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

SURGE 2019

Submitted by Jinang Rupeshbhai Shah Roll No: S190070
Department of Electrical Engineering, Indian Institute of Technology Kanpur,
Kanpur, UP, India – 208016.

Under the supervision of Prof. Preeti Malakar
Assistant Professor, Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur,
Kanpur, UP, India. 208016

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

TABLE OF CONTENTS

ABSTRACT

Our existing architectures have become so complex and have so many variables involved within that it leaves us with no obvious path to predicting performance of a parallel application. In these times of getting better performance with time as well as with power, it has become critical and inevitable to implement multi core processors across the devices, which then gave birth to parallel programming and parallel applications. Using the performance counters is one of the many ways to predict performance of a parallel application. It is relatively easier and rational idea to approach with. Having so many performance events, it creates the scepticism about the relevance of these many events with predicting performance of a particular application. Here we have selected seven events that are expected to affect the performance greatly from existing work. Iterative script was then used to collect the data from described counters which then transformed into datasets to be implemented in machine learning models. This paper demonstrates various models, Linear Regression, Decision Trees, Extra Trees Regressor and XGBoost for performance prediction. 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 process. With MiniFE we were able to achieve R2 score for more than 0.995 and median absolute percentage error of less than 4.5% for all the tested cases. For CoMD, 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. Which suggests that for CoMD, it Is more complex for the prediction of its performance using our selected performance events, which can be further improved. 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.

Keywords: Performance Prediction, Machine Learning, Hardware Performance Counters, Performance Events

ACKNOWLEDGEMENT

First and foremost, I would like to appreciate my family for always supporting me and encouraging me to do something different and unique. Thank you for always making me feel special.

I am deeply indebted to my project supervisor, Prof. Preeti Malakar for her valuable guidance, fruitful discussion, constant support and encouragement throughout my SURGE program and for inculcating in me the spirit of project work. I am really thankful for showing your confidence even in the times when I thought I was not going anywhere and believing me in that moment.

Further, I express my sincere gratitude to Surge program Prof. In charge Prof. Sudhir Kamle for constant maintaining supportive and healthy environment throughout the SURGE program. He really put in effort to ensure that we feel comfortable and smoothly adopt the new paradigm of research.

I would also like to thank IIT Kanpur for giving me this opportunity to understand the basics and be familiar with research area.

1. Introduction

Modelling the characteristics and performance of applications in an automatic way has been a long-standing goal of computer science research. Multicore and multithreaded processors have a prominent role in high-end computing. 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. Although multicore and multithreaded architectures have proven their potential and have been adopted in several commercial products, the programming technologies used for these architectures date from almost two decades ago.

Applications of high-end computing are designed to process at different core levels simultaneously to give a better performance to user. Nowadays almost every computing devices contain more than a core and there it gave birth to a new prism of parallel computing. With designing parallel application, it was crucial to better its performance and that’s when the dice of analysing performance was rolled out. Analysing performance requires an application to be run multiple times to gather enough data. In present times when software applications and respective updates have serious time constraints on development phase, it is not rational for developers to spend this much time on analysing the performance. A way was needed to be developed to predict the performance of a parallel application without having to run it.

Application performance monitoring is not new, but in the past, it was limited to the application development phase; it was just used to ensure that the application met demands at the time of deployment. In recent years, software applications have evolved to be more complex, dynamic, robust and distributed in nature. The current demand of the next-generation corporate world is to make applications available anytime, anywhere on any device. To meet this demand, the importance of application performance monitoring has also increased many folds. And recent development in machine learning has given us a chance to reshape the crystal of performance monitoring.

Machine learning provides systems the ability to automatically learn and improve from experience without being explicitly programmed. It focuses on the development of computer programs that can access data and use it learn for themselves. Machine Learning can be used to train a model that can predict performance of parallel application on the basis of defined variables. Here we propose to use the performance counters as variables to build a supervised learning model which can then predict the outcome.

In system, hardware counters are a set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems. The number of available hardware counters in a processor is limited while each CPU model might have a lot of different events that a developer might like to measure. Each counter can be programmed with the index of an event type to be monitored, like a L1 cache miss or a branch misprediction.

