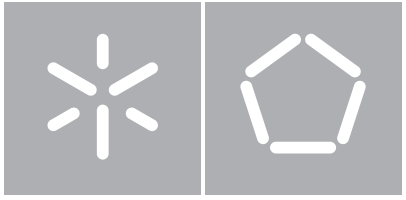
Universidade do Minho
Escola de Engenharia

Tiago Alves Carção

Spectrum-based Energy Leak Localization

Outubro 2014

This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, within projects: FCOMP-01-0124-FEDER-020484, FCOMP-01-0124-FEDER-022701, and grant ref. BI2-2013_PTDC/EIA-CCO/116796/2010.



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Alves Carção

Spectrum-based Energy Leak Localization

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Doutor João Saraiva
Professor Doutor Jácome Cunha

Outubro 2014

Anexo 3

DECLARAÇÃO

Nome

Tiago Alves Carção

Endereço electrónico: tiago.carcao@gmail.com Telefone: 932823790 / _____

Número do Bilhete de Identidade: 13370398

Título dissertação □/tese □

Sprectrum-based Energy Leak Localization

Orientador(es):

João Alexandre Saraiva, Jácome Miguel Costa da Cunha

Ano de conclusão: 2014

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado em Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 31 / 10 / 2014

Assinatura: Tiago Alves Carção

Acknowledgements

I want to thank both of my supervisors, Prof. João Saraiva and Prof. Jácome Cunha, due to their knowledge, dedication, experience, professionalism, innovative spirit, and their constant ability to discuss every detail throughout my Thesis, helped me greatly.

I also want to thank all of the members of the GreenLab @ Uminho where with the weekly meetings were able to provide useful contributions in this Thesis development.

To Joana, the person that was always on my side throughout this Thesis, that supported me in the hardest moments always with an incentive word, and undoubtedly without I could not finish this path. Thank you so much.

To my laboratory buddies, Claudio and Rui that had spent some great time with me, in Romania, Italy, The Netherlands, Póvoa de Varzim and Australia.

To all of my friends, specially, David and Casimiro that always provide such a good and fun time when I am with them, and João and Daniel that are always a source of interesting and enriching discussions.

To my little brother that is not so little anymore, a big thanks for being a person that I can always count on and discuss the various subjects of life and sports.

And finally, I would like to thank my parents that always supported me emotionally in my entire life, and gave me the opportunity to be who I am today. Thank you.

Abstract

For the past few years, we have begun to witness an exponential growth in the information and communication technologies (ICT) sector. While undoubtedly a milestone, all of this occurs at the expense of high energy costs needed to supply servers, data centers, and any use of computers. Associated with these high energy costs is the emission of greenhouse gases. These two issues have become major problems in society. The ICT sector contributes to 7% of the overall energy consumption, with 50% of the energy costs of an organization being attributed to the IT departments.

The rapid growth of internet-based businesses using computers, often referred to as "cloud computing", and the costs associated with the energy to run the IT infrastructure are the main drivers of green computing.

Most of measures taken to address the hight level of energy consumption have been on the hardware side. But, eventually, by physical limitations or not, the software will become the target (Green Software Computing). There are studies already that prove that developers are aware of energy usage on software issue, but complain about the lack of tools to aid the energy improvement process.

As a way to get software energy efficient, it is necessary to provide tools for the software developer to be aware of the energy footprint that he/she is creating with his/her application.

This thesis proposes and implements a methodology to analyze the software energy consumption. In one of the phases of the methodology, it also defines a technique that with uses a model to analyze and identify energy leaks in the software. This work is validated by comparing with other work already done.

With this results, one intendeds to provide some help to the development phase and to generate more energy efficient programs that will have less energy costs associated with, while support practices that promote and contribute to sustainability.

Resumo

Localização de falhas de energia baseada no espectro do programa

Nos últimos anos, temos vindo a assistir a um crescimento exponencial no sector das tecnologias de comunicação e informação (TIC). Contudo, apesar de, inquestionavelmente, se tratar um marco importante, tudo isto ocorre à custa de altos gastos de energia necessários para alimentar servidores, centros de dados e qualquer uso de computadores.

Paralelamente, associado aos altos custos de energia estão as emissões dos gases de efeito de estufa. Estas duas questões têm-se tornado grandes problemas da sociedade. O sector das TIC contribuí para 7% do consumo global de energia, o que representa, para o departamento TI de uma organização, 50% de custos, associados, à energia.

O rápido crescimento de negócios baseados na internet que utilizam computadores, frequentemente referidos como "cloud computing" e os custos associados à energia necessária para executar uma infra-estrutura TI são os principais promotores do "green computing".

A maioria das medidas adotadas para resolver o nível elevado do consumo de energia, têm sido feitas do lado do hardware. Existem alguns estudos que comprovam que os desenvolvedores estão cientes do consumo de energia associado ao software, mas queixam-se da falta de ferramentas que auxiliem o processo de melhorar energeticamente as aplicações.

Para se obter software energeticamente eficiente, é necessário fornecer, ao desenvolvedor, ferramentas que lhe permita estar consciente da "pegada" energética que está a ser criada pela sua aplicação.

Esta tese propõe e implementa uma metodologia para analisar o consumo de energia do software. Numa das fases desta metodologia, também define uma técnica que utiliza um modelo para analisar e identificar falhas energéticas no software. Este trabalho é validado por comparação com outro trabalho já efetuado.

Com estes resultados, pretende-se contribuir com ajuda para a fase de desenvolvimento e para gerar programas mais eficientes a nível da energia que terão menores custos de energia associados, ajudando a práticas que promovem e contribuem para a sustentabilidade.

Contents

Acronyms

API Application Programming Interface

IT Information Technology

ICT Information and Communication Technologies

IDE Integrated Development Environment

MBD Model-based Diagnosis

MHS Minimum Hit-set

RAPL Running Average Power Limit

SFL Spectrum-based Fault Localization

List of Figures

Listings

1. Introduction

Currently, we are witnessing a technological era where information media has grown exponentially, with billions of users. Almost everyone has access to computers, and the Internet is accessible virtually anywhere, which is undoubtedly a milestone in the field of content delivery [Guelzim and Obaidat, 2013].

The problem with this globalization is that all of this occurs at the expense of energy consumption that is the necessary and indispensable element to supply servers, data centers and any use of computers [Guelzim and Obaidat, 2013]. The energy required to meet the growing demand for power to run and storage, grows faster along with the widespread diffusion of cloud services over the internet [Ricciardi et al., 2013]. In this way, the fast and growing power consumption attracted the attention of governments, industry and academia [Zhang and Ansari, 2013]. Also, associated with this energy consumption, is the emission of greenhouse gases. These two issues are becoming a major problem in the society of information and communication [Ricciardi et al., 2013].

The information and communication technologies (ICTs) contribute 7% to the overall world energy consumption [Vereecken et al., 2010]. This percentage has the tendency to increase further, since the traditional technologies are migrating data from local servers to remote servers and data centers, forming a large number of clouds in the Internet infrastructure [Mouftah and Kantarci, 2013].

The energy consumption has an immediate impact on the business value. In fact, the energy costs associated with information technology departments constitute approximately 50% of the overall costs of the entire organizations [Harmon and Auseklis, 2009]. There is also electricity that is wasted, and potentially avoidable, that is leading to high operating costs [Zhang and Ansari, 2013]. Thus, this raises the need and expectation of reducing the energy costs and the impact on the environment, by directing attention to these issues [Harmon and Auseklis, 2009].

The energy efficiency requires a thorough investigation to discover and understand everything that is related with it [Zhang and Ansari, 2013]. ICT services require the integration of sustainable practices for green computing to meet sustainability requirements [Har-

mon and Auseklis, 2009]. This term, **green computing**, refers to the practice of using computing resources more efficiently, maintaining or increasing their overall performance. Although the original conceptual already exists for two decades now, only since the last decade has received more attention [Harmon and Auseklis, 2009].

Thus, **green computing** paradigms are emerging to reduce energy consumption, the resulting emissions of greenhouse gases and operating costs [Ricciardi et al., 2013], and trying to find solutions that make all these systems energy efficient [Mouftah and Kantarci, 2013].

Increasingly, the industry, in an attempt to reduce costs and energy consumption, is becoming more active in the area of green computing. For example, Symantec, using the monitoring of their resources, found there were some resources that were being squandered and by implementing measures to reduce this waste they saved close to $2 million and more than 6 million kilowatts of energy [Symantec, 2008b,a]. Google also made some changes, using customized cooling systems in their data centers improved the energy consumption [Google, 2014].

New challenges, research, and discussions are being addressed to enable new models and solutions that consider energy as an additional constraint, minimizing its consumption [Ricciardi et al., 2013]. This thesis aims to contribute to help to solve this issue, with a contribution that is useful and that enables a society of growth and prosperity eco-sustainable.

This project intends to address in detail green computing in the energy consumption of software. Nowadays when one says that a program is efficient, the term efficient encapsulates the notion that software is fast to execute and performs the task without requiring a lot of resources. However, when it comes to efficiency, the efficiency can also be energetic, and it is exactly this notion that one need to change in the consciousness of the programmer, the notion that it is also possible to have an efficient software in terms of energy.

All the hardware components of ICTs consume a constant power consumption to be running. When they are performing operations they increase this power consumption. This operations are directly related to the software needs what makes the study of energy consumption quite pertinent in software. Surely related with the increase of power consumption in the hardware components consumption that software provoques, up to 90% of the energy used by ICT can be attributed to software applications running on them [Standard, 2013].

The design of software has significant impact on the amount of energy used [Standard, 2013]. So it is very important that software engineers are aware of the consumed energy by the software they designed, in order to design more efficiently in regards to energy consumption, knowing precisely where the high consumption parts are and how to correct them.

1.1. Research Questions

1. *Can we define a methodology to analyze the energy consumption of software source code?*
2. *Is it possible to adapt a purpose fault localization algorithm to the context of energy consumption?*
3. *Can we find energy leaks in software source code?*

1.2. The solution

In this thesis the objective was to create a technique that could analyze a program's execution with a test suit and discover the energy leaks present in the program. With this objective in mind a methodology to accomplish this goal was elaborated and defined. This methodology has three different phases. For each of these phases, a sub-technique was developed. In the first phase the software code is prepared to extract the execution information of each program's constituent. This information is structured and represents the constituent energy consumption, the time of its execution and the number of times it was used. After this process the software is compiled and ran with a test suit. In the following phase, the energy results produced by the program's execution are collected and treated and the information is then passed to the final phase. In the final phase and using a technique based in the programs spectrum and its execution data, the data is evaluated and the information about which are the energy leaks is obtained. This methodology in conjunction with the three phases completely defined, accomplishes the objective previously set.

1.3. Structure of the Thesis

This Thesis is organized as follows:

Section 2 - Green Computing - contains the State of the Art, with information on Green Computing evolution and the emerging area of Green Software Computing.

Section 4 - Source Code Energy Consumption Analysis - contains the definition of a methodology to analyze program's source code energy consumption

Section 5 - Instrumentation and Results Treatment - specifies in detail the first and second phases, the instrumentation and results treatment, of the methodology presented in Section 4.

Section 6 - Results analysis: Spectrum-based Energy Leak Localization - contains information about the adaptation of a fault localization technique in the context of energy usage, third phase of the methodology defined in Section 4.

Section 7 - Spectrum-based Energy Leak Localization: The Tool - describes and showcases the tool developed, which implements all the techniques presented in Sections 5 and 6.

Section 8 - Validation - contains the process of validation that was made to the methodology and consequent technique.

Section 9 - Conclusion - concludes this Thesis with comments on the work done, results, and future work, along with answers to our Research Questions.

2. Green Computing

The concept of green computing despite being a hot topic is a relatively old concept. It has emerged around the 90s when it was raised the awareness of the energy that was being used by IT devices, which led the IT community to take some measures. One of the first measures taken under green computing was the assignment of a "certificate" to products that had a concern in terms of energy consumption minimizing it while maximizing efficiency. This certificate (Figure 1) was applied to different peripherals, computers, monitors, printers, etc. One of the first real results of this awareness was the appearance of the stand-by functionality in the devices that made them entering in sleep mode after a period of inactivity.

Figure 1: *Energy Star certification symbol*

Despite the fact that this awareness already started 20 years ago, only just more recently, in the last decade, has started to exist a more active concern with the reduction of energy usage.

The costs of energy consumption in the field of ICTs will be increasing over the next 20 years [Rühl et al., 2012] which alone is a great incentive for green practices. The ICT with its intrinsic properties and with their use, helps to reduce the energy consumption in other sectors. Nonetheless, it has a forecast increase in their energy consumption. Its share of 7% in global energy consumption will be increased to more than 14.5% [Vereecken et al., 2010].

Recently, one has witnessed an exponential growth of IT devices. Data centers are nowadays a common term in the vocabulary of informatics and all the big tech companies have this kind of infrastructure. Although these infrastructures endure what is widely known as the cloud, and upon all the benefits that this feature brings, maintaining data centers

have substantial energy costs of supply (huge set of machines and devices as well as cooling systems). Adding up to the costs, there is still a large amount of greenhouse gases in this eco-system. With what is expected to be a future reality very close, the internet of things, it is expected that the network of devices present increases significantly. That fact itself will imply that there is an infrastructure capable of handling this information increase which will naturally result in an boost of global energy consumption.

With these predictions and conditions, countless associations begun to focus their attention on this issue. A number of organizations, including the USA's Environmental Protection Agency (EPA), have identified a number of processes, optimizations and energy alternatives in data centers and even in home appliances [Fanara et al., 2009]. Google was another of the organizations that included in its research the topic of green computing and has already achieved some results [Google, 2014].

Another aspect of the ITs is the use of personal computers, and recently (and exponentially growing) smartphones and tablets. These devices have an intrinsic concern for energy usage since their power supply is taken from a battery which has a finite limited capacity. The less energy consuming components of these devices, the less power will be consumed, and therefore it is possible to use these devices during a longer period of time.

Having regard to energy concerns, version after version, Intel, the largest producer of processors for computers, smartphones, tablets, etc. has had a concern in obtaining maximum efficiency while lowering the power consumption of its processors. This development has permitted after each release, on the one hand to reduce the energy consumption in the use of processors, and on the other hand extend the usage time of portable battery powered devices.

The interest in this area exists and has strong promoters which is already remarkable. However, one can not ignore the fact that the ITs consist of two artifacts of different types: hardware and software. If on one side a lot has been done in order to decrease the power consumption of the hardware, as already shown – which is understandable since the hardware change does not alter the normal functioning of the software and allows immediate energy savings to be made – at some point, either by physical limitations or because more needs to be done to reduce the energy usage, software will be an obvious target.

2.1. Green Software Computing

This concern with the software has already started to happen although on a smaller scale when compared to the hardware, and has already been dubbed the *Green Software Computing*. Slowly we start to see some initiatives from companies that support some of the world's major operating systems such as Apple's Mac OS X and iOS, and Google with Android. On one hand Apple in its most recent versions of the operating system for desktop (Mac OS X), by using only the operating system software, could improve the energy performance of their computers, thus allowing the battery life to be prolonged, in some cases, up to 4 hours [Brownlee, 2013]. Regarding to mobile OS, iOS and Android devices already have tools that allow the user to check the battery consumption profile of applications. Apple already allows its developers in its development IDE (Xcode) to make an energy profiling to their applications. Android in its next version will also bring energy profiling tools aimed at developers.

