Comparative Analysis of Factors Affecting Performance of Multithreaded Applications

Aman Chopra*
Computer Science Department
New York University
New York, NY, USA
ac8511@nyu.edu

Siddhartha Singh*
Computer Science Department
New York University
New York, NY, USA
ss13793@nyu.edu

Mohamed Zahran
Computer Science Department
New York University
New York, NY, USA
mzahran@cs.nyu.edu

***Abstract*—In recent years, owing to the advancement in parallel programming, it is becoming crucial to evaluate the performance of multithreaded applications. Most of the times, just allocating resources and using expensive hardware does not guarantee speedup and hence it is important to do code instrumentation. According to Amdahl's law, many programs might not be parallelizable. Sometimes, the parallelisation overhead might not give the expected speedup. In this paper, we are proposing a learning-based approach to predict the execution time and hence the speedup of an unseen program on a given machine using application characteristics for specific hardware.**

The authors believe that the speedup depends on a lot of factors which we have showcased in this research. We are not only comparing several machine learning models to show which model works best to predict the performance but also evaluating the features that have a major impact on the performance of the multithreaded application. This research can help programmers rectify those bottlenecks for efficient performance.

***Index Terms*—Multithreading; PARSEC; Performance Prediction; Speedup; Random Forests; LASSO; Artificial Neural Network**

I. Introduction

In today's world, due to the increasing complexity of the processors and system architecture, accurately predicting the execution time is a challenging task. Most of the times, large scale complex parallel applications take a lot of time and hardware resources to execute. Sometimes the parallel overhead does not provide the expected speedup and we might waste time running a program with a sub-optimal number of threads. If we know the application features of a parallel program, we can predict the speedup given the number of threads. In this paper, we have showcased an approach to predict the execution time and hence the speedup compared to a single thread execution using application characteristics without actually physically executing the parallel application everytime. An early estimation of speedup is advantageous as it helps in designing prototypes and can help intelligently allocate resources.

In this paper, we have first extracted application characteristics from 11 workloads of the Princeton Application Repository for Shared-Memory Computers (PARSEC) Benchmark Suite [1]. The chosen application characteristics are finalised after extensive study of the most important features that affect the performance of the multithreaded application. We have then extracted the most influential features using Spearman's correlation coefficient, Random Forests, and Boruta algorithm. Finally, we have designed and compared several Machine-learning models to evaluate the performance out of which Artificial Neural Network (ANN) give the least error.

The paper has been laid out in the following manner. Section II, the objective section deals with the primary objectives of this paper. Section III of this paper talks about the key researches being already done in this field. Section IV lays down the proposed idea and methodology we have followed to solve this particular problem. Section V discusses the experimental setup. Sections VI and VII mention the results and the conclusion.

II. Objective

In this paper, we have compared and analyzed several regression models to predict the performance of multi-threaded applications. We have first extracted application dependent features from several parallel applications of the PRASEC Benchmark Suite after which we have applied feature extraction and regression techniques. The prime objective is to compare the accuracy of different models and suggest the one to predict the performance of a new parallel application with high accuracy where the measure of performance is the speedup relative to one thread execution.

*All the authors contributed equally to this work.

III. Literature Survey

A lot of studies has been done to make efficient models to predict the performance of multi-threaded applications. In [2], the authors have proposed a way to automatically develop models based on the training set of high-performance parallel application SMG2000. The paper got a decent accuracy using Artificial Neural Networks but they did not concentrate much on the feature importance and selection which we believe plays a crucial role in the accuracy of the model. Some work has also been done in cross-platform performance prediction of Parallel Applications. In [3], the authors have devised an observation-based approach using very short partial execution of parallel programs. The model though promising, does not yield high accuracy for applications with variable overhead per timestamp as no code analysis or program modeling is done. In another research, the authors have predicted performance based on inherent program similarity using Genetic algorithm by taking microarchitecture-independent characteristics on SPEC CPU2000 benchmarks [4]. Our research is inspired by [5] where we have identified the areas for improvement, took more application related characteristics which might improve the performance, and designed a new model for better accuracy.

IV. Proposed Idea and Methodology

Figure 1 shows the work flow chart. The idea is to study and identify the most important features which affect the performance given different state of the art parallel workloads and the same machine as we change the size of the problem and the number of threads. We want to predict the execution time of the application given these minimal features so that the programmer knows the speedup before physically executing the program and allocating resources. We have currently carried out the experiments on the same machine. The specifications of the machine are listed in Table I.

