BUAN 6341 Applied Machine Learning

ASSIGNMENT NO 1

SGEMM GPU Kernel Performance Prediction

Introduction:

The objective of this project was to predict the 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 based on 14 different configuration parameters of processor such as workgroup level and size, local memory shape, kernel loop unrolling factor, vector widths for loading and storing, enable stride for accessing off-chip memory within a single thread etc. Many techniques such as Gradient descent algorithm, linear regression model and logistic regression model were used, and various experimentation were performed.

About the Dataset:

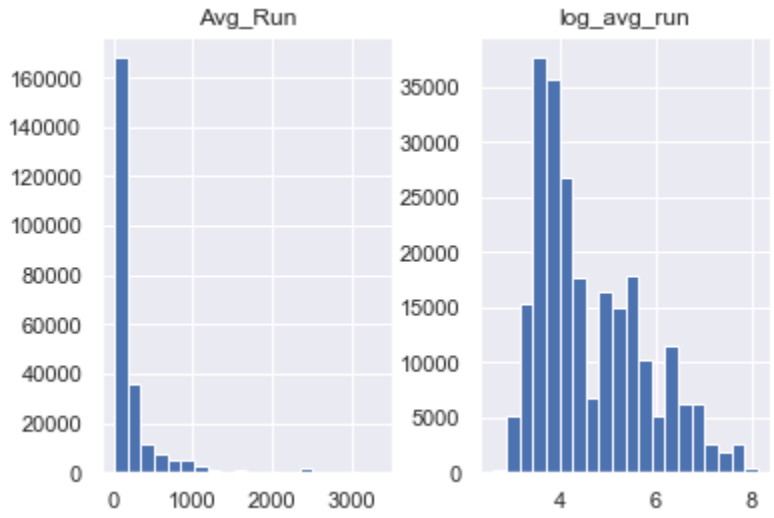
For this project, we have used the SGEMM GPU kernel performance Data Set. This data set measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations. For each tested combination, 4 runs were performed, and their results are reported as the 4 last columns. Below is the brief description about the dataset-

- There are 14 parameters, the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary.
- Out of 1327104 total parameter combinations, only 241600 are feasible (due to various kernel constraints). This data set contains the results for all these feasible combinations.
- The predictor variable or dependent variable for linear regression variable y is created by taking average of 4 run times.
- The predictor variable or dependent variable for logistic regression variable y is created by taking median and by assigning value 1 for above the threshold and value as 0 for below threshold.
- There is no NA or missing values in our dataset.
- The distribution of average run time is not normal as we have left asymmetry, for this reason I have used log (average run time) in my analysis for which distribution is more normal.

Please note that the experiment to obtain this dataset was run on a desktop workstation running Ubuntu 16.04 Linux with an Intel Core i5 (3.5GHz), 16GB RAM, and a NVidia GeForce GTX 680 4GB GF580 GTX-1.5GB GPU.

