Prediction of SGEMM GPU Kernel Performance using Supervised and Unsupervised Machine Learning Techniques

Sanket Agrawal[1], Akshay Bansal[2], Sandeep Rathor[3]
[1]Student, Dept of CEA, GLA University, Mathura, Uttar Pradesh, India
[2]Student, Dept of CEA, GLA University, Mathura, Uttar Pradesh, India.
[3]Assistant Professor, Dept of CEA, GLA University, Mathura, Uttar Pradesh, India.

Abstract- This paper proposes a novel approach for the prediction of computation time of kernel's performance for a specific system which consists of a CPU along with a GPU (Graphical processing unit). The prediction is done on the basis of 14 different configurations of processor such as local work group size, local memory shape, kernel loop unrolling factor, vector widths for loading and storing etc. A proposed system accomplished with multiple advanced techniques like PCA (Principal Component Analysis), backward elimination for feature reduction, and KNN Regression, and Random Forest Regression for making the predictions. Finally, the performance of the proposed system for predicting the running time of processors has the accuracy of 98.46% after using feature reduction techniques.

Keywords- Machine Learning, Backward elimination, GPU Kernel Performance, Random Forest Regression, KNN Regression.

I. INTRODUCTION

Traditionally, the computation ranging from small to large processing is performed using CPU (Central Processing Unit). Today, parallel computing or parallel processing or massively parallel systems are used for various purposes for example for performing matrix multiplication. The CPU takes high processing time for computation for example calculating multiplication of large matrices or convolution in image or video processing. This problem of time overhead was removed with the vector or super scalar processors such as GPUs (graphical processing units) [1]. It is also capable of performing various arithmetic and logical operations with more than two operands. It increases the computational speed and also reduces the computation time.

The increasing demand of GPUs for variety of multi-disciplinary areas such as image processing, predictive analysis, machine learning algorithms, etc leads to the analysis of running time of GPUs on computationally high vector operations [12]. The running time of a processor depends upon multiple factors such as bus interface, core clock frequency. The performance of a processor can be measured through number of operations performed per second. Through this factor one can easily compute the approximate computational time of a GPU. This will help engineers to decide the hardware configurations of a processing unit for the required computation time.

In this paper we propose an algorithm to predict the computational time of a SGEMM

GPU kernel for multiplication of two arrays each of size 2048 x 2048. Each run is based on certain attributes of GPU such as, local work group size, local memory shape, kernel loop unrolling factor, vector widths for loading and storing etc.

The proposed paper consists of five sections as follows: Section 2 related work proposed by various researchers section 3 represents data-set used in the process of prediction, Section 4 contains the proposed work and Section 5 represents elaborated results of the proposed scheme, finally, at last the conclusion and future scope of the proposed scheme.

II. RELATED WORK

Rafael et al. [2] propose the tensor train decomposition as an unified framework for surrogate modeling and global sensitivity analysis via sobol indices. The test set presented by researcher showed a relative error of 3.4%.

C. Nugteren and V. Codreanu [3] presented CLTune, an auto-tuner for OpenCL kernels. It evaluates and tunes kernel performance of a generic user defined search space of possible parameter value combinations. Used CLTune to explore a parameter space of more than two-hundred thousand configurations for matrix multiplication on GPU. However the author did not discuss about the accuracy.

J. López-Fandiño, et. al. [4] implemented the ELM (Extreme learning machine) algorithm for GPU by using several techniques such as ensembles, spectral-spatial classification scheme to improve the accuracy results of the ELM classifier. Author provided the better accuracy only on raw ELM.

M. Song et. al. [5] proposed Pervasive CNN (P-CNN), a user satisfaction-aware CNN inference framework. P-CNN is composed of two phases: cross-platform offline compilation and run-time management. Based on users' requirements, offline compilation generates the optimal kernel using architecture-independent techniques, such as adaptive batch size selection and coordinated fine-tuning. Proposed technique requires a high computation time.

