Designing Application Analysis Tools for Cross-Device Energy Consumption Estimation

Charalampos Marantos*, Nikolaos Maidonis‡, Dimitrios Soudris§
School of Electrical and Computer Engineering, National Technical University of Athens, Greece
*hmarantos@microlab.ntua.gr,‡nikos.maidonis.96@gmail.com,§dsoudris@microlab.ntua.gr

***Abstract*—Designing green and sustainable IoT applications makes energy consumption a key optimization goal of software development. Modern low-energy devices should be driven by energy-aware software. A promising solution to assist developers in this direction is provided by energy estimation tools. In this article, a method of designing flexible energy estimators is proposed. The introduced solution calculates the expected consumption of programs running on different devices and architectures by using synthetic datasets, the popular Valgrind and Pin profiling tools and the well-established Lasso regressor. In contrast to relevant studies, the emphasis is not on the construction of the most accurate tool, but on the characterization of the correlation between the various metrics (features) and energy consumption, on the comparison between predicting methods and on the construction of practical and easy-to-develop tools. The proposed approach is evaluated using the Polybench benchmark suite in widely used ARM-based systems, achieving an R2 score of 0.96, which is comparable to state-of-the-art approaches.**

***Index Terms*—Energy Consumption, Estimation, Correlation, Dynamic Instrumentation**

I. Introduction

Modern Internet of Things (IoT) smart systems and applications include a large number of connected deep edge devices installed in various environments (eg. industrial, sanitary, residential buildings etc). The ever-increasing attention to these applications pose new challenges to software developers as they target devices where energy is a critical design constraint, with great social and economic impact. As a result, bringing energy efficient edge applications development closer to the software engineering perspective by designing tools for analyzing source code applications, estimating and monitoring potential energy consumption during the Software Development Life Cycle is now an active research topic [1].

In this direction, this manuscript introduces a methodology for designing cross-device energy consumption estimation tools. It is considered very important to state the specific problem that the presented methodology aims to solve: The energy estimation tool runs on the programmer's workstation (PC), as part of the SDK tool used by the software developer. When the final application or part of it is ready, the developer uses the energy estimation tool to get an idea of the energy consumption that the application's code will consume if executed on a list of embedded devices (cross-device). Although measuring energy directly on the targeted edge device would give the most accurate results, not all hardware alternatives are accessible to developers and such a process may involve sophisticated equipment (eg special sensors) or expertise. These extra efforts would not only increase development time and cost, but may also not be feasible in very complex applications that involve a large number of different devices.

This work was funded by the EU's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

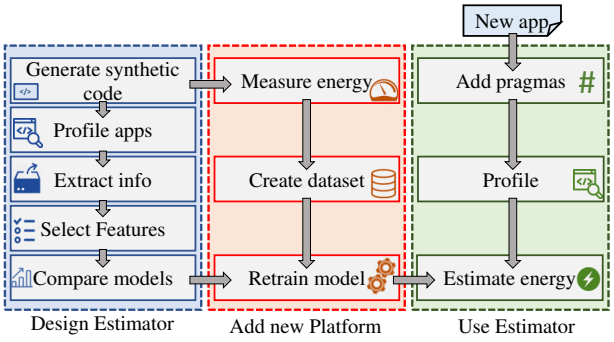


Fig. 1. Overview of the proposed framework

A large variety of approaches for solving this problem has been introduced in the literature. Some approaches are based on measurement based techniques that match assembly instructions directly with energy consumption for specific microarchitectures [2], while other techniques extend Worst Case execution Time (WCT) tools [3]. Machine learning techiniques use information retrieved from dynamic profiling [4] or static code analysis [5]. The present work is focused on the characterization of the correlation between the various metrics and energy consumption of devices, on the comparison between different estimation models and proposes a general methodology for building extensible and easy-to-develop tools.

II. Proposed Methodology

The introduced methodology is shown schematically in Figure 1 divided into 3 parts: The first one (blue) describes the central idea of designing energy estimators, while the second (red) shows the proposed process for adding new target devices to the introduced tool. The green box is the ready to use tool that analyzes and estimates the energy consumption of new applications. More details are provided in the next paragraphs.

