Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 27, Num. 3 (2020) 72-83

RESEARCH ARTICLE

Experiments on Model-Based Software Energy Consumption Analysis Involving Sorting Algorithms

Experimentos com Análise de Consumo de Energia Baseada em Modelos Envolvendo Algoritmos de Ordenação

Danilo S. Alves[1]*, Oseias A. Ferreira[1], Lucio M. Duarte[1], Davi Silva[2], Paulo H. Maia[2]

Abstract: Although energy has become an important aspect in software development, little support exists for creating energy-efficient programs. One reason for that is the lack of abstractions and tools to enable the analysis of relevant properties involving energy consumption. This paper presents the results of some experiments involving the gathering, modelling, and analysis of energy-related information, in particular, the costs of executing certain parts of a software. We combine some existing free and open-source tools to carry out the experiments, extending one of them to handle energy information. Our experiments consider a comparison of energy consumption of Java implementations of the Bubble Sort, Insertion Sort and Selection Sort algorithms using different data structures. We show how to combine an energy measurement tool and a model analysis tool to carry such a comparison. Based on this support and on our experiments, we believe this is a first step to allow developers to start creating more energy-efficient software.
Keywords: Energy consumption — Behaviour models — Sorting algorithms

Resumo: Embora o consumo de energia tenha se tornado um importante aspecto no desenvolvimento de software, existe pouco suporte para a criação de programas energeticamente eficientes. Uma razão para isto é a falta de abstrações e ferramentas que permitam a análise de relevantes propriedades relacionadas ao consumo de energia. Este artigo apresenta os resultados de experimentos envolvendo a coleta, modelagem e análise de informações sobre consumo de energia, em particular, o custo da execução de certas partes do software. Ferramentas gratuitas e de código aberto foram combinadas para realização dos experimentos, sendo que nós estendemos uma destas ferramentas para que ela trabalhasse com informações de energia. Nossos experimentos consideram a comparação do consumo de energia dos algoritmos de ordenação Bubble Sort, Insertion Sort e Selection Sort implementados com diferentes estruturas de dados. Demonstramos como combinar ferramentas de aferição de energia e de análise de modelos para realizar essa comparação. Com base no suporte utilizado e em nossos experimento, acreditamos que este é um primeiro passo para permitir que os desenvolvedores possam desenvolver software energeticamente eficiente.
Palavras-Chave: Consumo de energia — Modelos de comportamento — Algoritmos de ordenação

[1] *Institute of Informatics, Universidade Federal do Rio Grande do Sul, Brazil*
[2] *Distributed Software Engineering Group, Universidade Estadual do Ceará, Brazil*
***Corresponding author**: dsalves@inf.ufrgs.br
DOI: https://doi.org/10.22456/2175-2745.98904 • **Received:** 10/12/2019 • **Accepted:** 20/02/2020

1. Introduction

The number of developers concerned about the energy consumption of their software is growing over the last years. As software spreads to different types of platforms and devices - all of them requiring some amount of energy to execute the necessary computations -, this energy consumption becomes a big issue. Research has shown mobile applications that quickly drain battery energy tend to be rejected by users [1], indicating energy consumption to be a relevant aspect. Corporations have also come to the conclusion that small inefficiencies in software can significantly affect its operation [2]. For this reason, energy consumption is now an important factor during software development and evolution [3] [4].

An example of this is that, in 2010, the big clusters consumed 1.12% - 1.50% of the global energy consumption [5]. In [6], they show that, in 2010, 1.3% of all electricity in the world was consumed by data centers, being 2% of this total consumed by the United States, and there is an expected cost of $13 billions per year by 2020 [7]. Due to the high energy

consumption of data centers [8], some actions have been taken to advance energy efficiency, mainly in hardware and operating systems [9] [10]. Another possibility is to perform power optimisation in software, which is very difficult to happen if developers do not analyse the energy costs while - or before - they are devloping their systems [11].

It is our belief that energy will continue to increase its importance when comes to software, becoming as important as it already is in hardware design. Hence, we might soon start comparing programs in terms of energy complexity as well as we now do with time and space complexity. As systems grow in scale and complexity, mostly composed by multiple components geographically spread, which rely on power supply to keep running, reducing energy consumption might become one of the most important factors in software design.

Despite the current, and possible future, importance of energy consumption analyses, there is still little support for designing energy-efficient software. In fact, developers find it still unclear how to produce and evolve energy-efficient software [2] [12] [13], mainly due to the absence of combined abstractions and tools to collect and analyse energy consumption. If such support existed, developers could not only identify the costs of executing their software, but also compare different versions in terms of energy consumption and determine possible changes to improve energy efficiency.

In this paper, we investigate a combination of tools to analyse software energy consumption, considering the framework proposed in [14]. This is a first step towards identifying whether and how current tools could provide the necessary support for developers to understand energy costs associated to their software. We have executed experiments involving Java implementations of sorting algorithms (Insertion Sort, Bubble Sort, and Selection Sort). We work with Java because it is a widely used programming language and also used for the development of web and mobile applications. Carrying out the experiments using sorting algorithms makes it easier for the reader to understand both the software and the experiments. Moreover, sorting is a basic operation in Computing and we use the experiments to analyse the impact of different algorithms and data structures in terms of energy costs.

As our objective is to identify how a developer could more easily obtain information related to energy consumption, we use free and open-source tools. Values of energy costs were collected using jRAPL [15] and software behaviour was modelled using Labelled Transitions Systems (LTS) [16]. We take advantage of the open-source characteristic of the LoTuS tool [17], which supports models described as LTS and analyses on these models, to extend this tool to enable modelling and analysis based on energy costs associated to code elements. LoTuS, in this extended version, allows a user to graphically construct the model and assign energy costs to its transitions, which makes it easier to build and analyse LTS models with energy information.

The main idea behind our experiments was to evaluate the difficulties and limitations of existing support, whilst demonstrating how it is possible to obtain energy-related information that can be useful to a software developer, if tools are correctly combined and/or extended. Our ultimate goal was to identify whether it is possible to provide the necessary support for developing energy-efficient software. This goal includes the possibility of providing recommendations concerning modifications in the code that could improve efficiency, without affecting the original semantics of the program.

This paper contains the following parts: Section 2 presents some background information; Section 3 describes the experiments and the tool support used; Section 4 discusses some related work; and Section 5 contains the conclusions and possible future work.