Due to limitation of number of hardware counters one can only count limited number of events simultaneously. Drop in performance can be a cause of many events and we propose to introduce multiple models based on the selected events to predict the performance of a parallel application. It is supposed to be much feasible and effective way of predicting performance which can then be tested by various standard benchmarks. The benchmarks that we have used here to analyse are MiniFE and CoMD. Multiple models are applied to test the effectiveness of this proposal that to be able to predict the performance, in terms of selected set of performance events.

This paper is organized as follows. The next section provides a brief idea on our methodology and the aim we want to pursue. It also briefly explains performance counters, and machine learning models that we have used throughout the assessment. Section 3 familiarizes with the architecture, Software, libraries and applications used to conduct the experiment. It also describes the experimental setup and its implementation at length. Section 4 talks about error analysis and results followed by its discussion. This is followed by concluding remarks on our project.

2. Performance Prediction

2.1 Problem Statement

It's been a while since multicore processers were introduced and eventually it has established its necessity for higher performance. As most of high-end computing is done using parallel computing, it has become essential to establish a general approach towards dealing with performance of parallel applications. We present a proposal to use performance counters to predict the performance. From recent developments in machine learning, it has enabled us to design and build accurate and sustainable predictive models. Using the data gathered through performance counters we have built a multiple supervised learning models to establish our proposition. A detailed and thorough analysis of our proposed methodology is described in the following part of the paper.

2.2 Performance Counters

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. The operating system, network, and devices provide counter data that an application can consume to provide users with a graphical view of how well the system is performing. They are part of hardware that counts, measure 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 must to read the counter values at any requires time. This interface provides users ability to read, see and in a way to control the hardware counters. Performance counters are used to count performance events which relates to various occurrences which in a way affects the performance of respective application. Some of the known events are cache misses, branch predictions and instruction execution etc. For our purposes, we have found performance application programming interface (PAPI) to be useful as interface.

2.2.1 Interface for reading counters

As mentioned in Introduction, predicting performance can be done in various ways, but choosing hardware counters is the most feasible and still effective way to dealing with performance. In recent times, there has been already a bulk of research done in the many ways we can use the data gathered from hardware performance counters. And it has shown us the importance and significance of these counters in our systems, that is why nowadays almost all the systems have the hardware counters built in themselves. We have used Performance Application Programming Interface (PAPI) to read the counter values for selected events. And to read the counters using PAPI, code of the application has to be modified appropriately to get the counter values from all the cores which were included in execution. Further information on the modification is given in the later section of Experimental Setup.

2.2.2 Event Selection

In predicting performance, event selection is always a crucial part. There's always a doubt about correlation of these many existing events to the performance of an application. The primary aim is to select those events that can correlate in general to the performance as much as possible. There are so many events that can be in interdependence with the performance either directly or indirectly. And that is why it becomes critical to adopt a number of events that can give us sufficient idea of the performance of applications for a given configuration and existing architecture. After studying the utility of different events and from the existing research, we have selected in total following seven events for our analysis.

No.	Performance Events	Utility
1	PAPI_L1_TCM	Total L1 cache misses
2	PAPI_L2_TCM	Total L2 cache misses
3	PAPI_L1_LDM	Total L1 load data misses
4	PAPI_L2_LDM	Total L2 load data misses
5	PAPI_BR_PRC	Conditional branch instructions correctly predicted
6	PAPI_RES_STL	Total cycles stalled on any resource
7	PAPI_DP_OPS	Floating-point operations

To analyse the performance, we have used the event **PAPI_TOT_CYC**, which gives the total cycles during runtime of application which directly correlates to the execution time. For our purposes, execution time is the parameter we are considering for analysing performance and total cycles gives the exact value for that.

2.3 Machine Learning Models

With the recent trend of research in the field of machine learning, we have been introduced to various kinds of models. A set of techniques were used to build the respective models that can predict the performance of application which are described below. Variables to these models are also referred **features**. In our analysis, features will be the seven events described above.

Multiple Linear Regression, 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, is 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.

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. This leads to much faster implementation than random forest itself.