1) Design Estimator: The first component is responsible for generating synthetic code: the data-set used in the introduced tool. More specifically, a set of Python scripts creates C-language for-loops that include operations between random-sized matrices. The maximum number of integer and floating point arrays, as well as the maximum dimension size can be

configured by the user. In this work, we used up to 4 integer tables and 4 floating point tables that have a maximum of 6 dimensions. To indicate the loop of interest, the "#pragma scop" and "#pragma endscop" directives are placed. A representative example of a generated loop is given in Listing 1.

Listing 1. Generated Loop example

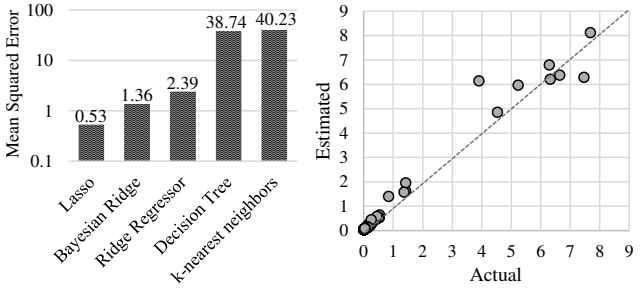
```
#pragma scop
for(i=0; i<18; i++){
    for(j=0; j<250; j++){
        for(k=0; k<167; k++){
            A0[i][j][k] = A1[i][j][k]*A1[i][j][k]/A1[i][j][k];
            B0[i][j][k] = B1[i][j][k]-B1[i][j]/B1[i][j]*B1[i][j];
        }
    }
}
#pragma endscop
```

The next stage (Profile apps) uses the popular Valgrind and Pin dynamic instrumentation tools to profile applications on the host machine (developer's PC) and generate the information to be used by the estimation models. Valgrind's Cachegrind tool simulates the cache architecture and gathers information about the *Instruction/Data cache reads, writes, misses* and monitors *Branching* behavior. Valgrind's Massif tool measures *Heap and Stack Memory usage*. Pin offers flexibility in designing specific monitoring metrics through custom C++ programs. Custom designed measurements were used to capture *branching divergence* information, as well as the number of *single-point and double-point operations* and the *types of arithmetic operations*. Pin metrics designed by MICA (https://github.com/boegel/MICA) were also used, including *Instruction-Level Parallelism, Instruction Types* (*memory reads, memory writes, control flow, arithmetic operations, etc.), Instruction and Data memory footprint, Memory reuse distances, conditional Branching predictability,* and *Memory stride* (distances between subsequent memory accesses).

The information produced by the previous step is then analyzed in order to extract information for feeding the energy estimation models. This step is very important as it identifies the energy related metrics. For this purpose, correlation and cross-validation techniques are used to produce the final features. These features are forwarded to the next component, which is responsible for comparing and selecting the models that estimate energy more accurately. These are key components of the proposed methodology. A detailed analysis of these aspects is presented in Section III.

2) Add new Platform: One of the main goals of the designed estimators is to be extensible: New devices (hardware platforms) can be added by the user easily following three steps:

- Run the synthetic codes on the new device in combination with a call to the energy metering script. The user adds commands to get energy information from external monitors, specific paths in the device tree or, if no sensors are available, user-defined metrics (eg. power-delay product).
- The energy measurements received from the previous step are processed and form an extra dataset file (csv).
- The estimation model is retrained using the new data as energy values (y values), maintaining the same feature values (x values) retrieved from the already profiled applications.



(a) Alternative models comparison (b) Lasso accuracy on test-set

Fig. 2. Choosing the best model to fit the synthetic data-set

A presentation of the extension to support an additional platform based on these steps is provided in Section IV.

3) Use Estimator: The final product of the proposed method for designing estimation tools (green box in Figure 1) analyses new applications and estimates energy. First, the user has to place the annotations to indicate the code block under interest, as shown in Listing 1 (*pragmas*). The next step runs the profiling tools and gathers the selected energy related features. Finally, the produced results are forwarded to the model that is responsible for estimating the potential energy consumption for running the application on the device imported by following the aforementioned process of adding a new platform.

