










Unified Power Modeling Design for Various Raspberry Pi Generations Analyzing Different Statistical Methods

Juan Manuel Paniego¹ , Leandro Libutti¹ , Martin Pi Puig¹ ,
Franco Chichizola¹ , Laura De Giusti^{1,2} , Marcelo Naiouf¹ ,
and Armando De Giusti^{1,3} 

¹ Computer Science Research Institute LIDI (III-LIDI) (CEA-CIC),
National University of La Plata, 1900 La Plata, Argentina
{jspaniego,llibutti,mpipuig,francoch,ldgiusti,
mnaiouf,degiusti}@lidi.info.unlp.edu.ar

² Scientific Research Agency of the Province of Buenos Aires (CICPBA),
La Plata, Argentina

³ National Council of Scientific and Technical Research (CONICET),
Buenos Aires, Argentina

Abstract. Monitoring processor power is important to define strategies that allow reducing energy costs in computer systems. Today, processors have a large number of counters that allow monitoring system events such as CPU usage, memory, cache, and so forth. In previous works, it has been shown that parallel application consumption can be predicted through these events, but only for a given SBC board architecture. In this article, we analyze the portability of a power prediction statistical model on a new generation of Raspberry boards. Our experiments focus on the optimizations using different statistical methods so as to systematically reduce the final estimation error in the architectures analyzed. The final models yield an average error between 2.24% and 4.45%, increasing computational cost as the prediction error decreases.

Keywords: Power · Raspberry Pi · Hardware counters · Modeling · Statistical models

1 Introduction

One of the main challenges in system design is the need for fast and accurate energy consumption prediction. In recent years, various innovations have helped meet this requirement.

The underlying idea is that power consumption depends not only on hardware, but also on the use of the software and its internal characteristics. For example, more complex software will require more CPU cycles, or a single huge disk write operation can require less power than multiple small write operations. In general, if a person is aware of how much power they are using, they can find their own suitable solution to save energy when using their device [1, 2].

While it is possible to perform hardware measurements on the system to implement fine-grained power control, such instrumentation is expensive and not frequently available.

However, with information about application execution, power consumption can be estimated. These data are collected through hardware counters that can be used to monitor a wide variety of events related to system performance with high precision. Since each event is associated with a certain level of energy consumption, any one of them can be used as a parameter in a performance model. While some events represent activities with little impact on energy, correlation is high in others. Even though the number of counters is usually limited, they can be used to count a wide variety of events.

One way to estimate energy consumption is by generating a statistical model based on these counters. To obtain a real-time prediction, a limited set of events must be selected that describes most of the variation in power.

In [3], a power estimation model was developed using hardware counters which did not contemplate the variation in the number of threads of the parallel application. Similarly, the statistical linear regression model was used, which is intrinsically simple to apply, but may result in greater estimation error. This model was implemented using the Raspberry Pi 3 model B (RPI3B) development board.

In [4], the error obtained using the aforementioned model on the successor board Raspberry Pi 3 model B+ (RPI3B+) is analyzed. Then, to improve predictions, the power model developed for multi-thread applications supported by both boards is optimized.

This work is an extension of [4]; here, different statistical methods are analyzed that allow obtaining better accuracy in the prediction.

The article is organized as follows: Sect. 2 presents an overview of related works in the energy consumption prediction field using different CPU and GPU architectures. Section 3 presents the process for obtaining a statistical model that is compatible with RPI3B and RPI3B+ boards and validates the model generated using linear regression. Section 4 analyzes other statistical methods present in the development tool and compares them to the one discussed in Sect. 3. Finally, in Sect. 5, conclusions and future works are presented.

2 Related Work

This section presents some related previous research works. Lee et al. [5] proposed regression modeling as an efficient approach to accurately predict performance and power for various applications that use any microprocessor configuration considering various microarchitecture designs, addressing cost simulation as a fundamental challenge to obtain correct values in the prediction. With the appearance of hardware counters, Weaver et al. [6] analyzed the values obtained with them to check if there is a good correlation with what was happening inside the processor architecture. From their results indicated that it is reasonable to expect that counters reflect processor behavior. This allowed many researchers to use tools to extract performance values.