2. Background

In this section, we present some basic ideas related to energy consumption evaluation. Following the framework proposed in [14], we divide this background section in the steps described as necessary for a developer to analyse their software energy consumption.

2.1 Collecting Energy Information

Software energy efficiency has gained the attention of the research community [1] [2] [15] [18]. The first step to make any analysis about the energy costs of executing a software is to collect such information. Collecting energy information refers to executing the software using some method to obtain energy costs as the code runs.

There are some available tools to collect energy information about performed operations, which allow the measurement of energy costs associated to code locations. It can be done using tools such as Gem5 [19] and McPAT [20], or jRAPL [15]. Whereas McPAT and Gem5 use abstractions to obtain estimates of energy information, simulating an execution, jRAPL allows the annotation of source code with methods to collect energy information, so that the user can select which parts of the code should be monitored for energy usage. However, jRAPL only works with some types of architectures and just with Java programs.

Whatever tool is used, collecting energy consumption information involves either running some simulations or executing the program a number of times to obtain a more precise cost information. Moreover, developers still need to understand how such tools can be used to increase energy efficiency and how the identified hotspots affect the overall energy consumption of their software.

2.2 Modelling Energy

A behaviour model is an abstraction used to describe the expected behaviour of a system. Using behaviour models, several types of analyses can be carried out, such as validation and verification of properties. These analyses would be very difficult - if not impossible - to execute in the actual system. Moreover, behaviour models can also be used to analyse the impact of changes and provide software documentation.

Usually, the natural way of modelling behaviours is using finite-state machines. A finite-state machine (FSM) is formed by a set of (abstract) states $Q = \{q_0, q_1, ..., qn\}$ that represent sets of possible concrete states of a system, and a set of transitions connecting these states. The combination of states and their transitions indicate the set of valid behaviours. A behaviour is then a path in the FSM, starting in a given state and proceeding through a transition to a next state, then moving to next state, and so on, until the last state of the path is reached. FSM have solid mathematical foundations, which makes them suitable for analysis and automatic verification of properties of systems. Furthermore, they are an intuitive way of describing system behaviour with some level of abstraction, enabling its presentation in a visual way and the analysis of valid and invalid behaviours.

Having data about energy consumption, it can be inserted in a model to be analysed in some available tool. A widely used model based on FSM is Labelled Transition Systems (LTS) [16]. In an LTS, the behaviour of a system is described by the sequences of actions that it can execute, where actions are defined to the level of abstraction involved, and they can represent method calls, variable assignments, task completion, or some other significant event.

An LTS $M = (S, s_i, \Sigma, T)$ is then a model where:

- S is a finite set of states,
- $s_i \in S$ represents the initial state,
- Σ is an alphabet (set of action names), and
- $T \subseteq S \times \Sigma \times S$ is a transition relation.

Transitions are labelled with the names of actions from the alphabet that trigger a change from the origin state to a destination state. Therefore, given two states $s_0, s_1 \in S$ and an action $a \in \Sigma$, then a transition $s_0 \xrightarrow{a} s_1$ means that it is possible to go from state s_0 to state s_1 executing action a.

A *behaviour* of an LTS M is then a finite sequence of actions $\pi = \langle a_1, ..., a_n \rangle$ such that $a_1, ..., a_n \in \Sigma$. The set $L(M) = \{\pi_1, \pi_2, ...\}$ of all behaviours of M is called its *language*. For a state $s \in S$, $E(s) = \{a \in \Sigma | \exists s' \in S \cdot (s, a, s') \in T\}$ represents the finite set of actions *enabled* in s. A *path* $\lambda = \langle s_1, a_1, s_2, a_2, s_3, ... \rangle$ is a sequence of alternating states $s_1, s_2, ... \in S$ and actions $a_1, a_2, ... \in \Sigma$ labelling transitions connecting these states, such that, for $i \geq 1$, for every transition $t = (s_i, a, s_{i+1})$ composing λ, $t \in T$. A path always starts and - if finite - ends with a state.

To model energy costs using an LTS, we add cost values to transition labels, thus associating each cost to an action. In this context, therefore, an action can be any code element to which we would like to associate an energy cost. Hence, a path in this energy-labelled model is a sequence of code elements and their respective costs, where the total cost of a path is the sum of the costs of each element composing this path. Representing energy in a behaviour model enables the application of techniques to find out relevant information that can collaborate in understanding how software is consuming energy and which parts have been influencing the most. Furthermore, the behaviour model can be used for software documentation, which is important for new versions of systems and for checking how software evolution affects energy consumption.

2.3 Analysing Energy Consumption

Research such as presented in [21] is one example of work on energy consumption analysis. They focus on finding excessive or anomalous energy consumption in software and use a methodology to optimise Java programs and decrease their energy consumption replacing data structures for their more energy-efficient alternatives. However, they do not offer any type of analysis tool, leaving the analysis for the developer, based on the provided information.

In fact, energy-consumption analysis has still little support, what makes it difficult to produce and evolve systems with low energy costs. This happens, essentially, because of the absence of software abstractions and tools [2]. A way of analysing energy costs is through a behaviour model of the system, as discussed before.

The only available tool supporting some type of cost representation is PRISM [22]. It supports modelling systems as Markov chains, which is used to model software considering stochastic behaviour. Hence, states represent possible states of the system and transitions are labelled with probabilistic information, indicating the probability of a particular transition occur. PRISM also supports the definition of transition/state costs/rewards, which could be used to model energy costs. With this information, questions about accumulated energy costs can be asked using a probabilistic temporal logic and a probabilistic model checker. However, modelling is not visual and analyses using PRISM can only refer to states or accumulated values for a certain path in a Markov chain. Important questions regarding comparisons between different executions (i.e., paths) cannot be asked.

In [14] they propose a model-based framework for analysing software energy consumption through the verification of energy-based properties, thus providing a novel approach for the development and evolution of energy-efficient software. The work uses an LTS for representing the system behaviour and its energy costs. As an LTS is a graph-like structure, graph-based algorithms can be used to calculate accumulated costs of paths and determine the most/least costly path (i.e., software behaviour).

