Programming languages and energy efficiency

Wessel Oele
Open university, The Netherlands

ABSTRACT
Many factors influence the energy efficiency of software applications. In this article two empirical studies about the relation between programming languages and the energy efficiency of programs that are written in them are compared. The results show that influence does exist and that there is a wide variety in the energy and performance behaviour of specific task/language combinations.

Keywords
Programming, programming languages, green programming, energy, environment, memory

1. INTRODUCTION
Every software application that is executed on some computational platform causes one or more processors of that platform to execute instructions and to allocate memory. Both processor and memory chips convert electrical energy into heat while doing so. Strictly speaking the computational platform that carries out the instructions increases the entropy of the physical system in wich it exists. Most people, however, will not dive into the physical laws of thermodynamics and simply state that the computational platforms they use (computers, phones, tablets, laptops, etc.) "consume" energy. In a world where climate change and both energy generation and consumption have become relevant issues on political and scientific agendas, the question emerges how energy efficient, or inefficient, the software we use actually is.

Although many factors affect the energy consumption of a running software application, this article focusses on the question whether the programming languages software engineers use, have any influence on the power consumption of the applications written in them? Does a relation between the programming language and energy efficiency exist and, if so, does it matter enough to make a difference for developers?

The remainder of this article is organised as follows. Section two covers two articles about a study by Pereira et al. Their first article introduces a framework for measuring the energy consumption of programs. In their second article they use their framework and extend their study with memory usage. Section three covers an article by Georgiou et al. It shows a different way of measuring energy consumption on multiple computational platforms. Section four covers a comparison between both studies and section five deals with threats to the validity of both studies.

2. RATING LANGUAGES
Pereira et al. conducted a study at analysing programming languages for energy efficiency in order to obtain what they call a "green rating" of languages [2]. They developed a framework (in C) that makes it possible to run an external program and measure the amount of energy the program consumed and the amount of time the program needed to finish its job.

The framework was used in a second study in which Pereira et al. extended their previous work [9]. Not only did they look at the energy and time behaviour of programs, but also at the amount of memory a program needs and how memory usage relates to power consumption. Four research questions were asked in their article:

RQ1: Is it possible to compare the energy efficiency of programming languages?

RQ2: Are the faster programming languages always the most energy efficient languages?

RQ3: Are memory usage and energy consumption related?

RQ4: Is it possible to choose the best programming language automatically when a programmer knows whether time, energy or memory is the most important?

2.1 Methodology
In order to compare the behaviour of programming languages, Pereira et al. looked for a collection of well known algorithms for which implementations exist in various languages. The *computer language benchmarks game* (CLBG) [5] offers exactly this. The CLBG is a repository of programs that implement various widely used algorithms. Amongst them are implementations for allocating and traversing a binary tree, replacing text based on regular expressions, generating an $n \times n$ bitmap of the Mandelbrot set, etc. It is well known that for many problems several different algorithms exist. In their study the algorithms that are known to be the most efficient were selected.

The repository not only holds implementations of these algorithms in various languages but also offers documentation about benchmarks of executed programs, the compilers/interpreters and hardware that was used, etc. Pereira

et al. selected ten different algorithms and for each of these algorithms implementations in 27 different languages.

They compiled and executed each of the algorithm/language combinations and measured the energy consumption with Intels running average power limit (RAPL) tool [3]. The amount of memory that is needed by a program was measured with the `time` tool that is available on the Unix operating system. They compiled and executed each of the algorithm/language combinations with the same versions of the compilers and interpreters as used in the CLBG and repeated this ten times to ensure that wrong measurements as a result of caching effects or cold starts were avoided. By doing so, they could check for consistency between their own results and the results of the CLBG.

The selected programming languages, sorted for convenience by paradigms, are the following:

Functional: Erlang, F#, Haskell, Lisp, Ocaml, Perl, Racket, Ruby, Rust

Imperative: Ada, C, C++, F#, Fortran, Go, Ocaml, Pascal, Rust

Objectoriented: Ada, C++, C#, Chapel, Dart , F#, Java, JavaScript, Ocaml, Perl, PHP, Python, Racket, Rust, Smalltalk, Swift, TypeScript

Scripted: Dart, Hack, JavaScript, JRuby, Lua, Perl, PHP, Python, Ruby, TypeScript

Determining the energy consumption of a program was based on a simple calculation: energy consumption (in Joules) = power consumption (in Watts) × time (in seconds).

2.2 Results

The following results emerge from the work of Pereira et al.

- It is indeed possible to compare the energy efficiency of programming languages. The results show that, overall, compiled languages are the most energy efficient while interpreted languages (like Python and Perl) are the least.
- There exists a top five of languages that are, overall, both faster and more energy efficient: C, Rust, C++, Ada and Java. For some specific computation problems other languages perform better.
- Faster languages are not always the most energy efficient. This is caused by the fact that the power variable in the equation $consumption = power \times time$ the power variable is not a constant. Thus optimizing for execution time frequently lowers power consumption, but not always.
- No relation between memory power consumption and peak memory allocation was found.
- Automatically selecting the best language is possible, given that a developer is interested in energy consumption and/or execution speed. If memory consumption is important, automatically selecting the most efficient language is not possible.

3. ENERGY ON DIFFERENT PLATFORMS

A different approach for measuring the energy efficiency of software applications was used by Georgiou et al. [4]. Like the study of Pereira et al. the algorithms of a fixed set of computation problems was used, but in this study problems were selected from the Rosetta code project.

The research questions Georgiou et al. asked in their study are the following:

RQ1: Which programming languages are the most energy efficient and inefficient for particular tasks?

RQ2: Which types of programming languages are, on average, more energy efficient and inefficient for different computational platforms?

RQ3: How much does the energy efficiency of each programming language differ among the selected platforms?

As can be deducted from the research questions Georgiou et al. wanted to gain an understanding of the energy efficiency of specific languages for specific tasks on a *per platform* basis. This is the main difference with the study of Pereira et al.

3.1 Methodology

Georgiou et al. selected 14 languages and 25 tasks from the Rosetta code repository. Language selection was based on the well known Tiobe index [8] with a couple of modifications: Proprietary and visually oriented languages were removed while the most promising languages like Go and rust were added to their selection.

Three different computational platforms were selected: a server, a laptop and a Raspberry Pi. For measuring power consumption over all these platforms Watts up Pro [6] was used and care was taken to make sure that in each of these platforms the consumed energy was indeed spent on the task and not on background processes like defragmentation, cron jobs, etc. They also measured the temperature of the platforms (using `lm_sensors` and `vcgencmd`) and compared it to the room temperature in order to make sure that no energy was consumed by cooling fans. Time was measured with the `time` tool.

For calculating the energy consumption an equation known as the *energy delay product* (edp) was used:

$$edp = E \times T^w$$

This equation was introduced by Horowitz et al. [7] and, later, used as a weighted function by Cameron et al. [1]. The multiplication of time (T) and power/energy(E) is the same as in the study of Pereira et al. but the exponent w is used as a weight for differentiating between energy efficiency and performance efficiency: A weight of 1 is used when energy efficiency is the most important, 3 is used when performance efficiency is the most important and 2 represents a balance between both.

3.2 Results

After calculating the edp for 25 tasks over 14 programming languages on three different platforms, Georgiou et al. found that the raw edp value of the most and least efficient programming language for a specific task on a specific platform differed considerably, up to a value of 10,000,000. For this reason, they took the language with the lowest edp value (p_{min}) on a task/platform combination and the edp values of each of the other languages (p) for that task/platform combination. The base 10 logarithm was then calculated according to the following formula:

$$log(p/p_{min})$$

The results were used to produce several box plots and heat maps on a per platform basis. This allowed Georgiou et al. to gain a fine grained insight in the energy and performance behaviour of a program per task per language per platform. Their conclusions are as follows:

- Compiled languages are, overall, more edp efficient than the interpreted languages.
- C, C++ and Go are the most edp efficient languages over all computational platforms.
- R, Java and Swift are the most edp inefficient languages over all computational platforms.
- Given a specific task on a specific platform the differences in edp efficiency between languages are big.

Georgiou et al. provide a table in which the most and least edp efficient languages (and their edp difference) are shown on a per task per platform basis. As an example (amongst many) of the last conclusion we can look at the edp performance difference between Rust and Swift on the Raspberry Pi platform for executing the lzw compression task. The edp difference between both languages is 9.56 in favor of Rust.

In order to gain insight in the edp differences of a specific language on different platforms, Georgiou et al. used the Wilcoxon' s signed-rank test in conjunction with a null hypotheses:

Hypothesis H_0**:** *A programming language' s average EDP, does not have a statistically important difference between the measurement platforms.*

From their calculations they concluded that no evidence was found for rejecting the null hypothesis for all languages, with the exception of the languages C, Swift and (a little weaker) C++ on the Raspberry Pi vs. laptop platform.

4. COMPARISON

Since both studies are partially similar in the research topic they cover, some of the found results can be compared. Both studies used a repository of computational problems, but where Pereira et al. used the CLBG, Georgiou et al. used the Rosetta code repository. This is interesting because it allows us to see whether the results of both studies can be generalised.

In both studies a clear winner in both energy and performance efficiency emerges: C. Further on, both studies conclude that interpreted languages are the least efficient while compiled languages are overall both faster and more energy efficient. In the work of Pereira et al. it was mentioned that some specific languages are far more efficient in executing specific tasks. This was acknowledged in the work of Georgiou et al. where several "champions" are mentioned. Rust, for example, is a clear winner when it comes to executing file i/o operations. Finally, both studies show that faster languages are not always the most energy efficient languages. C# and Java are a nice example: While Java is usually faster, C# is more energy efficient.

The main conclusion to be drawn is therefore that, indeed, programming languages *do* have influence on the energy efficiency of the programs written in them and that energy efficiency and execution speed are two different metrics. Although exceptions exist, a "big picture" seems to emerge.

Beyond this, both studies diverge into different directions. Pereira et al. extended their own work and tried to gain insight into the memory behaviour of programs, while Georgiou et al. looked into the performance and energy behaviour of programs on different computational platforms. As a consequence the results of both studies become more fine-grained in their own directions.

5. THREATS TO VALIDITY

In both studies several threats to validity are mentioned and in both studies these threats were divided in internal and external validity. Several commonalities between both studies were found. This can be regarded as a validity check in itself: Both studies follow roughly the same methodology. If the authors of the first study reported a completely different set of threats to validity than the other, one could ask the question how aware the authors of both studies are of potential pitfalls in their own work. In both studies, however, roughly the same threats to internal and external validity were reported. This suggests that there exists an overall awareness of the problems and pitfalls associated with this field of research. Only two studies are compared in this article, however, so we should be reticent in drawing conclusions.

5.1 Interal validity

Both Pereira et al. and Georgiou et al. state that measuring the energy consumption of a running program is complicated since many factors may interfere with obtaining precise measurements. Completely ruling out background processes of the operating system, for example, is hard to guarantee. The same problems arise when interpreters or virtual machines are used. In order to counter these problems, measurements were repeated several times after which outliers were removed and means and variances were calculated for checking the consistency of the results.

Finally, Georgiou et al. noted a potential problem with the energy measurement tool they used: It has a sample rate of one second so computations that last less than a second are not reported. These were therefore removed from the final results.

5.2 External validity

Both Pereira et al. and Georgiou et al. report that using only a fixed set of computation problems (from the CLBG and the Rosetta code repository) limits the generalizability of their work. The same can be stated for the set of programming languages. For the work of Georgiou et al. this limitation also applies to the selected computational platforms.

In both studies it was reported that determining what influence a programming language has on the energy consumption of the application written in it is very complicated as it equates to isolating a specific link in a large chain of dependencies. Although both studies offer all the information that is needed for replication, the results that emerge from them will change as compilers, algorithms, computational platforms, the languages themselves, virtual machines, interpreters, etc. change over time.

6. CONCLUSION AND FUTURE WORK

Two different studies that cover the relation between programming languages and the energy consumption of software applications written in them were compared in this article. The study of Pereira et al. also looked into memory consumption, while the study of Georgiou et al. measured the energy consumption of applications running on different platforms.

The results show that programming languages *do* have influence on the energy consumption of the applications written in them and that there exists a wide variety in both performance and energy efficiency across languages. In many cases faster languages are also the most energy efficient but this is not always the case. Automatically selecting the most suitable language for a given task is possible when energy or performance efficiency are important. When memory efficiency is important automatic selection can not be done.

Isolating the influence of a programming language on energy efficiency is hard as the language is only one link in large chain of dependencies. Many factors that influence the energy efficiency of software change over time. Studies like these will therefore have to be repeated in the future in order to identify changes in the current status quo.

Since programming languages are extended with newer features over time, it would be interesting to see how the energy and performance efficiency of individual languages evolve over time. Java, for example, has been extended with several features that are associated with functional programming. It would be interesting to see whether (and if so, how) this affects the energy efficiency of software written in it. Gaining a more fine-grained and historical insight in the energy profile of individual languages may help developers in making more informed choices when energy efficiency is important.

7. REFERENCES

[1] K. W. Cameron, R. Ge, and X. Feng. High-performance, power-aware distributed computing for scientific applications. *Computer*, 38(11):40–47, 2005.

[2] M. L. Couto, R. A. Pereira, F. J. Ribeiro, R. Rua, and J. A. Saraiva. Towards a green ranking for programming languages. *Proceedings of SBLP, Fortaleza, CE, Brazil, September 21âĂŞ22*, 2017.

[3] M. Dimitrov, C. Strickland, S.-W. Kim, K. Kumar, and K. Doshi. Intel power governor, 2015.

[4] S. Georgiou, M. Kechagia, P. Louridas, and D. Spinellis. What are your programming language's energy-delay implications? In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 303–313. ACM, 2018.

[5] I. Gouy. The computer language benchmarks game. *URL http://benchmarksgame. alioth. debian. org*, 2017.

[6] J. M. Hirst, J. R. Miller, B. A. Kaplan, and D. D. Reed. Watts up? pro ac power meter for automated energy recording, 2013.

[7] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Proceedings of IEEE symposium on low power electronics*, pages 8–11. IEEE, 1994.

[8] T. Index. Tiobe-the software quality company. *TIOBE Index| TIOBE–The Software Quality Company [Electronic resource]. Mode of access: https://www. tiobe. com/tiobe-index/-Date of access*, 1, 2018.

[9] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva. Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, pages 256–267. ACM, 2017.