Singh et al. [7] developed a model to measure processor power consumption in real time by compiling the information provided by the counters.

On the other hand, Bircher et al. [8–10] explored the use of performance counters to predict the energy consumption of various subsystems such as CPU, memory, chipset, I/O, disk, and GPU. It was developed and validated on two different platforms. Likewise, Rodrigues et al. [11] studied the use of applied performance counters for estimating energy consumption in real time on two different architectures – one oriented to high performance, and the other based on low consumption.

On the other hand, Lively et al. [12] developed a set of hybrid application-centric performance and consumption estimation models. They analyze a set of scientific codes in their MPI/OpenMP implementation, and generate an appropriate procedure to carry out modeling and validation.

Asymmetric core architectures have recently emerged as a promising alternative in an environment with power and thermal limitations. They typically integrate cores with different power and performance characteristics, which makes assigning workloads to the appropriate cores a challenging task. Pricopi et al. [13] presented a model for asymmetric multi-cores in which the performance and power consumption of the workloads assigned to each core can be obtained using the hardware counters.

More recently, with the use of FPGAs, O’Neal et al. [14] developed predictive performance and power consumption models for CPUs, GPUs, and FPGAs, saving simulation costs.

With the appearance of low consumption Single Board Computers (SBCs) boards, the authors of [3] designed a statistical model to predict the energy consumption of applications run on the RPI3B board. However, this model is limited in that it only allows consumption to be predicted for sequential execution and with four cores, added to the disadvantage that different prediction coefficients are used based on the number of threads. This article is an evolution of that previous work, and focuses on developing a model that allows evaluating parallel applications taking into account the variation in the number of cores used. Likewise, the need to build a multi-architecture model that considers the technological changes of new generations of SBC boards is highlighted.

3 Generating a Single Power Model for Various Raspberry Pi Generations

Because this work is based on statistical models, information extracted from applications with different computational behaviors should be used, so as to obtain data related to performance and energy consumption for the analyzed architectures.

In particular, the methodology used in our previous work [3] is applied: use of NAS [15] and RODINIA [16] benchmarks, which have different computational behaviors; instrumentation and parallel applications source code compilation; collection of performance counters and instantaneous power sampling; counter-power correlation and mutual correlation between counters; linear regression model training and model validation through the technique of leaving one out.

Taking into account the model obtained in [3] (which we will call the “Original Model”), the necessary modifications are applied to include the new Raspberry

generation, generating a new statistical model that allows reducing the final estimation error; we will call this updated version the “Unified Model”.

Since RPI3B and RPI3B+ are compatible SBC boards, the instrumented source code is reused for the different applications.

The model is based on linear regression, a statistical engine that presents a regressand, an offset constant and predictor or independent variables. Based on the architecture used, five independent variables are used, which correspond to the performance counters $L2_DCM$, $L2_DCA$, SR_INS , BR_INS and TOT_INS . In addition, the TOT_CYC counter is included in order to normalize previous events, obtaining five performance ratios.

First, the predictions obtained using the Original Model on the new generation of Raspberry boards are studied. All necessary optimizations are then carried out to reduce the error and thus obtain an accurate estimate [4].

3.1 Prediction with the Original Model

The model presents a constant that is added to the normalized values of the five counters, which have an associated weight. Equation 1 shows how to obtain the estimated power for the parallel applications.

$$Y_i = 1.595 + 14.696L2_{DCM} + 2.308L2_{DCA} + 0.108SR_{INS} + 0.093BR_{INS} + 0.058TOT_{INS} \quad (1)$$

To estimate the power required by the RPI3B+ board, we started by compiling all applications. Then, we proceeded to record used power and performance counters in each application running with 1, 2 and 4 threads.

Once the information is generated, the values obtained for each parallel application from the six counters used in the model are used to estimate power for the new board.