We have extended an LTS-analysis tool called LoTuS [17] to allow the inclusion of energy cost information as part of transition labels and implemented analysis algorithms to support answering relevant questions about software energy behaviour. LoTuS is an open-source tool that allows graphical modelling of software behaviour using LTS, providing a drag-and-drop GUI to create models.

3. Experiments and Results

In this section, we describe the methodology employed to perform the experimentation phase, the obtained results and a discussion about the experiments.

3.1 Methodology

The methodology applied to realise the experiments consisted on an instanciation of the framework proposed in [14], which is composed of four phases, described as follows:

- In the first phase, a behaviour model is created represented by an LTS. Model construction can be carried out manually, using the knowledge of the user about the software under analysis, or through a model extraction approach, such as demonstrated in [23]. In our experiments, we built the models by hand, based on the source code;
- The second phase consists in measuring the energy consumption in specific parts of the software under analysis. For this, we used the jRAPL library, which allows the measurement of energy consumption based on annotations in the source code. Using these annotations, it is possible to select which parts of the code are to be monitored for energy costs;
- At this point, it is possible to compose the software behaviour model with the energy information collected in the previous phase. This annotation of energy costs can be included automatically, i.e., using a script or an intermediate method that maps the source code to the behaviour model, or manually by the user after performing the measurements. At the end of this phase, we finally obtain an annotated LTS, where an energy cost is associated to each system transition. After this, we have basically a graph structure (states and transitions) with weights (energy costs) and graph-theory concepts and algorithms can be applied;
- Finally, with the energy-annotated model produced in the previous phase, it is possible to analyse some properties about energy consumption of the software, such as the ones proposed in [14].

The annotations in the source code were included as demonstrated in Algorithm 1 to make it possible to collect energy information. After selecting the parts of code that would be analysed, annotations were inserted before and after these parts to capture the values about the energy consumption at this point of the execution. After that, the energy cost of each annotated part would be the result of subtracting the values collected after executing that part from those values collected before the execution of the monitored part.Associating each annotated part to a model transition, we could add these values to the model.

For the experiments performed in this work, counters were included in each monitored part of the source code to obtain the average energy consumption value for these parts knowing the number of times that part was executed (see Algorithm 1. In case of control structures (loops and decisions) inside other control structures, the intern structure cost was subtracted from the external structure cost to obtain only the consumption of the analysed code transition and to represent this cost in the model correctly. With this, it was possible to travel in the behaviour model accumulating the costs to have the energy consumption of the whole path traversed.

3.2 Results

In this section, we describe the results of our experiments. We used the extended version of LoTuS to graphically specify the behaviour model, whereas energy-consumption measurement was obtained using jRAPL.

The experiments[1] consisted on analysing the impact of different storage structures in the Bubble Sort, Insertion Sort and Selection Sort algorithms, which were chosen because they consume significant energy values of energy when compared with other sorting algorithms, as demonstrated in [24]. Moreover, these algorithms have the worst complexity order among the most popular sorting algorithms. On top of all that, sorting algorithms are well-known and, thus, make it easier to understand the experiments and their results.

The algorithms were implemented to sort values stored in a `Vector`, in an `ArrayList` and in a `LinkedList`. They receive as input a parameter n, which consists of the number of elements that will be sorted, and a parameter *type*, which indicates the type of the data structure to be used. The values were stored in reverse order, to force the worst complexity case and maximise the energy consumption to obtain considerable results. The experiments were performed with n being 1500 and, as known, the sorting algorithms involved in this experiments have complexity $O(n^2)$, and added to the impact of energy measurement annotations, a long time of execution was required to collect significant results. To perform the experimental phase, the extended LoTuS tool has been used to include the possibility of representing costs in transitions and enable analyses about these costs, such as the cost of travelling a given path. The experiments were carried out using a PC with processor Core i5 3 GHz, with 16 GB of RAM running only the experiments and the Linux kernel base processes.

The LTS models presented in figures 1, 2 and 3 represent the behaviour models of Bubble Sort, Insertion Sort and Selection Sort algorithms, respectively. The energy values in the transitions consist of an average value from 10 executions of the respective algorithm and are represented in Watts (**W**). Highlighted transitions are the only ones with significant energy consumption values, which are related to methods `add()`, `get()` and `set()`. The remaining transitions represent variable declarations blocks, beginning of loop or decision structures, where the energy cost is deri-

[1]Data for these experiments can be found at ⟨https://github.com/VeriTesEnergyLab⟩

Algorithm 1: Code annotation example.

```
initialisation;
beforeA ← jRAPL_before(...);
for condition do
    forCounter++;
    [block code];
    beforeB ← jRAPL_before(...);
    if condition then
        ifCounter++;
        [block code];
    afterB ← jRAPL_after(...);
    consumptionB ← consumptionB + (afterB - beforeB);
afterA ← jRAPL_after(...);
consumptionA ← afterA - beforeA;
//Compute the difference between the measurements;
averageConsumptionB ← consumptionB ÷ ifCounter;
averageConsumptionA ← (consumptionA - consumptionB) ÷ forCounter;
```

sive and, for this reason, are not represented in the model. When a user informs an argument *type=1*, the algorithm executes the **opt1** path, which performs the sorting operations with a `Vector` structure; for *type=2*, **opt2** is executed, using an `ArrayList` structure; and the last option is *type=3* to execute **opt3** path, which consists in executing the sorting algorithm with a `LinkedList` structure.

As mentioned before, the LoTuS tool has been extended to enable analyses about a given path represented by an execution trace. With this functionality, it is possible to select a path to observe a specific property, such as the energy cost of an iteration in a particular data structure. These analyses can be observed in Table 1. Note that costs are small because the traces represent single paths executed by the program.

Table 2 contains the average total cost for 10 executions of the algorithms for each structure. It shows the impact on energy consumption of changing the storage structure, where the `LinkedList` structure has the highest energy consumption and the `ArrayList` has the lowest. As described in [21] and shown in their results[2], the `ArrayList` has the lowest energy consumption for `add()` and `set()` methods, while `Vector` has the lowest energy consumption for `get()` method. The `LinkedList` structure has the highest energy cost for all methods.

3.3 Discussion

The experiments presented here show the possibility of understanding software behaviour energy consumption through the use of a combination of an energy measurement tool and LTS analysis. Based on this information, it becomes possible to argue that the `ArrayList` is the best data structure for the sorting algorithms involved in this work in terms of energy efficiency, when one considers the structures involved in the experiments. The results also show the potential of this approach to provide support for the creation and evolution of energy-efficient software.