III. Energy Consumption correlation analysis and Model Comparison

The profilling stage produces more than 100 metrics related to CPU, memory behavior, operations types, branches etc. For choosing the best model, the generated dataset is splitted into training and test sets and cross-validation techniques are employed. Figure 2a presents the results of the best five models. We might observe that the Lasso regressor is superior to the competing models. This is due to its inherent feature selection capability, which gives a competitive advantage because we have a large number of features, where many express similar or proportional quantities. More specifically, the Mean Squared Error is 75 times smaller compared to using of the k-nearest neighbors regressor and 2.5 times compared to the results of using the Bayesian Ridge regressor. Figure 2b shows the results of estimating the energy using the Lasso model on a selected testset (1/5 of the generated dataset). According to these results, the R^2 score between actual and estimated values exceeds 0.98.

A key concept of the present work is to study the correlation between alternative metrics *collected in the host machine* (developer's workstation) and energy consumption of the applications *measured on the targeted embedded devices*. For this purpose, we first employed the Spearman's widely-used correlation method, which evaluates the monotonic relationship between two continuous variables: A coefficient is assigned to each feature, which varies from -1 to +1. -1 or +1 indicates an exact monotonic relationship. Positive correlations mean that as the feature value increases so does the energy, while negative correlations mean that the increase of the feature indicates decrease in energy. Zero implies no correlation. The most im-

TABLE I
FEATURES WITH LARGEST SPEARMAN CORRELATION

Feature	Spearman Correlation
Data L1 miss rate	0.97511
# Instruction Reads	0.97905
# Data reads	0.97897
# Data writes	0.97712
# Data Level 1 cache write misses	0.97581
# Data Last Level cache write misses	0.97734
# Instructions	0.97900

TABLE II
MOST IMPORTANT FEATURES ACCORDING LASSO MODEL

Feature	Lasso importance
Instruction level parallelism	2.48196
# Data writes	0.68814
# Memory blocks and pages	0.43879
Heap memory size	0.37906
# Arithmetic operations	0.32430
Memory stride	0.24014
# Conditional branches	0.18763
# Branch prediction misses	0.18500
Data Last Level cache miss rate	0.11928
Stack usage	0.09939
Data Level 1 cache miss rate	0.05392
# Memory reads	0.01440

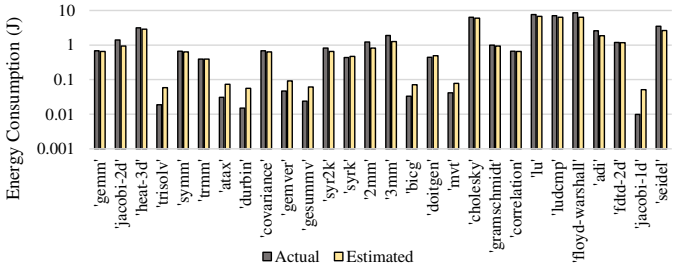


Fig. 3. Estimating Polybench apps energy on Jetson TX1 (ARM Cortex A57)

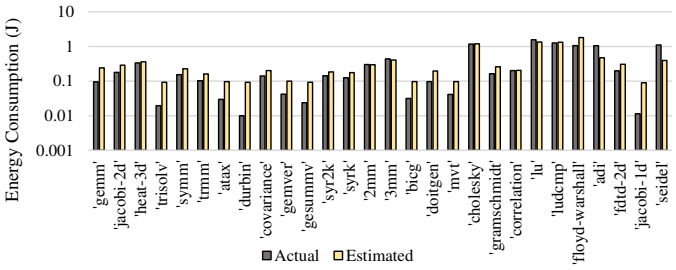


Fig. 4. Estimating Polybench apps energy on Jetson Xavier NX (ARM v8.2)

portant characteristics according to this analysis are presented in Table I alongside with the values of their coefficients.