This prediction has an average percentage error of 30% in the RPI3B+ , as opposed to the 6.8% recorded for RPI3B [3]. This increase in error for the new boards is explained by the architectural difference between generations, since the new version has a higher clock rate.

3.2 Generating the Unified Model

Figure 1 shows the actual power used by the different parallel applications on each board. The increase in power as more threads are used is similar in both architectures. The 23 sequentially run applications (1 thread), found in the first third in the figure, represent the lowest power consumption on both boards. On the other hand, with 2-thread and 4-thread parallel execution (second and third thirds in the figure, respectively) devices are better exploited and, therefore, power consumption is greater.

For the RPI3B board, consumption ranges between 2–3 watts, while for the RPI3B+ model, this range is between 3–4 watts approximately. The first step in finding a unified consumption prediction model for both boards is to study the cause of this difference.

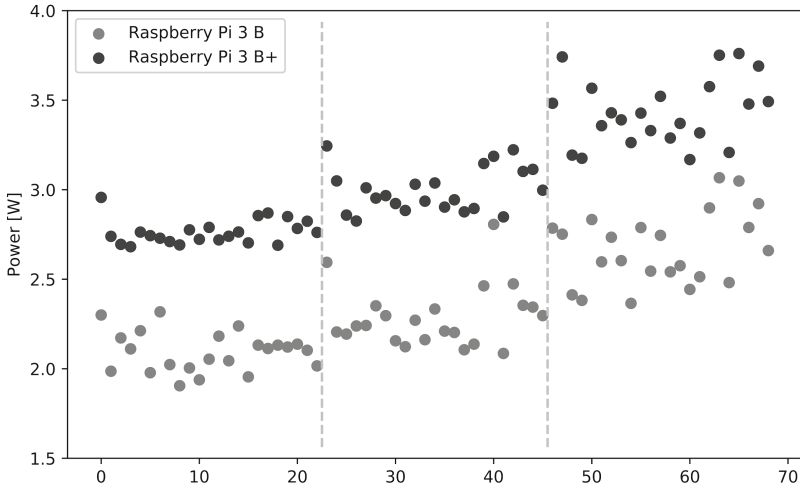


Fig. 1. Actual power for each application, considering number of threads.

First, the values recorded by performance counters feeding the model for each application are analyzed. In this case, the CFD application is chosen as an example; however, the other algorithms show the same behavior in their counters. Figure 2 shows that all events have the same trend in both development boards. Since these boards have several architectural similarities (cache levels and sizes, volatile memory, execution pipeline, etc.), running the same applications yields similar behavior results. Therefore, since there is no variation in performance counter values between the boards, the model cannot be used to differentiate application execution between them.

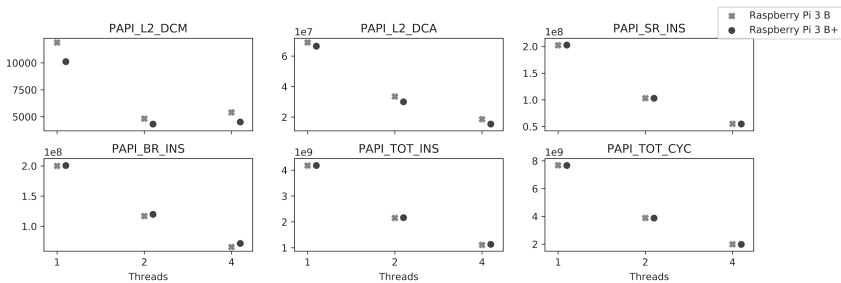


Fig. 2. Performance counter values on each development board considering number of threads running on the CFD application.

Subsequently, the generational changes between the two boards are analyzed to assess their effect on the prediction. The most significant change between both boards is the increase in clock rate from 1.2 GHz to 1.4 GHz. As a result, applications require more power to run, which can cause the increase analyzed in Fig. 1. To verify this, a predictor corresponding to maximum frequency value at runtime is added to the

statistical model. This change decreased the error in the estimate, reaching 14% on average.