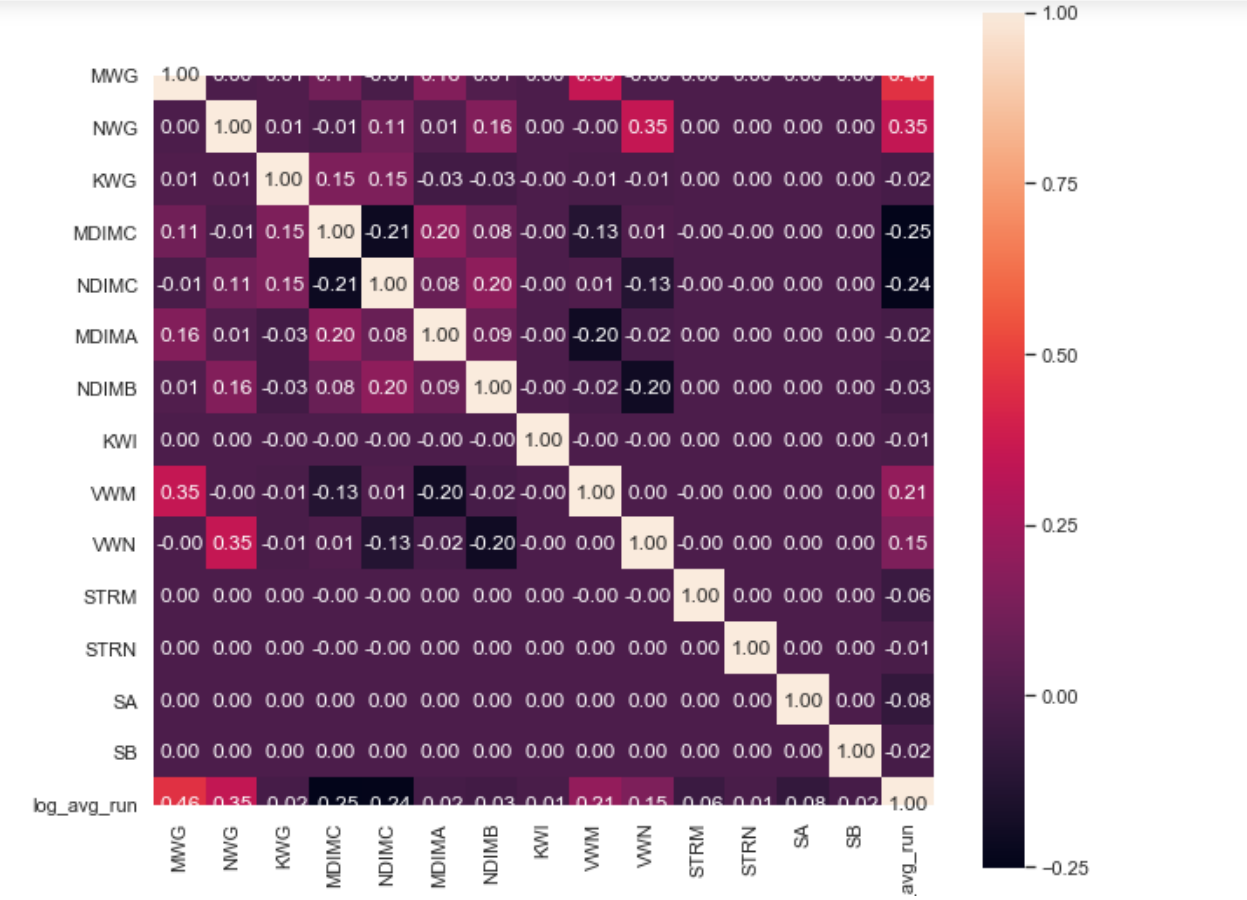
Data Preparation and Exploratory Data Analysis:

The objective of linear regression model is to accurately predict GPU run time. As seen in the figure below, the distribution of average GPU Run time is not normal and found to be right skewed , so we will convert the values of average GPU run time to log scale for better modeling and as shown from below figure distribution of log average GPU run time is comparatively normal.