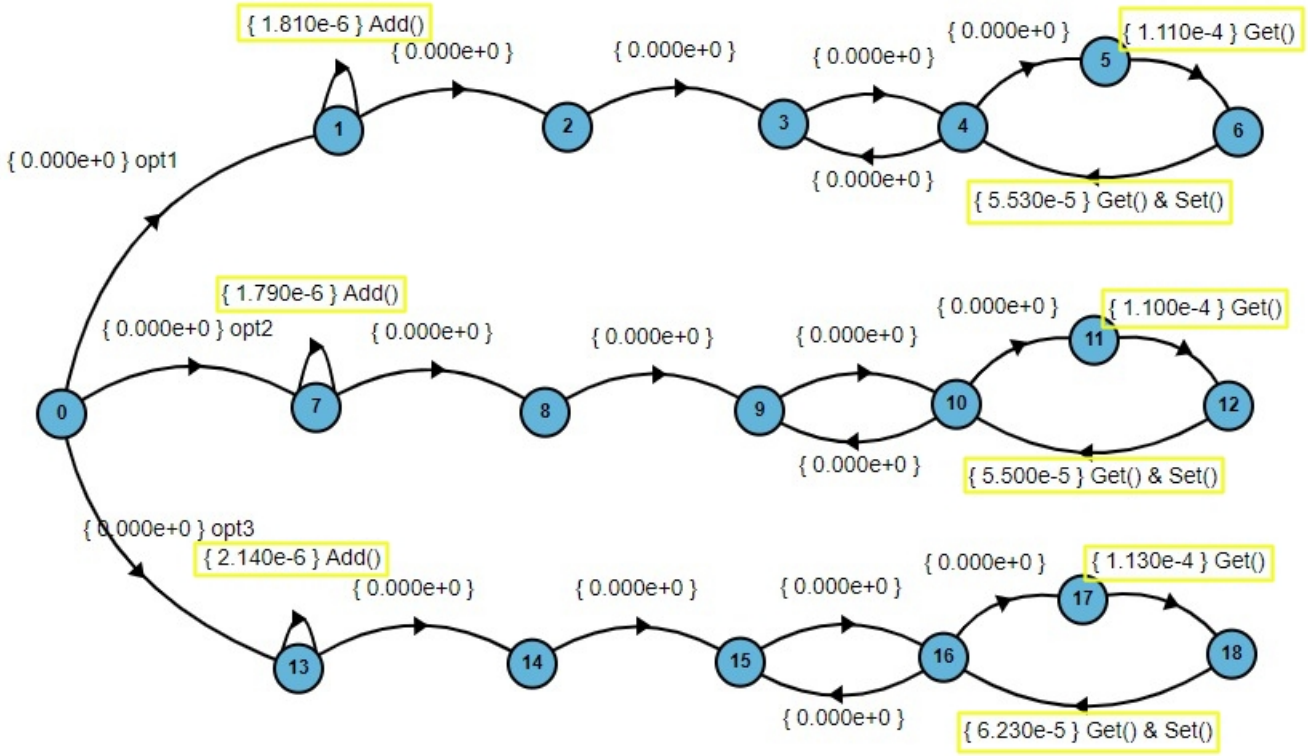
Although the sorting algorithms analysed have complexity $O(n^2)$, it is visible through the obtained results that the Bubble Sort algorithm demands a higher energy consumption than Insertion Sort and Selection Sort algorithms, consuming in some cases the double of energy. Hence, it demonstrates that there is not a strong correlation between time complexity and energy consumption. This result alone fosters our goal of investigating further an idea of energy complexity an how it compares to time and space complexity.

This work studied specifically some of the most popular and costly sorting algorithms, but the methodology employed to execute the experiments can be applied in other areas and, following, we present some of these areas where energy consumption can be a concern:

- **Energy Optimisation:** The techniques and software presented here can be an interesting application to optimise software during development or evolution with the goal of improving energy efficiency. With this, it is possible to discover and visualise what specific behaviours cause inefficiency on energy consumption, thus allowing that developers to achieve energy-efficient software;
- **Refactoring:** Refactoring is used to improve the source code organisation through the applications of modifications in code structure without affecting the software behaviour [25]. Applying the energy analysis, it is possible to evaluate the impact of these code transformations on the energy consumption providing to the developer information that is used to make the best choice during the refactoring process;
- **Mobile Applications:** Improving the energy consump-

[2] ⟨https://greenlab.di.uminho.pt/collections/⟩

Figure 1. Modelling of Bubble Sort performed in LoTuS tool.



tion of mobile applications, consequently the battery life is improved and, the methodology employed in this work can be used to identify the most costly behaviours and decrease the energy consumption of these applications.

- **Embedded Systems:** In particular, embedded systems are systems where hardware and software are combined and, this causes that the amount of available energy be used for both type of components. As is the case of mobile applications, embedded systems many times have limited battery to execute their functions. With the help of simulators of architecture and microprocessors, it is possible to evaluate the impact of replacing not only source code parts but also hardware components to check the influence of different settings in energy consumption;

- **Self-adaptative Systems:** Self-adaptive systems are able to adjust their behaviour or structure in response to their perception of the environment and the system itself [26]. Using the energy consumption feedback, it is possible to generate adjustments in the system to induce an adaptation strategy that reduces energy consumption.

We were confronted with some questions during the experimental phase, such as:

- How to define the abstraction level of the software behaviour during the model construction. As our modelling approach supports different levels of abstraction, finding the appropriate level for each type of analysis is some times not trivial. We are still not sure whether it would be possible to have models involving elements with different levels of abstraction, so that energy analysis could be better customised;

- How to model loop and nested structures and realise their energy measurement to perform better analyses. As energy costs may vary with each new iteration, account for this variation could improve precision, but it is still unclear how to model this. Nested structures can be modelled as one whole structure or as separate sequentially executed structures. It will require further investigation on what is the better option and whether we should support both, as they may be applied in different scenarios;

- How to measuring with precision the impact of energy annotations in source code, which makes the software take more time to execute and, consequently, increases the time to perform the experiments.

Figure 2. Modelling of Insertion Sort performed in LoTuS tool.

