Machine Learning-based Performance Prediction of Parallel Applications using Hardware Counters

By **Jinang Rupeshbhai Shah** S190070
Department of Electrical Engineering, Indian Institute of Technology Kanpur.

SURGE 2019
Indian Institute of Technology Kanpur

Under the supervision of **Dr. Preeti Malakar**
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur.

Introduction

- Modelling the characteristics and performance of applications in an automatic way has been a long-standing goal of computer science research. In a recent times, Multicore and multithreaded processors have a prominent role in high-end computing. And thread-level parallelism has long been recognized as the means to sustain performance scaling and overcome the limitations of conventional techniques for exploiting instruction level parallelism.
- Applications of high-end computing are being designed to process at different core levels to break the existing heights of achievable performance. With designing parallel application, it is now crucial to better its performance and that's when the dice of analysing performance was rolled out.
- With recent developments in machine learning,. We propose to use the performance counters for building the supervised learning models which can then predict the outcome. Drop in the performance can be a cause of many events and thus, these events has been selected after understanding utility of each event and more from the existing research.
- For our analysis, we have applied our methodology to multiple supervised learning models including Linear Regression, Decision Trees, Extra Trees Regression and XGBoost Regression. We have tested our models on standard existing frameworks of MiniFE and CoMD applications. Besides these counters, runtime configuration of parallel applications, is also considered as a variable in the training process of a model.
- From gathered results and following evaluation, we believe our paper suggests that such models represent the ability to transform prediction methods. Thus far more efficient and profitable development phase can be achieved through these methods and similar research.

Performance Prediction

- Counters are used to provide information as to how well the operating system or an application, service, or driver is performing. The counter data can help determine system bottlenecks and fine-tune system and application performance. They are part of hardware that counts, measures the events in software, that allow us to see performance patterns of an application from high-level view.
- We have proposed to use the performance counters for our prediction of performance and to accomplish that an interface is required to provide users ability to read, see and in a way to control the hardware counters. For our purposes, we have used **Performance Application Programming Interface (PAPI)**.
- Performance counters are used to count performance events which relates to various occurrences that in a way affects the performance. Some of the known events are cache misses, branch predictions and instruction execution etc.
- For our analytical purposes, here we have selected the seven events that are expected to affect the performance greatly from the existing work. These selected events are given below with their respective utility.

Events	Utility
PAPI_L1_TCM	Total L1 cache misses
PAPI_L2_TCM	Total L2 cache misses
PAPI_L1_LDM	Total L1 load data misses
PAPI_L2_LDM	Total L2 load data misses
PAPI_BR_PRC	Conditional branch instructions correctly predicted
PAPI_RES_STL	Total cycles stalled on any resource
PAPI_DP_OPS	floating-point operations

- For our analysis, execution time is the parameter we are considering for analysing performance where event PAPI_TOT_CYC gives the total cycles over runtime which directly correlates to the execution time. The training dataset contains eight events with configurational features.

Implementation

- For our motivation we have found PAPI to be useful for acting as an interface that provides users ability to read, measure or control the counters. The main file of the application has to be changed appropriately in order to get access to counters through PAPI.
- We have used MiniFE and CoMD applications for our evaluation. They both are considered as standard benchmarks to evaluate performance of parallel applications. MPICH library was used to implement the parallel processing where we have used **at max 60 cores** for our setup. Having data available for all the executed cores, average for seven performance events and maximum for the **PAPI_TOT_CYC** were stored in the dataset accordingly.
- **MiniFE has in total 12 features whereas CoMD has 14 including configurational features.** Normalization for each column of the dataset is necessary in order to achieve higher accuracy and sustainability for the model.
- Normalized dataset was then applied to multiple machine learning models such as Linear Regression, Decision Trees, Extra Trees Regressor and XGBoost Regressor for further evaluation in predicting the total cycles for a parallel applications. Here, MiniFE and CoMD were separately subjected to these models with their dataset.

Machine Learning Models

- With the recent trend of research in the field of machine learning, we have been introduced to various kinds of models in a very short span of time. A set of machine learning techniques has been used to build the respective models that can predict the performance of parallel application, which are described below. Variables to these models are also referred **features**. For our analysis, features are the seven events with configurational features that are selected from the existing research.
- Supervised learning models that are used for the evaluation are described below:

❑ **Multiple Linear Regression :** It is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables. the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data of the features.

❑ **Decision Trees :** It represents a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. The attached figure shows the decision tree chart for playing tennis with different possible weather forecasts.

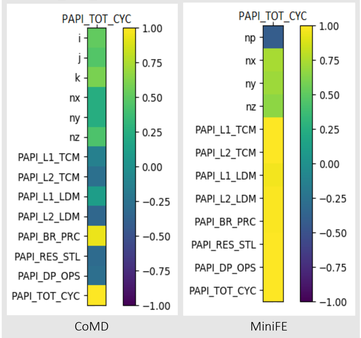
❑ **Extra Trees Regressor :** Decision trees, being prone to overfit, have been transformed to random forests by training many trees over various subsamples of the data. In extra trees regressor, instead of computing the locally optimal feature combination, for each feature under consideration, a random value is selected for the feature that leads to much faster implementation than random forest itself.