A study also proved that the developers are aware and interested in the green software domain [Pinto et al., 2014]. This study demonstrated that there is a community interest in learning more about this area and trying to find out what may be the causes of high energy consumption and possible ways to address them. Also note that developers feel there is a lack of tools that support this identification process and optimization.

To make greener software, besides requiring the energy consumption values, one also wants to know what zones of code are hotspots, or areas where the power consumption is excessive. These areas can be seen as *red* zones and are the firsts to be investigated.

In order to proceed with the identification of these red areas, one needs to be able to measure the energy consumption. As mentioned, research in Green Software Computing is still in an early stage and therefore the tools that exist to measure this consumption are incomplete and insufficient. To overcome this fact in some cases estimates are used. Some of these estimates are not reliable and are not precise. Although external devices can be used, they will only allow to measure the total power consumption for a period of time, this option may have read errors that are always associated with the reading of values in external devices. Adding to these difficulties, there is also the fact that the measurement of

consumption is done on the whole system and not only on the desired applications.

Intel, as a manufacturer of processors, and also as a promoter of Green Computing, since 2012 began to worry about providing tools to developers to gain access to energy consumption by existent on-chip components (either the processor, DRAM or even on-chip GPU). This tool is provided as an API and is named Running Average Power Limit (RAPL) Rotem et al. [2012]. The RAPL is an interface that allows system calls to consult the values of energy consumed by each component. These intakes are updated by the processor that will from time to time update some special registers in memory reserved for this purpose. Thus, by reading these registers one can read the recorded energy consumption. There are studies that prove that the measurements made by this interface are accurate and are trustworthy [Hähnel et al., 2012]. However, RAPL only reports on-chip consumption leaving aside peripherals such as the hard drive, and non-integrated GPUs and motherboard.

To address this lack of information, the academia started to construct their own tools that allows them to see the power consumption.

As previously said, the tools to run energy profiling are short in number and often lack the desired accuracy. Because of this, several laboratories have developed their own methods for monitoring power consumption. Software Improvement Group (SIG), a company that is linked to qualitative analysis of software, in collaboration with Amsterdam University, developed a laboratory related to energy. This laboratory, among other contributions to the field, [Arnoldus et al., 2013; Grosskop and Visser, 2013; Grosskop, 2013], has developed a piece of hardware that can be connected to any computer hardware component and also connected to a device (DAQ) that will produce as output the power consumption of the components connected to it (Figure 2) [Ferreira et al., 2013]. Li et al. [2014] also developed a similar technique but for mobile devices. One can also use hybrid variants for measuring the power consumption: Li et al. [2013] showed that combining hardware analysis based in power measurements, and software statistical modeling, at least in Android, is possible to calculate values of the energy consumption's source line.

Measuring the energy, which can be done using external or internal devices, and with a higher/lower level of refinement, some contributions have already been made.

Using a model-based policy, Zhang et al. [2010] during his PhD developed an applica-

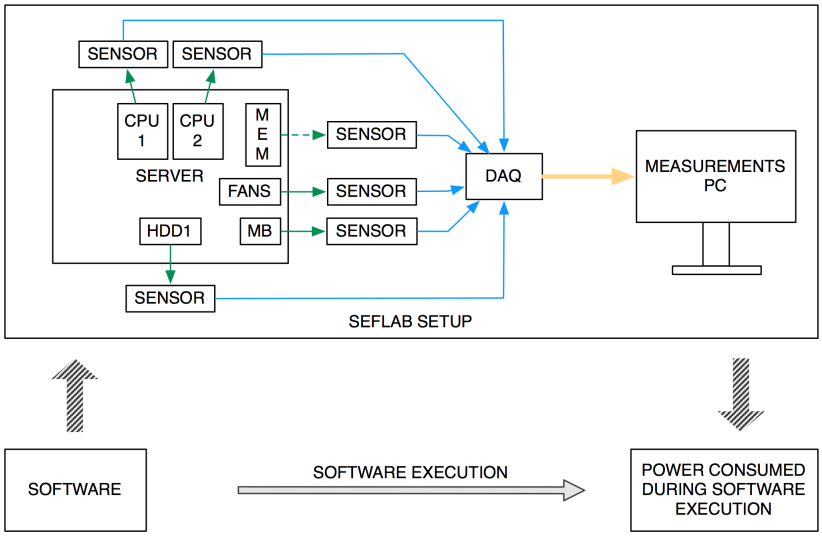


Figure 2: *SEFLab infrastructure [Ferreira et al., 2013]*

tion for Android that allows any application's energy consumption to be monitored. The limitations of this application are largely associated with the problems of using models, i.e., the need to calibrate the model for the environment where the application is running. Thus Couto [2014], attempts to solve this and other limitations by creating a dynamically calibration of the models for any smartphone.

Hönig et al. [2013] published a technique that uses a model-based technique to generate information about software energy consumption. This technique is presented in Figure 3, where using symbolic execution and execution knowledge stored in a database, this technique tries to predict energy consumption of a particular program.

Also in an attempt to provide energy information for a particular program, Noureddine et al. [2014] developed a technique for the instrumentation and collection of the energy usage data in Java (*JalenUnit*, as seen in Figure 4), and presented an approach to analyze the power consumption of a method by varying the method's data input.

One of the current practices in the development of applications is, before publishing them to the public, run an obfuscation tool on the source code trying to deter copying of software. Using this as motivation, Sahin et al. [2011] investigated and demonstrated that obfuscation has a significant statistical impact and is more likely to increase the energy usage. These conclusions are an indicator that the way the code is written has an influence

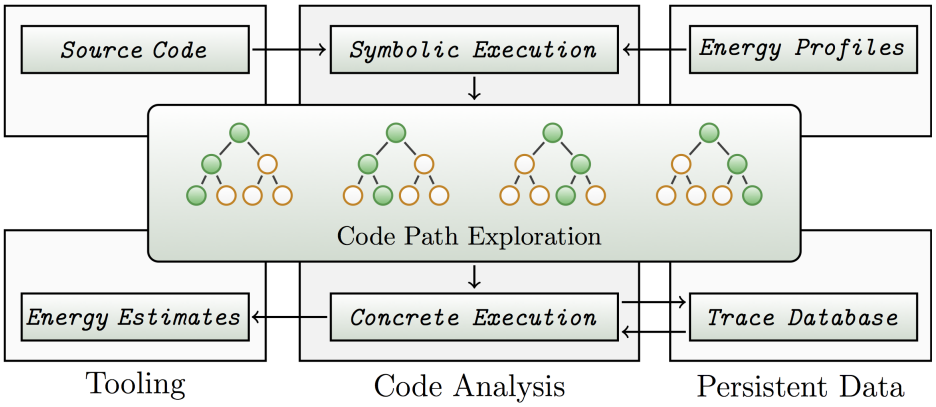


Figure 3: *SEEP technique that tries to bring energetic advices to developers development process [Hönig et al., 2013]*

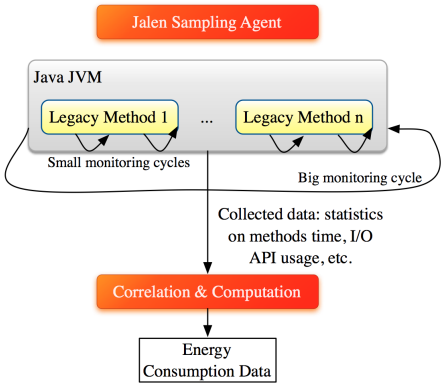


Figure 4: *A framework to retrieve energy consumption values named JalenUnit [Noureddine et al., 2014]*

on energy consumption.

Gutiérrez et al. [2014] did a energy consumption study in multiple Java Collections. They produced as results what were the collections that had higher intakes of energy or that were more energetically efficient. In conjunction with this analysis, they also developed a framework which taking into account the data obtained, refactors the java source code to use the collections that statistically consume less energy.

A common practice in the software world, the use of patterns, was also questioned at the energy level. Vásquez et al. [2014] presented a qualitative and quantitative study of the high energy consumption in API calls and patterns used in Android. Their findings indicate

that there are patterns that have a significant impact on the energy consumption, such as the Model-View-Controller pattern. Sahin et al. [2012] also did an analysis of the impact on the energy usage in software design patterns.

3. Fault Localization Techniques

It is becoming more common the IDEs (integrated development environment) offer tools to analyze the values of power consumption of the programs being written. Although this information is already provided to the users, the notion of what it means and what relevance that consumption of certain components have in relation to consumption of the program is yet to be determined.

In this thesis, a set of techniques and tools will be proposed to determine *red* areas in the software energy consumption. In this context, a parallel is made between the detection of anomalies in energy consumption in software during execution of the program and the detection of faults in the execution of a program. Establishing this parallelism, fault detection techniques, used to investigate the failures in the execution of a program, may be adapted to be used in the analysis of energy consumption.

When it comes to identify faults in programs there are two main analysis: model-based or statistically. The model-based analysis is an analysis that allows us to extract accurate conclusions about failures that can be happening. However, and because in a model we have to define the complete system, when applied to energy, at least for now, it would be impractical to obtain an energy model of the software. It would be necessary to take into account the system settings and all the implications that a change in the model would lead to energetically. The statistical analysis, based on the implementation of the program using the source code, does not allow taking totally accurate conclusions, but allows useful information to be extracted with relative ease.

So, knowing the two main analysis, the statistical analysis technique of fault localization is probably the most appropriate. Since its foundations rely on an analysis of the program based on its implementation (in its source code), one does not need to parameterize the entire system. Within the statistical analysis techniques for locating faults, the technique of using the program spectrum is more efficient than the use of dynamic slicing [Korel and Laski, 1988] and therefore the technique that stands out as candidate, with very good results in this field [Abreu et al., 2009], is the Spectrum-based based Fault Localization technique (SFL).

3.1. Sepectrum-based Fault Localization

A program spectrum [Reps et al., 1997] is a set of information of data run-time execution of a program. There are different types of program spectrum that can be used [Harrold et al., 2000]. For example, lets consider the use of the block-hit type, in the Listing 1. One can consult what is actually considered as a block-hit in the program execution. The spectrum of a block-hit program is a set of flags that will reflect if the condition of the block is used or not.

Listing 1: *Instrumented program to the block level*

```
int largestNumberAmongThreeNumbers(int a, int b, int c) {
    int res;

    if (a > b) {
    //block (c1)
        if (a > c) {
            //block (c2)
            res = a;
        }
        else {
            //bock (c3)
            res = b;
        }
    }
    else {
    //block (c4)
        if (b > c) {
            //block (c5)
            res = b;
        }
        else {
            //block (c6)
            res = c;
        }
    }

    return res;
}
```

In SFL, the hit spectrum is used to build a matrix A, of $N \times M$, where the M columns represent the different parts of the program during N executions (independent, i.e. the

$$
\begin{array}{c} \\ N \text{ spectra} \end{array}
\begin{array}{c} \overset{M \text{ components}}{} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix} \end{array}
\begin{array}{c} \text{error} \\ \text{detection} \\ \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \end{array}
$$

Figure 5: *The spectrum-based fault localization model (A,e)*

result of each execution does not influence the next) as can be seen in Figure 5. In this hit spectrum, the value 0 means that such part was not executed and the value 1 means it was. The SFL representation also presents one column vector corresponding to the errors, e. This vector represents all errors, which is the final disclosure of the existence of errors in the execution of the program where the value 0 means that no error occurred and 1 otherwise. The objective of spectrum-based fault localization is trying to find which parts of the program have a column that best explains the existence of the vector of errors. This similarity of vectors is quantified by coefficients of similarity [Jain and Dubes, 1988]. The existing test vector can be obtained in different ways. In SFL, there is the notion of an oracle that enables the vector error to be generated with the consultation of the oracle state. This oracle, in the case of detecting faults in a program, can be seen as the supposed output that the program may have. There may also be situations in which this oracle is not easily determined or given and to be used it must be generated, using other inherent techniques in the SFL domain.

Given the coefficients of similarity existing in SFL techniques the best performing coefficient is the Ochiai [Abreu et al., 2006]

$$S_O = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{01}(j)) \cdot (n_{11}(j) + n_{10}(j))}} \tag{1}$$

where $n_{11}(j)$ is the number of failed runs where part j was involved, $n_{10}(j)$ is the number of passed runs where part j was involved, $n_{01}(j)$ is the number of failed runs where part j was not involved and $n_{00}(j)$ is the number of passed runs where part j was not involved.

In the case of coefficients of similarity, it is normal that a value closer to 1 means that this vector is closer to explain the result of the vector of errors. To better understand the segmentation-based fault localization technique an example will be used. Figure 6 presents the values of the Ochiai coefficients calculated for each M column vector, of applying the SFL technique to the program shown in Listing 1 with the inputs $\langle 2, 4, 1\rangle$, $\langle 5, 3, 1\rangle$, $\langle 5, 2, 7\rangle$, $\langle 3, 9, 12\rangle$, $\langle 1, 3, 1\rangle$ and $\langle 2, 1, 4\rangle$ and with the outputs $4, 5, 7, 12, 3$ and 4 respectively.

	c_1	c_2	c_3	c_4	c_5	c_6	e
	0	0	0	1	1	0	0
	1	1	0	0	0	0	0
	1	0	1	0	0	0	1
	0	0	0	1	0	1	0
	0	0	0	1	1	0	0
	1	0	1	0	0	0	1
$n_{11}(j)$	2	0	2	0	0	0	
$n_{10}(j)$	1	1	0	3	2	1	
$n_{01}(j)$	0	2	0	2	2	2	
$s_O(j)$	0.82	0.0	1.0	0.0	0.0	0.0	

Figure 6: *Result of SFL technique applied to Listing 1, indicating c_3 as the faulty component*

The last row of the Figure 6 indicates that the component 3 (c_3) has the highest probability of being faulty, and the component 1 (c_1) is the closest second. In fact, if we consult the program in Listing 1 we can see that the block c_3 has an error, because it doesn't compare the value of b with c which will lead to failure for some inputs. The fact that c_1 has such a hight probability can be explained because this component enclosures the faulty component and so, will also fail for some inputs.

4. Source Code Energy Consumption Analysis

The process of energy analysis is dependent on the approach that one wants to define. The aim of this thesis is to conduct an energy consumption analysis of the software source code, so this process will focus on the source code level. The process takes as input a program yet to be compiled and a set of program tests, and provides information about the program's energy consumption.

What is proposed here can be seen as a generic methodology to be followed for the energy usage analysis on an application's source code. The method is generic and therefore can be applied to any language/programming paradigm.

The basis of this methodology is the methodology used in the SFL, i.e., we have a spectrum of a program that we want to identify in different tests in order to draw conclusions. However, the methodology defined here differs somewhat from the SFL approach. Because this is an energy consumption analysis the data collected must be more informative about the program's execution. This is why, at the end of the executions where it will exist the execution data non-structured, this data should be structured hierarchically so one can analyze it. After having this execution information, as in SFL technique, it is analyzed and conclusions are extracted.

In the proposed methodology, one can identify three distinct steps:

1. Instrumentation
2. Results Treatment
3. Results Analysis.

Initially, this methodology has as input the source code of the program. This source code will be instrumented and instructions that will extract the trace of program execution and relate energy consumption to the source code will be added. To make this instrumentation it will be necessary to define which are the components (packages/namespaces, functions/methods, the code block, the line of source code, etc.) to be instrumented and choose which tool will be used to perform the instrumentation. After this step, the code has to

be compiled and then executed. To fulfill the needs of running the program with different inputs it is fundamental to have a set of tests to run. Thus, compiling the code and running the program's test suit, one can obtain the energy consumption information produced by each test (Figure 7).