A. Dataset creation:

After extensive research, we started with the below listed 11 features as we think that these features combine both hardware and program features and they impact the performance of a multithreaded application the most. According to [5], Cycles, IPC, Number of Branches, Branch Predictability, Cache hit Rate (Both L1 and L3), and Cache miss Rate (Both L1 and L3) are the most important features. Taking inspiration from the paper, the features we have selected are:

- **branch-instructions:** It is a hardware event that tells the number of branch instructions. Performance might be negatively affected if the number of branch instructions is more.
- **branch-misses:** It is a hardware event that is an indication of branch-predictability. High % of branch-miss might negatively affect the performance of the application.
- **cache-misses:** It is a hardware cache event(which tells the number of L3-cache-misses. L3 is usually the last level of cache and a miss in that will result in fetching the data from Dynamic random-access memory (DRAM) which is slow. This event also includes the prefetch requests that miss the L3 cache.
- **L3-cache-miss-rate in %:** We are also calculating

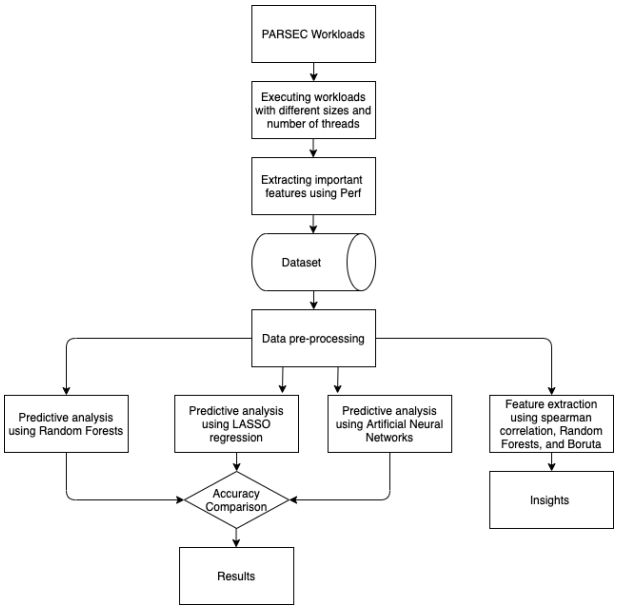

Figure 1. Research methodology flowchart

Table I
Machine specifications

Configuration	
Specification	**Value**
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	64
Thread(s) per core	2
CPU MHz	2099.972
BogoMIPS	4199.94
L1d cache	16K
L1i cache	64K
L2 cache	2048K
L3 cache	6144K

the percentage of last level cache miss rate to improve the accuracy of the model. Cache misses introduces overheads like cache coherence which negatively impact performance.

- **cache-references:** This hardware cache event counts requests irrespective of whether they miss the Last level cache which gives us an idea about how many requests the L3 cache is receiving.
- **cpu-cycles:** This software event is a characteristic of application size. Larger applications may benefit more from parallelisation.
- **instructions:** This hardware event tells the total number of instructions in the user space of the application.
- **IPC:** Instructions per cycle is calculated to know how efficiently the processor is performing for a given workload.
- **cpu-clock:** This software event measures the number of cpu-clock events in milliseconds.
- **page-faults:** This software event gives us the number of page-faults. An increase in page-faults might slow the performance.
- **L1-dcache-loads:** This hardware cache event tells us the number of L1 data cache load hits.
- **L1-icache-load-misses:** This hardware cache event tells the number of L1 instruction cache load misses.
- **LLC-load-misses:** This hardware cache event count only cacheable last level data read requests.

Other attributes we are adding to the dataset are the name of the PARSEC application, size of the application which takes three values 'simsmall', 'simmedium', and 'simlarge', the number of threads, the execution time, and the speed-up compared to a single thread execution. We extracted these attributes of each workload of different size executed over a different number of threads using the Perf profiling tool. We then parsed these values and created the dataset using a python script.

B. Data pre-processing:

For a model to perform well, data pre-processing is extremely important. The data needs to be processed based on the needs of an individual case study. To suggest a predictive model for performance prediction of multithreaded application, we have taken the following data pre-processing steps:

- We have removed the name column from our dataset as it has no significance in the prediction of decision attribute that is speed-up.
- We have changed the categorical size to numerical values using One-hot encoding which allows the representation of categorical data to be more expressive.