When training the original model, only the execution of 1- and 4-threaded applications was taken into account, since the goal in [3] was to evaluate the power for sequential and parallel executions with the maximum number of cores available in the processor. Therefore, to consider the impact of the level of parallelization used in the application, a predictor that considers the number of threads (cores) necessary for execution is added, since the applications are developed with the OpenMP shared memory multi-thread programming interface [17].

These optimizations allow estimating used power in both versions of the development board. Thus, the final statistical model based on linear regression (Unified Model) is described in Eq. 2.

$$Y_i = \beta_0 + \beta_1 L2_{DCM} + \beta_2 L2_{DCA} + \beta_3 SR_{INS} + \beta_4 BR_{INS} + \beta_5 TOT_{INS} + \beta_6 THREADS + \beta_7 MAX_{FREQUENCY} \quad (2)$$

To build the statistical model, the RapidMiner development tool was replaced by Python. This allows optimizing data cleaning time and the creation process of the prediction model with the inclusion of embedded libraries within the tool. It also provides different statistical models with efficient training. In addition, it allows to easily customize parameters to train each model. Table 1 shows weight β_i values after the model has been trained.

Table 1. Weights obtained for each predictor.

Predictor	Counter	Coefficient	Value
–	INTERCEPT	β_0	–2.656
X_1	L2_DCM	β_1	18.437
X_2	L2_DCA	β_2	4.381
X_3	SR_INS	β_3	0.037
X_4	BR_INS	β_4	–0.032
X_5	TOT_INS	β_5	0.169
X_6	THREADS	β_6	0.221
X_7	MAX_FREQUENCY	β_7	3.546

3.3 Evaluating the Unified Model

After training the model, which results in a set of weights that are applied to the aforementioned predictors, the result is evaluated against the actual power values measured. The predictions obtained for both Raspberry generations can be seen in Fig. 3, together with actual power used. Two types of samples are observed – circles represent the applications run on RPI3B, while triangles correspond to applications run on RPI3B+ .

Finally, prediction errors for each model can be seen in Fig. 4. The X axis lists each application analyzed, the Y axis represents the model used to run the applications, and the Z axis corresponds to the percentage error in the prediction. Model 0 corresponds to the Original Model developed in [3]. In model 1 (Model Opt. 1) clock rate is added as a predictor. As a last improvement, model 2 (Unified Model) adds the independent variable that corresponds to the number of threads that the application uses for its parallel execution.

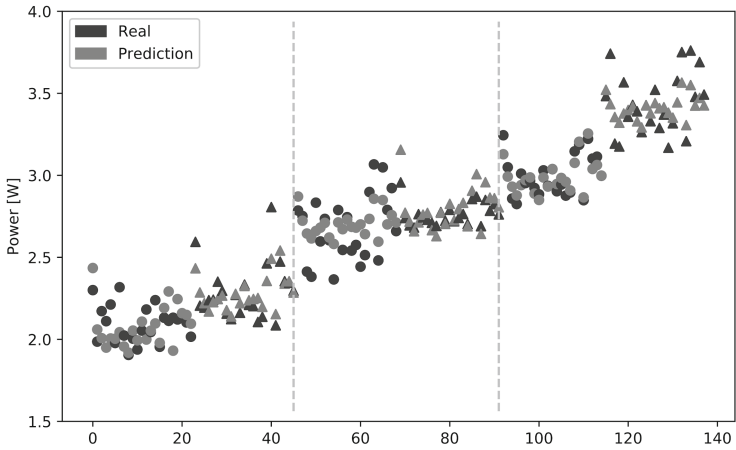


Fig. 3. Used power prediction for both boards.