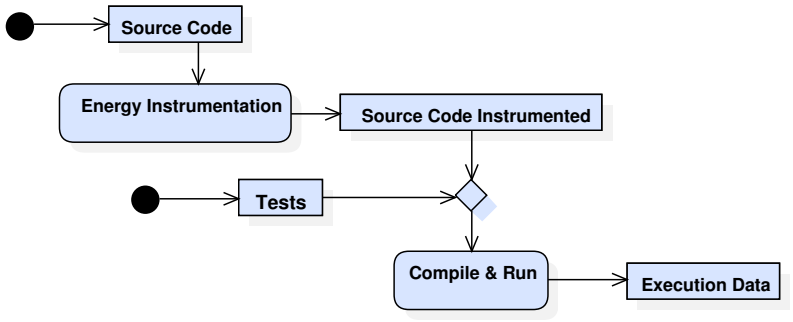


Figure 7: *Process of instrumentation*

In the next phase, and after obtaining the program's execution information of the different tests, it is necessary to structure these results. This structuring of the results is essential because, if by one side, the execution spectrum is needed, by the other side, in the context of energy consumption analysis the trace of execution containing the call hierarchy of components with the energy values is very informative (number of calls, energy consumption associated with the component and its derivatives, runtime, etc.). This process transforms the execution results and produces data in a format that can be fed into the following phase (Figure 8).

Figure 8: *Process of collecting results*

The data analysis will occur in the next phase. This analysis will try to relate the source code energy consumption with the data collected. After drawing conclusions about this data, it will be necessary to produce a report explaining the outcome from the analysis. So, at the end of this methodology as an output there will be a report with information about the analysis (Figure 9).

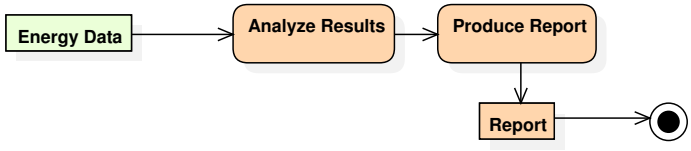


Figure 9: *Process of analysing results*

The entire process with this three phases is completely defined and can be seen in Figure 10.

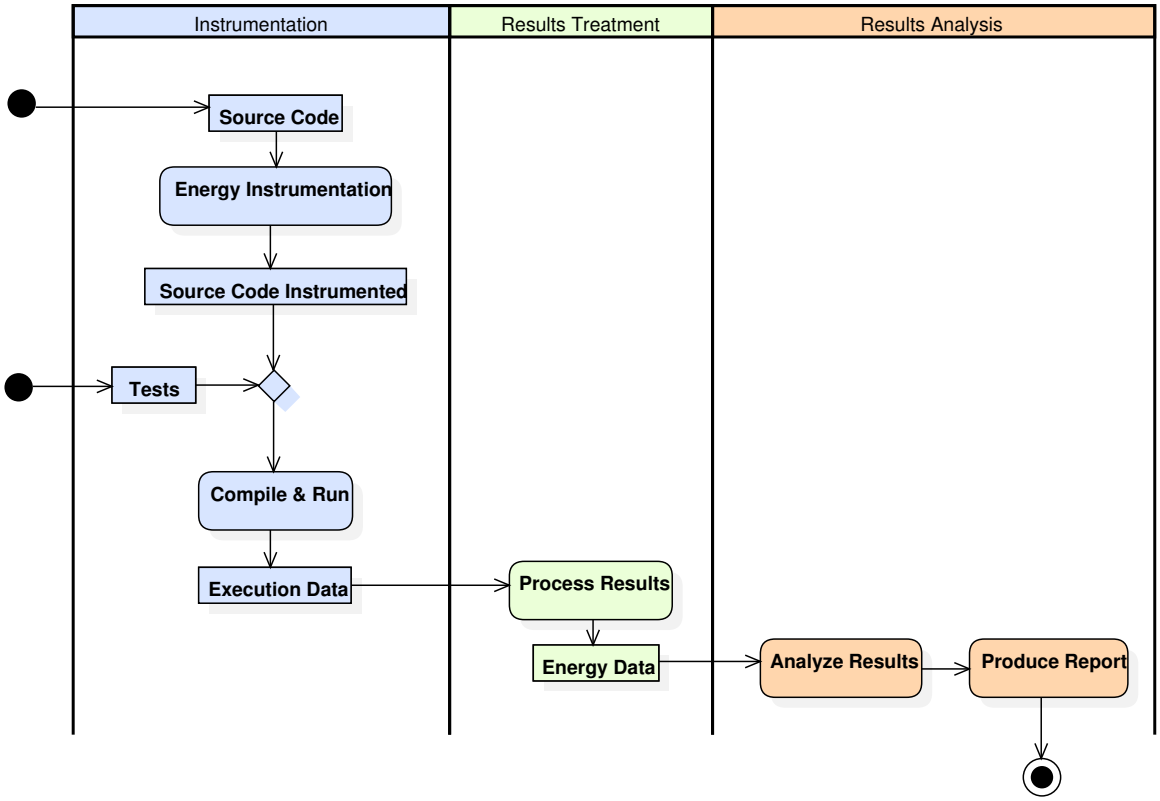


Figure 10: *Activity Diagram illustrating the Software Energy Analysis Technique*

5. Instrumentation and Results Treatment

Throughout the rest of this dissertation we will be diving into each phase of the methodology defined in Section 4. In this section the first phase will be addressed.

5.1. Instrumentation

As seen in Section 3.1, when one wants to identify the spectrum of a program implementation it must to specify the level on where the analysis will be performed. Depending on the programming language where the target program was developed, this granularity may vary. On one hand, in the C language, one can have Libraries > Files > Functions > Block of code > Line of code, on the other hand, in Java one can have Packages > Classes > Methods > Block code > Line of code, and in other languages there will also exist other components. Consequently, for each language will always be necessary to define the desired granularity.

After having defined the level of source code that one wants to retrieve the information, one also has to define the desired information to collect with the instrumentation. The process final goal is to analyze the program's energy consumption and therefore the logic data to be gathered is the information related with the energy consumption of the computer hardware components. As examples of hardware components there are the CPU, cache, DRAM, disk, fans, graphics card, motherboard, and other machine specific peripherals, and the program specific components (for example, the use of the mouse in a specific program). To complement this process, there is information that can be useful to retrieve conclusions about the profiling of energy: execution time, CPU frequency, CPU temperature, etc..

With the level of source code granularity and the information to collect chosen the next step is to perform the instrumentation. To do the instrumentation one can start by write by hand on the source code the instructions to collect the data after the execution, but this is an inefficient, long and not scalable process. So, in order to obtain an automated instrumentation, a structure that represents the program and can be modified to contain the collecting instructions must be defined. The use of such structure is a technique that modern compilers already use in their compiling processes and is called Abstract Syntax

Tree (AST). The AST represents the constituents of a program in a hierarchical manner (Figure 11). This structure allows changes by using operations without having concerns about the syntax structure of the source code file. This operations can be: add, remove or update nodes.

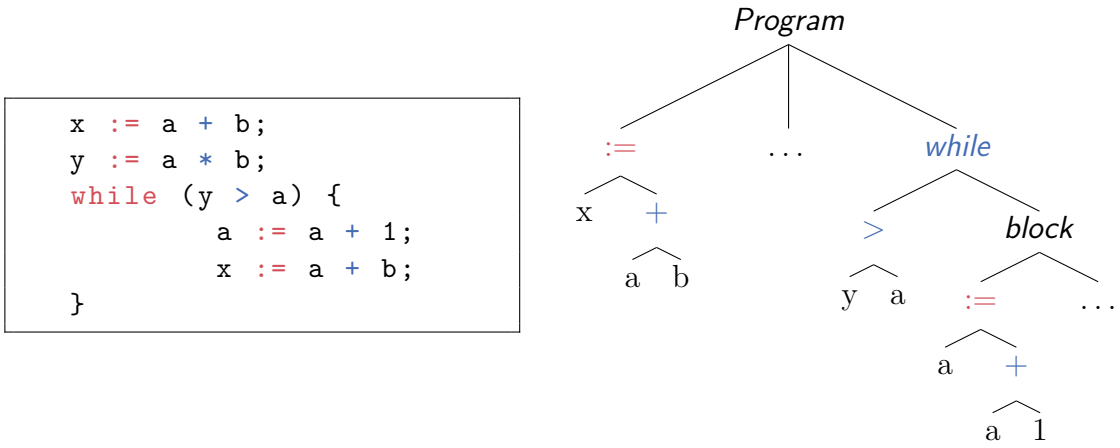


Figure 11: *A program with its Abstract Syntax Tree*

Having to collect energy information among other information, one needs to establish a source of this information. The source of this information can vary. It can be an external device that measures the overall energy consumption, a set of system calls that allow greater precision or even a pre-defined model. To the instrumentation here defined it is assumed that there is a framework that allows to accurately measure the power consumption within a certain range (depending on the granularity level). So, to define this range, reading and printing information nodes are both added before and after the granularity level content. An generic AST instrumented example can be seen in Figure 12. The syntax of this information

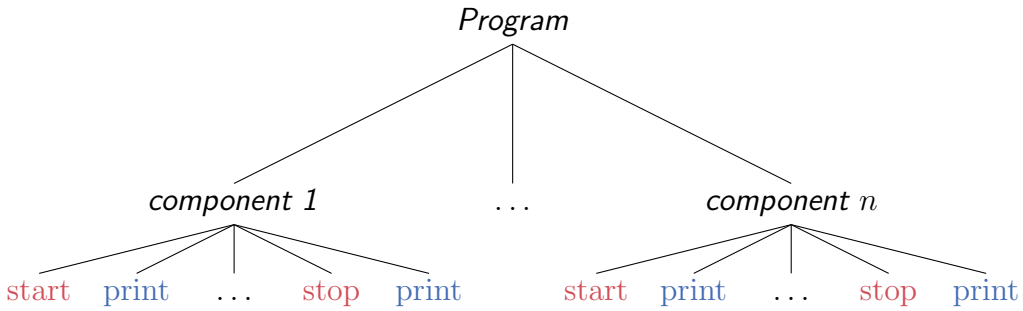


Figure 12: *Generic AST instrumented with nodes to extract energy information*

generated from all components has to be produced in a formatted way because it will serve as input in the next phase (Section 5.2).

After the instrumentation on the AST is made, the software source code has to be compiled, but now containing the needed instructions to collect the energy usage. After the compilation, the compiled program must be ran with a set of different inputs (test suit), that will test the program code. The more diverse and complete in terms of coverage of the program are these tests, the better analysis of the information extracted from the software implementation can be made.

5.1.1. Case Study: GraphViz

As a initial proof of concept and to apply the instrumentation in a robust application fully established, it was decided to choose a tool that had heavy processes and did some intensive processing to generate its output. The chosen tool was an open-source tool and is called GraphViz[1]. GraphViz is a software package that enables the design of graphs which then processes and generates the corresponding view. The Algorithm 1 (Section 7) was applied to the software package with about 18 tests. These tests ran GraphViz with different generation flags and different input graphs. In Figures 13 and 14 one can see the results (for the sake of visualization, some functions and tests are omitted).

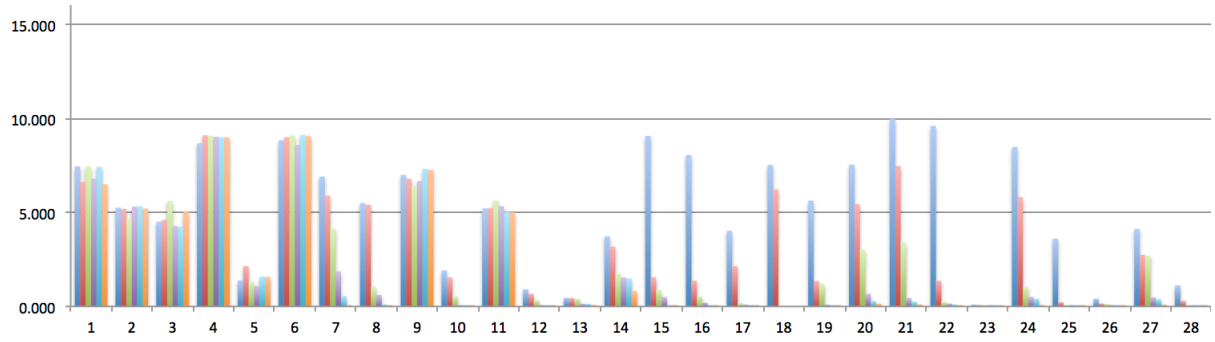


Figure 13: *Energy consumption of Graphviz functions*

These graphs show that different inputs and different flags have different energy consumption values which by itself is and indicator that an analysis can be made with different

[1] `www.graphviz.org`

tests to extract energy usage information. This was one of the first results that motivated further research.

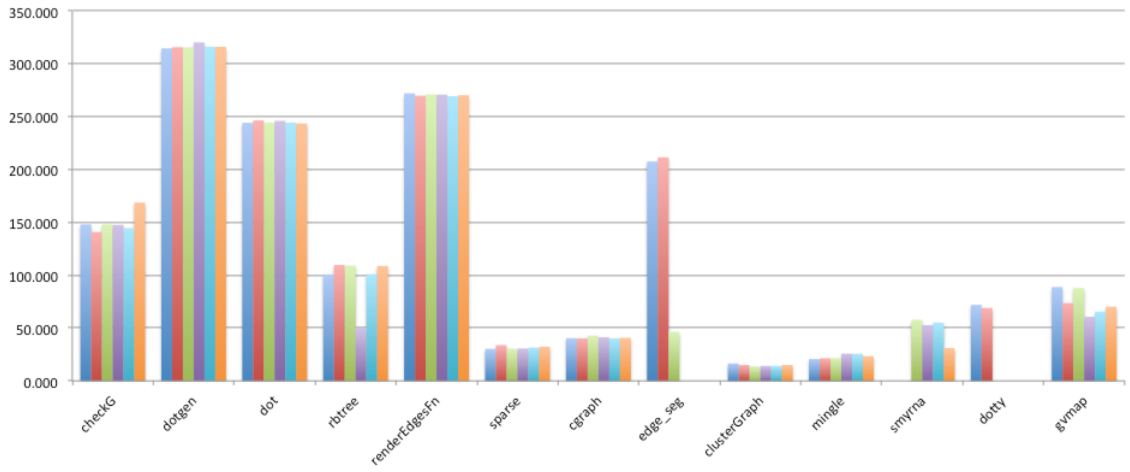


Figure 14: *Energy consumption of Graphviz modules*

5.1.2. The Influence of CPU Execution on the Energy Consumption Values

During the instrumentation and data collecting of the GraphViz application several tests were made. During this tests it was discovered that for some functions, the consumption values would increase in about 1000%. In a quick checkout to discover what was happening it was evident that something went wrong, and it was not a bad design of the function code. What was discovered was that when the program was executing, if the processor was working on one particular function that demanded a large computational resources, the processor would be put at 100% of its capabilities. When the CPU is running at full power it consumes more energy. Therefore the functions processed by the CPU when it was working at the maximum level had higher consumption values. The fact that this was happening had an impact on other functions besides the resource demanding ones. The functions that lead to this suspicion in fact, were being influenced because when the processor ended processing the resource demanding functions and started processing other functions, it was still working at high level when this was probably not needed.