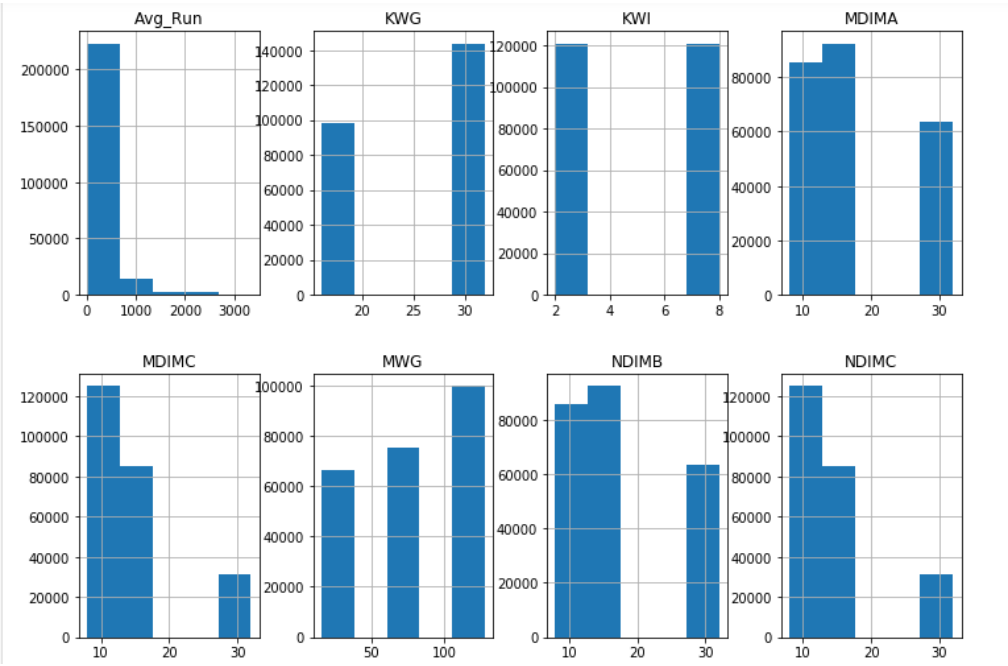


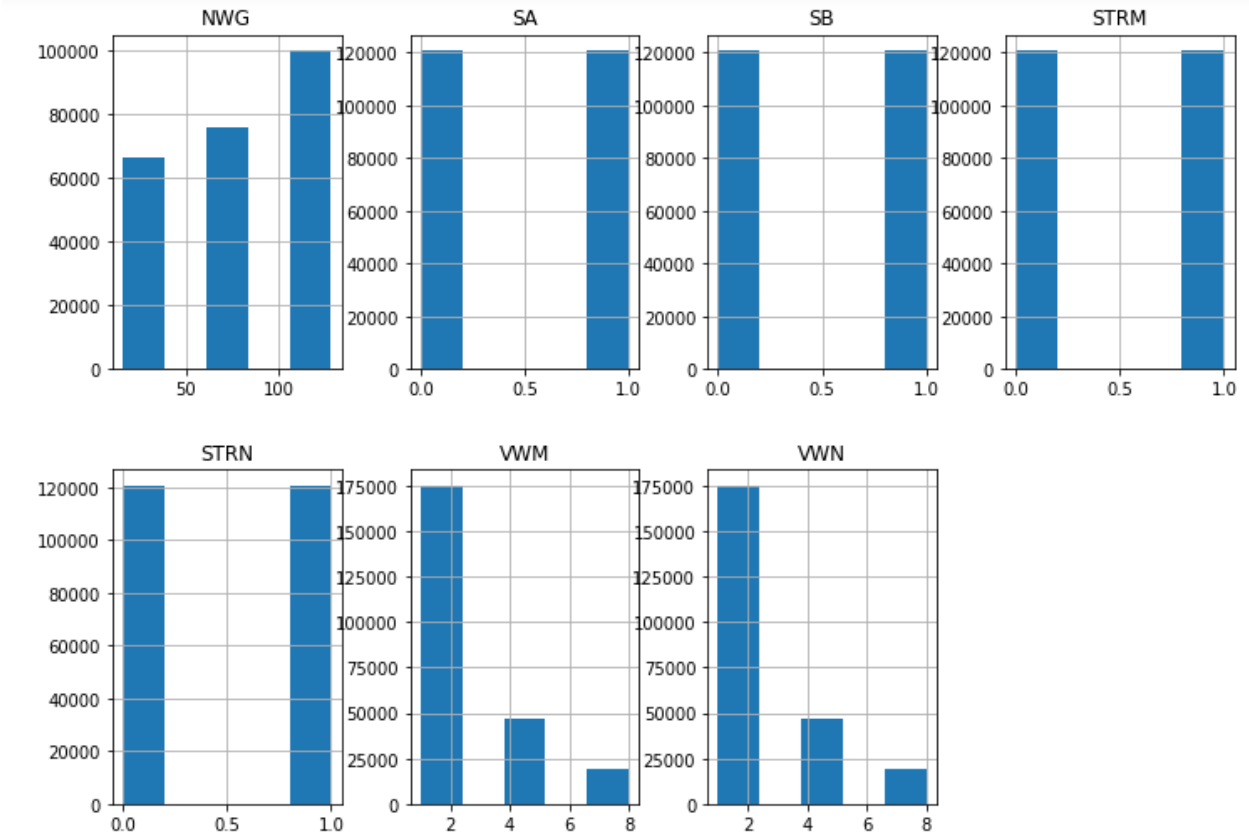
As checked, there is no missing value in this dataset.