Table II gives an alternative correlation analysis based on the weights calculated by Lasso. Lasso Regressor shrinks the weights of non-used features to zero, while it assignes a non-zero value to the rest of the features. Larger weights are given to the most important features, presented on Table II.

Based on this analysis, the following conclusions are drawn:

- *Instruction Level Parallelism* indicates how many instructions can be processed in parallel affecting the performance and respectively the energy consumption.
- Each *memory access* (especially *writes*) imposes a cost in terms of energy consumption. The amount of cost is affected by various parameters, such as the layer of the memory hierarchy in which the access occurs. When a *cache miss* occurs the CPU fetches requested data from the main memory, imposing an energy consumption overhead. The cache miss rate describes how effectively the application uses cache memories. The memory reuse distances (*stride*) heavily affect the potential cache performance.
- Modern CPUs use *branch prediction* mechanisms to guess the branch that will be selected and fetch the corresponding instructions in the CPU pipeline. When branch prediction fails, the CPU pipeline is flushed, which has a negative impact in application's energy consumption.
- Arithmetic operations are managed by the ALU and often consume a lot of energy (especially division). Thus, the number of operations can be used as an energy indicator.

While one could argue that the conclusions seem obvious, the profiling tools give us a list of more than 100 features related to memory, CPU behavior, and possibly energy consumption. The complete list of the examined features is not presented in this manuscript due to lack of space. According to the correlation results, instruction miss rate, some types of operations, branch divergence rates, etc are not included. For example, shift operations seem to have similar overhead to additions. Similarly, multiplications, although they have a higher overhead as they involve multiple additions, are only included coupled with the rest of arithmetic operations in a single feature metric. The choice of fewer metrics serves our purpose of creating a methodology for designing simple estimators, as it reduces the time overhead costs of the profiling phase (making the integration in SDK tools easier) and produces a hierarchical analysis of the alternative features.

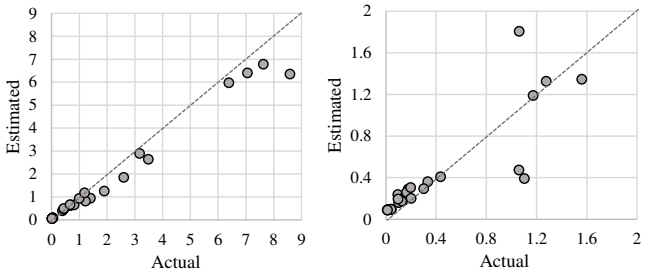
IV. ESTIMATION RESULTS

This Section evaluates the proposed methodology using the widely-used Polybench benchmark suite [6]. Using the models built on the synthetic data-set we first estimate the energy of the Polybench applications on Nvidia Jetson TX1 (using only the CPU part of the SoC which is an ARM Cortex A57) and then we evaluate the extensibility of the proposed method to support other devices without any change to the model parameters.

A. Experimental Results

Figure 3 presents the estimated and the actual energy of the 28 Polybench applications for the default input data sizes. Actual energy ranges from less than 0.01 to 10 Joules. Based on these results we might claim that the estimate can be considered very promising. More specifically, the Mean Absolute Error (MAE) is 0.17 Joules, while the estimations follow the real values very well, as their correlation according to the Spearman model is greater than 0.98 and the final R^2 score exceeds 0.96.

Following the procedure presented in Section II-2, we added an alternative device: The synthetic codes were executed on the NVIDIA Jetson Xavier NX board, which incorporates a SoC that contains a GPU (not used in the context of this manuscript)



(a) Jetson TX1 (ARM Cortex A57) (b) Jetson Xavier NX (ARM v8.2)

