Awad and Rahul Khanna. *Support Vector Regression*, pages 67–80. Apress, Berkeley, CA, 2015.

[2] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009.

[3] Ioana Baldini, Stephen J Fink, and Erik Altman. Predicting gpu performance from cpu runs using machine learning. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pages 254–261. IEEE, 2014.

[4] Laura Carrington, Mustafa M Tikir, Catherine Olschanowsky, Michael Laurenzano, Joshua Peraza, Allan Snavely, and Stephen Poole. An idiom-finding tool for increasing productivity of accelerators. In *Proceedings of the international conference on Supercomputing*, pages 202–212, 2011.

[5] Xianwei Cheng, Hui Zhao, Mahmut Kandemir, Beilei Jiang, and Gayatri Mehta. Amoeba: a coarse grained reconfigurable architecture for dynamic gpu scaling. In *Proceedings of the 34th ACM International Conference on Supercomputing (ICS)*. ACM, 2020.

[6] Xianwei Cheng, Hui Zhao, Mahmut Kandemir, Saraju Mohanty, and Beilei Jiang. Alleviating bottlenecks for dnn execution on gpus via opportunistic computing. In *Proceedings of the 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2020.

[7] Xianwei Cheng, Yang Zhao, Mohammadreza Robaei, Beilei Jiang, Hui Zhao, and Juan Fang. A low-cost and energy-efficient noc architecture for gpgpus. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. ACM/IEEE, 2019.

[8] Xianwei Cheng, Yang Zhao, Hui Zhao, and Yuan Xie. Packet pump: Overcoming network bottleneck in on-chip interconnects for gpgpus. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. ACM/ESDA/IEEE, 2018.

[9] gpgpu sim. Github - gpgpu-sim/ispass2009-benchmarks: Benchmarks used in the gpgpu-sim ispass 2009 paper, 2022.

[10] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *2012 IEEE International Symposium on Performance Analysis of Systems & Software*, pages 2–13. IEEE, 2012.

[11] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. Starchart: Hardware and software optimization using recursive partitioning regression trees. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 257–267. IEEE, 2013.

[12] Ali Karami, Sayyed Ali Mirsoleimani, and Farshad Khunjush. A statistical performance prediction model for opencl kernels on nvidia gpus. In *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, pages 15–22. IEEE, 2013.

[13] Benjamin C Lee and David M Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *ACM SIGOPS operating systems review*, 40(5):185–194, 2006.

[14] Jiayuan Meng, Vitali A Morozov, Kalyan Kumaran, Venkatram Vishwanath, and Thomas D Uram. Grophecy: Gpu performance projection from cpu code skeletons. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2011.

[15] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[16] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[17] 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*, 12:2825–2830, 2011.

[18] Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. A performance analysis framework for identifying potential benefits in gpgpu applications. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 11–22, 2012.

[19] Gene Wu, Joseph L Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. Gpgpu performance and power estimation using machine learning. In *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*, pages 564–576. IEEE, 2015.

[20] Yao Zhang and John D Owens. A quantitative performance analysis model for gpu architectures. In *2011 IEEE 17th international symposium on high performance computer architecture*, pages 382–393. IEEE, 2011.