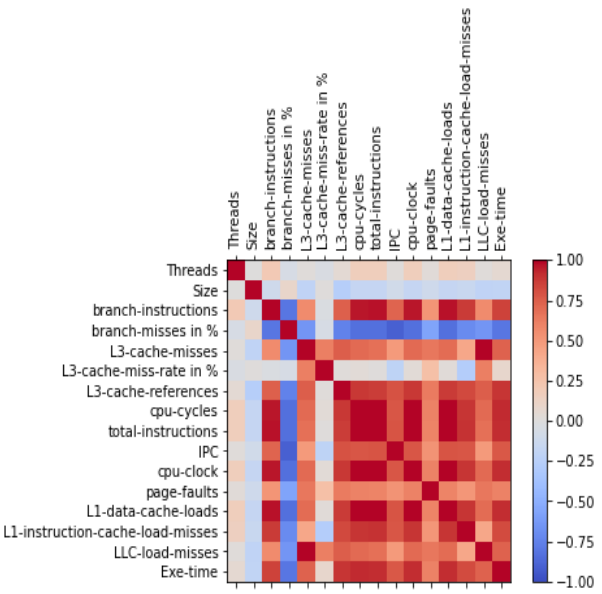

Figure 2. Spearman's rank correlation coefficient

C. Features extraction:

- **Spearman's coefficient** - The Spearman correlation evaluates a monotonic relationship between two variables — Ordinal or Continuous. Since it is based on ranked values of every variable instead of raw data, we consider Spearman's rank correlation coefficient. Figure 2 shows the correlation of each conditional variable with each other and the decision attribute. Eq 1 shows the formula to calculate the Spearman's coefficient where d is the difference between ranks of two observations and n is the number of observations.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{1}$$

- **Boruta** - This is a feature selection algorithm that is a wrapper on Random Forests. It creates something called as shallow attributes by creating shuffled copies of all attributes. This adds randomness to the data. We measure the Mean Decrease Accuracy to calculate the importance of attributes. We check whether a real feature has more importance than the best of its shallow attributes and constantly removes the unimportant features. Boruta hence confirms or rejects features from the dataset.

- **Random Forest** - In the Random Forest algorithm, we construct a decision tree and select r features in order to make a decision at a node. We then choose n random samples out of total N samples and then the best split attributed is decided for each node which finally gives us the most important attributes that affect the decision attribute the most. Figure 3 shows the top 5 features selected after running the Random Forest Algorithm.

Using the above three algorithms, we evaluate the top 5 application features and hardware/software events that affect the most the performance of multithreaded applications.

D. Predictive Algorithms

1) Random Forests: This ensemble technique uses multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging to predict the values of output variables based on inputs. It combines multiple decision trees in determining the final output rather than relying on individual decision trees. It has many decision trees as base learning models. We perform feature sampling and row sampling randomly from the dataset forming sample datasets for every model. We then train the model with training data and find the accuracy of the model using test and predicted data.

2) LASSO Regression: Least Absolute Shrinkage and Selection Operator is a regression analysis method that performs both variable selection and regularization to enhance the prediction accuracy. LASSO uses L1 regularisation where it adds "absolute value of magnitude" of coefficient as penalty term to the loss function and shrinks the less important feature's coefficient to zero thus, removing some feature altogether. Hence, this algorithm works well when we have a large number of features. To obtain better results, the alpha value is very important. We normalised the data and set the maximum number of iterations to 1000. We then set the value of alpha using Lassocv where we set the value of cross-validation to be 10 to obtain the best experimental result.

3) Artificial Neural Network: An Artificial Neural Network is a network structure that is believed to be similar in structure and function to a biological neural network. These networks are capable of learning patterns in data. The basic representation of simple feedforward neural network consists of multilayer perceptrons stacked into layers. Nodes in the input layer and output layer are equal to the number of input features and the number of output features respectively. Any layer between these layers is known as a hidden layer. Each node of the hidden layers is connected to each node of the previous and the next layer by a mathematical 'weight', and an optional bias weight may also be introduced into every perceptron.

In our problem statement, the artificial neural network is predicting the execution time based on the input attributes.

Min-max scaling: Before feeding data into ANN, it is important to normalize the numeric features to prevent unnecessary imbalance in the data while training. The data range in the input attributes is huge and hence data normalisation becomes extremely important to get better results. According to [6], "In practice, it is nearly always advantageous to apply pre-processing transformations to the input data before it is presented to a network. Similarly, the outputs of the network are often post-processed to give the required output values." We use Min-max scaling as shown in equation 2, which scales a range of values down, such as their values vary from 0 to 1.