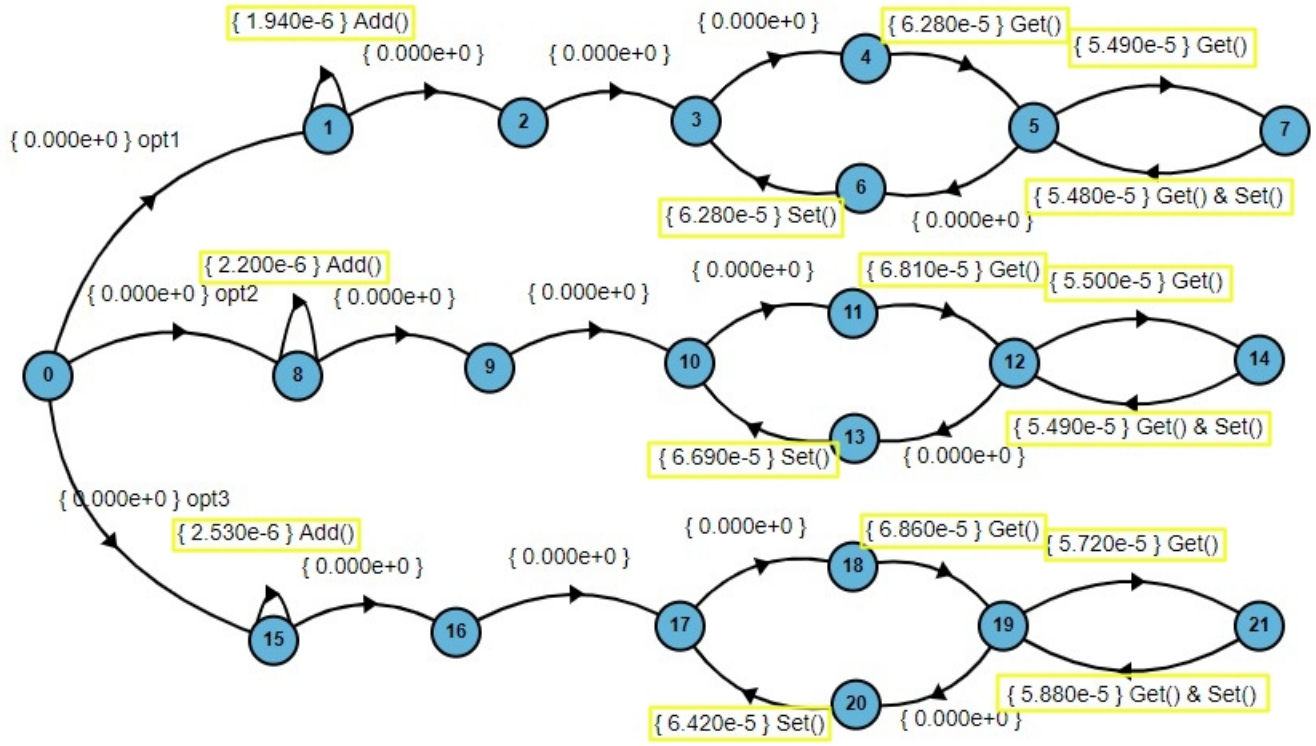


Table 1. Costs of paths for each data structure.

Algorithm	Structure	Execution Trace	Cost (W)
Bubble Sort	Vector	0-1-1-2-3-4-5-6-4-3	0.00016811
	ArrayList	0-7-7-8-9-10-11-12-10-9	0.00016679
	LinkedList	0-13-13-14-15-16-17-18-16-15	0.00017744
Insertion Sort	Vector	0-1-1-2-3-4-5-7-5-6-3	0.00023724
	ArrayList	0-8-8-9-10-11-12-14-12-13-10	0.00024710
	LinkedList	0-15-15-16-17-18-19-21-19-20-17	0.00025133
Selection Sort	Vector	0-1-1-2-3-4-5-6-5-7-3	0.00011344
	ArrayList	0-8-8-9-10-11-12-13-12-14-10	0.00011170
	LinkedList	0-15-15-16-17-18-19-20-19-21-17	0.00012570

4. Related Work

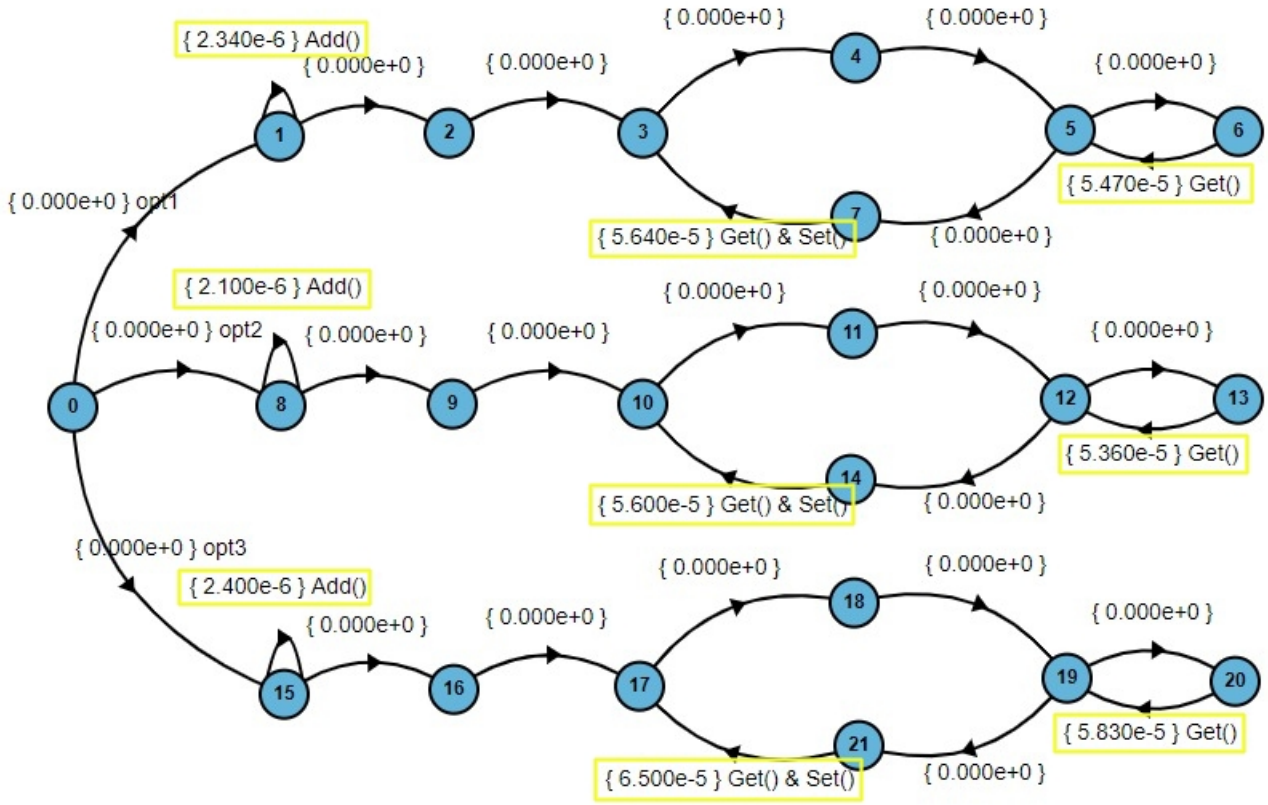
In this section, we describe some related work. We focus on two groups of work: previous studies involving behaviour models to represent energy costs and other research on how to analyse energy consumption. We do not report on approaches or tools for energy measurement because it is not our objective to propose a new way of doing that. Rather, we assume energy costs will be collected using one of the available tools and concentrate on how to extract the most from these data.

