Evaluating Neural Network Methods for PMC-based CPU Power Prediction

Mario Gutierrez, Dan Tamir, and Apan Qasem
Department of Computer Science
Texas State University
San Marcos, TX
{mag262, dt19, apan}@txstate.edu

***Abstract*—Emphasis on energy efficient computing has established power consumption, as well as energy and heat dissipation as determinant metrics for analyzing High performance computing applications. Consequently, optimizations that target High performance computing systems and data centers have to dynamically monitor system power consumption in order to be effective. Current architectures are exposing on-chip power sensors to designers and users. The general state of power measurement tools across different architectures, however, remains deficient. Recent research has shown that first-order, linear models can be effectively used to estimate real-time power consumption. This paper describes a neural-network based model for fine-grain, accurate and low-cost power estimation. The proposed model takes advantage of the wide array of performance monitoring counters available on current systems. We analyze the prediction capability of the model under various scenarios and provide guidelines for feature selection for other machine learning models for estimating power consumption on future architectures.**

Keywords–energy efficiency; power consumption; workload characterization, performance counters.

I. INTRODUCTION

The proliferation of portable wireless devices, along with the rapid growth of high-performance server farms and data centers, have made energy efficiency a central concern for the entire computing industry. Poorly managed and unbalanced power consumption of computing systems has direct economic and environmental impact in the form of high energy bills and large carbon footprints. Furthermore, it leads to device reliability degradation and high chip packaging costs. In recent years, numerous strategies for reduction in power usage have emerged. Current research is focused on hardware techniques such as developing new energy conserving components and on software strategies that aim to exploit existing hardware in an energy-efficient manner.

Most software techniques rely on methods for estimating system power in making optimization decisions. Several architectures provide a mechanism for measuring power directly. But for architectures that do not expose these metrics, obtaining system power involves attaching an external device or running costly simulations [1]. Neither method is suitable for software-based techniques that need to react to changes in power usage and make real-time decisions to conserve energy. Furthermore, even on platforms where the power counters are available, measurements incur a high overhead.

An efficient solution can be constructed via the reuse of hardware devices. One of the most important elements used in previous research is the set of built-in Performance Monitoring Counters (PMCs). These are registers built into the Central Processing Unit (CPU) and/or into the Performance Monitoring Unit (PMU) that can track performance-related information such as instruction counts, cache misses, and resource stalls.

Several models that correlate program behavior, captured via PMCs, to system power consumption have been proposed. In the body of research on modeling of CPU power consumption using PMCs, the dominant trend is to use linear regression. A linear framework is useful for understanding the relations between and importance of the independent variables used, but it is very rigid and requires tailoring to the specific problem.

This article presents an analysis of the power prediction behavior of a linear regression model from various perspectives and compares it to a neural network model. The purpose of this is to discover the underlying patterns of these models on the specific task of estimating and predicting power through the PMU hardware. This knowledge can serve as a guide for researchers for further investigation of power models.

The paper is organized as follows: Section II discusses related work. Section III details the research methodology, the experimental setup, the experiments, and the results obtained. Section IV includes result evaluation and Section IV presents conclusions and proposals for future research.

II. RELATED WORK

The use of PMCs for energy-aware applications has been heavily researched over the past few years. One of the first links between PMCs and power consumption has been pointed to by Bellosa [2]. In his paper, Bellosa has demonstrated the high correlation of PMCs to power consumption, and has presented strategies for energy-aware scheduling.

Contreras et el. have used PMCs in a linear model for the prediction of CPU (and memory) power consumption [3]. Singh et al. demonstrated the use of PMCs for power-aware thread scheduling [4]. Nagasaka et al. have been able to achieve good results by using PMCs to estimate GPU energy consumption [5]. Stockman et al. has examined machine learning techniques for the prediction of memory power usage [6]. Rodrigues et al. have shown that a power estimation model trained on one CPU can be used with reasonable accuracy on other CPUs with similar micro- architectures [7]. Lee et al. have investigated the use of PMCs for estimating micro-architectural components temperature [8]. Cavazos et al. have presented work on using PMCs to determine the best compiler optimization settings [9].

While these papers demonstrate the importance of PMCs for CPU performance evaluation, the use of PMCs for power estimation and prediction is concentrated on regression models. Our literature survey did not identify reference to the usage of neural networks for this process.