To try to obtain more information about this situation, it was made some research. The demanding resources functions were identified and a new instrumentation process was

made. In this new instrumentation and besides the instructions to collect the energy usage, a instruction to force the process to pause was added. After compiling this new version and execute it again, the new data was collected. The results of this instrumentation were somewhat positive but not conclusive:

- 19% of the functions that were firstly influenced had their consumption back on the normal values
- 15% of the functions that were firstly influenced increased their overall consumption values
- in the other cases there was no influence.

Because of this new instrumentation, the time that the program took to execute the input obviously increased but the energy consumption values were the same as before because the energy usage was not being tracked while the program was paused. This first results seems promising and would require more investigation. A more profound investigation on trying to find a win-win situation in the execution time and the energy consumption levels should be made. As this goes out of the scope of this thesis and due to time limits this was not explored any further and was addressed to future work.

5.2. Results Treatment

The results produced by the execution of the program instrumented and compiled, are not structured which difficult the task of extracting knowledge about each component's energy consumption. Therefore there is a need to build a structure that holds the information hierarchically so it can allow easy transformations to be made and immediate information calculation for each component.

The input to this phase is the output from the instrumented program execution. This output is written in a flat and sequencial form, representing the order that the components were used in the program. So, whatever is the language of the program instrumented, the output form will be in the same format for all languages and paradigms, which makes this phase independent of any programming language or paradigm.

So the previous phase output must be in a standard and defined manner so it can be parsed and treated in this phase. Therefore one needs to define a standard format for the input that the instrumentation output must follow. This representation must ensure the following format to be a valid input in this phase:

$$
\begin{aligned}
&\ldots \\
&\textit{begin} \text{ component } n \quad \textbf{where} \quad \text{time} = x, \text{CPU} = y, \text{DRAM} = z, \ldots \\
&\ldots \\
&\textit{end} \text{ component } n \quad \textbf{where} \quad \text{time} = x, \text{CPU} = y, \text{DRAM} = z, \ldots \\
&\ldots
\end{aligned}
$$

Having the input in a standard representation one can process this data and construct the structure needed to treat the information. This structure, and because the execution information is a hierarchy information (execution path), the representation chosen was a n-ary tree where the nodes represent the components identified and are characterized by the execution information.

In this new representation, each node contains information about the energy consumption as well as the time consumed and the number of times it was performed (Figure 15).

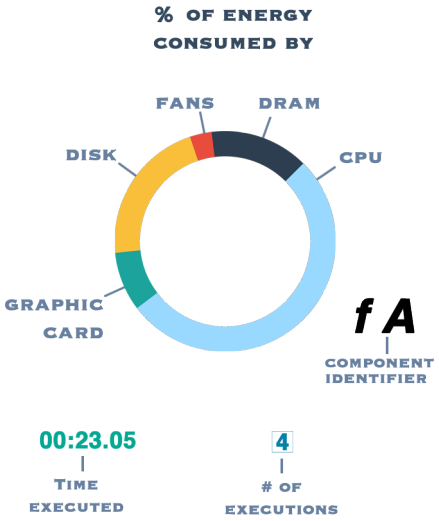


Figure 15: *Collected data node's information*

A graphical representation of the complete structure of all nodes can be seen in Figure 16. For each test run a related tree instance is created. The next step of the analysis of the software energy usage needs the program spectrum (instead of just having the binary spectrum-hit, it has the number of times executed per program component), and the time and energy consumption. The next phase also needs the data in a formatted form and with this tree structure, producing this information becomes trivial. For the n tests, and for each node in the tree a transformation to feed the next phase will be made. This transformation includes for each node aggregate all of its information in the tree and produce a row with all the components and its aggregated information.

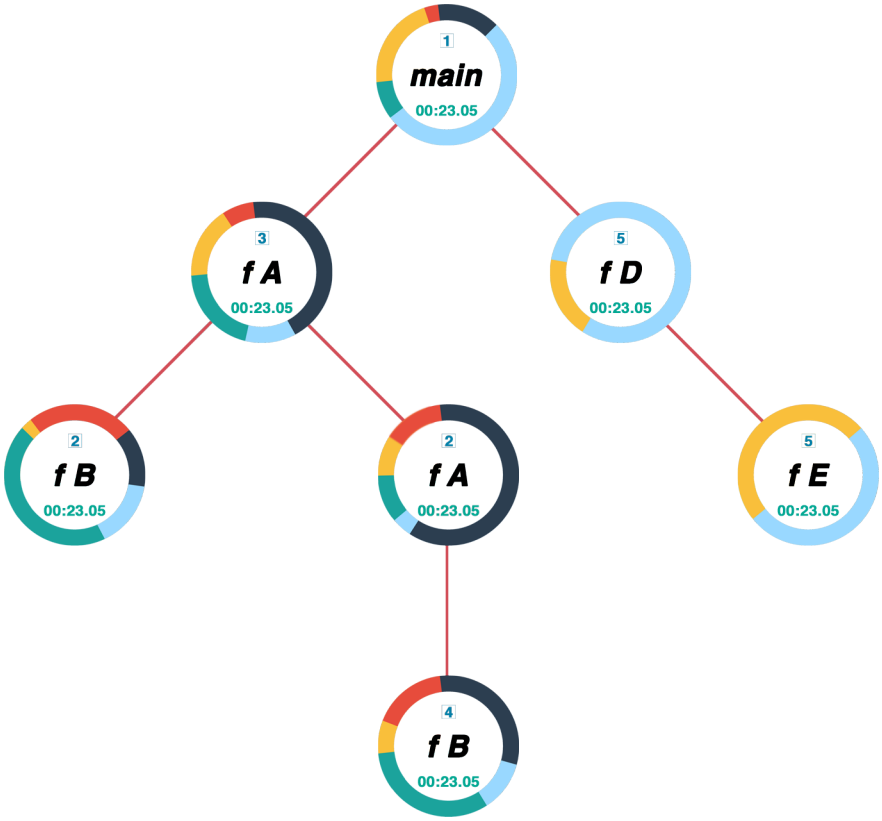


Figure 16: *An example tree of a test's data collected structure*

6. Results analysis: Spectrum-based Energy Leak Localization

The technique presented here, Spectrum-based Energy Leak Localization (SELL), is a technique that is independent of the programming language which means it is a generic technique and therefore be applied to different paradigms.

As seen in Section 3.1 and as the name so indicates, the SFL is based on the program execution hit-spectrum. In the technique developed throughout the thesis and presented here, a part of the knowledge used is also the spectrum of the program's execution. This spectrum allows the component to be discriminated whether if it was used or not and in the case where it has been used, extract more information about its execution. As in the SFL, the tests are also independent, i.e., the execution order of the tests is irrelevant because the state of a test does not affect the execution of another test. However, contrary to what the SFL defines, where there is an oracle to which we can ask questions about the validity of the output obtained by running a test, this analysis does not receives as input an oracle. This can be explained because, with regard to energy consumption, if one hand, there is still no known oracle to answer with 100% certainty to what is in excess energy consumption, on the other hand, what can really be seen as an excess of energy consumption? Therefore, the oracle is not a tool that can be obtained as an input.

Aside from the difference in the use of an oracle provided in the input, the technique presented here has important and complementary information to the spectrum of the execution that SFL does not need. This information can and is used as a way to obtain a more useful and complete analysis about the energy consumption of the program's components.

The input of this tool is a matrix A that has n lines which correspond to the number of the run tests and has m columns that are the m program's components (defined at the granularity level of the instrumentation) (Figure 17).

Each matrix element, λ_{mn}, if used in test n, contains information about the component m execution or contains no information when component m not used.

$$
\begin{array}{c} \\ n \text{ spectra} \end{array}
\begin{array}{c} \overset{m \text{ components}}{} \\ \begin{bmatrix} \lambda_{11} & \lambda_{12} & \cdots & \lambda_{1m} \\ \lambda_{21} & \lambda_{22} & \cdots & \lambda_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & \lambda_{nm} \end{bmatrix} \end{array}
$$

Figure 17: *The spectrum-based energy leak localization input matrix (A)*

$$
\lambda_{ij} = \begin{cases} \Big(E_{\text{nergy}}, T_{\text{execution}}, N_{\#}\Big)_{ij} & \text{if } c_j \text{ was used} \\ \\ \emptyset & \text{if } c_j \text{ was not used} \end{cases}
$$

This component execution data is segmented by 3 categories: energy consumption, execution duration and number of times executed. In the energy consumption category, values of energy of different hardware components are present: cpu (E_{cpu}), DRAM memory (E_{DRAM}), fans (E_{fans}), hard drive (E_{disk}) and graphic card (GPU) (E_{GPU}).

$$
E_{\text{nergy}_{ij}} = \Big(E_{\text{cpu}}, E_{\text{DRAM}}, E_{\text{fans}}, E_{\text{disk}}, E_{\text{GPU}}\Big)_{ij}
$$

All hardware components that consume energy may have its component represented on this data, but on this thesis we defined this components because they are representative of the differences between computers. All the consumption values are expressed in the unit of energy: *milliJoule* (mJ)[2]. The component's execution duration is represented in the attribute $T_{\text{execution}}$, this attribute is expressed in milliseconds. Finally, information about the number of executions (cardinality) is defined in $N_{\#}$ and is dimensionless.

With the matrix that contains, for each test, the execution information of each program component, the next steps are the processing and analysis of this information.

[2]There are two point of views in the energy consumption: the energy consumption and power consumption. While the energy consumption is the total energy consumed during a period of time and is measured in Joules (J), defined in the SI, the energy consumption is the energy consumed per unit time (J/s), or as is defined in the SI, Watts (W). The energy consumption indicates, for a given component, the total energy consumed which is the desired information when we want to extract information from an analysis on where we can make some changes that improve instantly the energy performance of the program's execution. On the other hand, the use of power consumption is useful in a situation where we want to realize when a component has a comparatively greater amount of energy usage per time; information that can be used to extract energy consumption patterns, which is set for future work.

As the SFL uses an oracle to check the validity of a test, the ideal would be an oracle adapted to the context of energy to better understand the values of the execution data. Thus, since we can not get the oracle, the first phase will be to build one that can be used. For the oracle construction there are several options. The first thought was to make a simple metric to calculate the average energy consumption of the program in all the tests and the oracle would determine if the test consumption was above average would be recorded as a failure and otherwise it was considered as a pass. However, this technique has some limitations, as well as the average energy consumption could hide significant statistical differences, one would be ignoring the other execution information such as execution time and the number of times the component was involved in the test.

Another possibility for this oracle would be building a base of prior execution consumption knowledge and use various programs to feed this knowledge base. The knowledge base can be segmented by type of software (image processing, graphs, etc.) and could be a correspondence between patterns of software execution and energy consumption. However, despite many positive points, the construction of this knowledge base would need a lot of different programs and for each one it would be necessary to catalog its execution pattern and the respective consumption. Another con is that the oracle would not obtain independence of the input tests for which the patterns were identified and that might differ between the testing and the pattern.

Thus, the solution that was defined for the oracle creation was premised on the fact that it had to be relative to the program implementation and use all available information to extract the best knowledge. Another point to consider is that while in the SFL, the oracle decides with a binary criterion (fail, pass) a test execution, here the criterion has to be a continuous value to represent the factor of a test *greenness*. Taking the example of what is usually done in the regulation of greenhouse gas emissions of the world's countries, where after assessing how much is the total emission of gases in the different years, depending on what each country contributed in gas emissions in those years, assigns the percentage of responsibility to each country. In this analysis one can try to establish an analogy, where the years are the different tests, the countries are the different components with the total for each category (energy, execution time and cardinality), and the goal is to try to assign

responsibilities to each component comparing with the total value.

To construct the oracle is then necessary, for each test, sum up all the values of the categories creating a total element. Because this is done for the n tests at the end we will get a vector here called t (total).

$$\begin{matrix} \\ \\ \\ \\ \end{matrix} \quad t$$

$$\begin{bmatrix} \sum_{i=1}^{m} \lambda_{1i} \\ \sum_{i=1}^{m} \lambda_{2i} \\ \vdots \\ \sum_{i=1}^{m} \lambda_{ni} \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}$$

With the oracle defined, the final model that this thesis introduces is totally defined and can be seen in Figure 18. It is based on this model that the remaining process of analysis will focus and is where from the knowledge will be extracted.

$$\begin{matrix} & m \text{ components} & & & & t \\ n \text{ spectra} & \begin{bmatrix} \lambda_{11} & \lambda_{12} & \cdots & \lambda_{1m} \\ \lambda_{21} & \lambda_{22} & \cdots & \lambda_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & \lambda_{nm} \end{bmatrix} & & \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \end{matrix}$$

Figure 18: *The spectrum-based energy leak localization input matrix (A)*

With the complete model defined, it is time to, and following the analogy of gas emissions, relate the data of each component with the total data. In the final, one wants to obtain a simple structure that makes the relationship between the similarity of any component (c_i)

and the total vector (t).

$$\begin{matrix} c_i \\ \\ \begin{bmatrix} \lambda_{1i} \\ \lambda_{2i} \\ \vdots \\ \lambda_{ni} \end{bmatrix} \end{matrix} \quad \overset{?}{\approx} \quad \begin{matrix} t \\ \\ \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \end{matrix}$$

The similarity between component i and the total vector t can be seen as how much component i is responsible for each execution information of the total vector. This association has as domain the current model and data, and therefore does not depend on prior knowledge, and is independent of other software, allowing conclusions regarding the software developed. Thus, it eliminates the dangers that could be introduced by comparing a program consumption with the consumption of other programs, since energy consumption is relative and it is totally dependent on what is the purpose of the program execution. As it would be expected, the smaller the number of components smaller is the components sample which influences the extracted similarity for each component. The quality of the test suit is also important because only with tests that provide global coverage and test the program for different inputs, one can hope to extract interesting information.

To obtain the component similarity with the oracle vector, there was a need to define a function that received the vector of a component and the total vector, and returned a structure with the similarity (α) for each of the constituints of component's execution information:

$$\text{similarity}\left(\begin{bmatrix} \lambda_{1i} \\ \lambda_{2i} \\ \vdots \\ \lambda_{ni} \end{bmatrix} , \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \right) = \phi_i$$

where,

$$\phi_i = \Big(\alpha(E_{\text{nergy}}), \alpha(T_{\text{execution}}), \alpha(N_{\#})\Big)_i$$

The chosen formula to calculate the similarity coefficient for each of the component's constituents, was the Jaccard similarity coefficient [Real and Vargas, 1996]. This formula, with two vectors $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ and where $x_i, y_i \geq 0$, calculates the similarity coefficient using the following formula:

$$J(x, y) = \frac{\sum_{i=1}^{n} \min(x_i, y_i)}{\sum_{i=1}^{n} \max(x_i, y_i)}$$

The Jaccard similarity coefficient is a well known formula to calculate the similarity coefficient between two vectors and has been used for a long period of time in the mathematics domain.