4.1 Energy Models

In [27], software energy consumption is described as behavioural contracts based on Power Consumption Automata (PCA). Properties written in terms of weighted linear temporal logic with freeze quantifiers are proposed to analyse the models. However, no example concrete example of usage of such model and logic was presented. Moreover, PCA describes the system in terms of power states, whereas we describe software behaviour and enhance it with energy information.

Considering energy-cost modelling, some other approaches, such as [28] [29], model energy costs using Markov chains and use PRISM [22] to run quantitative analyses based on probabilistic information. In the approach described in [28] the costs/rewards feature of PRISM is used to assign costs to states/transitions. A limitation of the aforementioned approaches is that analyses about paths are not supported, due to the type of logic adopted to describe properties. Hence, questions that either require producing sequences of actions or evaluating specific executions (e.g., the most/least costly behaviour) could not be easily executed, if at all. Another problem of using PRISM is the lack of model visualisation, which can not only help construct the model, but also make it

Figure 3. Modelling of Selection Sort performed in LoTuS tool.



easier to identify what could be modified to enhance energy efficiency.

4.2 Energy Consumption Analysis

Software energy consumption research has so far been focused on measuring energy costs. The work described in [4] concentrated on detecting excessive or anomalous energy consumption in software, focused on optimising the energy consumption in IT resources knowing how much power an application is consuming. Analyses have been conducted on the influence of data structures on energy consumption [21] [30] [31], introducing a methodology to optimise Java programs and decrease its energy consumption replacing data structures for a more energy-efficient alternative. The research presented in [32] described the development process of a profiler for measuring the energy consumption of source code points. In [33], critical energy areas were identified using a statistical method to associate responsibility for energy consumption to source code components. There have also been studies focusing on estimating energy costs with the goal of optimising and extending battery life in embedded systems [34] [35] and mobile devices [36] [37] [38], such in [39], where a study was performed to find alternative colours palette in user interfaces of mobile applications to optimise energy consumption while using consistent colours with respect to the original colour pallet. Nevertheless, none of the above studies employs a model as basis for their analyses, which limits their analysis capacity, since abstractions enable analyses that could not be or would be very difficult to be carried out directly on the actual software.

Some studies show how to curtail energy consumption during software development: this is done by using a search-based modification of the software system as an instance of Genetic Improvement. In this metric, it is sought to adapt the program and generate some related versions that hold some properties and improve others [40].

The work presented in [41] show how some decisions during the software development process can influence the energy cost. In their experiments, they performed energy measurements of various Java Application Programming Interface (API), evaluate for operations like file reading, file copy, file compression, and file decompression. Through this study, they found out that using APIs with different buffer sizes it is possible to save energy. Regarding user decisions, the work described in [42] shows how a software consumes more or less energy depending on user requirements and choices.

The impact of energy costs in software development has been studied [38] [43]. In [43], they checked a new way for aligning software design to energy consumption and is shown empirically in some different software implementations. For this experimentation, they used design patterns, which is a way of evaluating how implementation decisions can influence energy usage. The work of [38] exploits the consumption in different algorithms of machine learning on smartphones con-

Table 2. Average total cost for each algorithm executed with each data structure.

Algorithm	Structure	Total Consumption (W)
Bubble Sort	Vector	249.49515
	ArrayList	246.32837
	LinkedList	258.02736
Insertion Sort	Vector	123.96702
	ArrayList	124.29720
	LinkedList	130.98218
Selection Sort	Vector	123.51673
	ArrayList	120.90212
	LinkedList	126.06802

sidering aspects such as dataset size, number of data attributes, etc.

Many techniques have been created to get lower energy consumption from software systems. One of these techniques studied how to use data structures for energy efficiency [21] [30] and android language [44], as well as has been studied how different programming languages affect the final energy consumption, be it in mobile or desktop applications [18] [45] [46]. Some work [47] [36] seek to anticipate how much operations will be spent in software systems.

None of the above studies combines the measurement of energy with behaviour models. The experiments and analyses performed in our work demonstrated the possibility of combination of these approaches, providing to the user a way to improve their software behaviour when the goal is energy efficiency.

5. Conclusions and Future Work

With the performed experiments, it was possible to investigate how the combination of an energy measurement tool and an LTS-analysis tool works and how this combination can help developers produce and evolve energy-efficient software, providing a visualisation of the software behaviour and the energy consumption of the source code parts with a granularity that can be defined by the user.

We intend, as future work, to investigate whether this approach is the most suitable for energy-consumption analysis, developing other experiments, in particular involving multiple components. In the context of multi-component systems, the total energy consumption should account for the energy consumed by all of its components. How interactions of these components affect energy costs and how replacing one component by another may impact energy costs will a subject of study.

