A Preliminary Study of the Energy Impact of Software in Raspberry Pi devices

Kamar Kesrouani*, Houssam Kanso†, and Adel Noureddine‡
Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA
Anglet, France
*kkesrouani@univ-pau.fr, †houssam.kanso@univ-pau.fr, ‡adel.noureddine@univ-pau.fr

***Abstract*—Nowadays, the use of IoT devices is essential in most sectors along with a rising concern regarding their energy consumption. Therefore, an accurate estimation of the energy consumption of such devices is required for energy-efficient improvements. This paper presents Energy Measurement Model (EMM), an energy estimation tool for Raspberry Pi 3 Model B+. It is a software-based model used to estimate the local energy consumption of a device by taking into consideration CPU utilization. The error rate of our model averages at 1.25%. Using this model, we study the energy impact of a set of algorithms, programming languages and compilers.**

***Index Terms*—Connected Objects, Raspberry Pi, IoT, Energy Consumption, Energy Efficiency, Software Engineering**

I. Introduction

Energy consumption of IoT devices is an important topic and a rising concern with the widespread using of IoT equipment. Billions of IoT devices are expected to produce up to 14% of global carbon emissions by 2040 [1]. Measuring and monitoring the energy consumption of such devices is, therefore, a crucial step in order to minimize their energy impact and expand their lifetimes.

Many energy measurement tools and models have been proposed for PCs, servers, and other devices and equipment. However, IoT devices lack a software-based approach that can accurately monitor the energy consumption of devices, and scale up to thousands of devices. For instance, wattmeters or other hardware monitoring components offer accurate energy reading, but with a cost of limited scalability and important hardware and financial investment. On the other hand, software-based models scale up easily but their accuracy varies greatly.

In this paper, we conduct an empirical experiment on multiple algorithms, programming languages and compilers on a Raspberry Pi device. We analyze and compare the collected energy consumption data and draw our observations and insights on software in ARM-based Linux systems. To do so, we present our multi-model energy estimation approach: Energy Measurement Model (EMM).

The remainder of this paper is organized as follows: in section II, we review the state of the art of energy estimation and measurement. We propose our estimation model including its architecture and its validation in section III. In section IV, we conduct empirical experiments on different algorithms, programming languages and compilers, and draw our observations. Finally, we conclude in section V.

II. State of the art

In [2], a power model called PowerPi that measures the power consumption of Raspberry Pi 2 Model B has been proposed. It estimates the power consumption of CPU and network usage only. The measurement is done using an external power meter and the model is generated by adjusting a linear function to the measured data and minimizing the RMSE (Remaining Root Mean Square Error). We tested PowerPi model on Raspberry Pi 3 Model B+ . The error rate of the estimations provided by PowerPi model compared to the measurements from a powermeter is too high. In idle mode, the error rate is on average around 60% and when stressing the device using the stress command at 100% utilization, the error rate is on average around 72%.

In [3], an external current sensor was used to measure the power consumption of the Raspberry Pi. The sensor is inserted on the power lines of the USB connection in the 5 V line. Their solution is hardware-based, making it difficult and costly to deploy in an environment with multiple Raspberry Pis.

In [4], a model was proposed that converts data about resource usage into energy or power consumption information for Single Board Computers (SBCs). The model was tested with several real applications on a Raspberry Pi 2 model B. The model made up of power consumption characterization and power consumption estimation. CPU and Ethernet usage were considered as variables. It has an average error of 2.2%.

In [5], the authors evaluated the energy efficiency of SBCs (including Raspberry Pis) over the last years, by measuring and controlling the system utilization of multi-core systems. To model the power consumption of SBCs, they recorded the system utilization and the power consumption of devices under test and then run a regression analysis on the collected data. To measure the power consumption of SBCs, they monitored the system utilization (simple calls to /proc file system) and converted these to power consumption using their proposed models. For all devices, the error rate was smaller than 10%.

In [6], EMPIOT a power measurement platform was proposed. It is composed of hardware and software components, and measures the energy consumption of IoT devices. The effect of diverse design parameters on accuracy and overhead was studied. The authors used five different IoT devices performing sleep, software encryption, and transmission tasks.