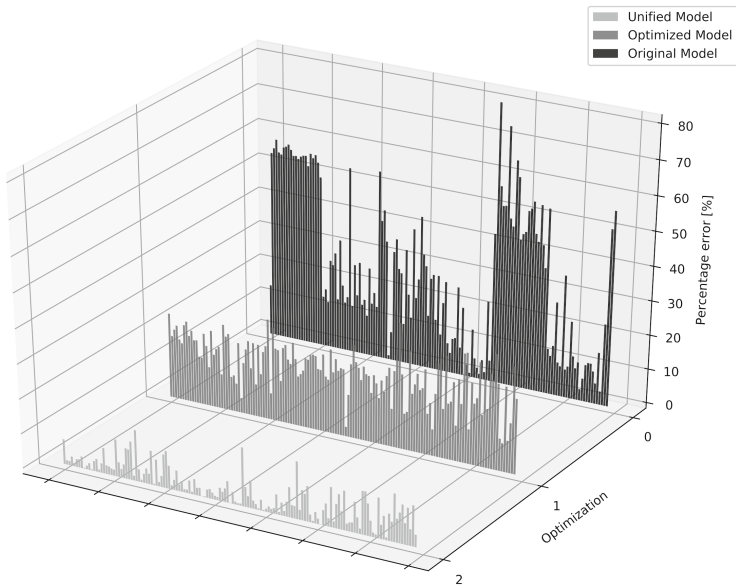


Fig. 4. Application prediction error comparison for the improvements proposed in the statistical model.

In the Unified Model, the average prediction error taking into account all tests carried out is 4.45%, the maximum prediction error is 16.95%, and error dispersion or standard deviation for the sample set is 3.85%.

3.4 Validating the Unified Model

To estimate model accuracy, the leave-one-out cross-validation (LOOCV) technique is used. This evaluation method is better than residual ones. The main problem with residual methods is that they do not generate an indicator of how the model behaves with predictions for applications not included in the training phase.

One possible solution is not using the entire set of applications for training. After obtaining the model, the set of applications that was removed for training is used to make the prediction. This type of evaluation methods is known as cross-validation.

Among the different cross-validation variants, the leave-one-out technique is used in our model, which allows separating the information in such a way that, for each iteration, a single sample is destined for the test data while the remaining set makes up the model’s training data. Then, the average of the errors in each iteration is calculated to obtain the final error for the validation.

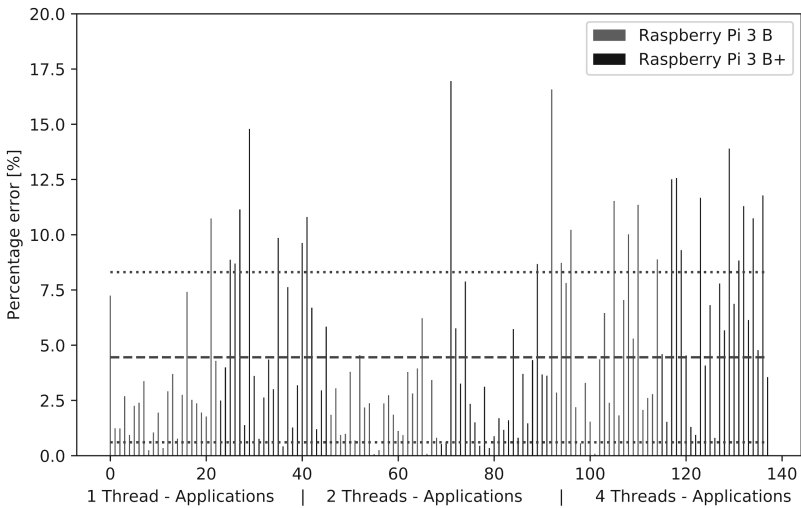


Fig. 5. Prediction error percentage for each application.

Figure 5 shows the error for each iteration with the validation technique used. The maximum prediction error is 18.46%. Error dispersion or standard deviation for the sample set is 4.14%. The average error for all iterations is 4.76%, which is considered acceptable and does not result in great differences between actual and estimated power to run a parallel application.

4 Analysis of Other Statistical Models

In an attempt to minimize the estimation error of the linear regression statistical method, various models from the Python's *Sklearn* library [18] were studied.

