Towards an Autonomous Framework for HPC Optimization: Using Machine Learning for Energy and Performance Modeling

Vinícius Klôh, Matheus Gritz, Bruno Schulze, Mariza Ferro

[1]Laboratório Nacional de Computação Científica – LNCC
Av. Getúlio Vargas, 333 – 25651-075 – Quitandinha, Petrópolis – RJ –Brasil

{viniciusk, masgritz, schulze, mariza}@lncc.br

Abstract. *Performance and energy efficiency are now critical concerns in high performance scientific computing. It is expected that requirements of the scientific problem should guide the orchestration of different techniques of energy saving, in order to improve the balance between energy consumption and application performance. To enable this balance, we propose the development of an autonomous framework to make this orchestration and present the ongoing research to this development, more specifically, focusing in the characterization of the scientific applications and the performance modeling tasks using Machine Learning.*

Resumo. *Alcançar altos níveis de desempenho com eficiência energética se tornou um grande desafio para a computação científica de alto desempenho. Para contornar esse desafio, espera-se que os próprios requisitos do problema científico orientem a orquestração de diferentes mecanismos de economia de energia, a fim de melhorar o equilíbrio entre o consumo de energia e o desempenho das aplicações. Para isso, é proposto o desenvolvimento de um framework autonômico para fazer essa orquestração. Neste trabalho são apresentadas as pesquisas em andamento para este desenvolvimento, mais especificamente, com foco na caracterização das aplicações científicas e das tarefas de modelagem de desempenho e consumo de energia utilizando técnicas de Aprendizado de Máquina.*

1. Introduction

One of the greatest barriers to make the new generation of supercomputers feasible is the ever-growing energy consumption in these environments. Today's expenses with electric energy in petaflopic supercomputers already reach high values, making the effort to ensure that the increase in energy consumption is not proportional to processing capacity even more important [Silva et al. 2018]. However, balancing computing power and energy efficiency is not a trivial task. Most of the time, there's a direct trade-off between energy efficiency and processing power where increasing one decreases the other. For example, the first supercomputer in Green500 list of June 2019 [1], is in the 469 position in the Top500. The major challenge for the viability of exascale processing is to find a balance between these two factors and therefore different research projects, such as ECP [2], Mont-Blanc [3] and HPC4e [4], and works, such as [Simon 2013], [Rajovic et al. 2013] and [Messina 2017], seek solutions on this aspect.

Studies point out that this next generation of supercomputers will need to be developed using approaches where the requirements of the scientific problem guide computer architecture

[1]https://www.top500.org/green500/lists/2019/06/
[2]ECP - Exascale Computing Projections - https://www.exascaleproject.org/
[3]https://www.montblanc-project.eu/
[4]https://hpc4e.eu/

and system software design. In addition, these requirements 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 will be fundamental (energy aware).

To achieve this balance between performance and energy saving, we propose the development of an autonomous framework to make the orchestration among applications, architectures and schedulers, based on the requirements of the scientific applications. However, to reach this level of orchestration, there are challenging tasks to solve: (a) how to collect the relevant parameters in those heterogeneous HPC environments; (b) 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; (c) performance and energy modeling.

In this paper we present all the steps to select the relevant parameters to predict the runtime of an application in a given computational architecture and the obtained results.

2. Proposed Autonomous Framework

The proposed framework (Figure 1), currently in development, will allow the orchestration of techniques to reach the balance of performance and power consumption of parallel applications. It depends on input data such as application characteristics, architectural parameters and runtime node configuration. 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 [Ferro 2015, Ferro et al. 2017a] and could figure out of the knowledge base (KB) of the framework. This KB would be used by the scheduler to predict the runtime and energy consumption in order to choose the best architecture for the job and also the feasibility to select the best frequency, using Dynamic and Frequency Voltage Scaling (DVFS) techniques.

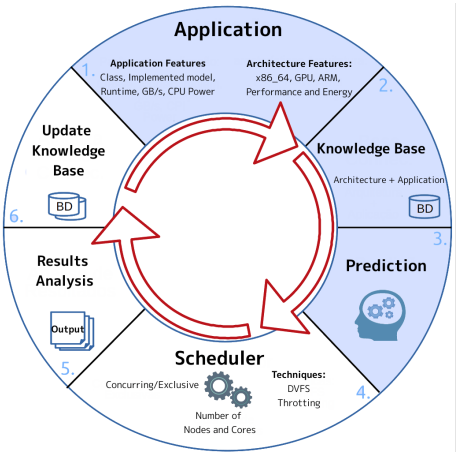

Figure 1. Proposed autonomous framework.

