Title of Your Work

Afonso Carreira - 62701
Estudo Orientado
Mestrado em Engenharia Informática
Faculdade de Ciências, Universidade de Lisboa
fc62701@fc.ul.pt

Abstract

your abstract (about 10 lines).

Keywords 5 keywords

1 Introduction

This thesis aims to make programmers aware of the energy consumed by their programs. The time it takes to execute an algorithm and the memory it takes to perform a given task have always been issues that developers have been concerned about, and recently there have been attempts to reduce the energy cost of programs. However, building energy-efficient code is not a simple task and can often be overlooked. Because of its difficulty, there is still a need for tools that can help with this task[1].

Saving energy in programs is crucial for the operation of certain devices, such as mobile phones or IoT devices, so certain techniques need to be applied in order to reduce the energy of a program. For mobile devices, techniques are already used to save the battery when necessary, but for systems that don't use batteries, such as servers, energy is rarely taken into account when developing a program. This lack of concern or awareness on the part of developers, although it appears to have a small impact, turns out to be quite significant, as in 2020 around 7% of global electricity use is due to information and communications technology, with an anticipated rise in line with the growing demand for new technologies[2]. This trend has become even more significant with the increased use of artificial intelligence [3], especially large scale models such as ChatGPT, which require significant computing resources to train and run. These energy-intensive processes contribute significantly to global energy consumption and carbon emissions, raising environmental concerns as AI adoption continues to grow.

There are several reasons why there is a need to save energy in software, be it for mobile systems or data centre applications.

- The dependence of mobile devices on batteries. All mobile devices rely on their batteries, so the software they run needs to make the best use of resources to conserve battery power.
- Reducing operating costs in data centers. It is crucial to reduce the operating cost of data centers by using energy-efficient programs. This reduction results in economic benefits for companies and contributes positively to environmental sustainability.
- Reducing energy consumption has a benefic impact in our environment, as it saves energy that can be used for other things.

When, trying to improve the energy efficiency of their code, programmers rely on blogs, websites and YouTube videos, which in most cases do not give completly right answers, and sometimes even the wrong ones. This is due to a lack of knowledge and guidelines. Also, current tools can measure the energy of programs and applications as they run, but this usually requires extra steps that many developers may not have the time or inclination to take, so there is a need for a tool that can help the developer without the need for extra effort[4].

Java is a good choice for building this tool because of its interoperability with different operating systems. It has a wide range of useful libraries (JRAPL, joularjx, Jalen) that help to measure energy accurately, and Java's typing and object-oriented features make the code easier to maintain and extend, so the tool can evolve with new energy metering standards and technologies.

In order to create this tool, static analysis techniques will have to be used to understand which methods are going to be used and, through inferences from previously made data, indicate the estimated energy levels of the program's execution. The inference will be made using energy data collected from low-level library functions. More complex functions are built on the basis of function composition, which means that, based on the estimated consumption of low-level functions, we can generalize our estimates to more complex functions and ultimately to the program as a whole.

Many devices rely on Java and the Java Virtual Machine (JVM), so it is important that the code they run is energy efficient. Several factors can affect the power consumption of Java applications, including the behavior of the garbage collector and the efficiency of the memory management system [5].

This work proposes the development of a solution capable of identifying the energy consumption of methods in programs and presenting this information quickly to the programmer, allowing him to make informed decisions in software design. **The goal is to create an extension for an IDE that integrates these functionalities. (para ja fica**

assim) This way, developers will have immediate feedback on the energy impact of their applications and can adjust them to meet efficiency requirements. Also the tool should be simple to use and developers should not need to know much about the energy to know how to use it.