With the application of this similarity function to all components of the matrix, the result will be a row vector that represents for each component and for each execution information their influence in the overall context. As already mentioned, this vector contains the similarity of each execution information for each component, which allows the similarity analysis to be made focusing the desired execution information. So, defining a sort criteria and sorting the similarity vector allows to realize which are the components that are closer to representing the totality of execution information. Thus, and relating to the sorting criteria, one can be realize what are the possible failures at energy level of the program.

$$\text{sortBy}\big(\phi, E_{\text{cpu}}, T_{\text{execution}}\big) = \big[\ldots\big]$$

With this similarity execution information of each component one can make one parameterized analysis, however, and complementary it would also be useful to have a value that translated all of the execution information. This value would allow a numerical and global comparison between the different components. This analysis would do the sorting of all components, where the components with highest value were likely to be faulty at energy level. To make the conversion, a function that translates execution information in a numeric value was developed. The function created, aims to convert the information available in a value, this value is dimensionless and therefore is not directly related to any of the units of information used. To obtain the desired value, one needs to multiply all the execution

Table 1: *Average power consumption for each hardware component*

Component name	**Power consumption (average) (W)**	**Formula factor**
CPU	102.5	0.34
DRAM	3.75	0.01
Fans	3.3	0.01
Hard Drive	7.5	0.02
GPU	187.5	0.62

information from different category, summing the information from the same category. The decision to multiply all categories have to do with the fact that it makes the final value grows depending on the proportion that that category adds, the higher the value of the category the higher is the proportion that it increases the overall value. Regarding information within the same category, they have a summative contribution within the category, and will influence in proportion the global value.

$$\text{globalValue}(\lambda_i) = E_{nergy_i} \times T_{execution_i} \times N_{\#_i}$$

In the energy category, there are different types of results on the hardware components' energy consumption. These hardware components have an usual power consumption values and it varies from component to component. Therefore, it makes sense that these energy information is standardized according to the naturalness of those components produce more power. This normalization allows for example two hardware components A and B with the same energy consumption, wherein the component A in mean consumes more energy than B, that B has its consumption value diluted. The Table 1 explains the average power consumption for each component[3] and the factor that it will have on the formula. This factor of a hardware component k is calculated using the following formula:

$$\text{factor}_k = \frac{\text{power}_k}{\sum_{i=1}^{n} \text{power}_i}$$

where $power_k$ represents the average power consumption of the hardware component k, and n is the number of hardware components available.

So, with the data from the Table 1 one can produce the following formula to calculate the energy category of global value:

[3] `http://www.buildcomputers.net/power-consumption-of-pc-components.html`

$$E_{\text{nergy}_{ij}} = 0.34 \times E_{\text{CPU}_{ij}} + 0.01 \times E_{\text{DRAM}_{ij}} + 0.01 \times E_{\text{fans}_{ij}} + 0.02 \times E_{\text{disk}_{ij}} + 0.62 \times E_{\text{GPU}_{ij}}$$

With this informations the full model and its operations are specified. In the following subsection a example using this technique is given.

6.1. An example

To understand how this analysis works and see how the analysis handles the execution data, an example can be studied.

Lets think of a program that could be written in any language known. This program has four different components (functions in C, modules in C, methods in Java, etc.) and is ran with a test suit of five different inputs. This program has previously been through the first two phases of the process described in the Section 4, and its energy, execution time and usage has been identified. Therefore, we can use the information of this program's execution and start the analysis. In Table 2 we can see the entire model of the SELL Analysis already defined but lets construct it step by step.

Firstly we have the input data. This data is the data seen the Table 2 where for each component and each test we have a triple of three categories. This triple contains the energy consumption value, the number of times that component was used and the consumption time:

$$\begin{pmatrix} E_{total} \\ N_{\#} \\ T_{exec} \end{pmatrix}$$

So, in Table 2 we can check all the data from the program's execution in the given tests.

Having this inputs, and as defined in SELL, we have to build the oracle (t vector). To do so, for each test, we sum all the values of each category of the component data. After doing this for every test we have built the oracle vector. The following step its to calculate each component's category similarity. To achieve this we apply for each component category vector and the oracle vector the Jaccard's coefficient similarity formula. For example, for c_1, and for the energy category similarity coefficient we will have the following formula:

$$E_{\text{similarity coefficient}} = \frac{min(37{,}140)+min(38{,}166)+min(36{,}129)+min(37{,}164)+min(39{,}209)}{max(37{,}140)+max(38{,}166)+max(36{,}129)+max(37{,}164)+max(39{,}209)} = 0.2314$$

Table 2: *SELL Matrix built for the example program*

test	c_1	c_2	c_3	c_4	t
1	$\begin{pmatrix}37\\1\\75\end{pmatrix}$	$\begin{pmatrix}61\\2\\102\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}42\\1\\34\end{pmatrix}$	$\begin{pmatrix}140\\4\\211\end{pmatrix}$
2	$\begin{pmatrix}38\\3\\77\end{pmatrix}$	$\begin{pmatrix}50\\1\\103\end{pmatrix}$	$\begin{pmatrix}34\\2\\42\end{pmatrix}$	$\begin{pmatrix}44\\1\\37\end{pmatrix}$	$\begin{pmatrix}166\\7\\259\end{pmatrix}$
3	$\begin{pmatrix}36\\1\\73\end{pmatrix}$	$\begin{pmatrix}58\\1\\102\end{pmatrix}$	$\begin{pmatrix}35\\1\\43\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}129\\3\\218\end{pmatrix}$
4	$\begin{pmatrix}37\\3\\74\end{pmatrix}$	$\begin{pmatrix}66\\2\\105\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\end{pmatrix}$	$\begin{pmatrix}61\\2\\43\end{pmatrix}$	$\begin{pmatrix}164\\7\\222\end{pmatrix}$
5	$\begin{pmatrix}39\\2\\75\end{pmatrix}$	$\begin{pmatrix}54\\3\\100\end{pmatrix}$	$\begin{pmatrix}51\\4\\60\end{pmatrix}$	$\begin{pmatrix}65\\2\\60\end{pmatrix}$	$\begin{pmatrix}209\\11\\295\end{pmatrix}$
similarity by component's category	$\begin{pmatrix}0.2314\\0.3125\\0.3104\end{pmatrix}$	$\begin{pmatrix}0.3577\\0.2813\\0.4249\end{pmatrix}$	$\begin{pmatrix}0.1485\\0.2188\\0.1203\end{pmatrix}$	$\begin{pmatrix}0.2623\\0.1875\\0.1444\end{pmatrix}$	
global similarity	0.020	0.0373	0.0116	0.0112	

The calculation for the other categories and the different components would be the same, and its results can also be consulted in Table 2 in the similarity by component's category row.

To finish the gathering and calculation of all the values needed to make the energy leak analysis in the program, we must calculate the global similarity of each component. To do so, we must apply the formula defined in the prior Section and calculate for each test and each component its global value. After this, we also must calculate the global value for each oracle test value. After this we calculate the similarity between the component's and the oracle's global value vector.

In Table 3 we can see, for the component c_1 and the oracle, its global value vector.

Table 3: *Component c_1 and oracle global value vector*

c_1	t
943.5	40174.4
2984.52	102325.72
893.52	28684.44
2792.76	86651.04
1989	230589.7

Using the Jaccard's coefficient similarity formula we can obtain the following similarity coefficient: 0.019661758. Doing this calculations for every component of the program the global value similarity coefficient can be consulted in Table 2.

Now that we have all the needed information to analyze we can extract some information. Reading the global similarity coefficient value we can see which component has the highest probability of have a energy leak. Sorting the components for this metric we obtain the following configuration: c_2, c_1, c_3 and finnaly c_4. This means that if I was a developer of this application I should consider looking first in the component c_2 to better improve the energy consumption of the program. The advantage of the SELL technique is that it can tell besides the global value, why the component is faulty. For example, c_2 is calculated as the most probable component to have a energy leak, why? Well, if we look into its categories similarity values we will see that this component ranks 1^{st} in the energy similarity value, 2^{nd} in the cardinality similarity value and 1^{st} in the execution time similarity. This ranks clearly points to this component. Also, there are some curious facts that can be seen in this analysis. For example c_4 has an energy category similarity value higher than the c_1, although and due to the other categories its ranked 4^{th} in the overall. Other curiousity is that c_3 has in one of the tests an higher value of energy consumed that any of the c_1 energy values retrieved, however and because we take into consideration multiple tests, c_3 is ranked 3^{rd} in the overall, when c_1 is ranked 2^{nd}. Other curious facts could be found and explained but, and to compare this analysis over a technique using only the energy consumption values, another fact will be given. If we calculate the components average energy consumption values we would obtain:

$$c_1 = 37.4,\ c_2 = 57.8,\ c_3 = 24,\ c_4 = 42.4$$

what would indicate the following ranking: c_2, c_4, c_1 and finnaly c_3. This rank is completely different from the obtained in the SELL Analysis because it ignores the other components influence. To prove that this technique produces some true conclusions in the following Section a validation of the technique will be done.

7. Spectrum-based Energy Leak Localization: The Tool

Throughout the thesis development, all the phases already identified in Section 5 and in 6 were materialized in a tool. This tool integrates the different phases developed and each phase was implemented as an individual module because that would allow them to be used in different contexts. One could use this modules, for example, in the CROSS platform[4]. In this portal, one could create a certification to represent the whole process of this tool and it would be automatically linked. A possible certification for the whole process using the modules built in this thesis is represented in Figure 19[5].

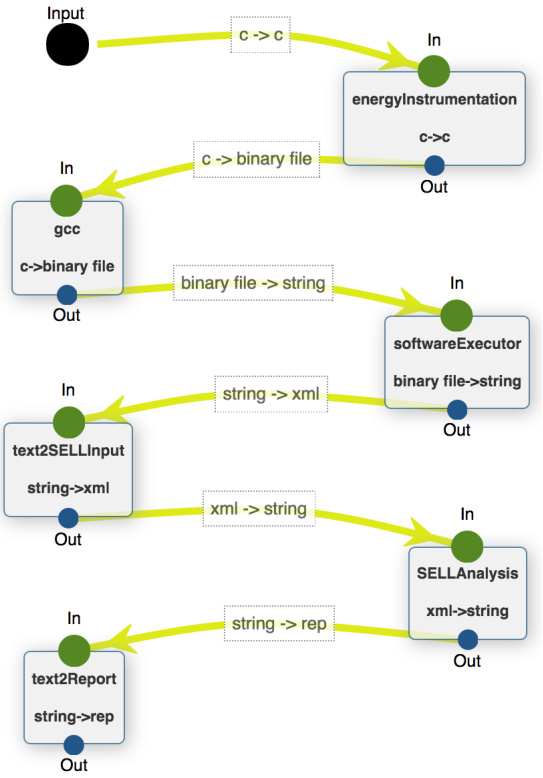


Figure 19: *A visual certification that represents the composition of the different modules to create a full process.*

The tool is composed by multiple components as can be seen in Figure 20.

In the following sections each of the components will be explained in detail.

[4]A platform that allows the construction of certifications to analyze open source software using different tools. This tools could be the modules here developed

[5]This visual language here seen was defined by Carção and Martins [2014] and is presented in [Carção and Martins, 2014]

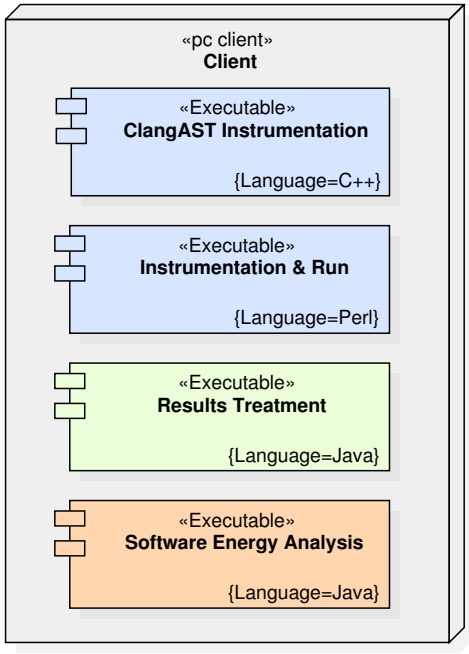


Figure 20: *Deployment Diagram of the tool developed*

7.1. The Instrumentation

The process of instrumentation was conceived to instrument programs written in C language, which, by doing so, allows the analysis of the energy consumption of C programs. This language was chosen due to that is a language well established in the community and allows access to a good number of repositories of robust open source software to be tested in terms of energy.

To do this instrumentation it was necessary to find an instrumentation tool that allows the extraction of the C language AST from a program. The natural language choice was a fairly complete tool that serves as a C front-end for the LLVM compiler and that among the many features already available, can build the program's AST from a file, and it is called the Clang[6] framework.

The accuracy chosen for the analysis was defined at the level of function which means that to retrieve the information required for the analysis one needs to know where is the beginning and the end of the function. The granularity choice is related with the precision

[6] `http://clang.llvm.org/`

one wants to extract information and analysis and in this case is also limited to the tools that exist and its accuracy.

The chosen framework to collect the energy data required for posterior instrumentation and analysis was the Intel Power Gadget framework. This framework works based on the framework RAPL and provides information on energy consumption and performance of the CPU. To measure the execution time it is used the Time library in C.

After having defined the parameters for the instrumentation, one can execute the instrumentation itself. The source code is processed and the AST is constructed. Then for all the functions we descend to the function level in the AST and add instructions to measure energy and time consumption and an instruction to print this information (Figure 21). The print information added was defined be in conforms with the data input of the following phase (grammar shown in Appendix A). It was developed a small program in C++ linked with Clang that allows to build the AST, add the nodes and regenerate the source code (Appendix D).

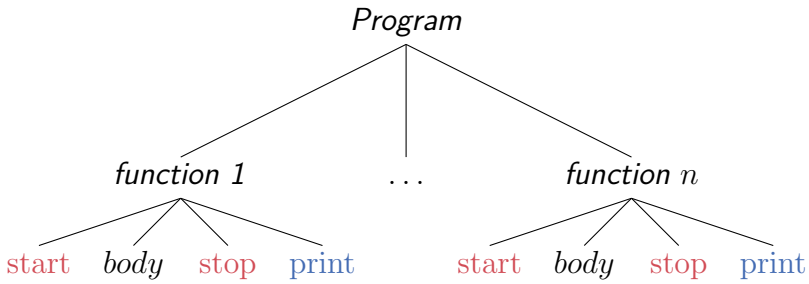


Figure 21: *AST of a program with a function granularity, each function is instrumented to extract energy information*

At the end of the instrumentation, one generates again the program's source code and compiles it.

Listing 2: *Generic instrumented C program with information to log energy consumption*

```
void function() {
    startMeasuring (Regist information)

    /* PROGRAM EXECUTION BEHAVIOR */

    endMeasuring (Display information)
}
```

The compiled program is then executed with a test suit and, for each test it produces the information about its energy consumption. This information will serve as input in the next phase, the results treatment.

For programs' instrumentation, compilation and the test suit execution, a Perl script (Appendix C) was developed. The script's algorithm is shown in Algorithm 1.