Correlation Matrix: From the correlation matrix as shown below, we can conclude that there is negative correlation between MDIMC and NDIMC (-0.21). Both MDIMC and NDIMC are negatively correlated with log average run time. MDIMC and VWM are negatively correlated (-0.13), NDIMC and VWN are negatively correlated (-0.13). MDIMA and VWM are negatively correlated (-0.20), NDIMB and VWN are negatively correlated (-0.20).



Feature Distribution: To understand the distributions of the features, I have plotted the histogram of all the features –





Standardization:

For our algorithm to perform better we have performed Standardization which involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

The idea behind performing standardization here so that our algorithm performs better or converge faster when features are on relatively smaller scale or close to normally distributed.

Experiment 1:

The linear regression equation of model is:

$$\text{log\_avg\_run} = \beta 0 + \beta 1 * \text{MWG} + \beta 2 * \text{NWG} + \beta 3 * \text{KWG} + \beta 4 * \text{MDIMC} + \beta 5 * \text{NDIMC} + \beta 6 * \text{MDIMA} + \beta 7 * \text{NDIMB} + \beta 8 * \text{KWI} + \beta 9 * \text{VWM} + \beta 10 * \text{VWN} + \beta 11 * \text{STRM} + \beta 12 * \text{STRN} + \beta 13 * \text{SA} + \beta 14 * \text{SB}$$

As part of this experiment, First I have taken initial coefficients as 0 and data set is split in 70:30, i.e. 70% training set and 30% test set. This is done using train_test_split function from sklearn library.

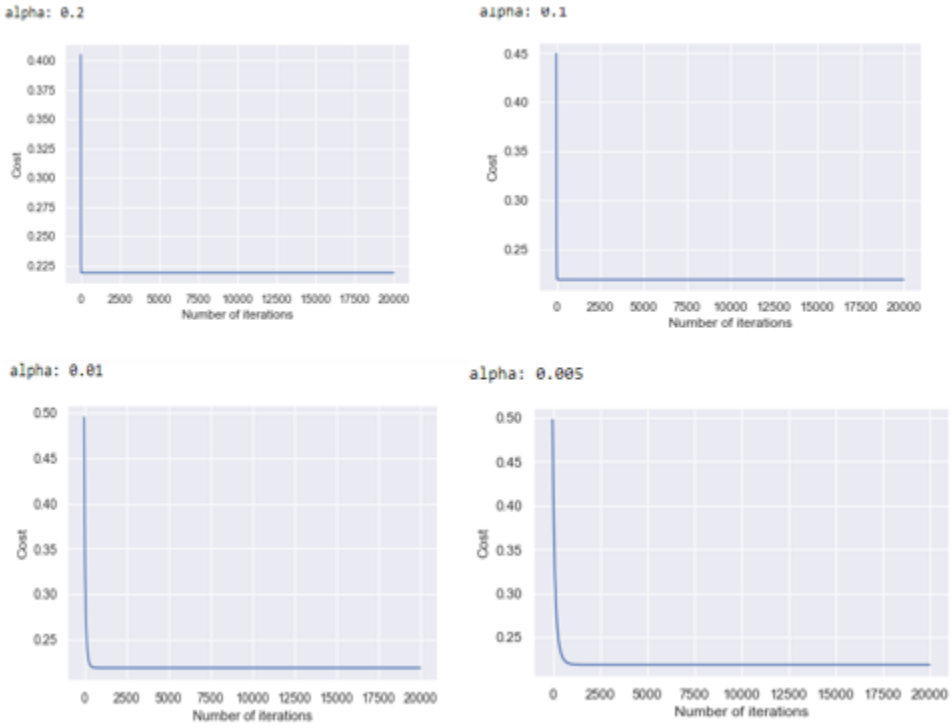
For this experiment, I have experimented with two thresholds i.e. 0.0000001 with 50,000 number of iterations and 0.001 with 20,000 number of iterations. And various learning rates are

used in the experiment i.e. 0.2, 0.1, 0.01, 0.005 and 0.0001. With these parameters I have calculated various parameters such as Cost Value, Converging Point, Train MSE and Test MSE.