Lin, Y. et. al. [6] has predicted the performance of GPU using the CPU runs as the input data to the machine learning techniques. They used 18 parameters for the prediction and got accuracy of 91%. While our proposed system works on GPU using GPU with higher accuracy.

Konstantinidis, E., & Cotronis, Y. [7] proposed a quantitative approach model for an automated GPU performance prediction. The experiments were performed on SGEMM for a total of 28 kernels. They showed that the prediction errors were comparable even when there were different hardware configurations.

Lin, Y. et. al. [8] proposed an approach MKL_DR, which generalizes the framework of multiple kernel learning for dimensional reduction.

Matsumoto, K. et. al. [9] implemented a code generator for fast matrix multiplication kernels. Given a set of parameters the code generator produces the corresponding code. They achieved an accuracy of 848GFlops/s and 2646 GFlops/s for SGEMM and DGEMM respectively.

Furthermore; Nakasato, N. [10] proposed an optimized way for dense matrix multiplication kernel for single, double, double-double precision. They achieved the 73% and 87% of the theoretical performance of GPU for single and double precision respectively.

Ardalani, Newsha et al. [11] proposed a Cross-Architecture performance predictor that uses a CPU thread to predict the GPU performance. They used machine learning for

predicting the GPU performance for a piece of code before writing a GPU implementation of it, with an error of 26.9% on a set of 24 real-world kernels.

M. Zhu et al. [13] implemented CNN and BFS algorithm for on an HBM-enabled GPU to evaluate the improvement brought by the adoption of the HBM, and to investigate techniques to fully unleash the benefits of such HBM-enabled GPU. Experimental results shows that the algorithms implemented are faster than normal algorithms.

In this paper, we are going to propose an efficient way of classification before and after applying principal component analysis (Unsupervised), and backward elimination (Supervised) techniques for feature reduction and then used KNN regression, and random forest regression to predict the run time performance of SGEMM GPU Kernel.

III. DATASET USED

For evaluating the performance of the proposed system, we used an existing data-set of SGEMM GPU kernel [2], which measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048. It has 14 independent variables in which 10 are ordinal and 4 variables are binary. The GPU is made to execute for four times and these four executions makes the dependent variable. Thus, the complete data-set has 18 variables along with a total of 241600 instances.

Variables MWG, NWG represents per-matrix 2D tiling at work-group level its values belongs to one of {16, 32, 64, 128}, variable KWG represents inner dimension of 2D tiling at work-group level and values belongs to {16, 32}, variable MDIMC, NDIMC represents local work-group size and value belongs to {8, 16, 32}, variable MDIMA, NDIMB represents local memory shape and value ranges to {8, 16, 32}, variable KWI is for kernel loop unrolling factor whose value belongs to {2, 8},variables VWM, VWN represents per-matrix vector widths for loading and storing and value belongs to one of {1, 2, 4, 8}, variable STRM, STRN represents enable stride for accessing off-chip memory within a single thread, and variable SA, SB represents per-matrix manual caching of the 2D work-group tile.

IV. PROPOSED WORK

In our proposed algorithm we used two feature reduction techniques and two prediction techniques, on 14 independent features and 4 dependent features.

PCA or principal component analysis is an unsupervised technique used for feature reduction in data analysis and machine learning. It transforms an N x M matrix of data set to N x K matrix were M<K. It actually maximizes the total variance of the independent variables. This is done by projecting the actual data set on a lower plane. In this we compose principal components, in which all the principal components are perpendicular to each other and first principal component has the highest variance, second principal component has the second highest variance and so on. Mathematically, the new reduced dataset can be expressed as equation (1).

$$Z = W^T X \tag{1}$$

Where Z is the reduced dataset, W is the principal components direction matrix and X is the original date set matrix.

