Classifying C++ Solutions Based on Their Energy Profile

Marcelo Borges Nogueira
marcelo.nogueira@ufrn.br
Federal University of Rio Grande do Norte
Natal, RN, Brazil

Sérgio Queiroz de Medeiros
sergio.medeiros@ufrn.br
Federal University of Rio Grande do Norte
Natal, RN, Brazil

Abstract

The algorithm classification problem consists of finding which algorithm a program implements among a given finite set of algorithms. We propose a novel classification strategy using the power slope that best represents an algorithm energy consumption profile. Then we classify a set of solutions by using a greedy approach to match two sets of solutions that are aimed to solve the same problem. To evaluate our approach, we used sets of random C++ solutions for 32 problems from CSES, a programming competition site. We executed two datasets twice in two different machines. In one machine, we restricted the classification task to a subset that varies from 14 to 16 CSES problems (a reduction of at least 50% in the search space). In the other one, we could reach a reduction in the search space that varied from 38% up to 53%.

CCS Concepts: • **Software and its engineering → Compilers**; **Software performance**.

Keywords: program classification, energy profile, C++

1 Introduction

The correct understanding of the meaning of a program is crucial for programming language processing [14], although it is impossible to exactly determine it [19]. Usually, the algorithm classification problem consists of finding which algorithm a program implements among a given finite set of algorithms.

We propose a novel approach to classify sets of programs independent of source code. It relies only on the energy consumption profile of executable programs. We use Intel's Running Average Power Limit (RAPL) interface to measure the energy consumption of a set of problems and then calculate, via linear regression, the power slope that best represents its energy consumption profile.

In our research, we constructed a dataset of randomly picked C++ solutions from Code Submission Evaluation System (CSES), a popular programming competition site. Then we performed experiments in two different machines, where, we could restrict the classification task to a subset that varies from from 38% up to 53% depending on the machine.

The rest of this paper is organized as follows: next section explains our methodology, while Section 3 presents our results. Section 4 discusses related work, and Section 5 presents our conclusions.

Config.	ELITE	THINK
CPU	i5-7500 @ 3.40GHz	i5-2400 @ 3.10GHz
RAM	8G DDR4 @ 2400 MHz	4G DDR3 @ 1333 MHz
L3 cache	6 MB	6 MB
OS	Ubuntu 22.04.3 LTS	Ubuntu 22.04.2 LTS
g++	11.4.0	11.4.0

Table 1. Configuration of Machines Used in the Experiment.

2 Methodology

CSES [9] is a programming competition site with 300 problems. Each problem has a set of test cases that consists of an input and its corresponding output. A user solution for a problem is accepted only when it produces, for each test case, the expected output. Moreover, a solution for a CSES problem usually must give the correct answer for a test case in at most one second and must use at most 512 MB of memory.

From CSES, we selected 32 problems of different topics (e.g., sorting and graphs). For each problem, we chose 100 accepted C++ solutions at random and we picked the inputs that would demand more running time from a solution. Our study focuses on C++ as it is the most used language in CSES and in programming competitions in general. To compile them we used the g++ flags `std=c++17` and `-O2`.

We used the RAPL interface provided by Intel [20] to get the energy consumption of the C++ solutions. We adapted a framework already used in previous works [15, 16, 18] to also measure the CPU time of C++ solutions through the `time` command provided by the operating system.

We measured the energy consumption and execution time of each solution for a certain problem by executing it, for each corresponding selected input, 10 times. Then we dropped the lowest and highest energy consumption (to remove potential outliers), resulting in 8 measurements.

The same measurements were performed in two different desktop computers with Intel x86-64 processors, which we labeled as **ELITE** and **THINK**. In Table 1 we describe the configuration of each computer.

For each one of the 32 problems, we divided their 100 gathered solutions into two groups: a training group, with 70 solutions, and a control one, with 30 solutions. Then, we followed the same approach of our previous work [18] to compute the power slope for each group of solutions. In short,

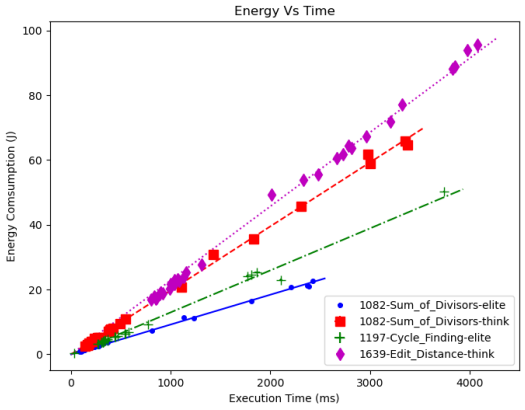

Figure 1. Maximum and Minimum Slopes for ELITE and THINK Considering the Control Dataset First Measurement.

