TOWARDS AN AUTONOMOUS FRAMEWORK FOR HPC OPTIMIZATION: A STUDY OF PERFORMANCE PREDICTION USING HARDWARE COUNTERS AND MACHINE LEARNING

Matheus Gritz
Gabrieli Silva
Vinícius Klôh
Bruno Schulze
Mariza Ferro
Laboratório Nacional de Computação Científica
Av. Getulio Vargas, 333 - Quitandinha, Petrópolis - RJ, 25651-075
masgritz,gabrieli,viniciusk,schulze,mariza@lncc.br

ABSTRACT

As the high processing computing becomes even more critical for scientific research across various fields, increasing performance without raising the energy consumption levels becomes an essential task in order to warrant the financial viability of exascale systems. This work presents the first step towards understanding how the many computational requirements of benchmark applications relate to the overall runtime through a machine learning model and how that can be used for the development of an autonomous framework capable of scaling applications to have an optimal trade-off between performance and energy consumption.

Keywords: Machine Learning; HPC; Performance; Energy Consumption.

Como Citar:
GRITZ, Matheus; SILVA, Gabrieli; KLÔH, Vinicius; SCHULZE, Bruno; FERRO, Mariza. Towards an Autonomous Framework for HPC Optimization: A Study of Performance Prediction using Hardware Counters and Machine Learning. In: SIMPÓSIO DE PESQUISA OPERACIONAL E LOGÍSTICA DA MARINHA, 19., 2019, Rio de Janeiro, RJ. **Anais** [...]. Rio de Janeiro: Centro de Análises de Sistemas Navais, 2019.

1. INTRODUCTION

High performance computing (HPC) is used to analyze, understand and solve problems in a wide range of applications, such as chemistry and biology (molecular dynamics, genome sequencing), geophysics (seismic wave propagation, earthquake predictions), and modeling of complex physical phenomena (climate and fluid dynamics). However, even with the current generation of petafloptic supercomputers, capable of performing complex simulations with realism and precision never achieved, some areas require more computational power in order to make their simulations even more precise or even feasible to perform. In view of this new reality, there is today a worldwide race to reach a new generation of supercomputers in the next decade: the exascale supercomputers.

A large number of works, such as [1], [2], [3], [4] and [5], consider that one of the greatest barriers to the feasibility of the exascale is the energy consumption of HPC environments. If today the expenses with electric energy of the petaflopic supercomputers reach high

values, to the new generation will depend on a lot of effort so that the energy consumption does not increase in the same proportion as the processing capacity [6]. However, balancing computing power and energy efficiency is not a trivial task. Most of the time you lose in energy efficiency when you gain in processing power or vice versa. For example, we can see that the Summit supercomputer is the first in the Top500[1] list of November 2018, but the third in Green500[2] list. The major challenge for exascale is to find a balance between these two factors and therefore different research projects (such as ECP[3], Mont-Blanc[4] and HPC4e[5]) seek solutions on this aspect. These solutions cover different levels, since the preparation of the applications (algorithms) so that in fact they can reach the levels of processing that will be offered by these future architectures, until the development of hardwares with faster processing and better energy efficiency (for example GPU and ARM).

To overcome this barrier, studies point out that the next generation of supercomputers will need to be developed using approaches where the requirements of the scientific problem guide computer architecture and system software design. In addition, these requirements of the scientific problem should guide the orchestration of different techniques and mechanisms of energy saving, in order to improve the balance between energy saving and application performance. For this, the use of autonomic techniques that allow from the best application scheduling to the frequency sizing of processors and memories (energy aware) will be fundamental.

To explore these aspects, it is critical to understand HPC applications and environments and how they relate to energy consumption. There is a need to deepen knowledge about the factors that limit application performance and interfere with power consumption, and map the architectures that represent the current state of the art in HPC. However, understanding the computational requirements and the energy consumption of the applications is a challenging task because involves the collection of different parameters, which are not always easily obtained due to the need for superuser privileges to be collected in the registers. Thus, to overcome this barrier, predictive models have been widely used in high performance computing, such as in the works of [7], [8], [9], [10], [11] and [12]. However, there are a number of factors for the development of these models, since they need to be efficient and effective, and for this, there is a need to be validated by different experimental sets. Appropriate parameters, possible to be collected and architecture independent are among the key pieces for prediction success.

To enable this balance between performance and energy saving, we propose the development of an autonomic framework to make the orchestration among applications, schedulers and architectures, based on the requirements of the scientific applications/simulations. In this work we present the ongoing research to this development and how to collect these relevant parameters looking to understand the applications and its performance for different models (Motifs/Dwarfs classes [13]) focusing on the development of the predictive techniques that will be part of the framework. The predictive tasks are being developed using Machine Learning (ML) techniques and in this paper we present all the steps to predict the execution time of an application in a given computational architecture

[1]https://www.top500.org/list/2018/11/?page=1
[2]https://www.top500.org/green500/list/2018/11/
[3]ECP - Exascale Computing Projections - https://www.exascaleproject.org/
[4]https://www.montblanc-project.eu/
[5]https://hpc4e.eu/

and the results obtained.