 ISBN: 978-1-61208-432-9

III. METHODOLOGY

This section includes details concerning the setup, data collection, and models used in the reported experiments. This information is useful as a point of reference concerning the machines and models in the experimental section as well as for implementing similar experiments on additional machines.

A. Selecting PMC Events

In the initial stages of the project, twelve PMCs of an Intel-based system have been considered for use in the models. For the final models, the five with the highest correlation value and the resource stalls counter have been chosen. Similar PMCs to those six have been located for the AMD machine and for the Power PC architecture.

B. Generating Training Data

To generate the training data for the learning models we have utilized a workload generation script that creates a large number of parallel workloads with diverse characteristics. The script selects a subset of the *PARSEC* programs and executes them by varying execution parameters such as the number of threads and data set size [1][3][7]. During each run of a workload, the PMC events listed in Table I are probed at a fixed interval. The power consumption values are recorded using the available power sensors on the *Sandybridge* architecture [1][3][7]. We have used an interval of 10 seconds.

The *Watts-Up-Pro* power meter is used for collecting power samples from the AMD machine since that machine does not have power sensors [7]. Initially, unnecessary background programs have been disabled, the workload script has been initiated, and peripherals have been disconnected. The workload has generated PMC event samples with an added time stamp component to assist in the synchronization with the power samples. Power samples have been collected every 2 seconds, and the PMC samples have been collected every 8 seconds (16 total as a sample is collected in two steps).

After the workload script has completed, the power samples have been collected from the memory of the PMU. To create the final dataset, the four power samples leading up to the elapsed time per PMC half-sample are found. Then, the eight power samples per PMC sample are averaged and used in the final dataset. This method has proved as highly reliable. The workload power patterns have been found to be well defined and prediction on the set has been very accurate.

C. Linear Regression Model

In these experiments, a straightforward multivariate linear regression model is used. The model is expresses via the following equation:

$$Y_{pwr} = a1 \times p1 + a2 \times p2 + \ldots + a6 \times p6 \quad (1)$$

Where p_i are the PMC counter values, a_i are the coefficients to be trained/identified, and Y_{pwr} is the power value. The model starts with $Y_{pwr} = a1 \times p1$ and a term, $a_i \times p_i$ is added at each step. The *lm* training function from R is used.

D. Neural Network Model

In contrast to the linear model, a neural network is more flexible, but more difficult to train. Additionally, because the weights are randomly initialized and the training only finds local minima, there is a problem with the results variance. There are several types of neural networks; for the reported experiments, a multi-layer, feed-forward structure utilizing the *neuralnet* implementation from R has been used. The default resilient back-propagation with weight backtracking procedures are used for training the model [10]

A neural network model consists of an input layer, one or more hidden layers, and an output layer. Our neural network model has one node in the input layer for each PMC event, and one node for the power value in the output layer. There are two hidden layers, one layer with 5 nodes and a second layer with 3 nodes. Originally, the model had only one layer, however the performance has been deficient. The addition of an extra layer has achieved satisfactory accuracy. The final values, which have yielded the local minimum, have provided significant improvement over the linear regression model.

Every node in a layer of the neural network model is mapped to every other node in the next layer. A mapping from one node to another can be represented by a weighted edge and finding these weights is the goal of training the network. The values of each node are calculated using: the values from the previous layer, the weights, and a squashing function. As an example, let v_{jm} be the j^{th} node in layer m, and let v_{in} be the i^{th} node in the previous layer n, let $w_{i,j}$be the weight from node i to node j, and let $sqsh$ be the sigmoid function. Then,

$$v_{jm} = sqsh\left(\sum_{i=1}^{n}(v_{in} \times w_{i,j})\right) \quad (2)$$

The hidden and output layers have an additional bias node whose value is always 1.

IV. EVALUATION

This section presents and discusses the results of various experiments. Each experiment examines a particular aspect of power prediction using the data, models, and setup described in the previous section.

A. Platforms

Table II provides details on the three systems that served as the evaluation platforms. In the rest of the paper, we refer to these platforms using the names listed in the header row. We have included both AMD and Intel-based systems in our experiments as the PMC units differ significantly.

B. Effect of Sample Size

The first experiment has explored the effect that the number of power samples has on the amount of prediction error. The purpose of this experiment is to determine the benefit of using more power samples for training. Figure 1 provides the results of this experiment.

At the beginning of the procedure, the samples of the dataset are normalized and shuffled. We have chosen to run the experiment for 24 trials of increasing sample size.