To reach this level of orchestration a number of different techniques must be studied, like: (a) how to access and collect performance counters and which are the relevant ones in those heterogeneous HPC environments; (b) the understand and modeling of the applications

and computational architectures; (c) Performance and energy modeling; (d) scheduling approaches and (e) the use of DFVS techniques. In this work we are focusing on (a) to (c), that comprises the blue part of the diagram on Figure 1.

The first phase involves the selection of performance counters that will serve as the basis of all the following steps. These performance counters are selected based on the main components of the hardware available and how their levels might impact the overall performance of an application. We presented our developments about the item (a) in [Silva et al. 2018] and [Ferro et al. 2017b] and in this work we are concerning only in, after enabling the collection, to define the relevant features to enable the performance and energy modeling.

At the second phase, a KB is built using the data collected during step one. At this point, the elements are cross-referenced to understand how they relate to each other and to the overall runtime to prepare the data to go through the prediction and scheduling phases. The third phase is the performance and energy modeling task, the main focus of this work, whose concepts are briefly discussed below.

2.1. Performance and Energy Modeling

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. In this work, we used a promising approach for empirical modeling, Machine Learning (ML) techniques, in order to predict the time an application takes to complete its task (performance) and the energy consumed for it, from the empirical data (sample set) collected from application runs in phase (a) – Figure 1.

This is an important and active area of research in HPC [Malakar et al. 2018]. But, accurate performance and energy modeling are complex because of the unknown interaction of the applications and system parameters in these complex systems. The input configuration, from where we receive the parameters for the prediction consists of the application (such as problem size, which can be multidimensional) and system parameters. The system parameters could change by each architecture and generation. So, the hardware counters to measure the performance and power consumption of an application, such as cache, cpu, IPMITool and so on, may have different access mode or may not even be present.

As mentioned, the main objective is to use these results to build an autonomous framework 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

This section will be focused on works that make use of hardware counters and ML to predict performance or energy.

Some works also tried to use ML models to predict and optimize applications based on data collected by performance counters. In [Martínez et al. 2017], the authors propose a specific ML 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 work of [Bhimani et al. 2017] propose a tool using ML 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 ML as their conclusion.

Other works use the approach to define an application signature, like ours, as important to predict performance, such as [Carrington et al. 2013] and [Wong et al. 2015]. However, they instrument the application for trace collection, which figure in significant effort spent on code instrumentation and incurs in high overheads of time and space.

4. Experimental Evaluation

In this study the experiments were divided in two phases: The first phase (Section 4.1) was the monitoring of the application, collecting the events described in the end of this Section. The second phase (Section 4.2) was the study of the ML techniques to gain knowledge about the hardware requirements and to define the relevant performance counters to enable the performance and energy modeling. Figure 2 presents the flowchart for these phases.

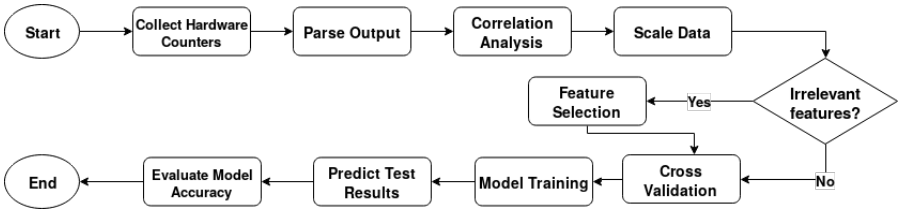

Figure 2. Flowchart detailing the project's workflow.

4.1. Experimental Setup - Monitoring Analysis

All data used in this work was collected in one of the nodes of the Beholder cluster at ComCiDis/LNCC [5]. It houses two Intel Xeon X5650S Hexa core CPUs with max clock of 2.67GHz 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 NAS Parallel Benchmark suite: 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 Motif's class (Dense Linear Algebra - DLA and Structured Grid - SG). These Motifs' classes represent applications with similar computational and communication characteristics and its use should enable a better understanding of the scientific applications [Mury et al. 2015]. For this study the input sizes chosen were A, B and C which correspond to the 3D matrix sizes 64, 102 and 162, respectively. For the LUD application, the 2D matrix size chosen was 128. Additionally, every application was executed 30 times varying the number of threads (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*. The collected data was done on a total of 2100 samples and it was used for the train and prediction phase.