XGBoost Regressor, 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. Some key algorithm implementation features include: **Sparse Aware** implementation with automatic handling of missing data values. **Block Structure** to support the parallelization of tree construction. **Continued Training** so that you can further boost an already fitted model on new data.

3. Experimental Setup

3.1 System Description

The system on which we have analysed our thesis is described below as to inform about the designing and architectural division of the cluster system that we have used for parallel application.

Class	Description
Bus	Cannon Lake PCH USB 3.1 xHCI Host Controller
Memory	15GiB System memory
Processor	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz
Bridge	8th Gen Core Processor Host Bridge/DRAM Registers
Generic	Xeon E3-1200 v5/v6 / E3-1500 v5 / 6th/7th Gen Core Processor Gaussian Mixture Model
Communication	Cannon Lake PCH HECI Controller
Storage	Cannon Lake PCH SATA AHCI Controller
Network	Ethernet Connection (7) I219-LM
Architecture	X86_64
CPU(s)	12
Threads per core	2
CPU MHz	855.644
L1 cache	32K
L2 cache	256K
L3 cache	12288K
Kernel Name	Linux
Kernel Release	4.15.0-50-generic
Kernel Version	#54-Ubuntu SMP Mon May 6 18:46:08 UTC 2019
OS Version	Ubuntu 18.04.2 LTS

3.2 Libraries used

3.2.1 MPICH

To analyse the performance of a parallel application, one must have the interface to interconnect different nodes altogether and should be able to make them communicate. Here to establish that communication we have taken use of Message passing Interface, and it is also referred “MPI”. MPI is a library specification for message passing, proposed as a standard by a broadly-based committee of vendors, developers and users. MPI was designed for high performance on both massively parallel machines and on workstation clusters. MPICH is a high performance and widely portable implementation of the **Message Passing Interface (MPI)** standard. MPICH and its derivatives form the most widely used implementations of MPI in the world. They are used exclusively on nine of the top 10 supercomputers (June 2016 ranking), including the world’s fastest supercomputer: Taihu Light. We have used the MPICH-3.3.1 for our implementation.

3.2.2 PAPI

As mentioned in 2.1, the interface used for reading performance counters in our application is Performance Application Programming Interface (PAPI). PAPI provides the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events. PAPI provides two interfaces to the underlying counter hardware; a simple, high level interface for the acquisition of simple measurements and a fully programmable, low level interface directed towards users with more sophisticated needs. Our application operates on PAPI-5.7.0. Modifications required to use PAPI is shown in the later part of implementations.

3.2.3 Bash

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. Bash is a command processor that typically runs in a text window where the user types commands that cause actions. Bash can also read and execute commands from a file, called a shell script. To collect the training and test datasets for our machine learning model, a script was built that can collect all the required performance counters and automatically stores them into appropriate files. Sample script to collect performance counters is given in the later discussion.

3.2.4 Python

Python is a widely used high-level programming language for general-purpose programming. Apart from being open source programming language, python is a great object-oriented, interpreted, and interactive programming language. Python is also usable as an extension language for applications written in other languages that need easy-to-use scripting or automation interfaces. Python along with R is gaining momentum and popularity in the

Analytics domain since both of these languages are open source. And that is why, python is widely considered as the preferred language for teaching and learning ML (Machine Learning). We have used python with some external library to perform the task and the short description of these libraries are as follow:

Numpy: NumPy is the fundamental package for scientific computing with Python. It can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Random: It is used to generate a random number within given range.
xgboost.XGBRegressor(**):** It is used to implement machine learning model for XGBoost Regressor with given multivariate dataset.
Scikit-learn: Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is distributed under many Linux distributions, encouraging academic and commercial use. The library is focused on modelling data not manipulating them. It provides some great features for modelling of data such as clustering, cross validation, regression, decision trees and multiple different models to analyse the data and ability of given data to generate supervised models on unseen data.

Some functions of scikit-learn used in our modelling are described below:

sklearn.model_selection.train_test_split(**arrays*, ***options*): Split arrays or matrices into random train and test subsets unless given the parameters.

sklearn.linear_model.LinearRegression(): Applies multiple linear regression to the dataset after splitting it into training and test dataset by former function

sklearn.tree.DecisionTreeRegressor(): Builds the supervised model of decision tree regressor with training dataset.

sklearn.ensemble.ExtraTreesRegressor(): implements Extra Trees Regressor

sklearn.metrics.r2_score(predicted values, test values): Finds the R2 score for the respective model using predicted and test values.