Threshold = 0.0000001			No of iterations = 50000	
Learning Rate	Cost Value	Converging Point	Train MSE	Test MSE
0.2	0.218580188	56	0.43715949	0.43881548
0.1	0.218580707	109	0.43715949	0.43881548
0.01	0.218589294	907	0.43715949	0.43881548
0.005	0.218598763	1686	0.43715949	0.43881548
0.0001	0.209531191	47157	0.41857115	0.41025014

Conclusion: As evident from the above table, the best alpha value is 0.0001 function where we received cost value as 0.2095311 and train MSE as 0.41857 and test MSE as 0.41025.

The variation of cost function for different learning rate is plotted as below –



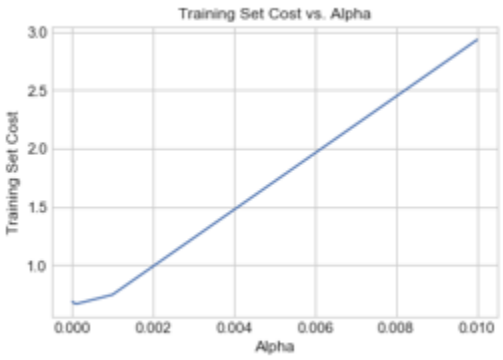
Logistic Regression:

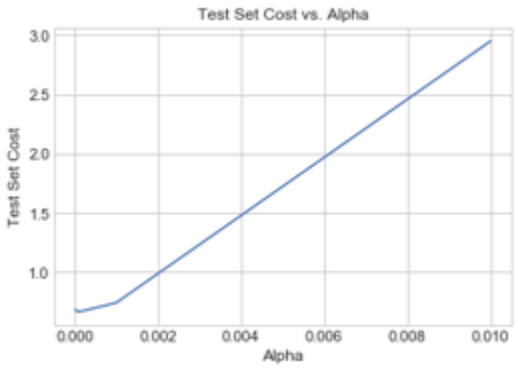
For Logistic Regression, I have created a new variable called "**gpuruntime_class**". It is a categorical variable and it takes "1" for log average GPU run time greater than 4.24 (median value) and "0" for less than 4.24.

Logistic regression algorithm does not make many of the key assumptions of linear regression and general linear models that are based on ordinary least squares algorithms – particularly regarding linearity, normality, homoscedasticity, and measurement level.

First, logistic regression does not require a linear relationship between the dependent and independent variables. Second, the error terms (residuals) do not need to be normally distributed. Third, homoscedasticity is not required. Finally, the dependent variable in logistic regression is not measured on an interval or ratio scale.

In this experiment, for 10 iterations, Minimum cost for training set is 0.665871 at alpha = 0.0001. Minimum cost for test set is 0.665318 obtained at same set of beta values.





Conclusion: Therefore, Best alpha value obtained from Logistic Regression is 0.0001.