This paper is organized as follow: In Section 2, an Autonomous Framework is proposed and briefly explained. In section 3, we present some related works. In Section 4, we detail the experimental setup, including architecture, hardware counters and how the data was parsed. In Section 5, we present the ML experiments and its results in Section 6. Finally in Section 7 we present our final considerations and future works.

2. PROPOSED AUTONOMOUS FRAMEWORK

The Figure 1 represents the autonomous framework that serves as the basis for this work and the various steps which will comprise it. The proposed framework, in development, will enable the orchestration of techniques to reach the balance of performance and power consumption of parallel applications. It depends on application characteristics, architectural parameters, runtime node configuration and input data. Based on these initial information, the characterization of the application and architectures available for the job execution will be defined, like an signature of it. This relationship leads to very specific conditions of performance and energy consumption, as demonstrated in [14], [15] and could figure out of the knowledge base (KB) of the framework. This KB would be used by the scheduler to predict the execution time and energy consumption in order to choose the best architecture for the job execution and also the feasibility to select the best frequency, using Dynamic and Frequency Voltage Scaling (DVFS) techniques. To reach this level of orchestration a number of different techniques must be studied, like the understand of the applications, computational architectures, how to access and collect performance counters and which are the relevant ones, predictive tasks for time of execution and power consumption, scheduling approaches and the use of DFVS techniques. In this work we are focusing on how to access and to collect the performance counters, to define the relevant ones to enable the prediction of the execution time. Based on the phases of the Figure 1 we present the work in development to achieve it.

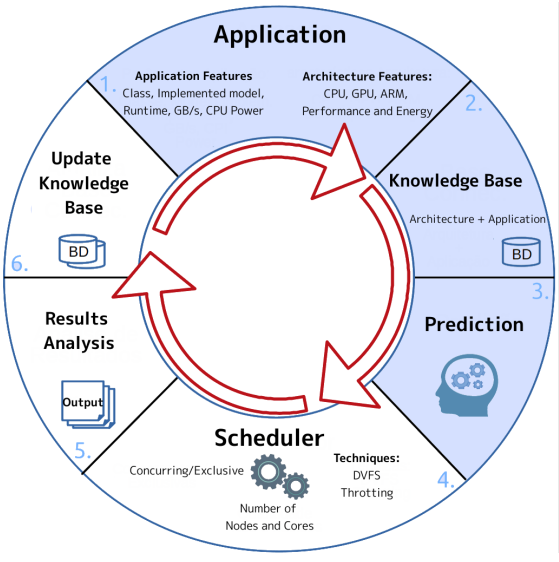

Figure 1: Autonomous Framework

The first phase, Application, involves the selection of performance counters that will serve as the basis of all the following steps. These performance counters, also known as

performance events, are selected based on the main components of the hardware available to us and how their levels might impact the overall performance of an application. Most events selected are based on either CPU, RAM memory or cache, considering an application access to these are the main culprits for extended runtime and resource consumption.

At the second phase, a knowledge base is built using the data collected during step one using a performance counter tool such as *perf* included in the *Linux kernel*. At this point, the elements are cross-referenced to understand how they relate to each other and to the overall runtime in order to prepare the data to go through the prediction phase.

Finally, at the third phase, a Machine Learning tool such as *scikit-learn* or *Weka* is used in order to train a regression model to understand the correlation and co-variance between our data and our target which, in this case, is the time an application takes to complete its task, and predict what could be the next instance of our target.

As mentioned, the main objective is to use these results to build a scheduling application that will be used to predict and scale applications at runtime in order to keep a balance between performance and resource consumption, using the data collected to further improve the knowledge base without need for human interference or input in a truly autonomous manner.

Next we present some related works which have some similarity of our work.

3. RELATED WORK

As this work involves the use of different techniques to enable the development of the proposed framework, like performance and energy consumption evaluation, the management of performance, predictive task using ML, scheduling, DFVS techniques and the development of the autonomous techniques to manage HPC environments, there are a large number of related works that can be mentioned. However, this section will be focused on works that make use of hardware counters and machine learning to predict performance.

As mentioned in Section 1, the use of autonomic techniques will be of utmost importance to allow a better application scheduling and CPU clock scaling. In order to ensure that the decisions taken by the framework are correct, it's important to be able to predict the behaviour of an application. A number of works make use of predictive models (such as [7], [8], [9], [10], [11] and [12]), which are classified according to their approach (analytic x empirical) ([16]).