Fig. 5. Actual vs Estimated energy on two different devices

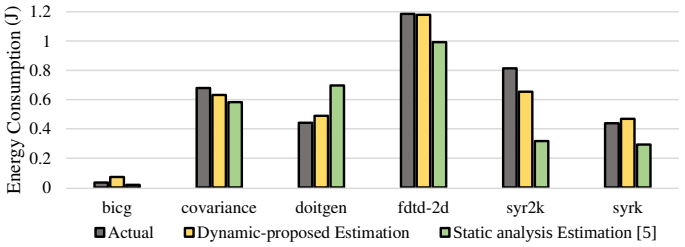


Fig. 6. Dynamic proposed approach against alternative static analysis approach

and a 6-core CPU ARM v8.2 with a 2-level cache and 8 GB 128-bit RAM. The model was retrained and the results for estimating the Polybench applications are presented in Figure 4. Based on these results, we can observe a high Spearman correlation between the predicted and the actual values (0.98). The MSE equals to 0.03 Joules. However, the R^2 score is reduced to 0.77 due to the fact that the model mispredicts some applications that consume less than 0.01 Joules of energy. However, we might conclude that the performance of adding a new platform can be considered acceptable as we did not make any calibration to the Lasso model's parameters.

The results of estimating the Polybench applications energy vs the actual values are also presented in Figure 5. As shown in this Figure, most of the points are very close to the ideal diagonal line, confirming the high level of prediction accuracy. We should mention that in the case of Xavier NX, for the applications that consume less energy than 0.2 Joules the predictions were mostly overestimating the actual energy. Additionally, there are three applications for which we have a relatively large error (more than 0.3 Joule). However, we might conclude that these misspredictions do not affect the overall quality of the presented methodology. The study of their characteristics will give a future direction for improvements, while a refinement of the model's parameters is expected to lead to better results. In the context of this work we focused on showing the results of using exactly the same model.

B. Comparison to related approaches

The relevant methods vary greatly in the mechanisms they use. Work in [2] employs a measurement-based method and achieves high accuracy ($R^2 > 0.99$). However, it targets only a specific microcontroller and uses a small data-set of 60 programs. The authors in [3] are inspired by the Worst Case execution Time (WCT) tools and design a energy estimating method based on iterative approaches. These tools are usually very slow and only support specific architectures. In addition, their accuracy is usually limited, but the experiments included in [3] achieve high a level of $R^2 = 0.95$. Our approach is inspired by projects like [4] that use dynamic instrumentation techniques. The work in [4] divides the application into phases and after collecting profiling results, it feeds estimation models achieving a final R^2 score of 0.97. We focus mainly on loops (the usual most energy-intensive phases) and have similar accuracy. Our difference is the fact that we provide correlation results and select a small subset of the features provided, to make simpler models and reduce the profiling phase time overhead, as well as our focus on designing a method for adding new targeted platforms easily. Finally, the comparison with a tool of similar purpose based on static code analysis approaches is very interesting for the completeness of this manuscript. The work in [5] proposes a simple and extensible energy estimation approach based on features retrieved from static analysis of the compiled application and more specifically from the assembly language. This approach is less accurate ($R^2 \approx 0.92$), while it requires additional user input that is not always easy to obtain (eg. the number of iterations of each loop body). Figure 6 presents the results for 6 representative applications from the Polybench suite. The average error is 0.14 Joules for the proposed approach, while the static method has an average error of 0.24 Joules.

V. Conclusions

A complete methodology for designing practical analysis tools to be used by developers for estimating energy consumption for running an application on different embedded devices was presented. The introduced framework uses random synthetic loops, popular profiling tools and regression methods. The proposed approach achieves similar effectiveness compared to related state-of-the-art tools but focuses on building an extensible solution that can be part of SDK tools. Particular emphasis was placed on studying the correlation between profiling results and energy as well as the tool's capabilities to add new targeted platforms in an easy and convenient way.

References

[1] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–33, 2019.

[2] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE transactions on instrumentation and measurement*, vol. 62, no. 7, pp. 1927–1934, 2013.

[3] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy consumption and execution time estimation of embedded system applications," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 426–440, 2011.

[4] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2016.

[5] C. Marantos, K. Salapas, L. Papadopoulos, and D. Soudris, "A flexible tool for estimating applications performance and energy consumption through static analysis," *SN Computer Science*, vol. 2, no. 1, pp. 1–11, 2021.

[6] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: http://www.cs.ucla.edu/pouchet/software/polybench*, 2012.