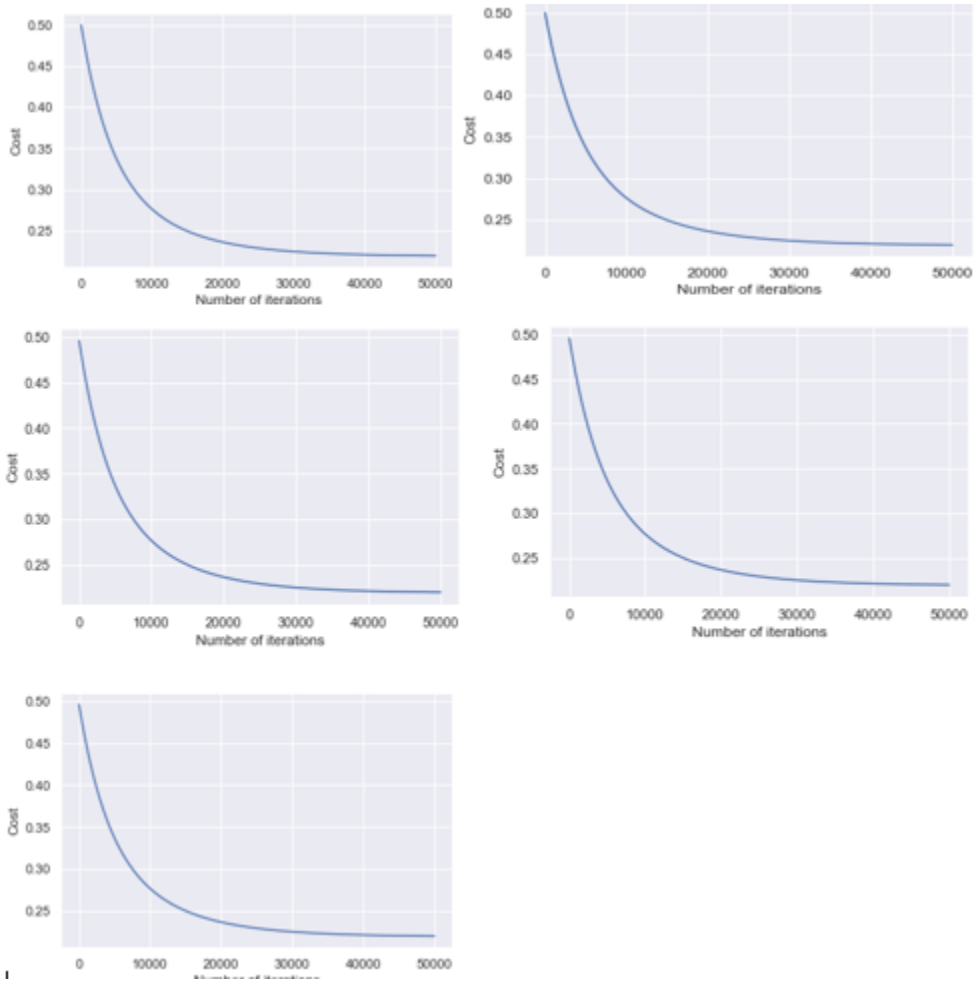
Experiment 2: In this experiment, we observed that for higher threshold values, algorithm converged fast but increasing threshold value also increases the cost as well as it can be shown from below observations that for threshold 0.0000001 first converging point received for learning rate 0.2 is 56, whereas for threshold 0.001 first converging point received for learning rate 0.2 is 14 –

Threshold = 0.0000001			No of iterations = 50000	
Learning Rate	Cost Value	Converging Point	Train MSE	Test MSE
0.2	0.218580188	56	0.43715949	0.43881548
0.1	0.218580707	109	0.43715949	0.43881548
0.01	0.218589294	907	0.43715949	0.43881548
0.005	0.218598763	1686	0.43715949	0.43881548
0.0001	0.209531191	47157	0.41857115	0.41025014

Threshold = 0.001			No of iterations = 20000	
Learning Rate	Cost Value	Converging Point	Train MSE	Test MSE
0.2	0.223420528	14	0.43715949	0.43881548
0.1	0.227784962	24	0.43715949	0.43881548
0.01	0.289575109	83	0.43715949	0.43881548
0.005	0.283284324	92	0.43715949	0.43881548
0.0001	0.279946455	102	0.42601706	0.42498195

In this experiment, the threshold value chosen are 0.001, 0.0001, 0.00001, 0.000001, 0.0000001 at the fixed learning rate of alpha 0.0001.

For Learning Rate 0.0001			
Threshold	Cost Value	Train MSE	Test MSE
0.001	0.499946455	0.43857115	0.44025014
0.0001	0.499946455	0.43857115	0.44025014
0.00001	0.290002663	0.43857115	0.44025014
0.000001	0.227846713	0.43857115	0.44025014
0.0000001	0.219531191	0.43857115	0.44025014



Conclusion: There are 2 main observations from above table –

- The best threshold value obtained in this case is 0.0000001 since we got less cost value at this threshold as compare to other thresholds for learning rate 0.0001
- Train MSE and Test MSE is approx. same for different threshold values.

Experiment 3: In this experiment, a selection of 8 random features (MWG, KWG, MDIMC, MDIMA, KWI, VWN, STRN, SB) were performed.