III. Model Generation

A. Model Architecture

Due to the high error rates found in the literature and incompatibility with new Raspberry Pis, we develop and generate our own power estimation model using empirical data and linear regression on a Raspberry Pi 3 model B+. It estimates the power consumption of the CPU (an ARM-based Cortex-A53 (ARMv8) 64-bit processor, running at 1.4 GHz). The device is running Raspberry Pi OS with kernel version 4.19.118∼v7+. Our model generates formulas by applying a linear regression function using the Weka tool [1]. We collect four metrics from the operating system for CPU cycles: user, nice, system and idle. These cycles are collected from the */proc/stat* folder, and we used them to calculate the CPU utilization using the following formula:

$$u[t] = \frac{c_{busy}[t] - c_{busy}[t-1]}{c_{total}[t] - c_{total}[t-1]} \quad (1)$$

where: $c_{total}[t]$ is the sum of $c_{busy}[t]$ and the number of idle cycles $c_{idle}[t]$. $c_{busy}[t]$ is the total number of busy cycles up to time t, it is defined by:

$$c_{busy}[t] = c_{user}[t] + c_{nice}[t] + c_{system}[t] \quad (2)$$

$c_{user}[t]$ is the number of user-generated CPU cycles, $c_{nice}[t]$ is the number of cycles created by low priority processes and $c_{system}[t]$ is the number of cycles created by the system.

In order to generate our estimation model, we stress the CPU in the range of 1% to 100% with a 1% step, using the Linux stress command. One sample was collected/measured each second. For each percentage, we did the following:

- Measure the power consumption P using a PowerSpy2 powermeter [2]. Powerspy2 is a hardware accurate measuring device used to measure a variety of metrics such as current, voltage, energy, power, and others. P is considered the true value of the power consumption of the device. It is used to train our model and validate it.
- Retrieve the CPU data (4 types of cycles and calculate the utilization rate u).

Using formula 1, we notice that multiplying u by 100, we obtain approximately the percentage of CPU utilization that we applied with the stress command. To generate the formula, the dataset containing u and P is given to Weka. The generated formula is the following:

$$Ppi = 2.1514 \times u + 4.142(W) \quad (3)$$

The Root-Mean-Square-Error (RMSE) of this formula is equal to 0.1471 and the normalized RMSE is equal to 2.81%.

Figure 1 shows a change in the slope approximately around the 50% CPU utilization. Using formula 3 only, we obtained higher error rate for the low percentages of CPU usage and around the 50% mark. Therefore, we generate two formulas using the same empirical approach and linear regression model: one trained with a dataset ranging from 1% to 50%, and the other with a dataset from 51% to 100% CPU utilization.

[1]https://www.cs.waikato.ac.nz/ml/weka/

[2]https://www.alciom.com/en/our-trades/products/powerspy2/

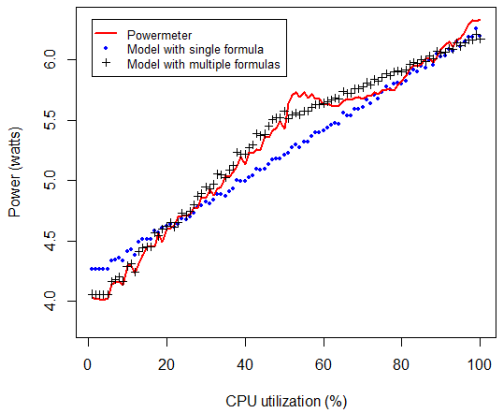

Fig. 1: Power consumption of the stress command measured by the powermeter, estimated by the single formula model (formula 3) and multi-formulas model (formulas 4 and 5)

The generated formula for the first 50% is the following:

$$Ppi = 3.4495 \times u + 3.8563(W) \quad (4)$$

The RMSE of this formula is equal to 0.0322 and the normalized RMSE is equal to 0.67%.

The generated formula for the second 50% is the following:

$$Ppi = 1.4584 \times u + 4.7788(W) \quad (5)$$

The RMSE of this formula is equal to 0.098 and the normalized RMSE is equal to 1.67%.

Since the error rate for each range of percentage is lower than the one obtained using formula 3, we build our power estimation tool using the formulas 4 and 5, and alternate between the formulas based on the CPU utilization. Our tool provides an estimation for power consumption per second.