It is done by firstly finding the co-variance matrix of the data set, which is used to find the Eigen values and in turn Eigen vectors. Then, this

Eigen vector matrix is multiplied by Original Dataset as shown in equation (2)

$$P = AU \quad (2)$$

Where P is the principal components matrix, A is the original matrix of attributes, and U is the Eigen vectors matrix.

Another feature reduction technique used is subset selection. Subset selection is used to reduce the dataset of size N x M size to N x K matrix, where K<M. It is a supervised approach as it uses the target variables along with the independent variables. Unlike PCA it does not project the data set to a lower plane, but instead it chooses the independent variables whose removal from the actual data-set increases the overall accuracy. The proposed methodology uses the backward elimination technique in which we start with the complete set and remove one independent variable which reduces the error. Repeat the same procedure until the removal of one more variable increases the error. Mathematically, it can be represented as equation (3)

$$\text{Re}move(X_i)\ if\ Error(X) > Error(X - X_i) \quad (3)$$

Where Xi is the ith independent variable and X is the set of all independent variables.

For comparing the prediction of running times of the GPU, two regressors are used namely KNN Regressor and Random Forest Regressor for comparison of performance. KNN or K-Nearest Neighbor predicts the value of a test case by taking the average of k most nearest neighbor's values. It finds the k training instances from the complete dataset which has the minimum distance from the test case. In this paper we used the Euclidean algorithm for measuring the distances. The Euclidean distance between two instances using can be calculated as equation (4):

$$D(X_i, X_j) = \sqrt{\sum_{k=1}^{n} (X_{ik} - X_{jk})^2} \quad (4)$$

Where D (X_i, X_j) represents Euclidean distance between X_i, X_j and X_i, X_j are the ith and jth instances each of dimension equal to number of independent variables in the dataset.

The second and the last classifier used for the predictions is Random Forest Classifier which is an ensemble of Decision trees. Instead of making one decision tree it evaluates on n (random integer) decision trees and averages the result, and these trees are constructed on subset of original dataset. An individual decision tree calculates the independent variables' gini index or entropy to rate the variables and the variable with the highest values is used as the first nodes of the tree and the process continues. The mathematical equation of gini index is shown as equation (5)

$$Gini(X_j) = 1 - \sum_{i=1}^{f} P_i^2 \quad (5)$$

Where f in the number of values variable X_j can take and p_i is the ratio of observations with target value as i.

V. RESULT AND ANALYSIS

For the purpose of training we used a total of 1, 41,160 training instances, 51,120 validation set and 48,320 test instances. In the algorithm first the best feature reduction technique is evaluated on validation set. This feature reduction technique is used to predict the test case instances. The first feature reduction technique is PCA.

The Figure 1 shows the cumulative variance and individual variance of principal components after applying PCA. This shows that 12 principal components are sufficient for prediction purpose.

Variance of independent variables before and after applying PCA is represented in table 1 with

the following abbreviations as: Var (independent variables), TS (Training Set before PCA) VS (Validation Set before PCA), PC (Principal Components), TS' (Training Set after PCA) and VS' (Validation Set after PCA).

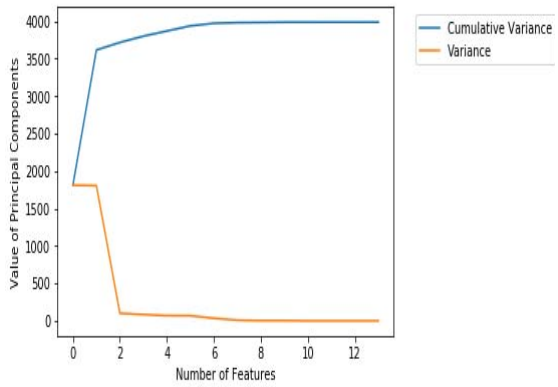

Figure 1. Cumulative and individual variance of Principal Components.