Organization

- Background (2) contains information of concepts that should be known to understand the complete report. It explains why it is difficult to predict and measure energy consumption of programs. It explains different energy tools, how they work, and what they do. It explains static and dynamic analysis and why it is important in this work.
- Related work (3) contains the initials solutions proposed to the theme of energy aware programming, how they changed during the years, and what the most recent tools do. And comparing with the proposed tool in this work.
- This section (4) explains in detail the existing problem and what is the solution, and a detailed explaination on how the solution will be built.
- The preliminary results (5) contains the tests made until the moment, and what results were obtained.
- Final conclusions (6) of the work done until the moment, and explaination of what work will be continued to be done.

2 Background

When programming, developers usually take into consideration the time the program takes to complete or the response time from client to server, or the memory usage. Most don't have the idea of how much energy their program uses or how much it can use in certain cases, and to get this idea is not as trivial as it seems. The energy usage is non deterministic and is not linear with the time the program takes to run.

2.1 Energy

There are some tools capable of displaying the energy consumption of programs.

Intel RAPL (Running Average Power Limit) is a tool for monitoring power consumption. It utilizes Model-Specific Registers (MSRs), which are used for program execution tracing, performance monitoring, and toggling CPU features. These registers store the total energy usage of the CPU, allowing it to be read and analyzed.

Perf is a command line tool already available in linux, and is mainly used for performance monitoring and profiling. Although its not specific for energy measurement, it can do it, with Intel RAPL but not as pratical as other tools, specially when its needed to meausre a single process energy consumption.

Powertop is another tool capable of providing the power consumption, however it only works for laptops, as it requires to check the battery to see how much energy was used and calculate the power consumption.

Joularjx is a tool based on powerjoular and capable to measure energy consumption on Java programs, displaying the methods consumption.

Experiment-Runner is a framework built in python made to facilitate experiments, it is easily customizable and can be used with energy measurement tools to monitor the power consumption of another process.

To perform this work a tool is needed in order to measure the energy of programs, methods or code snippets. The tool used was powerjoular [6], a open source tool, capable of measuring energy, from the CPU and GPU, using the Intel RAPL power data through the Linux powercap interface it can read the energy from the CPU and for the GPU it uses NVIDIA SMI to directly read the power consumption. To read the power consumption of specific processes, PowerJoular monitors the CPU cycles and utilization of each process. By knowing the total power consumption of the CPU through the RAPL interface, it can calculate the power usage of individual processes based on their CPU utilization. It is build in ADA, that is considered one of most energy efficient programming languages and it can monitor applications by name or PID.

2.2 Code static and dynamic analysis

Also, the tool proposed in this work will mainly use static analysis to achieve its goal, so it is crucial to understand why it is preferred over dynamic analysis in this context.

Static analysis, as the name implies, analyzes the code statically, meaning it examines the code without executing it. By examining the code, static analysis tools can understand how the program will behave at runtime[7], this analysis often aims for soundness, meaning that if the tool catches an error, it means that the error really exists, there are no false negatives. However, this can come at the cost of producing false positives, where issues that are not actually problems are flaged, so its important to keeps a balance between them. This analysis allows to check the entire source code and every path, much like compilers check syntax and types. Still, they can only predict some behaviors, as some can only be found when the program is executed, for example, by using dynamic analysis.

Dynamic analysis, on the other hand, executes the program and observes its exact behavior without having to estimate or predict. This type of analysis leaves no doubt about memory usage, output, the path taken, how much time it took[7]. A good example of dynamic analysis is unit testing, which tries to cover as many code paths as possible with different inputs, to understand as much as possible how the program works, and to find something that might be difficult to find with static analysis. However, dynamic analysis can

be time consuming, especially for programs that take a long time to complete.

For fast power estimation, static analysis is preferable. It analyzes code faster and is better suited for large projects with multiple dependencies, where dynamic analysis can be very difficult to achieve due to complex setup and long execution times. Although static analysis may not be as accurate as dynamic analysis, it is still a viable solution. In addition, static analysis is more portable because its setup is much simpler than the more complex setup required for dynamic analysis. Developers typically prefer not to run the program just to get an average measure of energy consumption for a code snippet or program, as it is time consuming and impractical. Therefore, using static analysis to infer energy consumption makes sense in this context.