The analytical approach includes the knowledge of experts involved in a project, so the analysed parameters are previously established based on their knowledge of the hardware and software. On the other hand, the empirical approach is based on observation, including a group of performance tests where different setups are collected in order to understand which features affect the performance of an application.

A large number of researchers explore the possibility of using Machine Learning techniques to obtain this kind of knowledge. Considering the different applications that are run on HPC environments and the complexity of these mediums, the amount of features that have to be analysed grows exponentially, making it the use of analytical approaches unfeasible. As such, with the advancement of Machine Learning techniques, the use of empirical approach has grown considerably and has been used in several different works such as [17], [18], [19], [20], [21] and [22].

Other works in the past also tried to use machine learning models to predict and optimize applications based on data collected by performance counters. In [17], the authors propose a specific machine learning approach tailored for stencil computation using Support Vector Machines and collecting data from two established stencil kernels with help of PAPI and conclude that their performance can be predicted with high precision due to the use of the appropriate hardware counters.

The authors of [23] propose a tool using machine learning as its base to predict the overall required time to complete multi-stage scientific data processing applications and using the Linux kernel's *perf* to collect hardware counters, presenting the effectiveness of using perf as well as machine learning as their conclusion.

What distinguishes this work from the others mentioned in this section is its approach, being a framework created without a specific application type as its target and capable of generalizing a vast gamma of scientific applications.

Next we present the complete experimental setup and the experimental results at this point of the work.

4. EXPERIMENTAL SETUP

All data used in this work was collected in one of the nodes of the Beholder cluster at ComCiDis/LNCC [6]. It houses two Intel Xeon X5650S Hexa core CPUs with max clock of 2.67GHz, two NVIDIA GF100GL GPUs, and 24GB of RAM memory split in 6 sticks of 4096MB with 1333MHz speed. It also has three levels of cache memory with the sizes of 64KB, 256KB and 12288KB respectively.

The experimental setup for this work is comprised of applications from the NASA Parallel Benchmark suite (NAS): Block Tri-diagonal solver (BT), Lower-Upper Gauss-Seidel solver (LU) and Scalar Penta-diagonal solver (SP) and the Rodinia Benchmark suite: Lower Upper Decomposition (LUD) mapped from two Moti's class (Dense Linear Algebra and Structured Grid). The approach used in this work is to understand the computational requirements of the applications from their classification as a Motif's class. These classification characterizes applications by common requirements in terms of computation and data movement.

The NAS benchmark suite allows the execution of the experiments with a vast range of problem sizes. For this study the size chosen was A, B and C which corresponds to the matrix sizes presented in the second column of Table 1, that offers a detailed view on the experiments used to collect the performance counters.

Additionally, every application was executed 30 times across a predefined thread interval (1, 2, 4, 6, 8, 10, 12) in order to observe the difference in the execution time to complete the task and how it impacts the performance counters collected by *perf* [24].

4.1. Performance Counters

The events were collected using *perf* tool, a performance counter tool available in the *Linux Kernel* since version 2.6.31. It performs counts of events accessing the Model-specific registers directly. Table 2 offers a look into the events and what they represent, the lines with an asterisk (*) were the ones selected during the Feature Selection phase in

[6] www.lncc.br

	Matrix Size	# of Runs	Motif
BT A	64x64x64	210 (30x7)	Dense Linear Algebra
BT B	102x102x102	210 (30x7)	Dense Linear Algebra
BT C	162x162x162	210 (30x7)	Dense Linear Algebra
LU A	64x64x64	210 (30x7)	Dense Linear Algebra
LU B	102x102x102	210 (30x7)	Dense Linear Algebra
LU C	162x162x162	210 (30x7)	Dense Linear Algebra
SP A	64x64x64	210 (30x7)	Structured Grids
SP B	102x102x102	210 (30x7)	Structured Grids
SP C	162x162x162	210 (30x7)	Structured Grids
LUD	128x128	210 (30x7)	Dense Linear Algebra

Table 1: Experimental Set

Section 5.2 and used for the ML predictive task.