Table 1. Variance of independent variables before and after applying PCA

Var	Before Applying PCA		After Applying PCA		
	TS	VS	PC	TS'	VS'
MWG	1804.313	1802.997	1	1810.15	1811.51
NWG	1803.69	1807.20	2	1805.06	1805.68
KWG	61.72	61.47	3	100.69	100.64
MDIMC	61.07	63.22	4	84.09	84.45
NDIMC	62.07	62.35	5	68.99	69.23
MDIMA	88.19	88.04	6	68.24	68.52
NDIMB	88.21	87.68	7	34.44	34.60
KWI	9.00	8.99	8	8.99	8.99
VWM	3.81	3.84	9	3.00	3.04
VWN	3.81	3.89	10	3.00	3.01
STRM	0.25	0.25	11	0.250	0.24
STRN	0.25	0.25	12	0.250	0.24
SA	0.25	0.25	13	0.24	0.24
SB	0.25	0.25	14	0.24	0.24

Table 1 shows the variance of 14 features on training and validation set before and after applying the PCA.

The second technique used for feature reduction is backward elimination technique. Table 2 shows the accuracy of Regressor after removing a specific variable from the data set. Initially, in the first pass, when no variable is removed from the dataset the removal of variable VWM shows the highest accuracy. In second pass the variable VWM is removed and now individually each variable is removed one by one and we find that the removal of variable MDIMA shows some increment in the accuracy. In third pass, there is no increment in accuracy on removal of any variable.

Table 2. Accuracy during Backward elimination

Var	First Pass	Second Pass	Third Pass
MWG	0.600	0.511	0.455
NWG	0.607	0.597	0.565
KWG	0.898	0.916	0.907
MDIMC	0.539	0.523	0.465
NDIMC	0.545	0.541	0.490
MDIMA	0.977	**0.985**	-
NDIMB	0.977	0.984	0.984
KWI	0.936	0.959	0.956
VWM	**0.978**	-	-
VWN	0.977	0.982	0.981
STRM	0.959	0.982	0.983
STRN	0.959	0.983	0.983
SA	0.879	0.891	0.879
SB	0.883	0.897	0.885

Table 3. Training accuracy of KNN on Original dataset, PCA, Backward Elimination

Runs	R^2 Values of Training Set for KNN		
	Without Reduction	After Applying PCA	After Backward Elimination
Run#1	0.9625	0.9693	0.9852
Run#2	0.9627	0.9695	0.9853
Run#3	0.9628	0.9695	0.9854
Run#4	0.9628	0.9695	0.9854

Table 4. Validation accuracy of KNN on Original dataset, PCA, Backward Elimination

Runs	R^2 Values of Validation Set for KNN		
	Without Reduction	After Applying PCA	After Backward Elimination
Run#1	0.9351	0.9357	0.9836
Run#2	0.9352	0.9359	0.9837
Run#3	0.9353	0.9360	0.9837
Run#4	0.9353	0.9360	0.9838

Tables 3, 4 shows the training accuracy and validation accuracy on original data, data after applying PCA, and data after removing two independent variables decided by the backward elimination technique for feature reduction. The Regressor used is the KNN Regressor with the number of nearest neighbors 5.

Table 5. Training accuracy of Random Forest on Original dataset, PCA, Backward Elimination

Runs	R^2 Values of Training Set for Random Forest		
	Without Reduction	After Applying PCA	After Backward Elimination
Run#1	0.9898	0.9896	0.9857
Run#2	0.9899	0.9897	0.9858
Run#3	0.9899	0.9897	0.9858
Run#4	0.9899	0.9897	0.9858

Table 6. Validation accuracy of Random Forest on Original dataset, PCA, Backward Elimination

Runs	R^2 Values of Validation Set for Random Forest		
	Without Reduction	After Applying PCA	After Backward Elimination
Run#1	0.9897	0.9882	0.9840
Run#2	0.9898	0.9883	0.9842
Run#3	0.9898	0.9884	0.9842
Run#4	0.9898	0.9883	0.9843

Tables 5, 6 shows the training accuracy and validation accuracy on original data, data after applying PCA, and data after removing two independent variables decided by the backward elimination technique for feature reduction. The Regressor used is the Random Forest Regressor with the number of estimators 100, and gini index as the criterion.