 ISBN: 978-1-61208-432-9

TABLE I. PERFORMANCE COUNTER EVENT NAMES PER MACHINE

Phenom	*Sandybridge*
CPU_CLOCKS_UNHALTED	UNHALTED_CLK_CYCLES
INSTRUCTIONS_RETIRED	INSTR_RETIRED_ANY
UOPS_RETIRED	UOPS_RETIRED_ALL
DISPATCHED_FAST_FPU	FP_COMP_OPS_EXE_X87
BRANCH_MISPREDICT_RETIRED	BR_MISP_RETIRED_ALL_BRANCHES
DISPATCH_STALLS	RESOURCE_STALLS_ANY

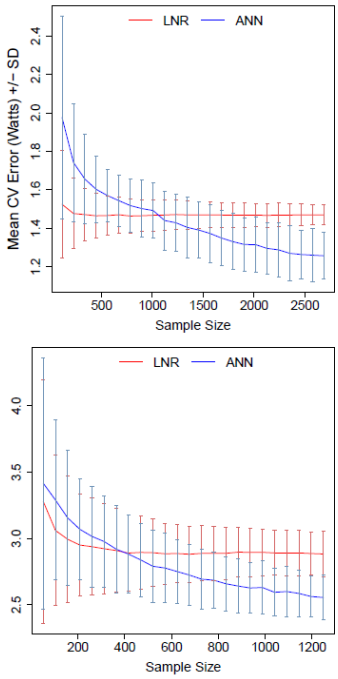

Figure 1. Sample Size Result on *Phenom* (l) and *Sandybridge* (r)

The next step is used to determine the number of samples Needed for increasing the dataset between each trial. After this, the dataset of each trial is split into 6 disjoint partitions for cross-validation (6-fold). Multiple runs of the procedure produce graphs with variability that makes it difficult to discern a clear pattern. To emerge the underlying pattern, the procedure has been repeated 50 times and the values are averaged. Both of the graphs for this experiment used the same procedure. One result used samples of the *Phenom* dataset [7], and the other used samples of the *Sandybridge* dataset.

The experiment results are interesting in several ways. First, the results are very similar despite the fact that each run used a dataset from different architectures. This is representative of the similar nature of the datasets, despite the fact they came from different architectures. Second, the linear regression model reaches its best performance for a small number of samples.

Third, the neural network eventually outperforms the linear regression model, but at different numbers of sample size. We speculate that the neural network overtakes the linear model at different points both because of the amount of noise in the *Sandybridge* data, and the inherent differences that are due to differing architectures.

The noisiness of the *Sandybridge* data has the larger effect as it prevents the linear regression model from converging to a lower error. Thus, the intersection point moves leftward. This assumes that the neural network can better handle noisy data.

In general, not much improvement in accuracy is seen between the models by the maximum amount of samples used. It is necessary, however, to have at least one thousand samples for obtaining reasonable variance. It is also expected that the neural network will continue to achieve slightly better accuracy for larger dataset sizes. A good strategy for quick power modeling is to collect one thousand samples and use a linear regression model. If accuracy the observed accuracy is not sufficient, a neural network model with a few thousand samples is recommended. We expect future collected datasets of the same type as used in this paper to behave similarly.

C. *Effect of the Number of PMCs*

The next experiment has examined the way that the performance of a power model behaves as more PMCs counters are included. Figure 2, depicts the experiment results. At the start of the procedure, the samples of the dataset are normalized and shuffled. The dataset at each trial is then split into 6 disjoint partitions for cross-validation. This procedure is conducted on each dataset to obtain the respective results.

 ISBN: 978-1-61208-432-9

TABLE II. EVALUATION PLATFORMS

	Core	*Sandybridge*	*Phenom*
Cores	4	6 (12 logical with HT)	4
Processor	2.40 GHz Intel Core 2 Quad	2.0 GHz Intel Xeon	1.00 GHz AMD Phenom
L1	32 KB (private)	32 KB (private per physical core)	64 KB (private)
L2	2 × 4 MB (shared 2 cores)	256 (private per physical core)	512 KB (private)
L3	none	15 MB (shared all)	2 MB (shared)
Compiler	GCC 4.8.2 -O2	GCC 4.8.2 -O2	GCC 4.8.2 -O2
OS	Ubuntu 14.04.1	Ubuntu 14.04.1	Ubuntu 14.04.1
Kernel	3.13	3.13	3.13