	Feature	Description
1*	Instructions	# of instructions sent to the CPU
2*	Cycles	# of CPU cycles completed
3	CPU Migrations	# of times that the processes moved to another CPU
4*	Branches	# of conditional instructions that alter the flow of a process
5*	Branch Misses	# of times the CPU failed to predict the outcome of a branch
6*	Context Switches	# of times a process was halted so another could be run
7*	Cache References	# of times any of the cache levels was accessed
8	Cache Misses	# of times something was not found in the cache
9*	L1 dcache Stores	# of times data was stored in the Level 1 dcache
10*	L1 dcache Loads	# of times data was loaded from the Level 1 dcache
11*	L1 dcache LoadMisses	# of times data was not found in the Level 1 dcache
12*	LLC Stores	# of times that data was stored in the Last Level cache
13	LLC Store Misses	# of times data failed to be stored in the Last Level cache
14	LLC Loads	# of times data was loaded from the Last Level cache
15	LLC Load Misses	# of times data was not found in the Last Level cache
16	Page Faults	# of times data stored at RAM was not found in the physical memory
17	Minor Faults	# of times a page fault was handled without accessing the disk
18	Runtime	the time (in ms) it took for an application to finish its execution

Table 2: Performance counters and features of the ML task used in this work

Due to hardware limitations of the Intel Westmere-EP processor family, events such as mem-load, mem-stores, major-faults and any power-related events could not be collected as it lacks Model-Specific Registers for these events.

Unfortunately, *perf*, the tool used to collect the performance counters doesn't generates an easily parse-able output and the development of a bash script to include the runtime in the original data file and a Python 3 module had to be developed in order to be able to parse the output and translate it in a format that could be used by the *pandas* library as a DataFrame was required.

The bash script makes use of awk to include the runtime to the original output file. The parser developed using Python 3 uses default functions like *replace* and loops to clean up the output accessed using the *os* module. After the data is cleaned up and reformatted, it's stored in a *pandas* dataframe and saved as a CSV file for posterior use.

5. MACHINE LEARNING

Machine Learning (ML) can be defined as an Artificial Intelligence (IA) subarea that searches for computational methods related to the automatic acquisition of new knowledge, new abilities and new forms of organizing the existent knowledge [25] [26]. Before using ML methods to extract knowledge or to construct predictors, it is necessary to define the learning task and the algorithm to be used. In the classification the learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples. Classification learning is a supervised task, because, in a sense, the scheme operates under supervision by being provided with the actual outcome for each of the training examples. Numeric prediction is a variant of classification learning in which the outcome is a numeric value rather than a category [27].

The input to a machine learning scheme is a set of instances. These instances are the things that are to be classified and are characterized by the values of a set of predetermined attributes. Each dataset is represented as a matrix of instances versus attributes. Each instance that provides the input to ML is characterized by its values on a fixed, predefined set of features or attributes.

The problem to be solved in this work is to learn the execution time to run one application. The instances are the rows of the tables that we have shown in Table 2. The ML algorithm used was the regression tree from the free and open source Python 3 library *scikit-learn*[28]. It offers a vast selection of supervised and unsupervised ML models as well as tools that can be used for feature selection, cross-validation and model optimization. This section describes the phases from preparing the data to training the machine learning model. Figure 2 presents a flowchart illustrating this process.

5.1. Data Preprocessing

As detailed in [29], data preprocessing is one of the most important steps when working with ML, as the quality of data and the usefulness of the information that can be derived from it, directly affects how our model learns and how well it can generalize our data. So, properly balanced and normalized data acts as a way to ensure the effectiveness of the model as well as to prevent common issues that are hard to detect at first glance such as overfitting.

The experimental steps, before the ML tasks, begin with the collection of the hardware counters and parse output, as described in Section 4. In sequence, the data preprocessing step begins.

The first step in preprocessing the data was to normalize it in order to prevent issues with the difference (scale data between 0 and 1). The main objective when doing a normalization tasks such as this is to prevent performance issues and reduced accuracy in ML models caused by wildly different scale of the parameters, something present in all of the collected data. When some features have different ranges in their values (for example, the feature Instructions and CPU Migrations) will affect negatively algorithms because the feature with bigger values will take more influence.

In order to scale the features, the package MinMaxScaler from scikit-learn was used. It transforms all numeric data in a scale between 0 and 1 by default.

$$X\_scaled = scale * X + min - X.min(axis = 0) * scale$$

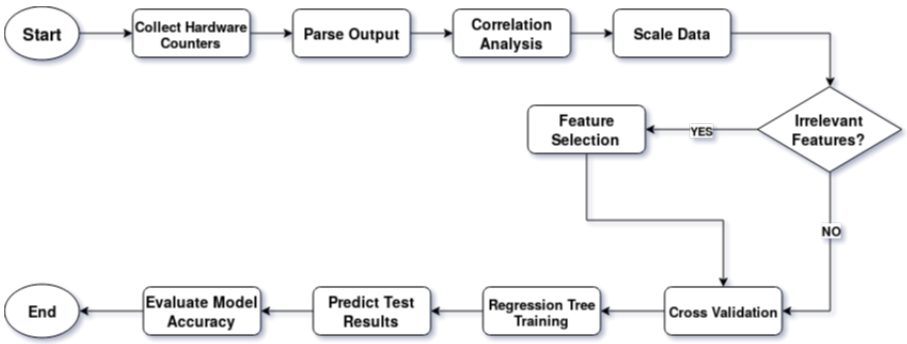

Figure 2: Project Flowchart

With the scale being represented as:

$$scale = (max - min)/(X.max(axis = 0) - X.min(axis = 0))$$

Another important preprocessing task that must be used in ML in order to better understand the data and how it relates to itself and our target is the Correlation Analysis. Figure 3 presents a correlation heatmap between the features of the dataset built using the Python data visualization libraries, *seaborn* and *matplotlib*. The parameters with the best correlation to our target (*runtime*) are instructions, branches, L1-dcache-stores, L1-dcache, L1-dcache-load-misses and LLC-loads, showing how events of CPU and cache are the ones with the strongest impact in performance.

5.2. Feature Selection

Feature selection (FS) is used to determine what features should be used in our ML model and whether they are effective to predict our target across the train and test phases [30]. Univariate feature selection was the technique used to select the features for its ease of use and overall reliability. It selects the best features based on univariate statistical tests, removing all features that do not attain to a predefined score level or threshold.

In this work, *scikit-learn*'s **SelectKBest** was the algorithm of choice. It removes all features based on a scoring function (f_regression, in this case), leaving only the best K features, where K is the number of features we want to use to train our model. The *f_regression* function uses the correlation between each feature and the target to calculate the overall score of a feature.

Despite we used feature selection algorithms, it is important to note that this task is not so trivial as to only use implemented algorithms. It involves the ad-hoc knowledge from the data scientist added to the knowledge from the specific domain (sometimes with an specialist of the domain). The data scientist need to have a deeper look and understand about the features to select the best ones and to judge the results of the algorithm correctly.

5.3. Cross Validation

Cross Validation (CV) is a common practice when working with supervised ML models and involves the concept of holding part of our data as a test set in order to prevent issues like *overfitting* and *underfitting*, that can negatively impact the accuracy and performance of a model.

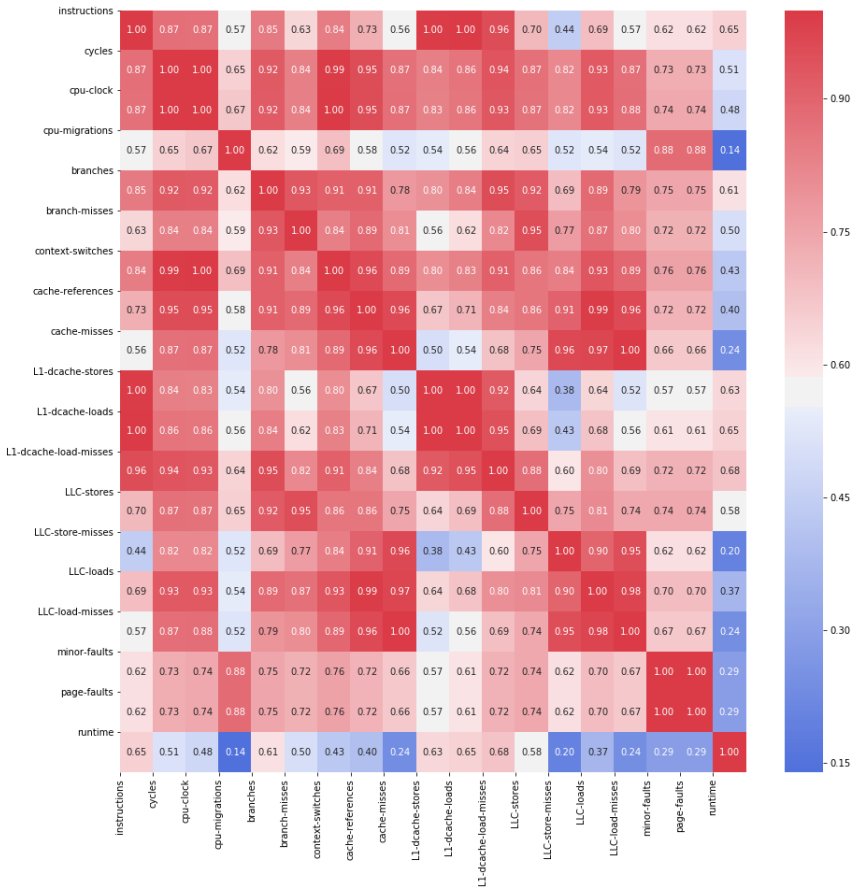

Figure 3: Correlation Heatmap

From *scikit-learn*'s *model-selection*, the **train_test_split** function was used to split all of data in four sets. It randomly selects lines from our original dataset, splitting it in four subsets called **train** and **test** following a predetermined ratio of 7.5:2.5.