Table 7. Test accuracy on KNN and Random Forest Regressors

Runs	R^2 value of KNN on test set	R^2 value of Random Forest on test set
Run#1	0.9839	0.9844
Run#2	0.9804	0.9845
Run#3	0.9840	0.9845
Run#4	0.9841	0.9846

Table 7 shows the final test accuracy on KNN and Random Forest regressors after applying the backward elimination. It is clearly shown in the table that the highest accuracy is obtained i.e. 98.46% by Random Forest Regressor. Therefore, for calculating and minimizing the computational time, proposed methodology provides a way to predict the execution time for the given hardware features.

VI. CONCLUSION AND FUTURE SCOPE

Today, when the price of systems increase with an increase in the number of hardware equipment, it is very important to calculate the requirements of a project in terms of its computational time and space efficiently. For calculating and minimizing the computational time, results analysis shows that the best result can be achieved from random forest Regressor after removing two features namely “VWM” and “MDIMA” from the set of independent variables. The proposed methodology is able to predict the running time of SGEMM GPU kernel (in milliseconds) with a test accuracy of 98.46%. In future, this area of study can be increased by adding more features to the dataset. Also, predictions can be done by using various other regressors such as SVM and ANNs.

VII. REFERENCES

[1] C. Gregg and K. Hazelwood, "WherC. Gregg and K. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer," (IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, 2011, pp. 134-144. doi: 10.1109/ISPASS.2011.5762730

[2] Ballester-Ripoll, Rafael et al. “Sobol Tensor Trains for Global Sensitivity Analysis.” CoRR abs/1712.00233 (2017): n. Pag.

[3] C. Nugteren and V. Codreanu, "CLTune: A Generic Auto-Tuner for OpenCL Kernels", IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, Turin, 2015, pp. 195-202.

[4] J. López-Fandiño, P. Quesada-Barriuso, D. B. Heras and F. Argüello, "Efficient ELM-Based Techniques for the Classificationof Hyperspectral Remote Sensing Images on Commodity GPUs", in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 8, no. 6, pp. 2884-2893, June 2015.

[5] M. Song, Y. Hu, H. Chen and T. Li, "Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures", IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 1-12.

[6] Baldini, I., Fink, S. J., & Altman, E. (2014), “Predicting GPU performance from CPU runs using machine learning”, In Proceedings – Symposium on Computer Architecture and High Performance Computing, pp 254-261.

[7] Konstantinidis, E., & Cotronis, Y., “A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling”, Journal of Parallel and Distributed Computing. Vol 107, sep 2017, pp 37-56, ISSN 0743-7315.

[8] Lin, Y., Liu, T., & Fuh, C., “Multiple Kernel Learning for Dimensionality Reduction”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, vol 33, no.-6, june 2011, pp 1147-1160.

[9] Matsumoto, K., Nakasato, N., & Sedukhin, S. G. (2012). “Implementing a code generator for fast matrix multiplication in OPENCL on the GPU”, In Proceedings - IEEE 6th International Symposium on Embedded Multicore SoCs, MCSoC, Nov 2012.

[10] Nakasato, N. (2011). “A fast GEMM implementation on the cypress GPU”, ACM SIGMETRICS Performance Evaluation Review, vol 38, issue 04, March 2011, pp 50-55.

[11] Ardalani, Newsha et al. “Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance.” *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2015): 725-737.

[12] J. Dean, D. Patterson and C. Young, "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution," in *IEEE Micro*, vol. 38, no. 2, pp. 21-29, Mar./Apr. 2018.

[13] M. Zhu, Y. Zhuo, C. Wang, W. Chen and Y. Xie, "Performance Evaluation and Optimization of HBM-Enabled GPU for Data-Intensive Applications," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 831-840, May 2018.