B. Validation

To validate our model, we proceed as follows:

- Measure P (using PowerSpy2) obtained by stressing the CPU from 1% to 100%. u is also collected respecting the same time unit as P and synchronized with the same clock attributing a timestamp for each measurement.
- Calculate the power using formula 3 for the whole percentage range (1% to 100%), formula 4 for the first percentage range (1% to 50%) and formula 5 for the second percentage range (51% to 100%).
- Calculate and compare the error rate of the power obtained by the different formulas for each percentage.

As seen in Figure 2, our multi-formulas model has a lower error rate than the one using a single formula. These results confirm that the multi-formulas is more accurate, with a maximum error rate of 3%, and an average error rate of 1.25%.

In the next section, we use our multi-formulas model to study the impact of different algorithms, programming languages and compilers on a Raspberry Pi's energy consumption.

IV. Enegy Impact of Algorithms, Programming Languages and Compilers

In order to study the impact of software choices, such as algorithms, programming languages or compilers, on the

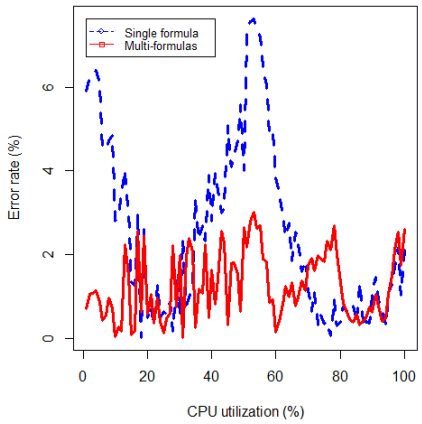

Fig. 2: Error rates of single and multi-formulas models using the stress command

energy consumption of software in Raspberry Pi devices, we conduct an empirical experimentation using various programs. The source codes were taken from the Rosetta Code website [3].

We compare the energy consumption of two sets of programs: the Fibonacci sequence, and Towers of Hanoi. We use the recursive and iterative version for each of these programs, implemented in the following programming languages: Java (openJDK 11.0.7), Python (2.7.16), C++, C and OCaml (version 4.05.0, only for Fibonacci). For the C language, we compiled the programs using GCC (version 8.3.0) and Clang (version 9.0.0). For GCC, we also apply the O2 and O3 optimizations flags during compilation. For OCaml, we use both the binary compiler (ocamlopt), and the byte-code compiler (ocamlc). For each execution, we run the program multiple times in a loop in order to collect sufficient power data, then we normalize the value for our comparison.

Fibonacci program calculates the number 30. In C and C++, it is executed 4000 times, and 200 times in Java and Python. In OCaml, it is executed 4000 times iteratively, and 200 times (ocamlc) and 400 times (ocamlopt) recursively. For Towers of Hanoi, we run the experiment 200 times with 10 towers for all languages, and for both iterative and recursive versions.

On average, the error rate for our experimental runs that last more than one minute is 2.5%. Smaller runs, which last for a few or dozens seconds, exhibit higher error rate at around 6.7% on average, with some experiments peaking to 10% or 28% (*e.g.*, for the Towers of Hanoi iterative version in C, the powermeter data averaged at 0.116 joules per execution, compared to 0.149 to our model. The entire 4000 runs of the C program lasted for only 5 seconds). This higher error rate is due to the granularity of both our model and the powermeter, which provided power data every second.

A. Results of the Fibonacci Sequence

From the experiments and the results in Figures 3 and 4, we draw the following observations:

- All tested programming languages and compilers have a higher energy consumption when running the recursive version in comparison to the iterative one.
- In the iterative version, Java consumes the most energy while C compiled with GCC or Clang compiler consumes the least. In the recursive version, the most energy is consumed by Python and the least energy is consumed by C compiled with GCC with O2 and O3 optimization.
- For the C compilers, in the iterative version, all compilers consume approximately the same amount of energy. We observe that the optimization flags in GCC did not improve the energy consumption. In the recursive version, the optimization flags O2 and O3 managed to provide a big improvement in the energy consumption compared to the default GCC or Clang settings.
- For OCaml compilers, in iterative and recursive versions, ocamlc compiler consumes more energy than ocamlopt.