3.3 Applications used for Analysis

3.3.1 MiniFE

MiniFE is a proxy application for unstructured implicit finite element codes. MiniFE also provides support for computation on multicore nodes, including pthreads and Intel Threading Building Blocks (TBB) for homogeneous multicore and CUDA for GPUs. FE is a Finite Element mini-application which implements a couple of kernels representative of implicit finite-element applications. It assembles a sparse linear-system from the steady-state conduction equation on a brick-shaped problem domain of linear 8-node hex elements. It then solves the linear-system using a simple un-preconditioned conjugate-gradient algorithm.

3.3.2 CoMD

CoMD is a reference implementation of typical classical molecular dynamics algorithms and workloads. It is created and maintained by ExMatEx: Exascale Co-Design Center for Materials in Extreme Environments. The code is intended to serve as a vehicle for co-design by allowing others to extend and/or reimplement it as needed to test performance of new architectures, programming models, etc. It is considered as a standard application for performance analysis.

4. Implementation

Methodology and required packages, libraries are introduced in the prior of this paper. Now this portion of paper will concentrate on the implementation of our methodology with the presented external libraries. This part is divided into multiple sections to make more clear understanding of each part individually.

4.1. Performance counters collection

After identifying the events now, the important task is to read all of them for same configuration and store them. But it faces limitation from hardware part. Possibility for counting multiple set of events simultaneously can be checked using the PAPI command **PAPI_event_chooser** in the bin of PAPI installation directory. Due to the architectural limitations all the required events cannot be counted simultaneously despite having 10 hardware counters available, which is why in addition to including PAPI code to main file to read the counters, we had to introduce a new variable to command line **' -nq '** which takes the value from 1 to 4 and gives the respective values of events assigned to the appropriate value. With that code of application also has to be changed in order to implement PAPI in the application. The Implementation of PAPI and the introduced variable to collect all required counters is given below:

Value of '-nq'	**Events associated**
1	PAPI_L1_TCM, PAPI_L2_TCM, PAPI_L1_LDM
2	PAPI_L2_LDM, PAPI_BR_PRC, PAPI_RES_STL
3	PAPI_DP_OPS
4	PAPI_TOT_CYC

Thus, by introducing an extra loop in the script which iterates from 1 to 4 for '-nq' can collect values of all the selected counters for a configuration.

4.2. Generating dataset for supervised model

After modifying code of an application to read the counter values for each process, the primary aim is to gather these values in the file column wise to generate the dataset for supervised learning model. For given '-nq' value, idea is to gather all the values of allocated counter from all the processes in one place. For a configuration, if performance counter is the **'feature' or 'variable'** for our model then its **average** is to be out of the values collected from multiple processes. Else if the performance counter is **'PAPI_TOT_CYC' i.e. '-nq' to be equal to 4**, then its **maximum** is to be taken from the values gathered across processes. Sample code from the script made to generate dataset is given below: After saving output of the application runtime to 'output' file,

For 'feature' performance counters:

```
grep "PAPI_L1_TCM" output | awk '{print $NF}' > l1_tcm
awk '{ total += $1 ; count++ } END { print total/count }' l1_tcm > avg1
paste avg1 >> output1
```

For PAPI_TOT_CYC, 'target' performance counter:

```
grep "PAPI_TOT_CYC" output | awk '{print $NF}' > tot_cyc
awk -f /users/misc/sjinang/max1.awk tot_cyc > maxima
paste maxima >> output4
```

"max1.awk":

```
BEGIN { max = "NaN" }
{
   max = (NR==1 || $1>max ? $1 : max)
}
END { print max }
```

Remember after generating dataset, it is necessary to normalize all the features(columns) separately, before using them as training or test dataset. Necessity and brief significance of normalization is given here.