The events were collected using *perf* tool, a performance counter tool available in the *Linux Kernel* since version 2.6.31: *Instructions**, *Cycles**, *CPU Migrations*, *Branches**, *Branch Misses**, *ContextSwitches**, *Cache References**, *Cache Misses*, *L1 dcache Stores**, *L1 dcache Loads**, *L1 dcache Load Misses*, *LLC Stores**, *LLC Store Misses*, *LLC Loads*, *LLC Load Misses*, *Page Faults*, *Minor Faults*, *Runtime**. The parameters with (*) were the ones selected and used for the ML predictive task.

[5] www.lncc.br

4.2. Experimental Setup - Machine Learning

A suitable approach for empirical modeling is supervised ML. This approach models the relationship between the output variable, such as runtime or energy consumption and one or more independent input variables (features) such as performance counters and application's size.

The first step in the Preprocessing phase was to scale the data in order to prevent issues with the difference in range between the features, which can lead to overfitting and underfitting among other common issues when using the data to train a ML model. The main objective when doing a scaling tasks such as this is to prevent performance issues and reduced accuracy in ML models caused by wildly different range 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 StandardScaler and MinMaxScaler from Scikit-learn [6] was used. It transforms all numeric data in a scale by removing the mean and scaling to unit variance.

For the training and test step, the supervised ML 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 to use it 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 for 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 and energy prediction.

A tree with the max depth between 3 and 8 was created and fitted using the train subsets randomly split during the step describe in Section 5, which are comprised of 75% of all the original data. Then, a prediction is generated using the test subsets. The metric used to measure how close the predictions are to the observed value was the Mean Absolute Percentage Error.

5. Experiments and Results

In this work, the targets are the runtime and the energy consumption, and the essential goal is to obtain knowledge that can be applied towards an equation that will allow to make a prediction of a estimated value. This is a complex task as it involves the features selection that are generic enough to allow accurate predictions and can be collected across different architectures. For this reason, three Feature Selection was made: The Feature Selection 1 (FS1) was based on the correlation analysis, the Feature Selection 2 (FS2) on the inclusion of the *matrix_workload* feature for the problem size performed and the Feature Selection 3 (FS3) including the feature *energy_joules* as a new target[7]. At this point, all features are numeric. Still, to reach knowledge of the application requirements, a new column with the sample classification, that corresponds to the Motif' class application, was included in the Feature Selection 4 (FS4).

Performance prediction: Initially, all the features presented in Section 4.1 (FS1) were used for the train and test phases, with its original scale and Decision Tree Regressor (DTR) with maximum depth between 3 and 8 were trained. The RT with maximum depth of 5 trained

[6] https://scikit-learn.org/stable/

[7] All the results can be accessed at `https://github.com/ViniciusPrataKloh/WSCAD_2019`.

with FS1 features returned a MAPE of 12.854%, showing the importance of these features. However, as too many rules had been created by the model, few samples were covered by each leaf, indicating that even though the error is acceptable, the generalization wasn't good and the result offers no real practical use for this problem. The DTRs with the best sample coverage were the ones with the maximum depth of 3 and 4 but their MAPE were far too high at 718.525% and 693.462%. This indicates that the generalization was bad for this dataset.

In order to improve the precision and generalization capabilities, new experiments were ran with the feature set FS2 to predict the performance and energy consumption an important feature is the problem size. However, this a difficult feature to be defined as it may vary with each type of problem (eg. text, arrays and pictures). In this work, the array's size was used provisionally as all applications executed involve some form of array. So, in this step, the feature *workload_size* was evidently important as it became the main feature used to split the tree, reducing the MAPE in DTRs with maximum depth of 3 and 4 to 38.383% and 23.801%, respectively, that is, a reduction of approximately 18.7x and 29x in error.

Figure 3 presents the DTR with maximum depth of 3 and its regression rules. Each node contains a MAE (Mean Absolute Error) value, which represents the mean of the difference between the actual and the predicted values, the number of samples covered in it and the predicted value for that node.