Algorithm 1 C program energy instrumentation and execution of set of tests

```
1:  procedure INSTRUMENTATE AND EXECUTE TESTS
2:
3:      for all module in software do
4:          programOutput ← clangEnergyInstrumentation(module)
5:          softwareInstrumented ← softwareInstrumented + moduleInstrumented
6:
7:      softwareCompiled ← compile(softwareInstrumented)
8:
9:      for all module in software do
10:         output ← execute(softwareCompiled)
11:         instrumentationOutput ← instrumentationOutput + output
12:
13:     return instrumentationOutput.
```

7.2. The Results Treatment

This phase receives as input the collected data from a program execution previously instrumented. To make this process cohere the input must be in a standard format. This format was defined with a grammar and is presented in Figure 22.

$$
\begin{aligned}
&\langle \textit{Input} \rangle && \longrightarrow \langle \textit{Data} \rangle^* \\
&\langle \textit{Data} \rangle && \longrightarrow \langle \textit{Component-begin} \rangle \, \langle \textit{Data} \rangle^* \, \langle \textit{Component-end} \rangle \\
&\langle \textit{Component-begin} \rangle && \longrightarrow \text{`>'} \, \langle \textit{Component} \rangle \\
&\langle \textit{Component-end} \rangle && \longrightarrow \text{`<'} \, \langle \textit{Component} \rangle \\
&\langle \textit{Component} \rangle && \longrightarrow \text{ID `('} \, \langle \textit{Params} \rangle \, \text{`)'} \\
&\langle \textit{Params} \rangle && \longrightarrow \langle \textit{Param} \rangle \, (\text{`,'} \, \langle \textit{Param} \rangle)^* \\
&\langle \textit{Param} \rangle && \longrightarrow \text{`time' `='} \, \text{NUMBER} \\
& && \quad | \; \text{`cpu' `='} \, \text{NUMBER} \\
& && \quad | \; \text{`dram' `='} \, \text{NUMBER} \\
& && \quad | \; \text{`gpu' `='} \, \text{NUMBER} \\
& && \quad | \; \text{`fans' `='} \, \text{NUMBER} \\
& && \quad | \; \text{`disk' `='} \, \text{NUMBER}
\end{aligned}
$$

Figure 22: *Input grammar of the results treatment phase*

A sample of the input of a program execution previously instrumented can be seen in Listing 3.

Listing 3: *Example of an input for the Results Treatment phase*

```
> main [ time = ..., cpu = ..., dram = ... ]
> fA [ time = ..., cpu = ..., dram = ... ]
> fB [ time = ..., cpu = ..., dram = ... ]
< fB [ time = ..., cpu = ..., dram = ... ]
> fA [ time = ..., cpu = ..., dram = ..., ]
> fB [ time = ..., cpu = ..., dram = ... ]
< fB [ time = ..., cpu = ..., dram = ... ]
< fA [ time = ..., cpu = ..., dram = ... ]
< fA [ time = ..., cpu = ..., dram = ... ]
> fD [ time = ..., cpu = ..., dram = ... ]
> fE [ time = ..., cpu = ..., dram = ... ]
< fE [ time = ..., cpu = ..., dram = ... ]
< fD [ time = ..., cpu = ..., dram = ... ]
< main [ time = ..., cpu = ..., dram = ... ]
```

This module was developed in Java, and so, this grammar was defined using the ANTLR framework (when dealing with Java is one of the most used frameworks to deal with grammars) and can be consulted in Appendix A. The semantic rules of this grammar were used to transform the collected data into a n-ary tree. The architecture of this module can be seen in Figure 23.

With the n-ary tree that represents the execution path of each of the program's component build, various operations are performed on the tree. Each component can appear multiple times as a node on the tree, so, its information is aggregated and refreshed. Traversing the tree and having done this for all of the components, the information aggregated is ready to be produced to the next phase. As this phase's input must be in a standard format, also the next phase's input must be in a standard format. This format is defined in the grammar of the Appendix B.

7.3. Results Analysis

The last module created was the component that made the full analysis. Because this phase is also a separated tool it must receive the input in a standard defined format. To define this format a grammar was created and can be seen in Figure 24. The development

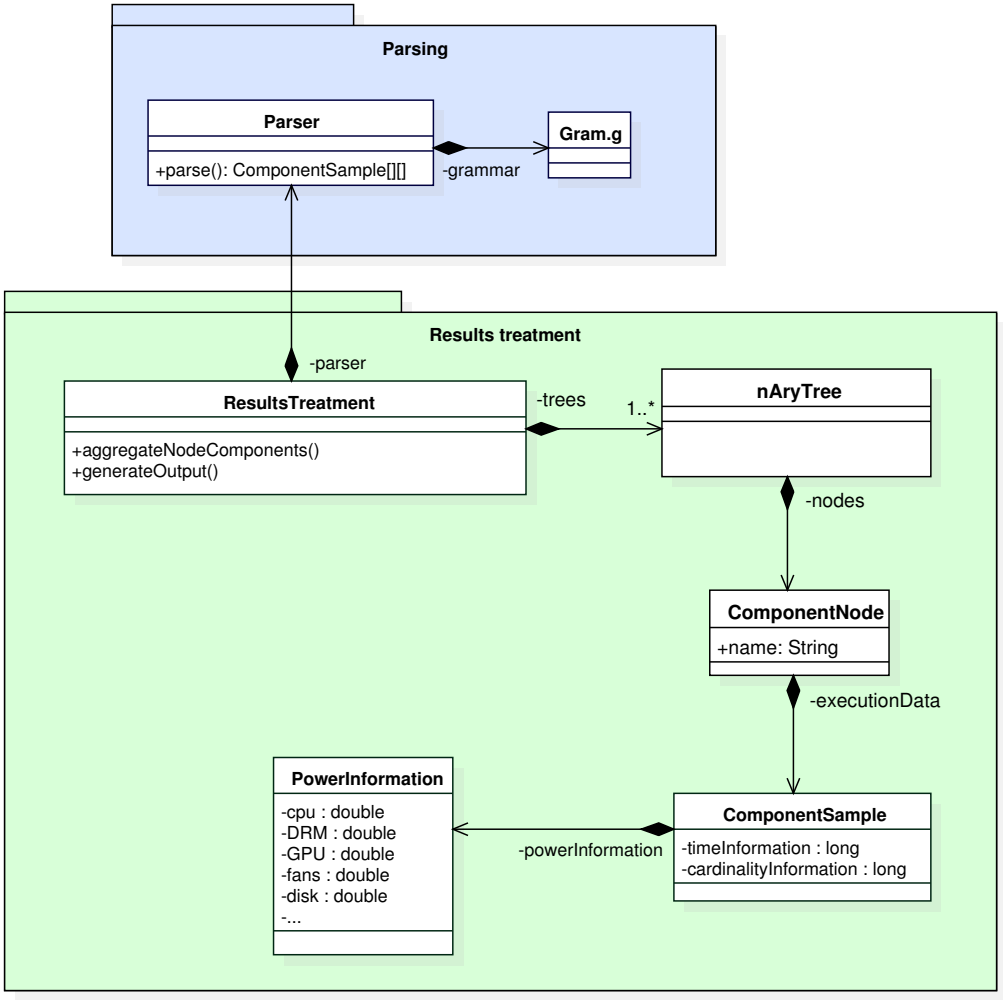


Figure 23: *Architecture of the Results Treatment Module*

language of this module was also Java and therefore the grammar was created using the ANTLR framework and can be consulted in the Appendix B.

As this module implements the SELL Analysis it must represent different concepts. Each of the concepts and its correlation is represented in the Table 4.

In the tool, there is a main class (SELLAnalysis) that is the center of this process. It has the model information and the operations to this model (defined in Section 6, i.e., calculate oracle, sortBy criteria and compute global value).

The architecture of this module is represented in the Class Diagram presented in the Figure 25.

$$
\begin{array}{lll}
\langle \textit{Matrix} \rangle & \longrightarrow & \langle \textit{Row} \rangle^* \\
\langle \textit{Row} \rangle & \longrightarrow & \langle \textit{Component-Sample} \rangle^* \\
\langle \textit{Component-Sample} \rangle & \longrightarrow & \text{`[' } \langle \textit{Params} \rangle \text{ `]'} \\
 & | & \text{`\_'} \\
\langle \textit{Params} \rangle & \longrightarrow & \langle \textit{Param} \rangle \text{ (`,' } \langle \textit{Param} \rangle)^* \\
\langle \textit{Param} \rangle & \longrightarrow & \texttt{`time'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`numberUsed'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`cpu'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`dram'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`gpu'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`fans'} \text{ `='} \text{ NUMBER} \\
 & | & \texttt{`disk'} \text{ `='} \text{ NUMBER}
\end{array}
$$

Figure 24: *Input grammar of the results analysis phase*

7.4. How to use the tool

To use the tool developed we first have to download it. This tool can be obtained in the url `url`. The package contains all of three modules and a README file. These modules must be ran separately or linked all together in a tool like CROSS portal. The README file gives instructions on how to run each of the modules and the pre-requisites to run them.

Table 4: *Correlation between the SELL concepts and its implementation in the tool*

SELL concept	**Implemented as**
Component's power information	**Class:** $PowerInformation$
Component	**Class:** $ComponentSample$
Matrix of components	**Instance Variable:** $ComponentSample[][]$
Oracle	**Instance Variable:** $ComponentSample[]$
Formula to calculate the similarity coefficient	**Class:** $SimilarityFormula$
Formula to apply the calculation of the similarity between the component and the oracle	**Class:** $ComponentSimilarityStrategy$
Component similarity	**Class:** $ComponentSimilarity$
An array of components' similarity	**Instance Variable:** $ComponentSimilarity[]$
Global value of component's	**Class:** $TotalValueComponent$
An array of components' global value	**Instance Variable:** $TotalValueComponent[]$

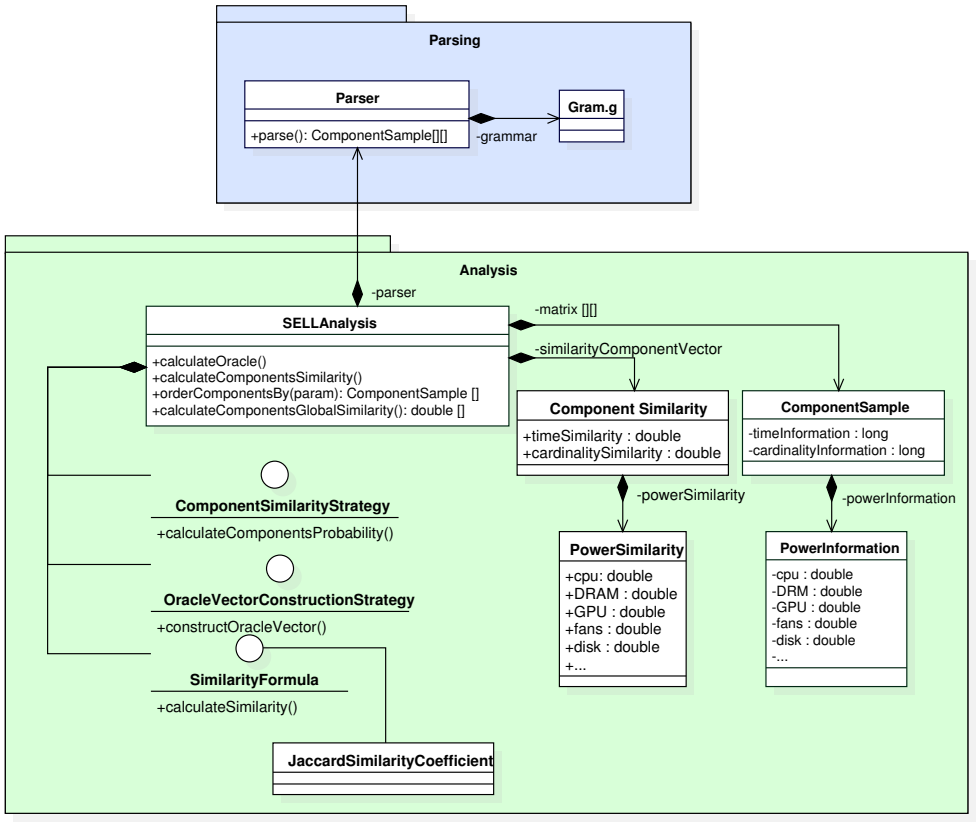


Figure 25: *Architecture of the System*

8. Validation

After defined a methodology to analyze the source code energy consumption, implemented that methodology along with a own technique to identify energy leaks a validation on the technique proposed was needed.

The chosen validation was to try to obtain the same results that other researches already accomplished and validated. By doing so, and if the technique here defined can also reach the same conclusions, then one could affirm conclusively that the technique works and is trustworthy.

In the Section 2 we already saw that Gutiérrez et al. [2014] made a research on how much the Java collections consumed and made a energy consumption rank of those collections, identifying the collections that perform best on energy usage. To build this rank they ran an well-known benchmark[7] and measured the energy consumption of the benchmark with the different collections. Their conclusions were in the form of how much times each collection when replaced by other collection had better/worse results. Given the times that a collection when replaced had better consumption values the following rank (worst to better) could be retrieved:

ConcurrentLinkedDeque
LinkedBlockingDeque
LinkedList
LinkedTransferQueue
ConcurrentLinkedQueue
ArrayList
PriorityQueue
CopyOnWriteArrayList
ConcurrentSkipListSet
TreeSet
CopyOnWriteArraySet
LinkedHashSet
HashSet

So, with the approach defined in this thesis we can also test the same benchmark and retrieve the conclusions. With the tool developed in this thesis, we wanted to test this

[7] http://java.dzone.com/articles/java-collection-performance

results. The problem was that the instrumentation module of the tool is constructed to C programs. So, to overcome this problem, we ran Java code from a proxy C program. This program instantiates a version of the Java Virtual Machine (JVM) and then uses it to run the desired code.

To apply the SELL technique we needed to define which are the components. As we want to know which collection has the highest probability of having energy leaks, the components here are the different collections. The test suit will feature the operations available in the benchmark (Table 5). For each operation of the benchmark a function was created a function. The energy is only measured after initializing the JVM, thus eliminating the energy usage of the initialization of the JVM. There is an excess of consumption for each method of the benchmark but is constant for every method therefore, in terms of methods comparison is negligible.

Table 5: *Operations performed in the benchmark for each collection*

Operations performed in the benchmark
add 100000 distinct elements
addAll 1000 times 1000 elements
clear
contains 1000 times
containsAll 5000 times
iterator 100000
remove 10000 elements given Object
removeAll 10 times 1000 elements
retainAll 10 times
toArray 5000 times

Using the tool the instrumentation and compilation, the results treatment and finally the SELL analysis were made to the benchmark. The analysis input and calculated similarities (as shown in Section 6) can be seen in Table 6.