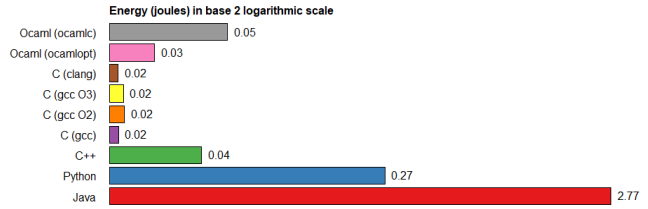

Fig. 3: Energy consumption of the iterative algorithm of the Fibonacci Sequence

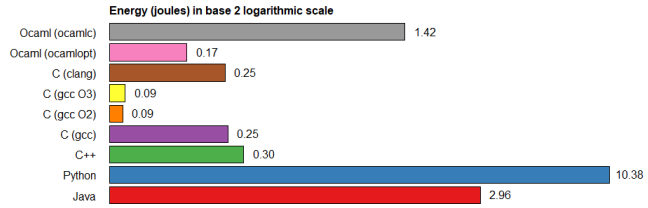

Fig. 4: Energy consumption of the recursive algorithm of the Fibonacci Sequence

B. Results of Towers of Hanoi

From the experiments and the results in Figures 5 and 6, we draw the following observations:

- Java and C++ consume more energy in the recursive version compared to the iterative version, while it is the opposite for the other programming languages.
- In the iterative version, the most energy is consumed by Java, while the least one is consumed by C with the GCC compiler. In the recursive version, Java also consumes the most energy while the least energy is consumed by C with the GCC compiler.
- For the C compilers, we observe that in the iterative version, the O2 and O3 optimization did not provide meaningful improvement in comparison to the default GCC settings. In the recursive version, the Clang compiler had on average a higher energy impact compared to GCC. The latter also did not provide meaningful improvements when using the O2 and O3 optimization.

C. Discussion

In our experiments on Fibonacci and Towers of Hanoi, Java was consistently the highest energy consuming programming

[3]https://rosettacode.org/wiki/Rosetta_Code

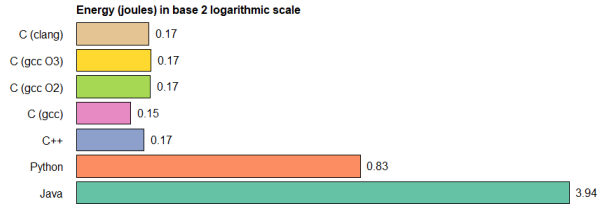

Fig. 5: Energy consumption of the iterative algorithm of the Towers of Hanoi program

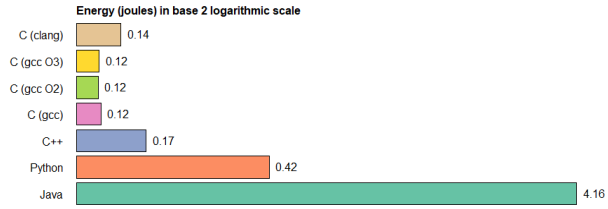

Fig. 6: Energy consumption of the recursive algorithm of the Towers of Hanoi program

language. However, we observe that it also had the least CPU utilization percentages, and took much more time to run the iterative and recursive version of the programs (with the only exception being the recursive version of Fibonacci, where Python consumed more energy). This might have two main explanations: 1) the impact of the Java Virtual Machine adding additional CPU cycles and thus consuming more energy, and/or 2) the state of the port of the JVM on Raspberry Pi's ARM-based architecture. For the first point, the energy consumption of OCaml's two compilers (binary and byte-code) provides insights on the impact of the virtual machine. Ocamlc, the byte-code compiler, constantly consumed more energy and took more time to execute compared to the binary compiler, ocamlopt. It also had, on average, a lower CPU utilization. For the second point, further investigations are needed to analyze whether the ARM port of openJDK is optimized compared to x86_64 devices.