Normalization:

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. It is not necessary to normalize dataset every time but the datasets which have features with different ranges and scaling, it becomes necessity and also beneficial for higher accuracy to normalize the given dataset. Normalization becomes essential in our case where values of different performance counters can differ drastically and even with different configuration same performance counter can give a very different value in terms of range. So here we have used simple formula for normalization where target is to scale the values in between 0 to 1 with respect to minima and maxima of respective feature.

For ith feature in the dataset named ‘cols’, sample python code to normalize given feature is given below:

```
cols[i] = [ (float(j)-min(cols[i])) / (max(cols[i])-min(cols[i])) for j in cols[i] ]
```

The given sample code is then applied to all the features individually of which normalization had to be done.

4.3. Building supervised Learning models

Now that normalized dataset is generated for our training and test datasets, the primary focus is to build the supervised learning model that can adapt new parameters according to the training data passed. As mentioned in 3.2.4, respective libraries and functions are used to build the machine learning model. Sample code is given below for linear regression. It can then be used for selected models with appropriate modifications for their implementation.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Here Y contains target ‘PAPI_TOT_CYC’ and X contains all other counters with configurational features
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=rn)
lin_reg_mod = LinearRegression()
lin_reg_mod.fit(X_train, y_train)
prediction = lin_reg_mod.predict(X_test)
```

5. Results and Discussion

5.1 Error Analysis

One of the most important part of machine learning analytics is to take a deeper dive into model evaluation and performance metrics, and potential prediction-related errors that one may encounter. The aim of error analysis and development of various techniques to improve model is to decrease the reducible error. R^2 provides the standard goodness of fir measure where as we have calculated error for each benchmark as given below and we refer it "Median Absolute Percentage Error (MdAbsPE)"

$$\text{error} = 100 \times \frac{\text{predicted} - \text{actual}}{\text{actual}}$$

5.2 Evaluation

We have applied our methodology to multiple supervised learning models on our benchmarks MiniFE and CoMD. In MiniFE, apart from seven counters there were 3 configurational features, thus in total dataset of MiniFE application has 11 features where as CoMD has 6 configurational features so in total, dataset for CoMD contains 13 features with target "PAPI_TOT_CYC". After collecting required data for MiniFE and CoMD, the collected results of R2 score and Median Absolute Percentage Error (MdAbsPE) are given below for different states and models. Note, to collect final result, each case is iterated 10 times with **random number generator** assigned to the 'random_state' parameter of **train_test_split** function and then its average is considered as final result for each respective case.

NOTE: Except the configurational features, all the columns/features are normalized including target i.e. 'PAPI_TOT_CYC'.

Model	Test_size	R2 Score	MdAbsPE (%)
Multiple Linear Regression	0.2	0.9984168009214096	2.7514101746963568
	0.3	0.9983948043806006	2.8221527995708993
	0.5	0.9982939449722771	2.9162722962338075
Decision Trees	0.2	0.9957465447347291	4.952625140484066
	0.3	0.9958785953704583	4.750514140045444
	0.5	0.9952305226861131	5.047406421754073
Extra Trees Regressor	0.2	0.9975551962284805	3.6691256490330386
	0.3	0.9974588960942233	3.7209452681690993
	0.5	0.9973200467457117	4.056139403417914
XGBoost Regressor	0.2	0.9975340167494992	3.724500708722579