	Control 1	Control 2	Training 1	Training 2
ELITE	40.8%	33.9%	39.2%	34.4%
THINK	15.7%	13.8%	15.0%	14.5%

Table 2. Normalized difference between the set of problems with the greatest power slope and the lowest one in each machine and for each measurement.

given the 8 measurements for each solution, we computed mean execution time, t, and mean energy consumption, c. With the mean values of all the solutions of a group (training or control) of a given problem, we fitted the function $\hat{c} = at$ using ordinary least squares [3].

For the classification task, given a set S of programs that solve a particular problem P, we tried to indicate, from a finite set of problems, a restrict subset which includes problem P. Notice that we have a uni-dimensional feature vector (the power slope) for each problem, and that to compute the power slope of a problem we need a fair number of solutions, such as 20. In our case, we will compute the slope of each problem belonging to the two groups mentioned previously: control and training. Hence, we did not use any machine learning methods, but just a simple greedy approach.

We will associate set S (from the control group) to the set S_1 (from the training group) with the nearest power slope. Let P_1 be the problem that S_1 solves, in case $P_1 \neq P$, we will check the set S_2, with the second nearest power slope, and see if P_2, the problem solved by S_2, is equal to P. We keep doing this until finding a set of solutions S_N that solves P. In case the subset formed by $\{P_1, P_2, ..., P_N\}$ is smaller than the original set of all problems, then we were able to reduce the search space. So, the smaller the N, the better the solution.

	C1 x T1	C1 x T2	C2 x T1	C2 x T2
100%	22 (16)	19 (14)	22 (16)	19 (14)
90%	9 (9)	12 (11)	11 (10)	12 (9)
80%	6 (7)	9 (8)	8 (8)	7 (8)
70%	6 (6)	6 (6)	6 (6)	6 (6)
60%	4 (4)	6 (5)	5 (5)	5 (5)
50%	2 (2)	4 (3)	3 (3)	4 (3)

Table 3. # of Trials to Classify the 32 Problems for ELITE.

	C1 x T1	C1 x T2	C2 x T1	C2 x T2
100%	19 (19)	15 (15)	20 (20)	17 (16)
90%	13 (13)	13 (12)	12 (11)	12 (12)
80%	11 (10)	9 (9)	11 (10)	9 (9)
70%	9 (9)	6 (6)	9 (10)	6 (6)
60%	6 (7)	5 (5)	6 (7)	4 (5)
50%	5 (5)	4 (4)	5 (6)	4 (4)

Table 4. # of Trials to Classify the 32 Problems for THINK.

3 Results and Discussion

In this section we will present and discuss our results. First, we will discuss the difference in the energy consumption profile of sets of programs aimed to solve different problems. The related artifacts are available online [2].

Consider the power slope variation among the different datasets. Figure 1 shows the minimum and maximum of such slopes for the first measurement of the control dataset in machines ELITE and THINK. Figure 1 suggests that computational solutions for different problems may have different energy consumption profiles, as there is a reasonable distance between the lines.

Table 2 shows the normalized difference between the maximum and minimum power slope for each machine in different measurements. According to it, it seems that there is a clear difference between the lowest and the greatest power slope. We can also see that this difference is more relevant in machine ELITE than in machine THINK. Although there is such difference, the energy consumption profile of a set of solutions to a given problem might not be unique.

Regarding the classification task, we tried to find which problem each one of the 32 control sets tried to solve based solely on its associated power slope. Table 3 shows our results when running the experiment at machine ELITE, while Table 4 encompasses the results for THINK. The values enclosed by parentheses should be ignored for now. We will explain them later.

As we performed the measurements twice for both datasets, we compared each measurement for the control dataset against each measurement for the training one. We used *C1* and *C2* to refer to the first, respectively to the second, measurement for the control dataset. Accordingly, we used

T1 and *T2* when referring to the first, respectively to the second, measurement for the training dataset.