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{2}$$

Model: We have used three hidden layers with 10, 8, 8 nodes respectively. The input layers consist of 15 nodes for 15 input features and the output layer consists of 1 node for one decision feature. The number of epochs were set to 200 and the batch size was set to 32.

Activation Function: The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. We have used ReLU as the conditional and decision attributes does not take negative values.

Optimizing Algorithm: The optimization algorithm that showed the best results for our problem was by using Adam optimizing algorithm.

V. Experimental Setup

- **Benchmarks:** We have used PARSEC 3.0 [1] as it consists of the state of the art multithreaded workloads for a variety of domains. PARSEC also comprises of applications parallelised using OpenMP. We have taken a total of 11 workloads which are listed in Table II.
- **Tools:** We have used the Perf Linux tool to extract the events and features for our dataset. We have used Python to parse the attributes and create the dataset. We have used several Machine Learning algorithms and models for data preprocessing, feature extraction, and execution time prediction.

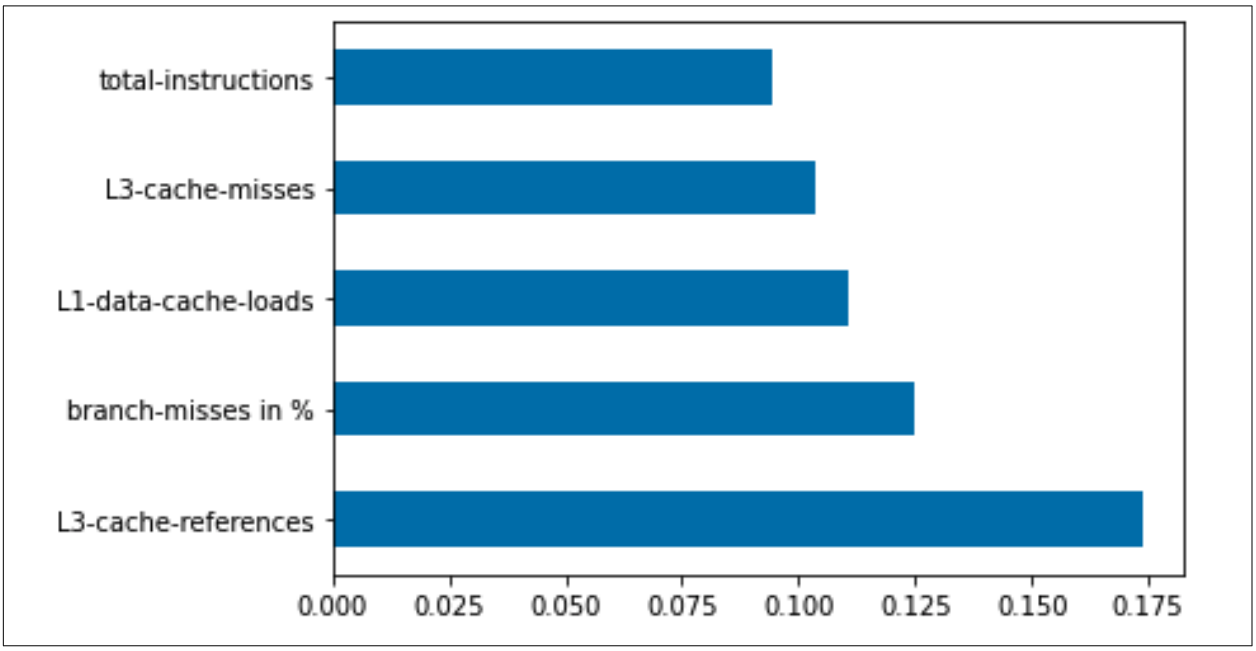

Figure 3. Top features that affect the performance of multithreaded application

Table II
PARSEC 3.0 WORKLOADS DESCRIPTION

PARSEC 3.0 BENCHMARK	
Application	**Domain**
Blackscholes	Financial Analysis
Bodytrack	Computer Vision
Canneal	Engineering
Facesim	Animation
Ferret	Similarity Search
Fluidanimate	Animation
Freqmine	Data Mining
Streamcluster	Data Mining
Swaptions	Financial Analysis
Vips in	Media processing
x264 in	Media processing

- **Dataset:** The dataset used in this study consists of various application-related features and hardware/software events as shown in Table III. We have executed each workload with three different problem sizes-small, medium, and large. We have executed each workload with a different number of threads i.e., 1,2,4, and 8. The dataset has 132 tuples and 18 columns. It has one decision attribute that is the execution time and the rest are conditional attributes. The dataset was shuffled with a random seed value and was divided into two sets of training and test data. 60% of the data was used to train all the models and the remaining was used to test the results. To train our model we have removed the name attribute. In our regression models, we are calculating the execution time as if we know the execution-time, we can also know the speedup compared to a single thread execution.