5.4. Regression Tree

For the train and prediction phase, the machine learning model from *scikit-learn*'s, **Decision Tree Regressor**, has been used. Popularly known as Regression Trees, they are a form of Decision Trees, which are non-parametric supervised models that predict values of a target feature by learning simple decision rules inferred from a dataset. The decision by its use was based on the success of this model to learn over numerical values. Also, it is not sensible to missing values, noise data and outliers. In addition, decision or regression trees are good at feature selection since the top nodes on which the tree is split are essentially the most important variables within the dataset. Finally, the results are very intuitive and easy to explain, which could help us to understand how the parameters are related with the runtime prediction.

A tree with the max depth of 4 is created and fitted using the train subsets randomly split during the step describe in section 5.3, which are comprised of 75% of all the original data. Then, a prediction is generated using the test subsets as its target.

6. RESULTS

This section provides the results and findings from this work. The training and test was done on a total of 1890 samples, which underwent a 4-fold CV (detailed in Section 5.3). The total number of features used, after the preprocessing and feature selection process (detailed in Sections 5.1 and 5.2), were 10 as described in Table 2 and highlighted with (*).

Initially, we did an experiment using all features available in Table 2 without limiting the max depth of the tree. While the result, precision-wise, was good (Mean Square Error: 0.00024206418887226167), we should not look solely at precision when analysing the obtained results, but also consider how well the model is able to generalize the data and if overfitting happened. A tree with many leaves with low coverage show that the model wasn't able to properly generalize the data used for training, creating a large number of rules with very few samples each. As such, it has little to no practical utility and that was the result for this initial experiment.

The features selected during the FS processes presented in Section 5.2 were the same used by the regression tree algorithm to create the tree as the more relevant one, confirming the results.

Seeking to improve the generalization capabilities of our tree, new experiments were executed using only the selected features and limiting the max depth of the tree to 4.

In order to evaluate the accuracy and effectiveness of the Decision Tree Regressor, explained in section 5.4, a number of test have been done to understand the degree of error in the predicted results. These test results are presented in Table 3 below.

Mean Absolute Error	0.02255048219039073
Mean Squared Error	0.0011791189649137987
Root Mean Squared Error	0.03433830171854454
Coefficient of Determination (R2)	0.9752297764075215

Table 3: Mean Error

The Mean Absolute Error is the measure of the average difference between continuous variables, while the Mean Squared Error measures the average squared difference between predicted values and what is being predicted and the Root Mean Square Error is the standard deviation of Prediction Errors.

As it can be seen, the model offers a good precision while predicting the results considering all of our data was scaled between 0 and 1. Also, according to the coefficient of determination, it's apt to predict new values with success. That is, it could successfully predict the runtime for new job executions in a accuracy way.

Figure 4 provides a visualization of the tree and its rules.

The tree uses the feature L1-dcache-load-misses as its root, splitting into its first two branches if it's larger than 0.7898 or not. Then, it splits based on L1-dcache-load-misses again or on Context Switches. As it can be seen, other branches split based on context-switches and other cache-related events, these being the responsible for prolonged runtime in the applications tested. In special, the left branches are better capable of generalizing our data.

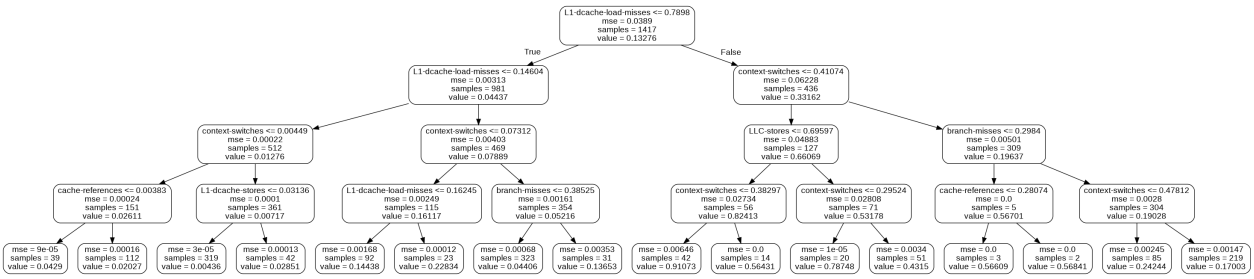

Figure 4: Regression Tree

A tree can be interpreted as a set of decision rules, as the example below shows.

```
if L1-dcache-load-misses > 0.14604 and
if context-switches > 0.07312 and
if branch-misses <= 0.38525 then
Mean Square Error = 0.00068, Samples = 323, Value = 0.04406
```

Upon observing the generated tree, the generalization process wasn't bad, considering 4 leaves have a high coverage of our original data. For example, these four rules cover 319, 323, 219 and 112 samples, each.