	0.3	0.997651546019757	3.807993205281532
	0.5	0.9968102880144925	4.211793987497884

MiniFE

Model	Test_size	R2 Score	MdAbsPE (%)
Multiple Linear Regression	0.2	0.9150859769075673	15.868559504622985
	0.3	0.9116287388880273	15.812968830126986
	0.5	0.9113087807392117	15.528636112027902
Decision Trees	0.2	0.87236075992283	13.308960020002502
	0.3	0.86383319456632	13.52708058254512
	0.5	0.865893038858653	13.568538959140955
Extra Trees Regressor	0.2	0.932049926868418	9.843190219266218
	0.3	0.9305755774035905	10.200037284638334
	0.5	0.9300053809343971	10.15174948233701
XGBoost Regressor	0.2	0.9259450067285874	10.229704524910002
	0.3	0.9234288724659679	10.688881372944351
	0.5	0.9176184139721381	10.77109073717429

CoMD

Note: For MiniFE, a total of 732 tests we conducted whereas for CoMD, the results were collected from the total dataset of 2832 test cases.

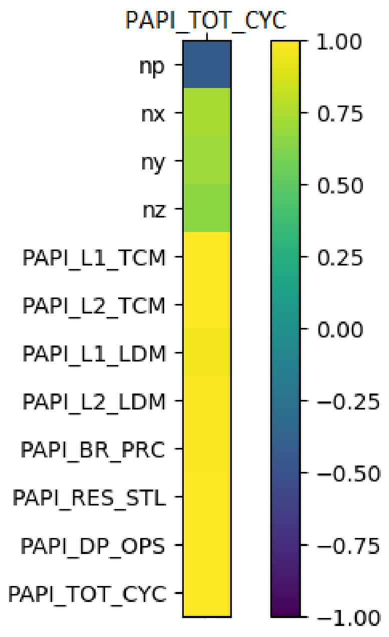
R2 score is a statistic used in the context of statistical models whose main purpose is the prediction of future outcomes. It provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. For MiniFE, we were 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.

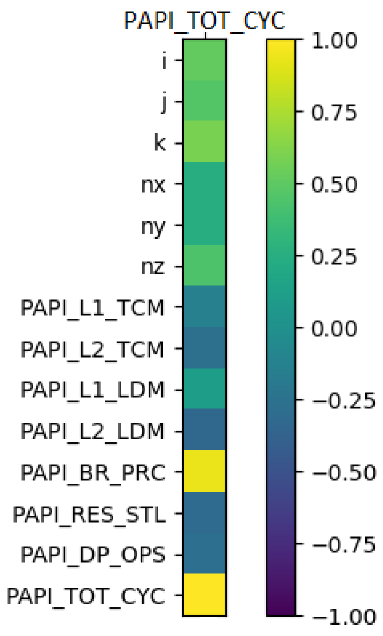
For CoMD, road is not as smooth as of MiniFE. Despite achieving R2 score of more than 0.86 for all the test cases, the error was lying in the range of 9.84 to 15.53. Achieved R2 score was good but the error was on and average more than 10 percent. This suggests that it is more complicated for the CoMD 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.

Correlation Matrix



MiniFE



CoMD

6. Conclusion

We have implemented multiple machine learning models to test if the selected performance events can give us the idea of performance as whole. Now with results we can say that in case of MiniFE, we are able to achieve median absolute percentage error less than 5% and R2 score is also greater than 0.99 in each possible scenario. It says that for MiniFE, the selected performance events are able to give us the idea of performance for whole application and with very small error. So, these results are backing our assumption that performance of an application can be predicted in terms of these events. But in case of CoMD, though we are able to achieve R2 score of more than 0.85 for each scenario but we are getting median absolute percentage error ranging from minimum 9.84% to 15.86%, which is significantly higher than what we saw in case of MiniFE. And its explanation may be given in terms of its calculational or parametrical complexity or inverse correlation with chosen events. But we can surely say that CoMD application cannot be predicted as accurately as MiniFE application with the selected events. And even further we can go to identify the best possible set of events to understand the performance of CoMD. Here we conclude that these selected set of events can predict the performance of MiniFE but its more complex for CoMD. But in general, these results back our hypothesis for predicting performance of a parallel application using performance counters. These 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.

7. References

1. Wikipedia
2. http://mezeylab.cb.bscb.cornell.edu/labmembers/documents/supplement%205%20-%20multiple%20regression.pdf
3. https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb
4. https://en.wikipedia.org/wiki/Decision_tree
5. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
6. https://ieeexplore.ieee.org/abstract/document/1522767
7. https://dl.acm.org/citation.cfm?id=1183426
8. https://ieeexplore.ieee.org/abstract/document/1392891
9. https://dl.acm.org/citation.cfm?id=1577137
10. https://ieeexplore.ieee.org/abstract/document/4145114
11. https://dl.acm.org/citation.cfm?id=1005691
12. https://ieeexplore.ieee.org/abstract/document/5598315
13. https://dl.acm.org/citation.cfm?id=1787310
14. https://www.icl.utk.edu/files/publications/1999/icl-utk-58-1999.pdf