TABLE III. PMC EVENT NAMES IN ORDER OF ADDITION FOR EXPERIMENT 4.2

	Phenom	*Sandybridge*
1	INSTRUCTIONS_RETIRED	UNHALTED_CLK_CYCLES
2	UOPS_RETIRED –	INSTR_RETIRED_ANY –
3	BRANCH_MISPREDICT_RETIRED	UOPS_RETIRED_ALL
4	CPU_CLOCKS_UNHALTED	FP_COMP_OPS_EXE_X87
5	DISPATCHED FAST FPU	BR MISP RETIRED ALL BRANCHES
6	DISPATCH STALLS	RESOURCE STALLS ANY

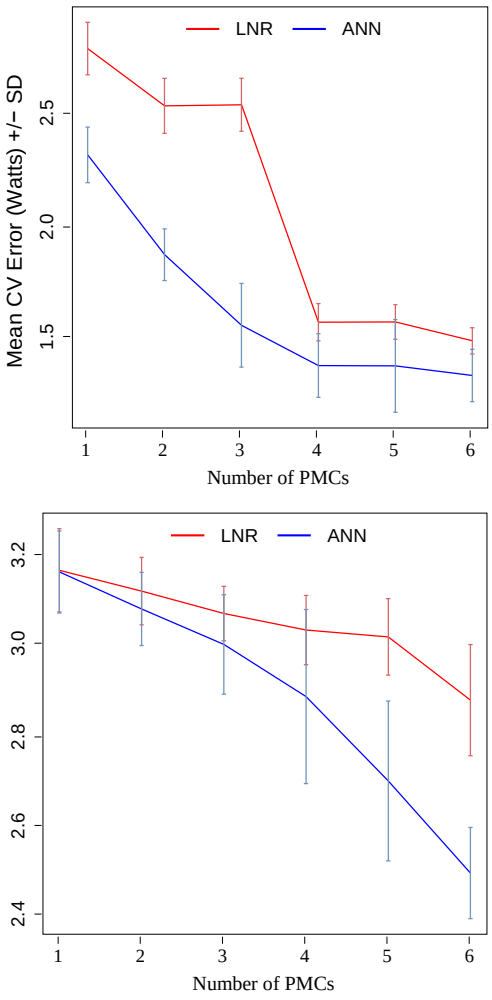

Figure 2. Number of PMCs Result on *Phenom* (l) and *Sandybridge* (r)

The PMCs have been added to the feature vector in order of decreasing Spearman correlation value. This method can be used with any set of PMC events. Table III provides the names of the PMCs in the order of insertion. The results from this experiment did not exhibit similar patterns.

For the *Sandybridge* dataset, both models have performed very similarly with one feature. Each subsequent feature improved performance for each model, showing stronger improvement for the neural network model. For the *Phenom* dataset, the neural network outperformed the linear regression model up until the fourth feature, after which the difference in performance became smaller. A few phenomena have been observed from these results. First, having more features improves accuracy. Second, a neural network model slightly outperforms a linear regression model with the same feature set.

Most machines have built-in PMCs, but only a few performance counters can be sampled at a time. From these experiment, it is easy to see that better results are obtained when more features are sampled. Furthermore, four metrics is a sufficient number of features to sample at a time. PMCs can be collected in alternating sets. This, however, slows down the rate of update in a real-time system by a multiple of the amount of sets. Additionally, there are problems if the PMCs are sampled over long periods of time.

D. Performance on New Workloads

To measure performance on new workloads, rather than unseen samples, the following experiment is performed. With this experiment, the error of the models on unseen workloads is observed. This may give an account the type of workloads that are not well predicted by our set of PMC events. The experiment results are depicted in Figure 3.

In this experiment, both the *Phenom* and *Sandybridge* datasets have been used. Initially, each dataset has consisted of samples collected from 66 different workloads. Both datasets shared a very similar distribution of samples for the respective workloads. The first step in preparing the datasets for the experiment is excluding very small workloads. The next step is trimming down all the remaining datasets (54 data sets) to the size of the sample with the minimum number of samples. This eliminates the likelihood that a large sample will cause a training. Next, for each reaming workload, the models are trained on samples from all other workloads and the prediction error is determined.