To confirm the validity of these results, we conducted a validation process by using the decision tree regressor model generated during the training and test phase in Section 5.4, that is, the runtime prediction model, on data never seen before by the model collected from 210 LUD runs with the same features and in the same architecture.

Mean Absolute Error	0.27762203139153246
Mean Squared Error	0.15581366404557717
Root Mean Squared Error	0.3947323954853176
Coefficient of Determination (R2)	-0.30343532487864144

Table 4: Mean Error with LUD

As it can be seen on Table 4, the three Error tests are far too high considering all of our data was scaled between 0 and 1 and coefficient of determination is negative, showing that the model is unable to predict new data with success. Some hypotheses were raised about the reason behind these results. The main one is the lack of a feature that represents the workload of each application in our experimental set, as this is a very relevant parameter that can be used to determine the runtime in a specific architecture. Besides that, a feature that contains architecture-specific information also needs to be added in order to give more significance to the data collected and then allow a better generalization process by the ML algorithm.

7. CONCLUSIONS AND FUTURE WORK

In this work we present the ongoing research to develop an autonomic framework to make the orchestration among applications, schedulers and architectures, based on the requirements of the scientific applications. We present how to collect the relevant parameters looking to understand the applications and its performance for different models

(Motif's class [13]) focusing on the development of the predictive techniques that will be part of the framework. The predictive tasks are being developed using Machine Learning (ML) techniques and in this paper we present all the steps to predict the execution time of an application using regression tree.

Training and test results had a good performance and also the generalization process sounds good when looking for the tree and the number of examples coverage by each leave. However, the domain's expert had suspicions about the predictive quality of the rules, since the some features considered representative for this domain had not been included in the learning task. Thus, the predictive model generated was validated with new examples and the suspicion was confirmed with poor results in the validation process.

The hypotheses for this results is that, although the features selected are relevant, new ones need to be included to give a real representation of the problem. Moreover, new examples using more applications' models and executed in different architectures need to be included in the database to be more representative and improve the generalization process. The results show that the development of a good predictive model using ML is an art and requires a good domain knowledge, a good quality and representative database and enough number of examples.

For example, the size of the problem (workload) is very important to characterize the runtime of an application, as well as in which architecture it is running. So, these new features will be included and new training and testing steps will be performed in future works. However, it is not trivial to define a feature that represents the workload problem, because an autonomous framework will receive different types of jobs to be executed, with multiple kinds of inputs. For example, the applications used in this work received as input matrices, which makes it easy to characterize it size, but applications could receive text files, matrices, vectors, complex data such as images and videos. Thus, a feature that generalizes the size of the problem is not trivial. In addition, it is important to characterize the computational architecture. If a nominal feature is used, such as CPU, GPU, ARM, it would reduce the representativeness of the data and another ML approach need to be used. Another option that will be evaluated is to use a set of features with the characteristics of the architecture, collected for each execution, like CPU Ghz, number of cores, memory size.

Also, in the future works experiments with new hardware counters, specially for power consumption measurements, need to be collected, the predictive task for energy consumption will be developed and more experiments with new applications of different Motif's class and using different architectures must be performed.

8. BIBLIOGRAPHIC REFERENCES

[1] SIMON, H. D. *Barriers to Exascale Computing.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. 1–3 p. ISBN 978-3-642-38718-0. Disponível em: <https://doi.org/10.1007/978-3-642-38718-0_1>. 1

[2] BERGMAN, K. et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, v. 15, 2008. 1

[3] ALVIN, K. et al. On the path to exascale. *International Journal of Distributed Systems*

and Technologies, IGI Global, v. 1, n. 2, p. 1–22, 2010. 1

[4] RAJOVIC, N. et al. Supercomputing with commodity cpus: Are mobile socs ready for hpc? In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2013. (SC '13), p. 40:1–40:12. ISBN 978-1-4503-2378-9. Disponível em: <http://doi.acm.org/10.1145/2503210.2503281>. 1

[5] MESSINA, P. The exascale computing project. *Computing in Science Engineering*, v. 19, n. 3, p. 63–67, May 2017. ISSN 1521-9615. 1

[6] SILVA, G. et al. Smcis: um sistema para o monitoramento de aplicaç oes cientıficas em ambientes hpc. SBC, São Paulo-SP, p. 277–288, Outubro 2018. Http://www2.sbc.org.br/wscad/current/anais/anais-wscad-2018.pdf. 2

[7] POVOA, L. V. et al. Predição de consumo energético com base na utilização de recursos computacionais. 2013. 2, 4

[8] FERREIRA, A. R. et al. Um modelo analítico para estimar o consumo de energia de sistemas multi-camadas no nível de transação. Universidade Federal de Goiás, 2017. 2, 4