Due to size constrains, the components are the rows and the tests (methods) are the columns[8]. The collections are ordered by the its global similarity value. So, by comparing the rank obtained by the SELL technique and the rank obtained by [Gutiérrez et al., 2014]

[8]Because each method was called once in the execution to simulate the test, the usage cardinality of each matrix element is allways 1

Table 6: *SELL Matrix built for the benchmark test. Collections are the components (rows) and the operations to the collections are the tests (columns).*

	add	**addAll**	**clear**	**contains**	**containsAll**	**iterator**	**remove**	**removeAll**	**retainAll**	**toArray**	**similarity by component's category**	**Global similarity**
LinkedBlockingDeque	796	614	918	1293	1241	1101	1387	1137	1247	1306	0.096	0.1160
	1	1	1	1	1	1	1	1	1	1	0.0769	
	770	636	4936	5120	5212	3782	5290	4144	5183	4086	0.0949	
ConcurrentLinkedDeque	709	550	1046	1394	945	1035	1257	1399	1086	1145	0.0919	0.1080
	1	1	1	1	1	1	1	1	1	1	0.0769	
	705	619	5007	4313	5300	3728	4985	4335	5123	3983	0.0923	
LinkedList	770	524	1048	926	996	811	1008	1075	1194	1400	0.0848	0.0935
	1	1	1	1	1	1	1	1	1	1	0.0769	
	694	817	4980	2961	4607	3198	4951	4373	4764	4370	0.087	
LinkedTransferQueue	760	722	1250	702	1226	920	1159	647	1226	989	0.0835	0.0881
	1	1	1	1	1	1	1	1	1	1	0.07692	
	787	815	4409	3219	4618	3142	5414	3513	4438	3987	0.0832	
ConcurrentLinkedQueue	741	700	1163	999	954	701	1164	810	1277	889	0.0817	0.0831
	1	1	1	1	1	1	1	1	1	1	0.0769	
	724	852	4391	3359	3838	3339	5352	3381	4396	3622	0.0806	
ArrayList	503	747	970	1024	1335	533	1106	728	961	746	0.0752	0.0776
	1	1	1	1	1	1	1	1	1	1	0.0769	
	430	2712	2785	3268	5431	1835	4442	4653	4906	2221	0.0792	
PriorityQueue	721	998	1134	908	514	1069	631	545	883	1182	0.0746	0.0761
	1	1	1	1	1	1	1	1	1	1	0.0769	
	3182	3906	4690	4503	481	3781	2449	1816	2596	4980	0.0785	
CopyOnWriteArrayList	827	1063	772	533	704	727	1089	1217	739	768	0.0734	0.0696
	1	1	1	1	1	1	1	1	1	1	0.0769	
	3678	4163	763	3121	3493	1965	4779	3866	3595	2363	0.077	
ConcurrentSkipListSet	625	865	1016	1043	687	1362	597	566	554	964	0.0720	0.0657
	1	1	1	1	1	1	1	1	1	1	0.0769	
	2977	3566	3603	3890	618	3881	2505	2208	2060	4319	0.0718	
TreeSet	787	909	591	935	199	1020	1070	741	705	981	0.069	0.065
	1	1	1	1	1	1	1	1	1	1	0.0769	
	2980	3790	382	4882	370	3887	3492	1995	2244	4826	0.0699	
CopyOnWriteArraySet	1307	975	685	550	708	725	1067	742	543	520	0.0680	0.0644
	1	1	1	1	1	1	1	1	1	1	0.0769	
	5160	5070	3055	2271	435	2229	5443	2054	2048	956	0.0696	
HashSet	738	622	1042	730	896	581	706	785	636	859	0.066	0.0482
	1	1	1	1	1	1	1	1	1	1	0.0769	
	2245	2794	3201	2001	2787	2314	2367	2214	2304	2874	0.0608	
LinkedHashSet	556	625	970	586	765	673	738	732	639	1073	0.064	0.0447
	1	1	1	1	1	1	1	1	1	1	0.0769	
	980	2397	3240	1160	1017	3184	1968	2729	2626	3771	0.0559	
Oracle	9840	9914	12605	11623	11170	11258	12979	11124	11690	12822		
	13	13	13	13	13	13	13	13	13	13		
	25312	32137	45442	44068	38207	40265	53437	41281	46283	46358		

(Table 7) we can see that 9 of 13 collections have the same rank and only four collections are misplaced.

From this four collections we can see that the difference in the collections that do not have

Table 7: *Rank from [Gutiérrez et al., 2014] on the left vs our analysis rank on the right*

ConcurrentLinkedDeque	*LinkedBlockingDeque*
LinkedBlockingDeque	*ConcurrentLinkedDeque*
LinkedList	*LinkedList*
LinkedTransferQueue	*LinkedTransferQueue*
ConcurrentLinkedQueue	*ConcurrentLinkedQueue*
ArrayList	*ArrayList*
PriorityQueue	*PriorityQueue*
CopyOnWriteArrayList	*CopyOnWriteArrayList*
ConcurrentSkipListSet	*ConcurrentSkipListSet*
TreeSet	*TreeSet*
CopyOnWriteArraySet	*CopyOnWriteArraySet*
LinkedHashSet	*HashSet*
HashSet	*LinkedHashSet*

the same rank, is one position. This collections have energy consumption similarity values and very alike (0.064 vs 0.066 and 0.096 vs 0.0919) as well as the global similarity values and therefore, a possible lack of precision on the energy measure may be the explanation here. Is also important to mention that this collections also have close values in the rank defined by [Gutiérrez et al., 2014]. Note that this differences only exist in one position and in any case misplace a supposedly energy leak free collection as a collection with a high probability of being faulty in energy terms. So with our analysis we came to very much the same conclusion of which Java collections were the better and the worse in terms of energy, which means that our solution works and may be used to identify energy leaks in the software with the capability of giving a extra report on why is that happening.

9. Conclusion

When I started doing this Thesis I faced an area that was still in a very early stage of development but had already great interest. The fact that Green Software Computing area is pretty new can be exciting because we can investigate and produce work in some areas that were never done or investigated, but on the other side, it also means that the tools available are not always complete and the community is still small.

As stated before, the tools available in this domain are scarce, and the developers actually do not have any tool that helps them to improve their software to be more energy efficient. Thus, the main goal of this Thesis was to propose a technique that could execute a software package, read its execution data and produce a report with the program's energy leaks. With the work already presented here we clearly have shown that this goal was accomplished.

Throughout the Thesis research and development multiple contributions were made. The main contributions of this Thesis are:

- A methodology to analyze a program's source code energy consumption

This methodology defines what are the steps to be taken in order to execute, read and analyze a program's energy usage.

- A software module that allows the instrumentation

This module allows the instrumentation of C programs to extract the execution data of the source code.

- A software module that processes the execution data

This module is independent of the program language and accept as input the program's execution data. It processes this data and aggregates the information by component.

- A software module that analyzes the execution data

This module (SELL) is also program language independent. It analyzes the program execution data and produces as output which are the energy leaks in the program.

- A tool that supports all the modules above mentioned.

9.1. Research Questions Answered

During my Thesis work three questions, that i presented in the beginning of this dissertation (Subsection 1.1 - Research Questions), appeared. This questions are related to the concept, the design and implementation of this Thesis. Now, and with the conclusion of my Thesis, I can answer these questions.

Q1 *Can we define a methodology to analyze the energy consumption of software source code?*

A1 Yes, in Section 4 - Source Code Energy Consumption Analysis we defined such methodology that we detailed in the Sections 5.1, 5 and 6.

Q2 *Is it possible to adapt a purpose fault localization algorithm to the context of energy consumption?*

A2 Yes it is. In Section 6 - Results analysis: Spectrum-based Energy Leak Localization we adapt and refine a fault localization technique to become a energy leak localization technique.

Q3 *Can we find energy leaks in software source code?*

A3 Indeed. After defined and implemented the SELL technique, Section 8 - Validation we demonstrated that in fact, we can identify the energy leaks in the software's source code.

9.2. Other contributions

Added to this Thesis contributions during its work I was involved in other works that gave fruit to three publications and one prize award:

- **A Visual DSL for the Certification of Open Source Software**, Tiago Carção, Pedro Martins. In the proceedings of the *International Conference on Computational Science and Its Applications (ICCSA'14)*, Guimarães, Portugal, June 30 - July 3, 2014.
- **Detecting Anomalous Energy Consumption in Android Applications**, Marco Couto, Tiago Carção, Jácome Cunha, João Paulo Fernandes, João Saraiva. In the

proceedings of the *Brazilian Symposium on Programming Languages (SBLP'14)*, Maceio, Brazil, October 2-3, 2014.

- **Measuring and visualizing energy consumption within software code**, Tiago Carção. In the proceedings of the *Visual Languages and Human-Centric Computing (VL/HCC'14)*, Melbourne, Victoria, Australia, July 28 - August 1, 2014.

- **Energy consumption detection in LabView**, Tiago Carção, Jácome Cunha, João Paulo Fernandes, Rui Pereira, João Saraiva. Grand prize ($2000) of a competition on innovating ideas applied to a specific software, awarded by National Instruments

9.3. Future Work

With the contributions of this Thesis as basis, some research should be made to further improve the tools available to help the development of software applications. This research can target the following topics:

- **Identify patterns of energy usage (bad smells)** - Having a tool to identify the energy leaks in a software program, one can run multiple software packages and identify some bad smells in terms of energy;

- **Propose some refactoring to those bad smells** - With the bad smells identified, multiple techniques to refactor them with a greener version can be researched;

- **Develop a visual tool to present the information collected** - With all of the information – the energy leaks, the bad smells and consequent refactors – we need to present this information. Thus, a visual tool, that can also be integrated in the IDEs, that implement these techniques can be developed. This tool would be a major contribution to the daily tasks of the software developer.

References

Abreu, R., Zoeteweij, P., Golsteijn, R., and van Gemund, A. J. C. (2009). A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792.

Abreu, R., Zoeteweij, P., and van Gemund, A. J. C. (2006). An evaluation of similarity coefficients for software fault localization. In *12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), 18-20 December, 2006, University of California, Riverside, USA*, pages 39–46.

Arnoldus, J., Gresnigt, J., Grosskop, K., and Visser, J. (2013). Energy-efficiency indicators for e-services. In *2nd International Workshop on Green and Sustainable Software, GREENS 2013, San Francisco, CA, USA, May 20, 2013*, pages 24–29.

Brownlee, J. (2013). OS X Mavericks Will Improve Your Battery Life By As Much As 4 Hours. `http://www.cultofmac.com/251135/os-x-mavericks-will-improve-your-battery-life-by-as-much-as-4-hours/`. Accessed: 2014-09-23.

Carção, T. and Martins, P. (2014). A visual DSL for the certification of open source software. In *Computational Science and Its Applications - ICCSA 2014 - 14th International Conference, Guimarães, Portugal, June 30 - July 3, 2014, Proceedings, Part V*, pages 602–617.

Couto, M. (2014). Monitoring Energy Consumption in Android Applications. Master's thesis, University of Minho.

Fanara, A., Haines, E., and Howard, A. (2009). The state of energy and performance benchmarking for enterprise servers. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pages 52–66.

Ferreira, M. A., Hoekstra, E., Merkus, B., Visser, B., and Visser, J. (2013). Seflab: A lab for measuring software energy footprints. In *2nd International Workshop on Green and Sustainable Software, GREENS 2013, San Francisco, CA, USA, May 20, 2013*, pages 30–37.

Google (2014). Better data centers through machine learning. `http://googleblog.blogspot.pt/2014/05/better-data-centers-through-machine.html`. Accessed: 2014-09-23.

Grosskop, K. (2013). PUE for end users - are you interested in more than bread toasting? *Softwaretechnik-Trends*, 33(2).

Grosskop, K. and Visser, J. (2013). Energy efficiency optimization of application software. *Advances in Computers*, 88:199–241.

Guelzim, T. and Obaidat, M. S. (2013). Chapter 8 - Green Computing and Communication Architecture. In Obaidat, M. S., Anpalagan, A., and Woungang, I., editors, *Handbook of Green Information and Communication Systems*, pages 209–227. Academic Press.

Gutiérrez, I. L. M., Pollock, L. L., and Clause, J. (2014). SEEDS: a software engineer's energy-optimization decision support framework. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 503–514.

Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using RAPL. *SIGMETRICS Performance Evaluation Review*, 40(3):13–17.

Harmon, R. R. and Auseklis, N. (2009). Sustainable IT services: Assessing the impact of green computing practices. pages 1707–1717. IEEE.

Harrold, M. J., Rothermel, G., Sayre, K., Wu, R., and Yi, L. (2000). An empirical investigation of the relationship between spectra differences and regression faults. *Softw. Test., Verif. Reliab.*, 10(3):171–194.

Hönig, T., Eibel, C., Schröder-Preikschat, W., Cassens, B., and Kapitza, R. (2013). Proactive energy-aware system software design with SEEP. *Softwaretechnik-Trends*, 33(2).

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice-Hall.

Korel, B. and Laski, J. W. (1988). Dynamic program slicing. *Inf. Process. Lett.*, 29(3):155–163.

Li, D., Hao, S., Halfond, W. G. J., and Govindan, R. (2013). Calculating source line level energy information for android applications. In *International Symposium on Software Testing and Analysis, ISSTA '13, Lugano, Switzerland, July 15-20, 2013*, pages 78–89.

Li, D., Jin, Y., Sahin, C., Clause, J., and Halfond, W. G. J. (2014). Integrated energy-directed test suite optimization. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pages 339–350.

Mouftah, H. T. and Kantarci, B. (2013). Chapter 11 - Energy-Efficient Cloud Computing: A Green Migration of Traditional {IT}. In Obaidat, M. S., Anpalagan, A., and Woungang, I., editors, *Handbook of Green Information and Communication Systems*, pages 295–330. Academic Press.

Noureddine, A., Rouvoy, R., and Seinturier, L. (2014). Unit testing of energy consumption of software libraries. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1200–1205.

Pinto, G., Castor, F., and Liu, Y. D. (2014). Mining questions about software energy consumption. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 22–31.

Real, R. and Vargas, J. M. (1996). The probabilistic basis of jaccard's index of similarity. *Systematic biology*, pages 380–385.

Reps, T. W., Ball, T., Das, M., and Larus, J. R. (1997). The use of program profiling for software maintenance with applications to the year 2000 problem. In *Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th*

ACM SIGSOFT Symposium on Foundations of Software Engineering, Zurich, Switzerland, September 22-25, 1997, Proceedings, pages 432–449.

Ricciardi, S., Palmieri, F., Torres-Viñals, J., Martino, B. D., Santos-Boada, G., and Solé-Pareta, J. (2013). Chapter 10 - Green Data center Infrastructures in the Cloud Computing Era. In Obaidat, M. S., Anpalagan, A., and Woungang, I., editors, *Handbook of Green Information and Communication Systems*, pages 267–293. Academic Press.

Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. (2012). Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27.

Rühl, C., Appleby, P., Fennema, J., Naumov, A., and Schaffer, M. (2012). Economic development and the demand for energy: A historical perspective on the next 20 years. *Energy Policy*, 50:109–116.

Sahin, C., Cayci, F., Gutiérrez, I. L. M., Clause, J., Kiamilev, F. E., Pollock, L. L., and Winbladh, K. (2012). Initial explorations on design pattern energy usage. In *First International Workshop on Green and Sustainable Software, GREENS 2012, Zurich, Switzerland, June 3, 2012*, pages 55–61.

Sahin, C., Tornquist, P., McKenna, R., Pearson, Z., and Clause, J. (2011). How Does Code Obfuscation Impact Energy Usage? *conferences.computer.org*.

Standard, R. (2013). GHG Protocol Product Life Cycle Accounting and Reporting Standard ICT Sector Guidance. In *Greenhouse Gas Protocol*, number January, chapter 7 - Guide.

Symantec (2008a). Corporate responsibility report. `http://www.symantec.com/content/en/us/about/media/SYM_CR_Report.pdf`. Accessed: 2014-09-23.

Symantec (2008b). Environmental progress and next steps. Email to Everyone Symantec (Employees). Accessed: 2014-09-23.

Vásquez, M. L., Bavota, G., Bernal-Cárdenas, C., Oliveto, R., Penta, M. D., and Poshyvanyk, D. (2014). Mining energy-greedy API usage patterns in android apps: an empirical study. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 2–11.

Vereecken, W., Van Heddeghem, W., Colle, D., Pickavet, M., and Demeester, P. (2010). Overall ict footprint and green communication technologies. In *Communications, Control and Signal Processing (ISCCSP), 2010 4th International Symposium on*, pages 1–6.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2010, part of ESWeek '10 Sixth Embedded Systems Week, Scottsdale, AZ, USA, October 24-28, 2010*, pages 105–114.

Zhang, Y. and Ansari, N. (2013). Chapter 12 - Green Data Centers. In Obaidat, M. S., Anpalagan, A., and Woungang, I., editors, *Handbook of Green Information and Communication Systems*, pages 331–352. Academic Press.