3 Related Work

Energy efficiency is a critical focus across industries, as it directly impacts global sustainability, economic costs, and product quality. The goal is to reduce greenhouse gases to create a sustainable future, reduce infrastructure costs, and improve product quality[8].

In particular, large scale computation and communication consume a lot of global energy, and these values have been increasing in the last decades, so the topic of energy aware programming and energy efficient software has been targeted by many researchers in recent years with the objective of reducing energy costs in large IT infrastructure. This improvement can be considered a optimization problem and can be tackled in several ways for example an heuristic approach by adjusting the hardware performance dynamically, or completing tasks in their deadlines, using the least energy possible. However some of this implementations can only be short term solutions and in long term, the focus will be toward more complex models that can predict and optimize performance relative to hardware configurations[9]. In addition, new languages can be developed to address these goals, as demonstrated by ENT[10]. ENT is a Java extension that empowers programmers with more direct control over the energy consumption of their applications. ENT's type system enables applications to adapt dynamically to power constraints by switching operational modes based on resource availability, such as battery level or CPU temperature, allowing for software-level energy optimization. However, the language introduces complexity, making it potentially challenging for developers to learn and adapt to existing codebases.

————————- This paper[11] shows that system calls are directly related to energy consumption in Android applications. With this insight, it's possible to use static analysis to identify system calls within the code. This information can then be used to infer potential energy usage patterns, providing an early indication of where higher energy consumption may occur. ————————-

To tackle the problem of energy consumption in IT, some solutions have been presented. Some researchers focused on using energy measurement tools, like JRAPL to measure common libraries in Java and understand how much energy they use and what are the best alternatives to improve the energy efficiency of the code[12]. Observing common libraries for the implementation of list, sets and maps, is possible to see which ones have the better energy efficiency and what changes could improve the code. Hasan et al.'s [13] research adds to this by creating detailed energy profiles for various Java collection classes, including lists, maps, and sets, across different implementations (Java Collections Framework, Apache Commons Collections, and Trove). Their work presents concrete quantification of energy consumption in these collections based on common operations such as insertion, iteration, and random access, and highlights the performance impact of collection types on energy efficiency for different input sizes.

However, because these collections are often used with threads, it is important to understand how much energy efficiency can be improved without compromising thread safety. The energy consumption of Java's thread-safe collections was studied in [14], where researchers demonstrated that switching to more energy-efficient collection implementations can reduce energy usage while maintaining thread safety.

Building on these efforts, [15], Pereira et al. introduced a static analysis tool (Jstanley), as part of a Eclipse plugin, that can detect energy inefficient collections and recommend better alternatives. While jStanley demonstrated notable improvements in energy efficiency within its specific context, it has several limitations. For example they only account for 3 collections, (Lists, Sets and Maps), they only account for 3 sizes of the collections (25,000, 250,000 and 1,000,000), it does not account for loops, thread safe and thread unsafe collections. Compared to our approach, Jstanley is limited, as it does not provide the actual information about the energy spent, it just shows recommendations. While the tool shows great improvements in its tested environment, replacing collections may not be enough in many practical cases. A more extensive tool, capable of analyzing a wider range of collections and providing energy metrics, would enable developers to achieve even greater energy efficiency and awareness.

In this work[16], a tool (CT+) has been proposed that is capable of performing static analysis of the code and recommending changes that reduce energy consumption. It improved from previous works by taking into account more collections implementations, more operations, thread safety and support for mobile applications.