Concerning the measurement of energy consumption of sorting algorithms, we intend to compare all the popular sorting algorithms, such Quicksort, Heapsort, Merge Sort, Shellsort and others with different storage structures and different number of elements to be sorted, investing more time on the experiments an perform the analysis of more energy properties.

We would like to evaluate and analyse different levels of abstraction in the LTS model. Although associating energy costs to method executions seems natural, it may be necessary to work with specific code elements, such as iteration blocks, selection blocks, etc. Moreover, we have to investigate how to represent energy costs regarding nested blocks and iterations.

When the subject is software energy efficiency, a question comes up: a quick execution of a program means to have an energy-efficient program? Some studies such as [12], [48], and [49] say that substituting energy optimisation by performance optimisation is insufficient and some times incorrect. In [50] a study was performed with the objective to relate energy consumption, time of execution and memory and they affirm that energy consumption does not depend only on the execution time. On the other hand, the study described in [51] supports that energy and time are directly related. Therefore, a possible analysis would be whether time-efficient algorithms are also energy-efficient algorithms. This could lead to the definition of energy complexity classes and create a new aspect related to software development that could be considered when comparing different solutions.

Finally, we plan to combine energy costs with probabilistic behaviour, enabling a developer to discover what is the probability of executing the most/least costly behaviours and decide whether it is worth - or necessary - to make some change in the their code to improve software efficiency. We will also study whether it is possible to provide some recommendations on how a developer should carry out these necessary changes. In addition to all that, the use of a model extraction approach, such as [23], could ease the modelling effort, thus making it easier to adopt energy analysis as part of a software development process. However, we would have to determine how to combine a extracted models with the collected energy information.

Acknowledgements

This work is partially supported by CNPq/Brazil under the grant Universal 131260/2019-7.

Author contributions

Danilo S. Alves: Carrying out the experiments and paper writing.

Oseias A. Ferreira: Carrying out the experiments and paper writing.
Lucio M. Duarte: Paper writing and revision.
Davi Silva: Carrying out the experiments.
Paulo H. Maia: Paper revision.

References

[1] KHALID, H.; SHIHAB, E.; AL. et. What Do Mobile App Users Complain About? *IEEE Software*, v. 32, n. 3, p. 70–77, May 2015.

[2] PINTO, G.; CASTOR, F. Energy efficiency: A new concern for application software developers. *CACM*, ACM, New York, NY, USA, v. 60, n. 12, p. 68–75, December 2017.

[3] ALBERS, S. Energy-efficient algorithms. *CACM*, ACM, New York, NY, USA, v. 53, n. 5, p. 86–96, maio 2010. Disponível em: ⟨http://doi.acm.org/10.1145/1735223.1735245⟩.

[4] SINGH, V. K.; DUTTA, K.; AL. et. Estimating the energy consumption of executing software processes. In: *GreenCom, iThings and CPSCom 2013*. [S.l.: s.n.], 2013. p. 94–101.

[5] KULKARNI, P. et al. Fast searches for effective optimization phase sequences. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 39, n. 6, p. 171–182, jun. 2004. Disponível em: ⟨http://doi.acm.org/10.1145/996893.996863⟩.

[6] KOOMEY, J. G. Worldwide electricity used in data centers. *Environmental research letters*, IOP Publishing, v. 3, n. 3, p. 034008, 2008.

[7] WHITNEY, J.; DELFORGE, P. Data center efficiency assessment. *Issue paper on NRDC (The Natural Resource Defense Council)*, 2014.

[8] DORN, J. et al. Automatically exploring tradeoffs between software output fidelity and energy costs. *IEEE Transactions on Software Engineering*, IEEE, v. 45, n. 3, p. 219–236, 2017.

[9] HUANG, C.; TSAO, S. Minimizing energy consumption of embedded systems via optimal code layout. *IEEE Transactions on Computers*, v. 61, n. 8, p. 1127–1139, Aug 2012.

[10] RAGHUNATHAN, V. et al. Energy-aware wireless systems with adaptive power-fidelity tradeoffs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 13, n. 2, p. 211–225, Feb 2005.

[11] CHAI, Q. et al. Empowering Designers to Estimate Function-Level Power for Developing Green Applications. In: *Proceedings - 2013 International Conference on Cloud and Service Computing, CSC 2013*. [S.l.: s.n.], 2013. p. 57–62.

[12] PANG, C. et al. What do programmers know about software energy consumption? *IEEE Software*, IEEE, v. 33, n. 3, p. 83–89, 2015.

[13] MANOTAS, I.; POLLOCK, L.; CLAUSE, J. Seeds: a software engineer's energy-optimization decision support framework. In: ACM. *Proceedings of the 36th International Conference on Software Engineering*. [S.l.], 2014. p. 503–514.

[14] DUARTE, L. M. et al. A model-based framework for the analysis of software energy consumption. In: ACM. *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. [S.l.], 2019. p. 67–72.

[15] LIU, K.; PINTO, G.; AL. et. Data-oriented characterization of application-level energy optimization. In: EGYED, A.; SCHAEFER, I. (Ed.). *Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. p. 316–331.

[16] KELLER, R. M. Formal Verification of Parallel Programs. *CACM*, v. 19, n. 7, p. 371–384, July 1976.

[17] BARBOSA, D. M. et al. Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-adaptive Systems. In: *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Piscataway, NJ, USA: IEEE Press, 2017. (SEAMS '17), p. 24–30. Disponível em: ⟨https://doi.org/10.1109/SEAMS.2017.18⟩.

[18] LI, D.; HALFOND, W. G. J. An Investigation into Energy-saving Programming Practices for Android Smartphone App Development. In: *Proceedings of the 3rd International Workshop on Green and Sustainable Software*. New York, NY, USA: ACM, 2014. (GREENS 2014), p. 46–53. Disponível em: ⟨http://doi.acm.org/10.1145/2593743.2593750⟩.

[19] BINKERT, N.; BECKMANN, B.; AL. et. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, ACM, New York, NY, USA, v. 39, n. 2, p. 1–7, ago. 2011.

[20] LI, S. et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. [S.l.: s.n.], 2009. p. 469–480.

[21] PEREIRA, R.; COUTO, M.; AL. et. The influence of the Java collection framework on overall energy consumption. In: ACM. *Proceedings of the 5th International Workshop on Green and Sustainable Software*. [S.l.], 2016. p. 15–21.