Python was the second most consuming programming language, which can be explained by its interpreted nature, consuming CPU cycles and energy to compile and run the code on every iteration. In contrast, compiled languages consumed, by far, the least energy and took the least time to execute. Although the variation between them is low, we still observed that GCC was better optimized compared to Clang. The O2 and O3 optimization flags were only useful in the recursive versions, which confirms the results obtained on x86_64 computers in [7] (in which, the authors explained that the O2 and O3 flags activate dozens of optimization flags including the *Predictive Commoning optimization* that is used to eliminate redundancies across the iterations of a loop).

However, several limitations can be improved in our work. First, our model only takes into consideration the CPU. Therefore, it can be used to estimate the energy consumption for CPU-intensive workloads. Other components need to be taken into consideration in a future power model, such as the network adapter. Second, in our experiments we used a limited set of software, such as the stress command, the Fibonacci Sequence or the Towers of Hanoi program. More extensive empirical experiments are needed to better understand the impact of algorithms, programming languages or compilers on energy consumption. Finally, our model was tested on one model of Raspberr Pi (version 3 B+). Therefore, it may not be as accurate on older or newer Raspberry Pi models. However, we argue that our multi-formulas approach is an interesting aspect to build more generic power models for IoT devices.

V. Conclusion

In this paper, we presented a software-based and multi-formulas power estimation model. The formulas used to estimate the power consumption alternate according to the CPU utilization. We validated our model with an average error rate of 1.25% and a maximum one of 3% in stress tests, and an average of 2.5% for our software experiments. We then compared the energy impact of algorithms, programming languages and compilers across two sets of programs.

For future work, we intend to model the power consumption of other hardware components of Raspberry Pi devices, such as the network interface (both Ethernet and Wifi), and the RAM memory. Additional empirical experiments will be conducted to understand the impact of byte-code virtual machines, both for Java (openJDK) and OCaml (ocamlc). We also plan on providing a cloud-based interface and repository for power estimation models of multiple IoT devices, therefore allowing the community to share their metrics and train more accurate models for a wide variety of devices.

Acknowledgement

The project leading to this publication has received funding from Excellence Initiative of Université de Pau et des Pays de l'Adour - I-Site E2S UPPA, a French "Investissements d'Avenir" programme.

References

[1] "Tsunami of data" could consume one fifth of global electricity by 2025," Climate Home News, Dec. 11, 2017. https://www.climatechangenews.com/2017/12/11/tsunami-data-consume-one-fifth-global-electricity-2025/.

[2] F. Kaup, P. Gottschling, and D. Hausheer, "PowerPi: Measuring and modeling the power consumption of the Raspberry Pi", in 39th Annual IEEE Conference on Local Computer Networks, Edmonton, AB, Sep. 2014, pp. 236–243, doi: 10.1109/LCN.2014.6925777.

[3] F. Astudillo-Salinas, D. Barrera-Salamea, A. Vazquez-Rodas, and L. Solano-Quinde, "Minimizing the power consumption in Raspberry Pi to use as a remote WSN gateway", in 2016 8th IEEE Latin-American Conference on Communications (LATINCOM), Medellin, Colombia, Nov. 2016, pp. 1–5, doi: 10.1109/LATINCOM.2016.7811590.

[4] L. Ardito and M. Torchiano, "Creating and evaluating a software power model for linux single board computers", in Proceedings of the 6th International Workshop on Green and Sustainable Software - GREENS '18, Gothenburg, Sweden, 2018, pp. 1–8, doi: 10.1145/3194078.3194079.

[5] F. Kaup, S. Hacker, E. Mentzendorff, C. Meurisch, and D. Hausheer. The Progress of the Energy-Efficiency of Single-board Computers. Technical report, Otto-von-Guericke-University, Institute for Intelligent Cooperative Systems, 01 2018.

[6] B. Dezfouli, I. Amirtharaj, and C.-C. Li, "EMPIOT: An Energy Measurement Platform for Wireless IoT Devices," ArXiv180404794 Cs, Dec. 2018, Accessed: Apr. 10, 2020. [Online]. Available: http://arxiv.org/abs/1804.04794.

[7] A. Noureddine, A. Bourdon, R. Rouvoy and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT," 2012 First International Workshop on Green and Sustainable Software (GREENS), Zurich, 2012, pp. 21-27, doi: 10.1109/GREENS.2012.6224251.