We can see that to correctly classify all the problems in ELITE it was necessary between 19 and 22 trials, while this amount varied from 15 up 20 for THINK. In both machines, it was possible to correctly classify at least 90% of problems with at most 13 trials.

When analysing the results for ELITE, we identified some long-running programs with an energy consumption pattern different from the faster programs, as we can see in Figure 2, which shows the linear regression calculated for the training dataset of problem 1668.

The slowest program is an outlier, according to our classification presented in [18]. We could see that this tail outlier also occurred in the training dataset of other problems (e.g., 1666, 1669). We then decided to recompute the power slope after removing these tail outliers. We performed this only when slowest program of a given dataset is an outlier and its execution time is at least 50% greater than the execution time of the second slowest program of the dataset.

Tables 3 and 4 show between parentheses the result after discarding tail outliers. By removing them, it was possible to correctly classify all the problems in at most 16 trials.

There are some factors that might affect the quality or validity of our acquired data, analysis, and conclusions. We performed all tests in terminal mode, after restarting the system and killing some background processes, such as network and Xserver. Nevertheless, other OS processes are still running and could affect the measurements. However, as the `time` command considers only the CPU time used by the C++ solution being measured, we believe we have mitigated this issue.

RAPL reads Machine-Specific Registers (MSRs) once every 1 ms to update the energy measurements, which may lead to not-so-accurate results in case of short-running code paths [5]. We mitigated this by picking programs that would run for at least 10 ms, but the measurements of short-running programs may be affected by transient factors, such as scheduler decisions, OS services, and CPU temperature. We mitigated this issue by taking actions such as restarting the machines and taking cool down periods. Moreover, the energy consumption measurements were consistent.

We randomly selected 100 solutions for each of the 32 problems and we randomly split them between the control and the training sets. A different split of the solutions between these two sets could lead to different results, but we consider that it would not invalidate the approach.

4 Related Work

The interest in the energy consumption of computer applications increased in the last decade [8, 10, 11, 17], mainly after the release of Intel's RAPL [20], as it simplified the task of measuring their energy efficiency [7]. In newer Intel processors, RAPL energy measurements are nearly equal to plug power readings [7], enabling the emergence of several energy profiling tools on top of it [1, 6, 13, 18], and it was used in many works focused on the energy efficiency of computational applications.

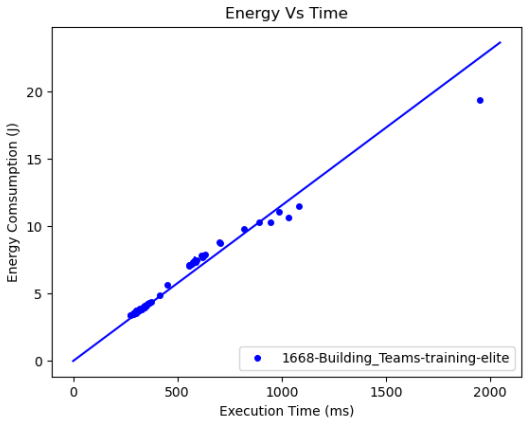

Figure 2. Linear Regression for 1668 - Training at ELITE showing a tail outlier.

Lately, several approaches, such as [12], based mainly on machine learning techniques, tried to solve the algorithm classification problem [4]. They take as input a representation derived from the source code of a program (such as its LLVM intermediate representation) and try to relate it to a given finite set of problems. Our classification approach has a similar goal, but it does not depend on a program source code, it relies only on the energy consumption profile produced by an executable program. On the other hand, our approach classifies problems for a particular machine configuration and also needs a set of solutions for a single problem to be able to compute its power slope, while approaches based on some form of source code could provide a more machine-independent classification and use a single solution.

5 Conclusions and Future Work

We presented a new approach to classify a set of problems based on its energy consumption profile. Our approach computes the power slope for a control set with several solutions for a given problem and tries to associate it with a training set that has a similar power slope.

Our results indicate that this approach could be useful to reduce the search space of possible matches for a given problem. Overall, we were able to reduce the search space by more than one third, and often by 50%.