[22] KWIATKOWSKA, M.; NORMAN, G.; AL. et. PRISM: Probabilistic symbolic model checker. In: KEMPER, P. (Ed.). *Proc. of the Tools Session of Aachen 2001 Intl Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*. [S.l.: s.n.], 2001. p. 7–12. Available as Technical Report 760/2001, University of Dortmund.

[23] DUARTE, L. M.; KRAMER, J.; AL. et. Using contexts to extract models from code. *Software and Systems Modeling*, v. 16, n. 2, p. 523–557, 2017.

[24] BUNSE, C. et al. Exploring the energy consumption of data sorting algorithms in embedded and mobile environments.

In: IEEE. *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*. [S.l.], 2009. p. 600–607.

[25] FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 2018.

[26] CHENG, B. H. et al. Software engineering for self-adaptive systems: A research roadmap. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 1–26.

[27] NAKAJIMA, S. Model checking of energy consumption behavior. In: CARDIN, M.-A. et al. (Ed.). *Complex Systems Design & Management Asia*. Cham: Springer International Publishing, 2015. p. 3–14.

[28] BAIER, C.; DUBSLAFF, C. et al. Probabilistic model checking for energy-utility analysis. In: ____. *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*. Cham: Springer International Publishing, 2014. p. 96–123. Disponível em: ⟨https://doi.org/10.1007/978-3-319-06880-0_5⟩.

[29] DUBSLAFF, C.; KLÜPPELHOLZ, S.; BAIER, C. Probabilistic model checking for energy analysis in software product lines. In: *MODULARITY '14*. New York, NY, USA: ACM, 2014. (MODULARITY '14), p. 169–180. Disponível em: ⟨http://doi.acm.org/10.1145/2577080.2577095⟩.

[30] HASAN, S. et al. Energy profiles of java collections classes. In: ACM. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. [S.l.], 2016. p. 225–236.

[31] OLIVEIRA, W.; OLIVEIRA, R.; AL. et. Recommending energy-efficient java collections. In: IEEE PRESS. *MSR 2019*. [S.l.], 2019. p. 160–170.

[32] SCHUBERT, S. et al. Profiling software for energy consumption. In: *2012 IEEE International Conference on Green Computing and Communications*. [S.l.: s.n.], 2012. p. 515–522.

[33] PEREIRA, R. Locating energy hotspots in source code. In: *39th International Conference on Software Engineering (ICSE 2017)*. [S.l.: s.n.], 2017. p. 88–90.

[34] BRANDOLESE, C. et al. The impact of source code transformations on software power and energy consumption. *Journal of Circuits, Systems, and Computers*, World Scientific, v. 11, n. 05, p. 477–502, 2002.

[35] JAYASEELAN, R.; MITRA, T.; LI, X. Estimating the worst-case energy consumption of embedded software. In: IEEE. *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. [S.l.], 2006. p. 81–90.

[36] HAO, S. et al. Estimating mobile application energy consumption using program analysis. In: IEEE PRESS. *35th International Conference on Software Engineering (ICSE 2013)*. [S.l.], 2013. p. 92–101.

[37] COUTO, M. et al. Detecting anomalous energy consumption in android applications. In: SPRINGER. *Brazilian Symposium on Programming Languages*. [S.l.], 2014. p. 77–91.

[38] MCINTOSH, A.; HASSAN, S.; HINDLE, A. What can android mobile app developers do about the energy consumption of machine learning? *Empirical Software Engineering*, Springer, v. 24, n. 2, p. 562–601, 2019.

[39] LINARES-VÁSQUEZ, M. et al. Optimizing energy consumption of GUIs in Android apps: a multi-objective approach. In: ACM. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. [S.l.], 2015. p. 143–154.

[40] PETKE, J. et al. Genetic improvement of software: A comprehensive survey. *IEEE Transactions on Evolutionary Computation*, v. 22, n. 3, p. 415–432, June 2018.

[41] SINGH, J.; NAIK, K.; MAHINTHAN, V. Impact of developer choices on energy consumption of software on servers. *Procedia Computer Science*, Elsevier, v. 62, p. 385–394, 2015.

[42] ZHANG, C.; HINDLE, A.; GERMAN, D. M. The impact of user choice on energy consumption. *IEEE software*, IEEE, v. 31, n. 3, p. 69–75, 2014.

[43] SAHIN, C. et al. Initial explorations on design pattern energy usage. In: *2012 First International Workshop on Green and Sustainable Software (GREENS)*. [S.l.: s.n.], 2012. p. 55–61.

[44] OLIVEIRA, W.; OLIVEIRA, R.; CASTOR, F. A study on the energy consumption of android app development approaches. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. [S.l.: s.n.], 2017. p. 42–52.

[45] LINARES-VÁSQUEZ, M. et al. Mining energy-greedy api usage patterns in android apps: An empirical study. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014. (MSR 2014), p. 2–11. Disponível em: ⟨http://doi.acm.org/10.1145/2597073.2597085⟩.

[46] PEREIRA, R. et al. Helping programmers improve the energy efficiency of source code. In: *Proceedings of the 39th International Conference on Software Engineering Companion*. Piscataway, NJ, USA: IEEE Press, 2017. (ICSE-C '17), p. 238–240. Disponível em: ⟨https://doi.org/10.1109/ICSE-C.2017.80⟩.

[47] COUTO, M. et al. Products go green: Worst-case energy consumption in software product lines. In: ACM. *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. [S.l.], 2017. p. 84–93.

[48] LIMA, L. G. et al. Haskell in green land: Analyzing the energy behavior of a purely functional language. In: IEEE. *2016 IEEE 23rd international conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.], 2016. v. 1, p. 517–528.

[49] PINTO, G.; CASTOR, F.; LIU, Y. D. Understanding energy behaviors of thread management constructs. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2014. v. 49, n. 10, p. 345–360.

[50] PEREIRA, R. et al. Energy efficiency across programming languages: How do energy, time, and memory relate? In: ACM. *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. [S.l.], 2017. p. 256–267.

[51] YUKI, T.; RAJOPADHYE, S. Folklore confirmed: Compiling for speed = compiling for energy. In: SPRINGER. *International Workshop on Languages and Compilers for Parallel Computing*. [S.l.], 2013. p. 169–184.