[9] SIEGMUND, N. et al. Performance-influence models for highly configurable systems. In: ACM. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. [S.l.], 2015. p. 284–294. 2, 4

[10] JAIANTILAL, A.; JIANG, Y.; MISHRA, S. Modeling cpu energy consumption for energy efficient scheduling. In: ACM. *Proceedings of the 1st Workshop on Green Computing*. [S.l.], 2010. p. 10–15. 2, 4

[11] BALLADINI, J. et al. Metodología para predecir el consumo energético de checkpoints en sistemas de hpc. In: *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*. [S.l.: s.n.], 2014. 2, 4

[12] WU, X. et al. Using performance-power modeling to improve energy efficiency of hpc applications. *Computer*, IEEE, v. 49, n. 10, p. 20–29, 2016. 2, 4

[13] ASANOVIC, K. et al. A View of the Parallel Computing Landscape. *Commun. ACM*, ACM, New York, NY, USA, v. 52, n. 10, p. 56–67, oct 2009. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1562764.1562783>. 2, 12

[14] FERRO, M. *Avaliao de Sistemas de Computao Massivamente Paralela e Distribuda: Uma metodologia voltada aos requisitos das aplicaes cientficas*. Tese (Tese de Doutorado) — Laboratrio Nacional de Computao Cientfica, Petrpolis - RJ, Maio 2015. 3

[15] FERRO, M.; EVOY, G. M.; SCHULZE, B. Analysis of high performance applications using workload requirements. In: _____. *High Performance Computing for Computational Science – VECPAR 2016: 12th International Conference, Porto, Portugal, June 28-30, 2016, Revised Selected Papers*. Cham: Springer International Publishing, 2017. p. 7–10. ISBN 978-3-319-61982-8. Disponível em: <https://doi.org/10.1007/978-3-319-61982-8$_2$¿.3

[16] KALTENECKER, C. Comparison of analytical and empirical performance models: A case study on multigrid systems. 2016. 4

[17] MARTÍNEZ, V. et al. Performance improvement of stencil computations for multi-core architectures based on machine learning. *Procedia Computer Science*, Elsevier, v. 108, p. 305–314, 2017. 4, 5

[18] SIEGMUND, N. et al. Performance-influence models for highly configurable systems. In: ACM. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. [S.l.], 2015. p. 284–294. 4

[19] BERRAL, J. L.; GAVALDA, R.; TORRES, J. Power-aware multi-data center management using machine learning. In: IEEE. *2013 42nd International Conference on Parallel Processing*. [S.l.], 2013. p. 858–867. 4

[20] BERRAL, J. L.; GAVALDA, R.; TORRES, J. Adaptive scheduling on power-aware managed data-centers using machine learning. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. [S.l.], 2011. p. 66–73. 4

[21] BERRAL, J. L. et al. Towards energy-aware scheduling in data centers using machine learning. In: ACM. *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*. [S.l.], 2010. p. 215–224. 4

[22] BERRAL, J. L.; GAVALDÀ, R.; TORRES, J. Empowering automatic data-center management with machine learning. In: ACM. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. [S.l.], 2013. p. 170–172. 4

[23] BHIMANI, J. et al. Fim: performance prediction for parallel computation in iterative data processing applications. In: IEEE. *2017 IEEE 10th International conference on cloud computing (CLOUD)*. [S.l.], 2017. p. 359–366. 5

[24] MELO, A. C. D. The new linux'perf'tools. In: *Slides from Linux Kongress*. [S.l.: s.n.], 2010. v. 18. 5

[25] MITCHELL, T.; UTGOFF, P.; BANERJI, R. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In: MICHALSKI, R.; CARBONNEL, J.; MITCHELL, T. (Ed.). *Machine Learning: An Artificial Intelligence Approach*. Palo Alto, CA: Tioga, 1983. 7

[26] FERRO, M.; LEE, H. D.; ESTEVES, S. C. Intelligent Data Analysis: A Case Study of the Diagnostic Sperm Processing. In: *Proceedings ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA-02)*. Foz do Iguau, Brazil: [s.n.], 2002. p. 116–120. 7

[27] WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Second. [S.l.]: Morgan Kaufmann, 2005. 525 p. (Morgan Kaufmann Series in Data Management Systems). ISBN 0120884070. 7

[28] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. 7

[29] KOTSIANTIS, S.; KANELLOPOULOS, D.; PINTELAS, P. Data preprocessing for supervised leaning. *International Journal of Computer Science*, Citeseer, v. 1, n. 2, p. 111–117, 2006. 7

[30] MOTODA, H.; LIU, H. Feature Selection, Extraction and Construction. In: *Proceedings of Foundation of Data Mining, PAKDD*. [S.l.: s.n.], 2002. p. 67–72. 8