In addition, SEEP [17] uses symbolic execution for energy profiling, generating multiple binaries representing different

code paths and input scenarios. By analyzing these binaries with hardware-based energy measurement devices, SEEP provides energy consumption data, offering a deeper understanding of code efficiency across various inputs and paths. This approach complements other tool, such as PEEK, which builds on SEEP to help developers optimize energy usage with minimal effort [18]. PEEK is an IDE-integrated framework that guides developers in writing energy-efficient code. It has a front end for IDE interfaces (e.g., Eclipse, Xcode), a middle end to manage data and versioning via Git, and a backend where energy analysis is performed—either through SEEP or hardware devices. Through these layers, PEEK identifies inefficiencies and suggests optimizations, supporting efficient coding practices. However, it has some limitations when compared to the proposed approach in this work, it uses dynamic analysis instead of static analysis, which was already explained in, 2.2, why it was choosen over dynamic analysis.

This solutions have some limitations, like the need to execute the code before showing the average energy cost to the developer and have limited collections.

This work has the objective of creating a user friendly tool that quickly estimates energy consumption through static analysis and providing developers recommendations. This energy will then be displayed to the developer making him more aware of the energy consumption of the program and help make better decisions.

Where the existing related technologies are described and explored. What is the state of the art on the topic you are working?

4 «Section(s) about your Work»

As described in the previous sections, the challenge is to make developers aware of the energy consumption of their programs. By using a simple and practical tool, they can quickly and accurately estimate their program's energy consumption. This allows them to get immediate feedback on energy consumption with every code change, facilitating energy-efficient development.

To provide this insight to developers it is necessary to build a tool that can provide all of that. The tool needs to be pratical, which means that integrating it in an IDE is a very good option. With this the developer only needs to download an extension for an IDE and will have the insights desired. The tool will be an extension using Language Server Protocol (LSP), so it can be integrated in all the IDEs that support this feature (VScode, Eclipse, Neovim, Intellij IDEA). This will make it accessible to most developers wanting feedback on the energy consumption. To make it fast, it will use static analysis to parse the code into an Abstract Syntax Tree (ABS), from there its capable of analyzing the code and using an inference function it will output the estimated cost. **It will also be possible to make a more precise estimation, by running the code and also provide recommendations. optional/if i have time**.

————————-

The inference function will work like this **Understand how to do this** First it is important to understand which snippets of code influence the energy consumption, what are the most common ones (sys calls/ CPU usage/ recursion/ loops). Maybe collect data from the powerjoular and use a ML model. Maybe do it by hand with data analysis and obtain a formula capable of infering the energy. ————————-

Here you should describe in as much detail as possible the problem and your plan to tackle it (use appropriate section titles for the described work).

What are the methods you are planning to use, or already started to use, to tackle your problem.

This should be based on related word, your understanding of the problem and eventually preliminary results.

5 Preliminary Results (optional)

In the start of the project, some tools were tested in order to see how to get the energy profiles for later use. The tools tested were PowerJoular, powertop, perf and joularjx.

Perf is a Linux tool primarily designed for analyzing application performance characteristics rather than precise energy measurement. While it can provide some energy-related metrics, its measurements tend to be imprecise. In this context, perf was used mainly to get a rough idea of energy consumption and to serve as an alternative when more accurate tools were unavailable.

Powertop was also tested, but it could only perform energy measurements on laptops, as it relies on battery drain data to calculate energy consumption. Since this approach doesn't align with our specific requirements, we considered Powertop as a last-resort option.

Joularjx is a tool based on PowerJoular capable of measuring energy consumption of Java applications and it provides the energy used by the methods and overall program. Although it uses PowerJoular, when measuring the same program it gives different measurements.**why**

As described in the the 2.1, PowerJoular is the best option. As a command line program it can be easily adpated to measure any program or code snippet in most languages. So it was combined with the framework experiment-runner, that facilitates the process.