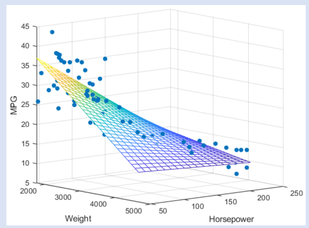
❑ **XGBoost Regressor :** It is an implementation of gradient boosted decision trees designed for speed and performance. The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model.

- These models were then subjected to training data collected from the normalized dataset. The learning models were evaluated over different range of test size. And the results were collected after taking average from 10 iterations of model training with different training dataset each time.

Correlation Matrix

- Here Each cell in the spectrum represents the correlation between respective feature in row and 'PAPI_TOT_CYC'.

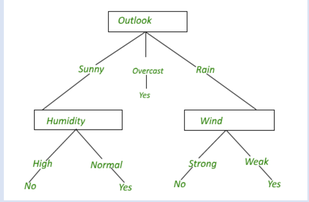




https://in.mathworks.com/help/stats/regress.html

Attached figure shows the implementation of **Multiple Linear Regression** on a sample dataset of cars. It maps data from cars' features such as "Weight" and "Horsepower" to its "Mileage(MPG)". In 3D, The mapping takes the shape of Plane.

Decision Tree for *PlayTennis*



https://www.geeksforgeeks.org/decision-tree

Given figure shows the **decision tree** chart for playing tennis with different possible weather forecasts. It evaluates the possibilities and takes the decisions whether tennis should be played or not.

Results and Analysis

Model	Test_size	R2 Score	Error(%)
Multiple Linear Regression	0.2	0.9984	2.75
	0.3	0.9984	2.82
	0.5	0.9983	2.92
Decision Trees	0.2	0.9957	4.95
	0.3	0.9959	4.75
	0.5	0.9952	5.05
Extra Trees Regressor	0.2	0.9976	3.67
	0.3	0.9974	3.72
	0.5	0.9973	4.06
XGBoost Regressor	0.2	0.9975	3.72
	0.3	0.9976	3.81
	0.5	0.9968	4.21

TABLE 1

Model	Test_size	R2 Score	Error(%)
Multiple Linear Regression	0.2	0.9151	15.87
	0.3	0.9116	15.81
	0.5	0.9113	15.53
Decision Trees	0.2	0.8724	13.31
	0.3	0.8638	13.53
	0.5	0.8659	13.57
Extra Trees Regressor	0.2	0.9320	9.843
	0.3	0.9306	10.20
	0.5	0.9300	10.15
XGBoost Regressor	0.2	0.9259	10.23
	0.3	0.9234	10.69
	0.5	0.9176	10.77

TABLE 2

- Table 1 and Table 2 present the data collected over various configurations and models. Each test case was run over ten different random states and the average of those values are presented in these tables. Here evaluation is based on two parameters, R2_score and Error. **For MiniFE, a total of 732 tests we conducted whereas for CoMD, the results were collected from the total dataset of 2832 test cases.**
- R2 score is a statistic used in the context of statistical models whose main purpose is the prediction of future outcomes. It provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. For our purposes, we have used the Median Absolute Percentage Error for the Error parameter, which was calculated as the median of absolute percentage error, where **Absolute Percentage error** = $100 * \left| \frac{\textbf{predicted - actual}}{\textbf{actual}} \right|$
- For MiniFE, we were able to achieve R2 score of more than 0.995 for all the configurations with different models. With Multiple Linear Regression and test size of 0.2, we were able to achieve error for as low as 2.75 percent and were able to maintain error in **5.05** for all the tested cases. This shows the remarkable correlation with our selection of events and the execution time.
- For CoMD, road is not as smooth as of MiniFE. Despite achieving R2 score of more than 0.86 for all the test cases, the error was lying in the range of **9.84 to 15.53**. Achieved R2 score was good but the error was on and average more than 10 percent. This suggests that it is more complicated for the CoMD to predict its execution time from selected seven events.
- The derived results can also be understood from correlation matrices of MiniFE and CoMD. For MiniFE, all the seven selected performance events seem to be in greater correlation with PAPI_TOT_CYC which is also reflecting on its results. Due to higher correlation between features and target, we are able to build a supervised learning models with lower error margins and higher accuracy in predicting the performance. Whereas for CoMD, the correlation between the selected events and PAPI_TOT_CYC is not good as it is with MiniFE. And which led to the clouding of accurate prediction with the machine learning model and the error turned out to be higher.

Conclusion

- We have implemented multiple machine learning models to test if the selected performance events can give us the idea of performance as whole. The results are suggesting that it is relatively tough to predict the performance for CoMD than MiniFE with the selected performance events. But even further, with some more analysis and effort, a more appropriate set of events can be identified for CoMD. But in general, these results back our proposition for predicting performance of a parallel application using performance counters. This results add one more shine in the utilities of performance counters. We believe that the similar research and more utilization of performance counters can lead us to show a more easy and feasible paths to the complex problems in the area.