4.1 Support Vector Regression (SVR)

This algorithm is a modified version of SVM (Support Vector Machine) used for classification in machine learning. SVM generates a hyperplane that separates data maximizing margin. To create the margin, two lines including all data are defined. Each predictor used generates one dimension, resulting in a hyperplane with D (number of predictors) dimensions. Kernels allow decreasing the number of input dimensions. Finally, Support Vectors refers to all the data enclosed by margin lines [19].

The model generated with this statistical method is evaluated for each application, obtaining the percentage error relative to actual power consumption. Figure 6 shows the application errors for each generation of RPI3 for different number of execution threads (1, 2 and 4 threads). SVR generates an average prediction error of 2.24% and a maximum error of 17.66%.

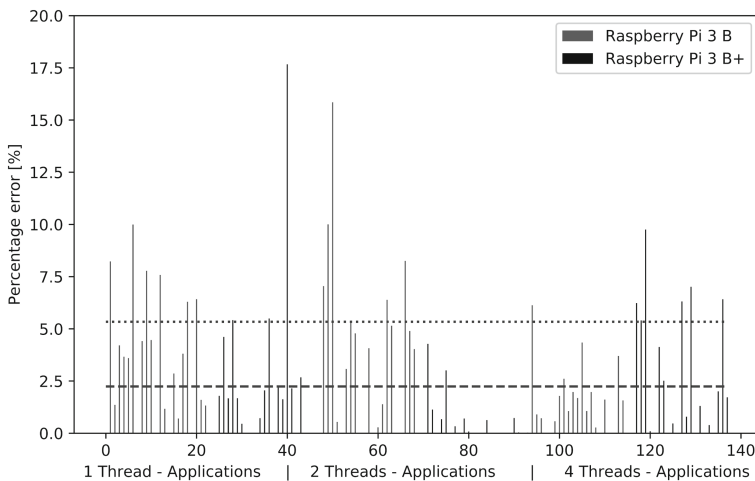


Fig. 6. Prediction error for each application evaluated with the SVR model.

4.2 Gaussian Regression (GR)

The regression of the Gaussian process is not parametric (that is, it is not limited by a functional form) so, instead of calculating the probability distribution of the parameters of a specific function, GPR calculates the probability distribution over all allowable functions that fit the data. GPR has several benefits, including good performance in small data sets and the ability to provide uncertainty measurements on predictions [20]. For this model, the procedure used in the previous method is repeated. Figure 7 shows an average error of 2.58% and a maximum error of 11.21%.

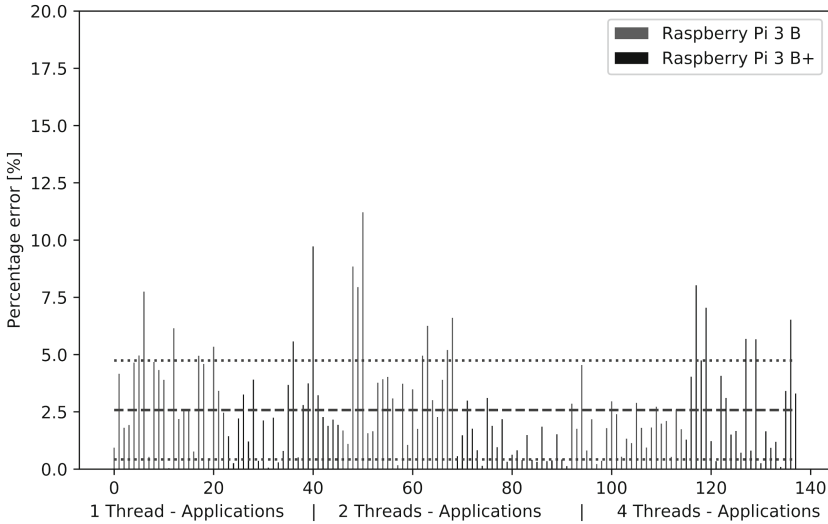


Fig. 7. Prediction error for each application evaluated with the Gaussian Regression model.