It uses the feature *workload_size* as the root, splitting in *branches* and *L1-dcache-load-misses*. The "False" path had a good generalization, covering 484, 437, 161 and 337 samples (out of a total of 1575 samples) per leaf. On the other hand, the generalization on the "True" path wasn't as good because of how the samples were distributed during training, meaning that few samples in the dataset were covered in these nodes. New experiments are needed with a larger range of workload sizes in order to attain better results.

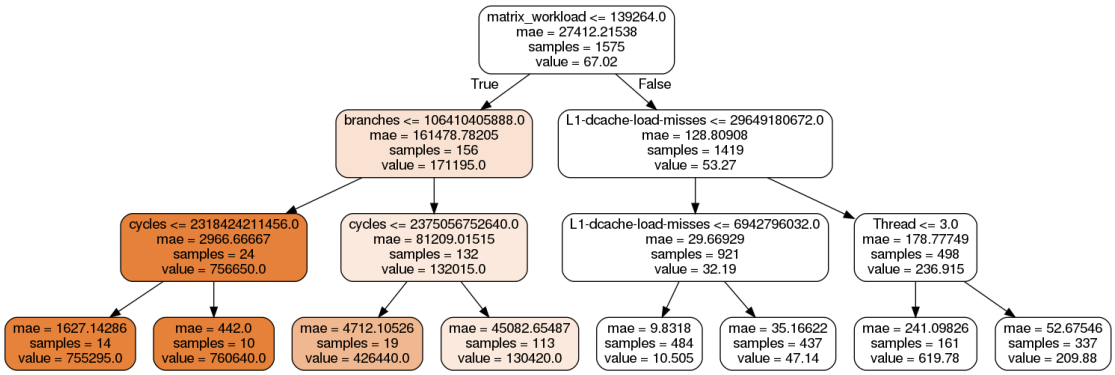

Figure 3. Regression Tree for the Performance.

To evaluate the relevance of scaling the data, two more groups of experiments were ran, one using StandardScaler and the other MinMaxScaler with a range between -1 and 1. The MAPE in DTRs with maximum depth of 3 and 4 were 2.848% and 1.703% (StandardScaler), 2.401% and 1.18% (MinMaxScaler), respectively. A reduction of approximately 13.5x and 14x in error.

All the results for the runtime can be observed in left side of the Table 1 and for the energy, int the right side. As can be seen, the MAPE was reduced with the increase of the depth (from 3 to 8), and the MinMaxScaler presents the better results for the prediction.

Energy prediction: For the *energy_joules* as the target (FS3), the MAPE for the DTRs with

maximum depth of 3 and 4 without scaling the data was 6663.895% and 5626.854%. After scaling, the MAPE went down to 79.8% and 85.212% and 11.135% and 3.605% after applying the StandardScaler and MinMaxScaler (with a range of -1 and 1), respectively.

The DTR selected different features from that used to predict the runtime, with *context-switches*, *L1-dcache-loads*, *caches-misses*, *LLC-load-misses*, *L1-dcache-stores*, *cpu-migrations* being its main rules. This shows the complexity of reaching a set of generic features for accurately modeling and predicting performance and energy consumption.

Runtime

	No Scaling		StandardScaler		MinMaxScaler	
	MAPE	**R2**	**MAPE**	**R2**	**MAPE**	**R2**
3	38.383	0.977	2.848	0.977	2.401	0.959
4	23.801	0.991	1.703	0.991	1.18	0.973
5	12.759	0.999	0.708	0.999	0.49	0.98
6	6.766	0.999	0.425	0.999	0.395	0.981
7	3.668	0.981	0.284	0.999	0.297	0.981
8	2.146	0.999	0.366	0.999	0.38	0.981

Energy

	No Scaling		StandardScaler		MinMaxScaler	
	MAPE	**R2**	**MAPE**	**R2**	**MAPE**	**R2**
3	6663.895	0.915	79.8	0.915	11.135	0.915
4	5626.854	0.96	85.212	0.961	3.605	0.96
5	4603.152	0.964	51.688	0.965	2.761	0.939
6	4402.886	0.974	18.152	0.975	1.512	0.974
7	3179.646	0.975	19.906	0.975	1.335	0.975
8	2669.094	0.972	19.264	0.972	1.318	0.972

Table 1. Regression Metrics.