 ISBN: 978-1-61208-432-9

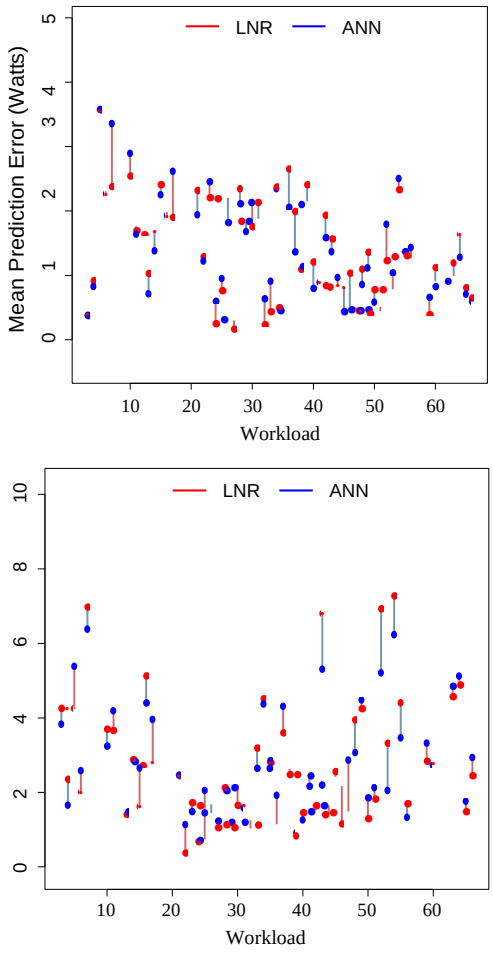

Figure 3. Feature Selection Result on *Phenom* (l) and *Sandybridge* (r)

The procedure is executed 50 times and averaged for the final result. This method is used to produce the resulting graph for the *Phenom* and *Sandybridge* datasets (Figure 3).

The graphs of Figure 3 are structured in a way to display the error per workload as a set of distinct trials, and to accentuate the relative error between the linear regression and neural network. Each point plots the prediction error on a workload after being trained on all the other workloads. There are two points per workload trial, a blue one for the neural network result and a red one for the linear regression result. A line connects a pair and has the color of the best performing model.

The models have very similar performance on *Phenom* 35.2% of the time (less than 0.15 Watt difference). They demonstrate similar performance on *Sandybridge* 27.8% of the time (less than 0.20 Watt difference). With the *Phenom* data, the neural network outperforms the linear regression model 59.3% of the time. With *Sandybridge*, the linear regression model won out 55.6% of the time. On average, *Phenom* had error of 1.42 +/- 0.68 Watts for the linear regression model and 1.40 +/- 0.68 Watts for the neural network model. On *Sandybridge*, the average error is 2.75 +/- 1.60 Watts for the linear regression model and 2.78 +/- 2.37 Watts for the neural network model.

A. *Cross-Architecture Power Prediction*

The final experiment performed discovers whether patterns in power prediction are reasonably similar between different architectures. If this is the case, it would be reasonable to train a predictive model on one architecture and use it for prediction on other machines. The experiment results are included in figure 4.

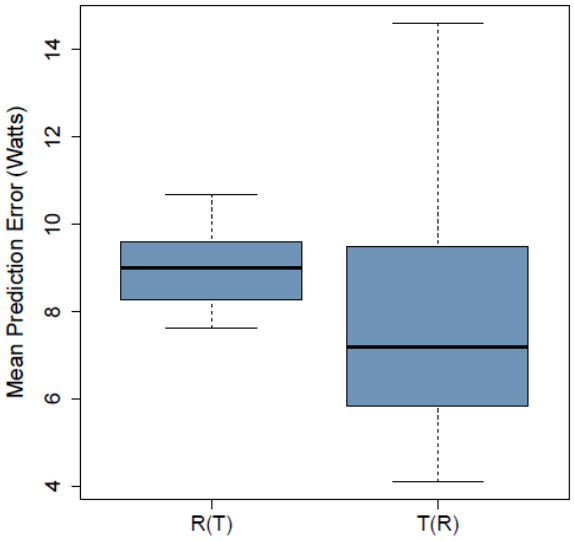

Figure 4. Cross-Architecture Result for Neural Network Model

The procedure for this experiment starts with the normalization of datasets to the range of [0, 1]. After this step, the models are trained on the data from one machine and used for prediction on the other machine. To settle the intrinsic variance in the neural network model, we have averaged the results over 50 experiments.

The linear regression model is able to achieve a prediction error of 8.13 +/- 5.42 Watts when trained on the *Sandybridge* data and used for prediction on *Phenom*. The error is 9.28 +/- 5.32 Watts when trained on *Phenom* and used for prediction on *Sandybridge*. These are moderately acceptable results. The box objects in Figure 4 show the performance of the neural network when trained on *Phenom* and used for prediction on *Sandybridge* ($R(T)$ and $T(R)$). The $R(T)$ performance is about the same as the linear regression model achieved. The $T(R)$ performance had a relatively low median error, but high variance.