4.3 Kernel Ridge Regression (KRR)

Kernel ridge regression is a non-parametric form of *ridge regression* [21, 22]. The aim is to learn a function in the space induced by the respective kernel k by minimizing a squared loss with a squared norm regularization term. The form of the model learned by *KernelRidge* is identical to SVR. However, different loss functions are used: KRR uses squared error loss while support vector regression uses e-insensitive loss.

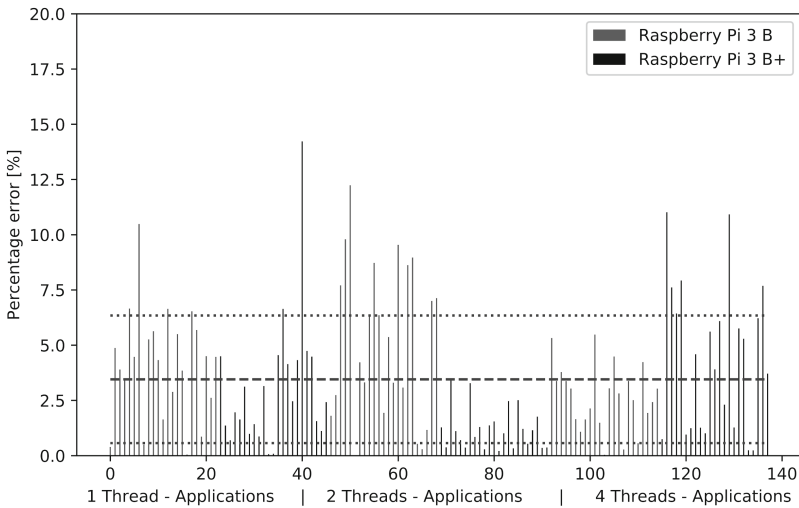


Fig. 8. Prediction error for each application evaluated with the Kernel Ridge Regression model.

Figure 8 shows that the average prediction error is 3.45%, and the maximum error is 12.23%.

4.4 Statistical Models Comparison

After generating the prediction models based on the different statistical methods, each behavior should be analyzed to choose the one with the best performance and the lowest cost at runtime.

Table 2 shows, for each statistical method, prediction average and maximum error, as well as the increase in training time compared to the Linear Regression model.

To achieve a training process with lower execution time but higher error percentage, the Linear Regression method should be used. On the other hand, if a higher prediction accuracy is desired, regardless of how long training takes, then the SVR model is better.

Table 2. Average prediction error for supplementary models.

Model	Mean error [%]	Max error [%]	Increase training time
SVR	2.24	17.66	55.9X
Gaussian Regression	2.58	11.21	1.56X
Kernel Ridge Regression	3.45	14.22	1.25X
Linear Regression	4.45	16.95	1X

5 Conclusions and Future Work

In this article, power consumption prediction for various parallel applications based on readings from performance counters present on the RPI3B and RPI3B+ development boards was discussed.

We considered the possibility of adding to a previously built power model (compatible only with RPI3B), new predictors that allow integrating new board generations, as well as considering the level of parallelism applied to each algorithm.

A statistical power model based on linear regression was designed for multi-thread applications, and it was validated using samples obtained from the analyzed boards. The previously created model was optimized by adding the number of threads used by the application and maximum board frequency as predictors.

Model validation (using LOOCV) yielded an average prediction error of 4.76%, meaning that power consumption was estimated with a high degree of accuracy for both boards.

Different statistical techniques were analyzed in an attempt to reduce the estimation error: SVR, KRR, and GR. These achieved estimation average error reductions between 22.48% and 49.67%, compared to the Linear Regression method, which help achieve significant improvements in the prediction. As a disadvantage, there is an increase in training time as prediction accuracy increases.

Based on the target error margin for the prediction, one of the different models can be chosen:

- Linear Regression is simple and requires less training time. Its disadvantage is that it has a high average prediction error.
- SVR yields the lowest average error, but it is the most expensive method in terms of training time. Additionally, it does not minimize maximum prediction error.
- On the other hand, GR has a low average error (close to that achieved by SVR), but with the advantage that it minimizes maximum prediction error.