Initially, JoularJx and the experiment-runner using powerjoular were used to explore their capabilities and familiarize with the tools. A sample Fibonacci program written in Java and C was used as a test case. However, the energy measurements provided by the two tools differed, and the experiment runner occasionally encountered errors. Later, it

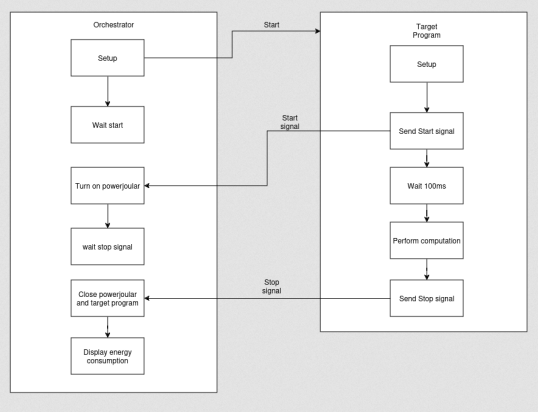

Figure 1. orchestrators process

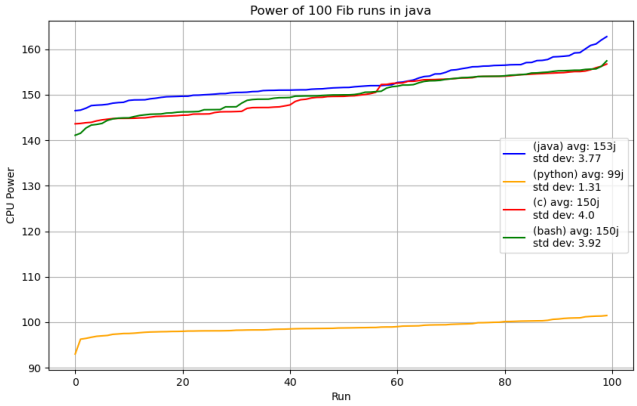

Figure 2. orchestrators comparison

was determined that these problems were due to incorrect use of the framework. However it was still decided the the best approach was to make a new orchestrator similar to the Experiment-Runner, but simpler and specifically focused on measuring process energy consumption using PowerJoular. A Java-based orchestrator was initially developed because the Fibonacci implementation was also in Java. However, when tested, the energy measurement results differed significantly from those of the experiment runner, even though the main difference was the programming language (Java vs. Python).

To further analyze these inconsistencies, another orchestrator was developed in Python. This allowed for a closer examination of the differences in energy measurements and a deeper understanding of the behavior of the tools.

To better understand the results and the problem encountered, its better to see, first, how the orchestrators behave:

- On the start of the orchestrator it launches a command to start the target Java Program and waits a signal.
- The Java program start and setup the necessary things to run (starting threads, reading/writing files, etc.) and then it sends a start signal to the orchestrator to start monitoring the computation wanted, and waits for some miliseconds.
- The orchestrator receives the start signal and reads the PID of the target program from a file and starts PowerJoular using that PID. Then it waits for the stop signal.
- The target Java program when it finishes the computation it sends a stop signal back.
- The orchestrator on receiving the stop signal, first stops PowerJoular and then stops the target program, if needed. Then it displays the energy measurement information stored in the files created by PowerJoular.

This process is more clearly illustrated in Figure 1

With this process it was noticeble that the Java orchestrator was getting significantly more energy consumption than the Python one, which is not very logical, since they both target the same program. So, to try and check which one was having problems, two more orchestrators were implemented, one in C and another in bash.

After running the tests again it was possible to see that the Python orchestrator was getting values way more different than the other three orchestrators as show in Figure 2. The figure contains 100 runs of the Fibonacci recursive program written in Java and order by the less energy to highest energy. And it shows the energy reads for the four different orchestrators used. The labels contain the average energy values and its standard deviation.

Further analysis of the orchestrators revealed a notable difference in behavior. When the Python orchestrator was running, both the parent and child processes consumed CPU resources. In contrast, the other orchestrators (Java, C, and Bash) showed CPU usage only in the child process. This disparity may explain why powerjoular reported lower energy consumption for the Python orchestrator. Since the CPU load was shared between the parent and child processes, powerjoular, which measures energy only for the child process (the target Fibonacci program), captured less total energy usage. Since the experiment runner included an example demonstrating how to use the framework with powerjoular, the authors were made aware of this potential conflict when launching powerjoular from Python.