In conclusion, the result of this experiment is quite surprising. We did not expect the models to perform as well as they did. However, the performance of prediction between machines with vastly different architectures is still quite poor and not a recommended alternative to the collection of training data from the target machine itself. On the other hand, it is very these results signify that prediction between similar machines could be performed much more accurately. This is in line with the cross-architecture prediction results of [7].

 ISBN: 978-1-61208-432-9

V. CONCLUSION AND FURTHER RESEARCH

Several aspects of power prediction using PMCs are examined in this paper. These are: the effect of the number of power samples used, the effect of the number of performance counters used, the predictive accuracy on unseen workloads, and the prediction accuracy using training data from machines with different architectures.

In the sample size experiment, it is concluded that more samples improved the performance of neural network model consistently. On the other hand, the linear regression model settled onto its best accuracy after only a relatively small number of samples. The immediate conclusion is that using a linear regression model is probably fine for most applications and in cases where only a small number of samples are available. Otherwise, if high accuracy is desired, it would be better to use a neural network with a high number of samples.

The next experiment has examined the way that an increasing number of performance counters affects prediction accuracy. It has been concluded that using more performance counters (starting with the highest correlated counter) generally leads to better accuracy. Four counters has provided sufficient accuracy for both machines tested. The experiment for prediction ability on new workloads showed no obvious patterns between the two machines. This result may indicate that the prediction error of this type of workload, cannot be accurately determined given a set of PMCs. This could be, however, the cause of a difference between implementation of the PMCs on the different machines.

The final experiment has examined the prediction performance of training the models on one machine and using them for prediction on the other machine. The result is not "bad" given that the two machines used are from very different architectures. This shows that there exist common patterns in the collected PMC data between machines. While this would not be useful for very different machines, it could be useful when considering machines with similar architectures. A power-aware scheduling program built on one machine can work well on a similar machine without having to collect any new samples.

In tandem, these experiments highlight useful behavior of power-PMC modeling concentrating on the use of neural networks for this purpose.

In the future we plan to explore additional neural networks and perform additional experiments for PMU counter selection. Additionally, the power estimation and power prediction models will be embedded in a meta-scheduler developed by the research team.

VI. ACKNOWLEDGMENTS

Financial support for this work is provided by the Semiconductor Research Consortium (SRC) under contract no. 2011-HJ-2156 and the National Science Foundation through awards nos. CNS-1305302 and CNS-1253292.

REFERENCES

[1] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in the IEEE/ACM International Symposium on Microarchitecture, pp. 469-480, 2009.

[2] F. Bellosa, "The benefits of event: Driven energy accounting in power-sensitive systems," in Proceedings of the 9th Workshop on ACM SIGOPS European Workshop, pp. 37–42, 2000.

[3] G. Contreras and M. Martonosi, "Power prediction for intel xscale reg; processors using performance monitoring unit events," in the Proceedings of the International Symposium on Low Power Electronics and Design, pp. 221–226, 2005.

[4] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," SIGARCH Computer. Architecture News, 37(2), pp. 46–55, 2009.

[5] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in the proceedings of the International Green Computing Conference, pp. 115–122, 2010.

[6] M. Stockman, M. Awad, R. Khanna, C. Le, H. David, E. Gorbatov, and U. Hanebutte, "A novel approach to memory power estimation using machine learning," in the proceedings of the International Conference on Energy Aware Computing, pp. 1–3, 2010.

[7] R. Rodrigues, A. Animalia, I. Koren, and S. Kundu, "A study on the use of performance counters to estimate power in microprocessors," IEEE Transactions on Circuits and Systems II: Express Briefs, 60(12), pp. 882–886, 2013.

[8] K.-J. Lee and K. Skadron, "Using performance counters for runtime temperature sensing in high-performance processors," in the Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 8, 2005.

[9] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. O'Boyle, and O. Temam, "Rapidly selecting good compiler optimizations using performance counters," in the proceedings of the International Symposium on Code Generation and Optimization, pp. 185–197, 2007.

[10] F. Gunther and S. Fritsch, "neuralnet: Training of neural networks," The R Journal, 2(1), pp. 30-38, 2010.

 ISBN: 978-1-61208-432-9