Application classification: For the Application Classification, a new featured (target) called "Classes" was added to represent the classes DLA and SG. The Confusion Matrix generated by the test data was $\left[\begin{smallmatrix} 371 & 0 \\ 0 & 154 \end{smallmatrix}\right]$, representing 100% of accuracy in the test phase.

The features *L1-dcache-load-misses*, *branch-misses* and *context-switches* were also selected in the Regression Tree at maximum depth of 4, for runtime prediction. This shows the importance of these features in the composition of an application's signature, in the performance prediction and to classify them according to their hardware requirements.

According to the domain expert, the features used in the DTC are not sufficient to clearly classify new data. This was confirmed in the validation stage, as only approximately 59% of correct answers were obtained, showing that the generalization process needs to be improved. Still, it is well known that the application classes used, even the CPU intensive DLA class, uses a large amount of memory [Ferro 2015]. Even though there is an overfitting in the classification, the attributes selected by the tree still comply with the classes' requirements.

6. Final considerations

This paper proposes a empirical modeling to predict the performance and energy consumption, and to classify the application according to the hardware requirements, presenting all the steps to select the relevant parameters. This work contributes with the importance of the feature selection and data scaling, as the error metrics had significant reductions in the trained models.

Also, the performance counters were proven to be effective feature in the proposed prediction and classification problems. As observed, each application class had a different impact on hardware usage, but in general, features related to cache were present in all tests, following the computational requirements of the selected application classes.

In future works, new experiments will be carried out with a larger range of applications, classes and performance counters in order to improve the model's generalization capabilities and better understanding of these features.

Acknowledgments

Authors would like to thank the financial support from CNPq, CAPES and Faperj.

References

Bhimani, J., Mi, N., Leeser, M., and Yang, Z. (2017). Fim: performance prediction for parallel computation in iterative data processing applications. In *2017 IEEE 10th International conference on cloud computing (CLOUD)*, pages 359–366. IEEE.

Carrington, L., Laurenzano, M. A., and Tiwari, A. (2013). Inferring large-scale computation behavior via trace extrapolation. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 1667–1674.

Ferro, M. (2015). *Avaliação de Sistemas de Computação Massivamente Paralela e Distribuída: Uma metodologia voltada aos requisitos das aplicações científicas*. Tese de doutorado, Laboratório Nacional de Computação Científica, Petrópolis - RJ.

Ferro, M., Mc Evoy, G., and Schulze, B. (2017a). *Analysis of High Performance Applications Using Workload Requirements*, pages 7–10. Springer International Publishing, Cham.

Ferro, M., Silva, G. D., Klóh, V. P., and Schulze, B. (2017b). *Challenges in HPC Evaluation: Towards a methodology for scientific applications' requirements*. IOS Press, Amsterdam. accepted to publish.

Malakar, P., Balaprakash, P., Vishwanath, V., Morozov, V., and Kumaran, K. (2018). Benchmarking machine learning methods for performance modeling of scientific applications. *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*.

Martínez, V., Dupros, F., Castro, M., and Navaux, P. (2017). Performance improvement of stencil computations for multi-core architectures based on machine learning. *Procedia Computer Science*, 108:305–314.

Messina, P. (2017). The exascale computing project. *Computing in Science Engineering*, 19(3):63–67.

Mury, A. R., Schulze, B., Licht, F., de Bona, L. C., and Ferro, M. (2015). A concurrency mitigation proposal for sharing environments: An affinity approach based on applications classes. In Al-Saidi, A., Fleischer, R., Maamar, Z., and Rana, O. F., editors, *Intelligent Cloud Computing*, volume 8993 of *Lecture Notes in Computer Science*, pages 26–45. Springer International Publishing.

Rajovic, N., Carpenter, P. M., Gelado, I., Puzovic, N., Ramirez, A., and Valero, M. (2013). Supercomputing with commodity cpus: Are mobile socs ready for hpc? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 40:1–40:12, New York, NY, USA. ACM.

Silva, G. D., Klôh, V. P., Yokoyama, A., Ferro, M., and Schulze, B. (2018). Smcis: Scientific applications monitoring tool for hpc environments. In *2018 Symposium on High Performance Computing Systems (WSCAD)*, pages 148–154. https://ieeexplore.ieee.org/document/8748925.

Simon, H. D. (2013). *Barriers to Exascale Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wong, A., Rexachs, D., and Luque, E. (2015). Parallel application signature for performance analysis and prediction. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2009–2019.