6 Forthcoming Work and Conclusions

Possible work timeline

It should include subsections that describe the work to be carried out during the rest of the year, and what its objectives are.

It should also include planning until the end of the work, in chronological order.

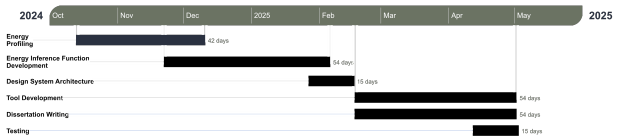

Figure 3. Work

Include a final concluding subsection (may be a separate section) with a summary of contributions already made, a preliminary self-assessment of the work done so far, and difficulties encountered.

References

[1] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 22–31, New York, NY, USA, 2014. Association for Computing Machinery.

[2] Anders Andrae. New perspectives on internet electricity use in 2030. *Engineering and Applied Science Letters*, 3:19–31, 06 2020.

[3] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[4] Gustavo Pinto and Fernando Castor. Energy efficiency: a new concern for application software developers. *Commun. ACM*, 60(12):68–75, November 2017.

[5] N. Vijaykrishnan, M. Kandemir, S. Kim, S. Tomar, A. Sivasubramaniam, and M. J. Irwin. Energy behavior of java applications from the memory perspective. In *Proceedings of the 2001 Symposium on JavaTM Virtual Machine Research and Technology Symposium - Volume 1*, JVM'01, page 23, USA, 2001. USENIX Association.

[6] Adel Noureddine. Powerjoular and joularjx: Multi-platform software power monitoring tools. In *18th International Conference on Intelligent Environments (IE2022)*, Biarritz, France, Jun 2022.

[7] Michael D Ernst. Static and dynamic analysis: Synergy and duality. In *WODA 2003: ICSE Workshop on Dynamic Analysis*, pages 24–27, 2003.

[8] Kenneth Gillingham, Richard G. Newell, and Karen Palmer. Energy efficiency economics and policy. *Annual Review of Resource Economics*, 1(Volume 1, 2009):597–620, 2009.

[9] David J. Brown and Charles Reams. Toward energy-efficient computing. *Commun. ACM*, 53(3):50–58, March 2010.

[10] Anthony Canino and Yu David Liu. Proactive and adaptive energy-aware programming with mixed typechecking. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 217–232, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Karan Aggarwal, Z Chenlei, J Campbell, Abram Hindle, and Eleni Stroulia. The power of system call traces: Predicting the software energy consumption impact of changes. 2014.

[12] Rui Pereira, Marco Couto, João Saraiva, Jácome Cunha, and João Paulo Fernandes. The influence of the java collection framework on overall energy consumption. In *Proceedings of the 5th International Workshop on Green and Sustainable Software*, GREENS '16, page 15–21, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Samir Hasan, Zachary King, Munawar Hafiz, Mohammed Sayagh, Bram Adams, and Abram Hindle. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 225–236, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Gustavo Pinto, Kenan Liu, Fernando Castor, and Yu David Liu. A comprehensive study on the energy efficiency of java's thread-safe collections. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 20–31, 2016.

[15] Rui Pereira, Pedro Simão, Jácome Cunha, and João Saraiva. jstanley: placing a green thumb on java collections. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE '18, page 856–859, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Wellington Oliveira, Renato Oliveira, Fernando Castor, Benito Fernandes, and Gustavo Pinto. Recommending energy-efficient java collections. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 160–170, 2019.

[17] Timo Hönig, Christopher Eibel, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Seep: exploiting symbolic execution for energy-aware programming. *SIGOPS Oper. Syst. Rev.*, 45(3):58–62, January 2012.

[18] Timo Hönig, Heiko Janker, Christopher Eibel, Oliver Mihelic, and Rüdiger Kapitza. Proactive Energy-Aware programming with PEEK. In *2014 Conference on Timely Results in Operating Systems (TRIOS 14)*, Broomfield, CO, October 2014. USENIX Association.