VI. EVALUATION

A. Results:

The dataset consists of 132 tuples and 18 attributes out of which "Exe-time" was the decision attribute and the remaining were conditional attributes. The top 5 features which affect the execution time of a multi-threaded application and hence the performance the most are: L3-cache-references, branch-misses in %, L1-data-cache-loads, L3-cache-misses, and total-instructions. We got the same 5 attributes after running Random Forests, Boruta, and calculating Spearman's correlation.

We compared several regression models to predict the execution time and hence the speedup of the parallel application. Out of Random Forests, LASSO regression, and Artificial Neural Network, ANN gives the least mean average error(MAE) and maximum R2 score which is a statistical measure of how close the data are to the fitted regression line. A high R2 score indicates how well our model fits the data. Table IV shows the final results.

Table III
DATASET DESCRIPTION

Conditional Attributes	
Attributes	**Description**
Name	The name of the PARSEC Workload
Threads	The number of threads used to execute the program
Size	The size of the application ranging from small, medium, to large
branch-instructions	The number of branch instructions
branch-misses in %	The percentage of branch misses
L3-cache-misses	The prefetch requests and code fetch requests that miss in the L3 cache
L3-cache-miss-rate in %	The percentage of L3-cache miss rate
L3-cache-references	Total number of requests that hit L3-cache
cpu-cycles	The number of CPU cycles
total-instructions	The total number of instructions in user space
IPC	The number of instructions per cycle
cpu-clock	Indication of how parallel the job has been
page-faults	The number of page faults
L1-data-cache-loads	The number of L1 data cache load hits
L1-instruction-cache-load-misses	The number of L1-instructions cache load misses
LLC-load-misses	The Last Level Cache load misses
Exe-time	The execution time of the application
Speedup	The ratio of execution time of single thread vs multiple threads

Table IV
COMPARISON OF DIFFERENT REGRESSION MODEL

MAE between Predicted and actual Execution time		
Algorithm	**Mean Average Error**	**R2 Score**
Random Forests	0.41	60%
LASSO Regression	0.37	83%
ANN	**0.032**	**86%**

B. Observations:

We can see that the ANN performs the best for the given problem statement. Neural networks are suitable for large dimensionality datasets where little knowledge about the underlying process exist. Neural networks scale well, hence, when we add data from more benchmarks, we will obtain better results. As far as the top features are concerned, the results seem intuitive. The number of L3-cache-references sure will play a huge role in the performance of the application. We see that branch-misses in % impact the performance more than the total number of instructions or the total number of branch-instructions. Of course, a high number of L3-cache-misses will negatively impact the performance as now the data has to be fetched from the Dynamic random-access memory (DRAM) which is slower. The total number of instructions and L1-data-cache load hits are the other important attributes which affect the performance of multithreaded applications.

VII. CONCLUSION AND FUTURE WORK

We have analyzed the performance of multiple regression models to predict the performance of multithreaded applications. According to our analysis, Artificial Neural Network gives the maximum accuracy and is the appropriate model for the dataset and the problem statement, among others. We have also analysed the minimum number of application features that affect the performance of multithreaded applications the most. This research can be used to find and rectify bottlenecks to improve the efficiency of parallel programs.

In the future, we intend to execute the model on different workloads of multiple benchmarks to test the validity of the model. We also intend to find the optimal number of threads which can give the maximum speedup given the program characteristics. Finally, we want to extrapolate this research to multiple platforms and machines to come up with a unified model to predict the performance of parallel applications.

REFERENCES

[1] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[2] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Euro-Par 2005 Parallel Processing*, J. C. Cunha and P. D. Medeiros, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 196–205.

[3] L. T. Yang, Xiaosong Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 40–40.

[4] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. John, and K. D. Bosschere, "Performance prediction based on inherent program similarity," *2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 114–122, 2006.

[5] T. Jain, N. Agarwal, and M. Zahran, "Performance prediction for multi-threaded applications," *2019 International Workshop on AI-assisted Design for Architecture (AIDArc)*, 2019.

[6] C. M. Bishop, *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1996.