We believe that our approach could be combined with other ones to help them to solve the classification problem more efficiently. As future work, we want to improve the accuracy of our method and to apply it in the context of other programming languages, such as Python and Java.

References

[1] Dirk Beyer and Philipp Wendler. 2020. CPU Energy Meter: A tool for energy-aware algorithms engineering. In *Tools and Algorithms for the Construction and Analysis of Systems: 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings, Part II 26*. Springer, 126–133.

[2] Marcelo Borges Nogueira and Sérgio Queiroz de Medeiros. 2024. *Dataset and Software Supporting the Paper "Classifying C++ Solutions Based on Their Energy Profile"*. https://doi.org/10.5281/zenodo.13235026

[3] Richard Burden and J. Douglas Faires. 2016. *Análise numérica* (3ª edição ed.). Cengage Learning.

[4] Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, and Fernando M. Quintão Pereira. 2023. A Game-Based Framework to Compare Program Classifiers and Evaders. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization* (Montréal, QC, Canada) *(CGO 2023)*. Association for Computing Machinery, New York, NY, USA, 108–121. https://doi.org/10.1145/3579990.3580012

[5] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review* 40, 3 (2012), 13–17.

[6] Jaimie Kelley, Christopher Stewart, Devesh Tiwari, and Saurabh Gupta. 2016. Adaptive power profiling for many-core HPC architectures. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 179–188.

[7] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. 3, 2 (2018), 1–26. https://doi.org/10.1145/3177754

[8] Lukas Koedijk and Ana Oprescu. 2022. Finding Significant Differences in the Energy Consumption when Comparing Programming Languages and Programs. In *2022 International Conference on ICT for Sustainability (ICT4S)*. 1–12. https://doi.org/10.1109/ICT4S55073.2022.00012

[9] Antti Laaksonen, Roope Salmi, and Topi Talvitie. 2015. *Code Submission Evaluation System*. https://cses.fi/

[10] Luís Gabriel Lima, Francisco Soares-Neto, Paulo Lieuthier, Fernando Castor, Gilberto Melfe, and João Paulo Fernandes. 2019. On Haskell and energy efficiency. *Journal of Systems and Software* 149 (2019), 554–580. https://doi.org/10.1016/j.jss.2018.12.014

[11] Charalampos Marantos, Lazaros Papadopoulos, Christos P Lamprakos, Konstantinos Salapas, and Dimitrios Soudris. 2022. Bringing Energy Efficiency closer to Application Developers: An Extensible Software Analysis Framework. *IEEE Transactions on Sustainable Computing* (2022).

[12] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 1287–1293.

[13] Adel Noureddine. 2022. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. In *2022 18th International Conference on Intelligent Environments (IE)*. 1–4. https://doi.org/10.1109/IE54923.2022.9826760

[14] Dinglan Peng, Shuxin Zheng, Yatao Li, Guolin Ke, Di He, and Tie-Yan Liu. 2021. How could neural networks understand programs?. In *International Conference on Machine Learning*. PMLR, 8476–8486.

[15] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* (Vancouver, BC, Canada) *(SLE 2017)*. Association for Computing Machinery, New York, NY, USA, 256–267. https://doi.org/10.1145/3136014.3136031

[16] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2021. Ranking programming languages by energy efficiency. *Science of Computer Programming* 205 (2021), 102609.

[17] Gustavo Pinto and Fernando Castor. 2017. Energy Efficiency: A New Concern for Application Software Developers. *Commun. ACM* 60, 12 (nov 2017), 68–75. https://doi.org/10.1145/3154384

[18] Sérgio Queiroz de Medeiros, Marcelo Borges Nogueira, and Gustavo Quezado Gurgel Magalhães. 2023. Analyzing the Time x Energy Relation in C++ Solutions Mined from a Programming Contest Site. In *Proceedings of the XXVII Brazilian Symposium on Programming Languages (SBLP '23)*. Association for Computing Machinery, New York, NY, USA, 64–72. https://doi.org/10.1145/3624309.3624312

[19] Henry Gordon Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society* 74, 2 (1953), 358–366. https://doi.org/10.1090/S0002-9947-1953-0053041-6

[20] Vincent M Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Dan Terpstra, and Shirley Moore. 2012. Measuring energy and power with PAPI. In *2012 41st international conference on parallel processing workshops*. IEEE, 262–268.