The linear regression equation of model would be:

$$\log\_avg\_run = \beta 0 + \beta 1 * MWG + \beta 2 * KWG + \beta 3 * MDIMC + \beta 4 * MDIMA + \beta 5 * KWI + \beta 6 * VWN + \beta 7 * STRN + \beta 8 * SB$$

Below table represents the comparison between Random Feature Selection and Best Selection Features with alpha value as 0.0001 with threshold value as 0.0000001, we have observed the Cost Value, Converging Point, Train MSE, Test MSE, Train R Square and Test R Square –

Feature Selection	Alpha = 0.0001			Threshold = 0.0000001		
	Cost Value	Converging Point	Train MSE	Test MSE	Train R Square	Test R Square
Random Features	0.336007477	31758	0.67092076	0.67311134	0.329079237	0.321082619
Best Selection Features	0.225673668	41246	0.45238357	0.45239629	0.543714311	0.543714311

Conclusion: There are 3 main observations from above table –

- The Cost Value of the model with 8 random features is very high i.e. 0.336 as compared to the cost value of the model with 8 best selected features which is 0.2256.
- Training R Square and Test R Square value i.e. 0.32 of random featured model is less than as compare to best selection feature model which is 0.543.
- The Errors of random feature selection is more than the best selection feature and also with the model with all variables.

Experiment 4:

A selection of 8 best suited features (MWG, NWG, KWG, MDIMC, NDIMC, MDIMA, NDIMB, KWI) were performed.

The linear regression equation of model would be:

$$\log\_avg\_run = \beta 0 + \beta 1 * MWG + \beta 2 * NWG + \beta 3 * KWG + \beta 4 * MDIMC + \beta 5 * NDIMC + \beta 6 * MDIMA + \beta 7 * NDIMB + \beta 8 * KWI$$

Feature Selection	Alpha = 0.0001			Threshold = 0.0000001		
	Cost Value	Converging Point	Train MSE	Test MSE	Train R Square	Test R Square
Random Features	0.336007477	31758	0.67092076	0.67311134	0.329079237	0.321082619
Best Selection Features	0.225673668	41246	0.45238357	0.45239629	0.543714311	0.543714311
Model with all Features	0.219531191	47157	0.41857115	0.41857115	0.553714311	0.553714311

Conclusion: There are 2 main observations from above table –

- The Cost Value of the model with 8 best selection features is less i.e. 0.225 as compared to the cost value of the model with 8 random features.
- The 8 selected features performed better as compare to random chosen features but didn't perform better as compare to model with all features because we are restricting the important features from the model.

What do you think matters the most for predicting the value and category/class of GPU run time?

To predict the value and category/class of GPU run time, we checked AIC value in R as shown below –

	Df	Sum of Sq	RSS	AIC
+ SA	1	88746165	1.9568e+10	2730609
+ KWI	1	34851386	1.9622e+10	2731273
+ NDIMB	1	29993465	1.9627e+10	2731333
+ MDIMA	1	25935767	1.9631e+10	2731383
+ VWN	1	14899757	1.9642e+10	2731519
+ VWM	1	6300626	1.9650e+10	2731625
+ STRM	1	5203966	1.9652e+10	2731638
<none>			1.9657e+10	2731700
+ STRN	1	382	1.9657e+10	2731702

Since AIC value of feature SA (per-matrix manual caching of the 2D workgroup tile) is less as compare to other features, residual sum of squares is also less as compare to other features, therefore we can say that feature SA matters the most. As observed for other features such as KWI, NDIMB, MDIMA, VWN, VWM, STRM, STRN AIC value and residual sum of squares value is almost same therefore in this model, almost all the features hold the equal importance for predicting the GPU run time.

What other steps you could have taken with regards to modeling to get better results?

Below are some of the steps which can be considered to get better results –

- Hyperparameter tuning can be performed better using algorithm such as Decision Tree Regressor, Support Vector Regressor, Gradient Boosting and Neural Networks etc., in order to get lower MSE values.
- Performance can be improved using ensembles where we can often get good predictions from combining the predictions from multiple “good enough” models rather than from multiple highly tuned models.