Appendices

A. ANTLR Grammar of Results treatment

Listing 4: *"ANTLR Grammar of Results treatment"style*

```
grammar Output;

@lexer::header { package output; }
@parser::header {
  package output;
  import genericTree.*;
}

output returns [ Tree<ComponentNode> tree ]
        :       l1 = line (l2 = line { $tree.getRoot().addChild($l2.node);
| } )* { $tree = new Tree($l1.node); }                                      |
        ;

line returns [ Node<ComponentNode> node ]
        scope { ArrayList<Node<ComponentNode>> childList; }
        :       component_begin (l1 = line      {
        if ($line::childList == null)
        $line::childList = new ArrayList<Node<ComponentNode>>();

        $line::childList.add($l1.node);
        })* component_end
                {
                if ($component_begin.nodeBegin.getId().equals($component_en
                ComponentNode componentNode = new ComponentNode($component.
                $node = new Node<ComponentNode>(componentNode);

                        if ($line::childList != null) {
                                for(Node<ComponentNode> child : $line::chil
                                        $node.addChild(child);
                                }
                        }
                }
                }
        ;

component returns [ OutputNode outputNode ]
@init{ $outputNode = new OutputNode(); }
        :       ID '(' params ')' { $outputNode.setId($ID.text); $outputNod
```

```
        ;


component_begin returns [ OutputNode nodeBegin ]
        :       '>' component { $nodeBegin = $component.outputNode; }
        ;

component_end returns [ OutputNode nodeEnd ]
        :       '<' component { $nodeEnd = $component.outputNode; }
        ;

params returns [ OutputNodeParameters parameters ]
@init{ $parameters = new OutputNodeParameters(); }
         : p1 = param { $parameters.put($p1.key, $p1.value); } ( ',' p2 = p
         ;

param returns [ String key, String value ]
        : 'm' '=' ID { $key = "m"; $value = $ID.text; }
        | 't' '=' NUMBER { $key = "t"; $value = $NUMBER.text; }
        | 'e' '=' NUMBER { $key = "e"; $value = $NUMBER.text; }
        ;

ID
        :   IdentifierNondigit
            (   IdentifierNondigit
            |   DIGIT
            )*
        ;

NUMBER
        : DIGIT+ ('.' DIGIT+)?
        | '.' DIGIT+
        ;

fragment IdentifierNondigit
    :   Nondigit
    ;

fragment Nondigit : ('a'..'z'|'A'..'Z'|'_');


fragment DIGIT   : '0'..'9'+;
WS : (' ' | '\t' | '\n' | '\r' )+ {skip();};
```

B. ANTLR Grammar of SELL Analysis

Listing 5: *"ANTLR Grammar of SELL Analysis"*

```
grammar Matrix;

@lexer::header { package sell; }
@parser::header {
  package sell;
  import java.util.TreeMap;
  import sell.input.*;
}

parse returns [ DataRetrieved matrix ]
@init{ $matrix = new DataRetrieved(); }
 : (line { $matrix.addVector($line.vector); } )+ EOF
 ;

line returns [ DataLineRetrieved vector ]
@init{ $vector = new DataLineRetrieved(); }
 : (c1 = component { $vector.addComponent($c1.component); } )+ ('>'
    ↪  c2 = component { $vector.addTotal($c2.component); } )? ('|'
    ↪ c3 = component { $vector.addError($c3.component); } )? (NL |
    ↪ EOF)
 ;

component returns [ ComponentSample component ]
 : param
 | '[' params ']' { $component = new ComponentSample($params.
    ↪ parameters); }
 | '_' { $component = new ComponentSample(false); }
 | '0' { $component = new ComponentSample(false); }
 | '1' { $component = new ComponentSample(true); }
 ;

params returns [ ComponentParameters parameters ]
@init{ $parameters = new ComponentParameters(); }
 : p1 = param { $parameters.put($p1.key, $p1.value); } ( ',' p2 =
    ↪ param { $parameters.put($p2.key, $p2.value); } )*
 ;

param returns [ String key, String value ]
 : 'n' '=' NUMBER { $key = "n"; $value = $NUMBER.text; }
 | 't' '=' NUMBER { $key = "t"; $value = $NUMBER.text; }
 | 'e' '=' NUMBER { $key = "e"; $value = $NUMBER.text; }
 | 'cpu' '=' NUMBER { $key = "cpu"; $value = $NUMBER.text; }
 | 'ram' '=' NUMBER { $key = "ram"; $value = $NUMBER.text; }
```

```
 | 'disk' '=' NUMBER { $key = "disk"; $value = $NUMBER.text; }
 | 'fans' '=' NUMBER { $key = "fans"; $value = $NUMBER.text; }
 | 'gpu' '=' NUMBER { $key = "gpu"; $value = $NUMBER.text; }
 ;


NUMBER
 : DIGIT+ ('.' DIGIT+)?
 | '.' DIGIT+
 ;


fragment DIGIT   : '0'..'9'+;
NL    : '\r'? '\n' | '\r';
Space : (' ' | '\t')+ {skip();};
```

C. Perl script to apply instrumentation to every C module

Listing 6: *"Perl script to apply instrumentation to every C module"*

```
#!/usr/bin/perl -w

use strict;
use warnings;
use 5.010;
use Cwd 'abs_path';
use File::Copy;

my $path = shift || '.';

my $absoluteDirPath = abs_path($path);

my @folder_split = split('\/',$absoluteDirPath);
my $name_dir = $folder_split[-1];

my $target_dir =  $absoluteDirPath . "/../". $name_dir . "_copy";

$target_dir = abs_path($target_dir);

mkdir($target_dir) or die "Could␣not␣mkdir␣$target_dir:␣$!";

copy_recursively($absoluteDirPath, $target_dir);

traverse($absoluteDirPath);

sub traverse {
    my $thing = shift @_;
        my $filename = shift @_;

        #say $thing;
        if (defined($filename) and not -d $thing) {
                my $firstChar = substr($filename, 0, 1);
                my @stringSplitted = grep { /.+\.c$/ } $filename;

                if ((not $firstChar eq ".") and (scalar
                   ↪ @stringSplitted) > 0) {
                     #file it is not an hidden file and is a C
                        ↪ language file
                     my $absolutePath = abs_path($thing);
                     my $command = "/Users/tac/Dropbox/MEI/Tese/
                        ↪ CInstrumentation/CInstrumentation␣" .
                        ↪ $absolutePath;
                     system($command);
```

```
                }
        }

    return if not -d $thing;
    opendir my $dh, $thing or die;
    while (my $sub = readdir $dh) {
        next if $sub eq '.' or $sub eq '..';
                #say "$thing/$sub";
        traverse("$thing/$sub", $sub);
    }
    close $dh;
    return;
}

sub copy_recursively {
    my ($from_dir, $to_dir) = @_;
    opendir my($dhh), $from_dir or die "Could not open dir '
       ↪ $from_dir': $!";
    for my $entry (readdir $dhh) {
                next if ($entry eq '.' or $entry eq '..');
        my $source = "$from_dir/$entry";
        my $destination = "$to_dir/$entry";
        if (-d $source) {
            mkdir $destination or die "mkpath '$destination' failed
               ↪ : $!" if not -e $destination;
            copy_recursively($source, $destination);
        }
        else {
            copy($source, $destination) or die "copy failed: $!";
        }
    }
    closedir $dhh;
    return;
}
```

D. Clang C++ program to instrumentate a C program with energy instructions

Listing 7: *"Clang C++ program to instrumentate a C program with energy instructions"*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <vector>

#include "llvm/Support/Host.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/ADT/IntrusiveRefCntPtr.h"
#include "llvm/ADT/StringRef.h"
#include "llvm/Support/FileSystem.h"

#include "clang/Basic/DiagnosticOptions.h"
#include "clang/Frontend/TextDiagnosticPrinter.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Basic/TargetOptions.h"
#include "clang/Basic/TargetInfo.h"
#include "clang/Basic/FileManager.h"
#include "clang/Basic/SourceManager.h"
#include "clang/Lex/Preprocessor.h"
#include "clang/Lex/Lexer.h"
#include "clang/Basic/Diagnostic.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/AST/ASTConsumer.h"
#include "clang/Parse/ParseAST.h"
#include "clang/Rewrite/Frontend/Rewriters.h"
#include "clang/Rewrite/Core/Rewriter.h"

using namespace clang;

char functionName[256];

// RecursiveASTVisitor is is the big-kahuna visitor that traverses
// everything in the AST.
class MyRecursiveASTVisitor
    : public RecursiveASTVisitor<MyRecursiveASTVisitor>
{

 public:
  MyRecursiveASTVisitor(Rewriter &R) : Rewrite(R) { }
  void InstrumentStmt(Stmt *s);
  bool VisitStmt(Stmt *s);
  bool VisitFunctionDecl(FunctionDecl *f);
```

```
    bool VisitReturnStmt(ReturnStmt *Return);

  Rewriter &Rewrite;
};

// Return Statements
bool MyRecursiveASTVisitor::VisitReturnStmt(ReturnStmt *Return)
{
        //      Stmt *BODY = Return->getBody();
        SourceLocation ST = Return->getLocStart();
        char fc[256];
        sprintf(fc, "//Retrieve data %s\n", functionName);

        Rewrite.InsertText(ST, fc, true, true);

    return true; // returning false aborts the traversal
}

bool MyRecursiveASTVisitor::VisitFunctionDecl(FunctionDecl *f)
{
  if (f->hasBody())
  {
    SourceRange sr = f->getSourceRange();
    Stmt *s = f->getBody();

    // Make a stab at determining return type
    // Getting actual return type is trickier
    //QualType q = f->getReturnType();
    QualType q = f->getResultType();
    const Type *typ = q.getTypePtr();

    std::string ret;
    if (typ->isVoidType())
       ret = "void";
    else
    if (typ->isIntegerType())
       ret = "integer";
    else
    if (typ->isCharType())
       ret = "char";
    else
       ret = "Other";

    // Get name of function
    DeclarationNameInfo dni = f->getNameInfo();
    DeclarationName dn = dni.getName();
    std::string fname = dn.getAsString();
```

```
    // Point to start of function body
    SourceLocation ST = s->getLocStart().getLocWithOffset(1);

    // Add comment
    char fc[256];
    sprintf(fc, "\n// Retrieve data\n");
    sprintf(functionName, "%s", fname.data());
    Rewrite.InsertText(ST, fc, true, true);

    if (f->isMain())
      llvm::errs() << "";

    SourceLocation END = s->getLocEnd();
    Rewrite.InsertText(END, fc, true, true);
  }

  return true; // returning false aborts the traversal
}

class MyASTConsumer : public ASTConsumer
{
 public:

  MyASTConsumer(Rewriter &Rewrite) : rv(Rewrite) { }
  virtual bool HandleTopLevelDecl(DeclGroupRef d);

  MyRecursiveASTVisitor rv;
};


bool MyASTConsumer::HandleTopLevelDecl(DeclGroupRef d)
{
  typedef DeclGroupRef::iterator iter;

  for (iter b = d.begin(), e = d.end(); b != e; ++b)
  {
    rv.TraverseDecl(*b);
  }

  return true; // keep going
}


int main(int argc, char **argv)
{
  struct stat sb;
```

```
if (argc < 2)
{
   return 1;
}

// Get filename
std::string fileName(argv[argc - 1]);

// Make sure it exists
if (stat(fileName.c_str(), &sb) == -1)
{
  perror(fileName.c_str());
  exit(EXIT_FAILURE);
}

CompilerInstance compiler;
DiagnosticOptions diagnosticOptions;
compiler.createDiagnostics();

// Create an invocation that passes any flags to preprocessor
CompilerInvocation *Invocation = new CompilerInvocation;
CompilerInvocation::CreateFromArgs(*Invocation, argv + 1, argv +
   ↪ argc,
                                   compiler.getDiagnostics());
compiler.setInvocation(Invocation);

// Set default target triple
llvm::IntrusiveRefCntPtr<TargetOptions> pto( new TargetOptions())
   ↪ ;
pto->Triple = llvm::sys::getDefaultTargetTriple();
llvm::IntrusiveRefCntPtr<TargetInfo>
   pti(TargetInfo::CreateTargetInfo(compiler.getDiagnostics(),
                                   pto.getPtr()));
compiler.setTarget(pti.getPtr());

compiler.createFileManager();
compiler.createSourceManager(compiler.getFileManager());

HeaderSearchOptions &headerSearchOptions = compiler.
   ↪ getHeaderSearchOpts();

headerSearchOptions.AddPath("/usr/local/include",
        clang::frontend::Angled,
        false,
        false);
```

```
  headerSearchOptions.AddPath("/usr/local/opt/llvm/llvm/tools/
    ↪ clang/include",
                                clang::frontend::Angled,
                                false,
                                false);
headerSearchOptions.AddPath("/Applications/Xcode.app/Contents/
  ↪ Developer/Toolchains/XcodeDefault.xctoolchain/usr/include",
        clang::frontend::Angled,
        false,
        false);
headerSearchOptions.AddPath("/System/Library/Frameworks",
        clang::frontend::Angled,
        false,
        false);
headerSearchOptions.AddPath("/Library/Frameworks",
        clang::frontend::Angled,
        false,
        false);
headerSearchOptions.AddPath("/usr/include",
        clang::frontend::Angled,
        false,
        false);

// Allow C++ code to get rewritten
LangOptions langOpts;
langOpts.GNUMode = 1;
langOpts.CXXExceptions = 1;
langOpts.RTTI = 1;
langOpts.Bool = 1;
langOpts.CPlusPlus = 1;
Invocation->setLangDefaults(langOpts,
                            clang::IK_CXX,
                            clang::LangStandard::lang_cxx0x);

//compiler.createPreprocessor(clang::TU_Complete);
compiler.createPreprocessor();
compiler.getPreprocessorOpts().UsePredefines = false;

compiler.createASTContext();

// Initialize rewriter
Rewriter Rewrite;
Rewrite.setSourceMgr(compiler.getSourceManager(), compiler.
  ↪ getLangOpts());

const FileEntry *pFile = compiler.getFileManager().getFile(
  ↪ fileName);
```

```
compiler.getSourceManager().createMainFileID(pFile);
compiler.getDiagnosticClient().BeginSourceFile(compiler.
   ↪ getLangOpts(),
                                                  &compiler.
                                                     ↪ getPreprocessor
                                                     ↪ ());

MyASTConsumer astConsumer(Rewrite);


// Convert <file>.c to <file_out>.c
std::string outName (fileName);
size_t ext = outName.rfind(".");
if (ext == std::string::npos)
   ext = outName.length();
outName.insert(ext, "_out");

//llvm::errs() << "Output to: " << outName << "\n";
std::string OutErrorInfo;
llvm::raw_fd_ostream outFile(outName.c_str(), OutErrorInfo, llvm
   ↪ ::sys::fs::F_None);

if (OutErrorInfo.empty())
{
  // Parse the AST
  ParseAST(compiler.getPreprocessor(), &astConsumer, compiler.
     ↪ getASTContext());
  compiler.getDiagnosticClient().EndSourceFile();

  // Output #include
  outFile << "//#include␣<PowerGadgetTool.h>\n";

  // Now output rewritten source code
  const RewriteBuffer *RewriteBuf =
    Rewrite.getRewriteBufferFor(compiler.getSourceManager().
       ↪ getMainFileID());
  outFile << std::string(RewriteBuf->begin(), RewriteBuf->end());
}
else
{
  llvm::errs() << "Cannot␣open␣" << outName << "␣for␣writing\n";
}

outFile.close();

int result = rename(outName.c_str(), fileName.c_str());
```

```
  if (result != 0) {
          llvm::errs() << "Cannot␣save␣file␣as␣original\n";
  }

  return 0;
}
```