As a future line of work, we are planning to apply the methodology developed to the new *Raspberry Pi 4* development board. Similarly, we are planning on implementing a tool that collects all required information in real time, applies the proposed statistical model and generates a report on power consumption for a given parallel application.

References

1. Bekaroo, G., Santokhee, A.: Power consumption of the Raspberry Pi: a comparative analysis. In: 2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies, EmergiTech, pp. 361–366. IEEE (2016)
2. Procaccianti, G., Ardito, L., Vetro, A., Morisio, M., Eissa, M.: Energy efficiency in the ict-profiling power consumption in desktop computer systems. In: Eissa, M., (ed.) Energy Efficiency-The Innovative Ways for Smart Energy, the Future Towards Modern Utilities/InTech, pp. 353–372 (2012)
3. Paniego, J.M., et al.: Modelado estadístico de potencia usando contadores de rendimiento sobre Raspberry Pi. In: XXIV Congreso Argentino de Ciencias de la Computación. (2018)
4. Paniego, J.M., et al.: Armando De Giusti Modelado de potencia en placas SBC: integración de diferentes generaciones Raspberry Pi. In: Libro de actas XXV Congreso Argentino de Ciencias de la Computación, CACIC 2019, pp. 159–169 (2019). ISBN 978-987-688-377-1
5. Lee, B.C., Brooks, D.M.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. *ACM SIGOPS Oper. Syst. Rev.* **40**(5), 185–194 (2006)
6. Weaver, V.M., McKee, S.A.: Can hardware performance counters be trusted?. In: 2008 IEEE International Symposium on Workload Characterization, pp. 141–150. IEEE (2008)
7. Singh, K., Bhadauria, M., McKee, S.A.: Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Comput. Archit. News* **37**(2), 46–55 (2009)
8. Bircher, W.L., John, L.K.: Complete system power estimation using processor performance events. *IEEE Trans. Comput.* **61**(4), 563–577 (2011)
9. Bircher, W.L., John, L.K.: Complete system power estimation: a trickle-down approach based on performance events. In: 2007 IEEE International Symposium on Performance Analysis of Systems & Software, pp. 158–168. IEEE (2007)
10. Bircher, W., Law, J., Valluri, M., John, L.K.: Effective use of performance monitoring counters for run-time prediction of power. University of Texas at Austin Technical report TR-041104, 1 (2004)
11. Rodrigues, R., Annamalai, A., Koren, I., Kundu, S.: A study on the use of performance counters to estimate power in microprocessors. *IEEE Trans. Circuits Syst. II Express Briefs* **60**(12), 882–886 (2013)

12. Lively, C., et al.: Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems. *Comput. Sci. Res. Dev.* **27**(4), 245–253 (2012)
13. Pricopi, M., Muthukaruppan, T.S., Venkataramani, V., Mitra, T., Vishin, S.: Power-performance modeling on asymmetric multi-cores. In: *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, p. 15. IEEE Press (2013)
14. O’Neal, K., Brisk, P.: Predictive modeling for CPU, GPU, and FPGA performance and power consumption: a survey. In: *2018 IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, pp. 763–768. IEEE (2018)
15. Bailey, D.H., et al.: The NAS parallel benchmarks. *Int. J. Supercomput. Appl.* **5**, 63–73 (1991)
16. Che, S., et al.: Rodinia: a benchmark suite for heterogeneous computing. In: *IEEE International Symposium on Workload Characterization, IISWC*, pp. 44–54 (2009)
17. OpenMP. <https://www.openmp.org/>
18. Scikit-learn. <https://scikit-learn.org/stable/>
19. Epsilon-Support Vector Regression. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>
20. Quick Start to Gaussian Process Regression. <https://towardsdatascience.com/quick-start-to-gaussian-process-regression-36d838810319>
21. Kernel Ridge Regression. https://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.KernelRidge.html
22. Ridge Regression. https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression