Green Software

Lukas Koedijk
lukaskoedijk@gmail.com

June 17, 2019, 26 pages

Research supervisor:	A. M. Oprescu, a.m.oprescu@uva.nl
Host/Daily supervisor:	D. Stam, stam.dennis@kpmg.nl
Host organisation/Research group:	KPMG Digital Enablement, www.kpmg.nl

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Here goes your abstract. Be concise, introduce context, problem, known approaches, your solution, your findings.

Contents

Chapter 1

Introduction

Currently more and more people are concerned with global warming. Global warming is a result of the emission of greenhouse gasses during energy generation of not green energy options. Because of this a lot of people want to change to green energy generation to help solve this problem. Another solution is to decrease the energy consumption. Which is not only good for the environment but can also save a lot of money on the energy bill. The energy consumption of communication networks, personal computers and data centers world wide is increasing over the years [1]. This happens with a growth rate of 10%, 5% and 4% respectively [1]. Therefore it is important to research ways of decreasing the energy consumption. In the field of hardware there is, according to Koomey's law, an increase of the number of computations per Joule. However this is not enough, because tasks need more computations to complete due to the confidence in the improvement of hardware [2]. For this reason we need to look at possibilities in decreasing the energy consumption from a software perspective.

1.1 Problem statement

The energy consumption of communication networks, personal computers and data centers are increasing yearly. There are two ways of decreasing the energy consumption, by decreasing the energy consumption of hardware or software. Scientific research is mostly focused on decreasing the energy consumption of hardware. There are also some papers about reducing the energy consumption in the software process and the decision making process. A small bit of research is done on the energy consumption of software, but their research goal is to estimate the energy consumption. Therefore what I miss and want to look into is if there is a good way of writing code regards the energy consumption.

1.1.1 Research questions

During this research I want to answer the following two existence questions: (1) **Is there a difference in the energy consumption of software projects in different programming languages that have the same functionality** and (2) **Is there a difference in energy consumption of different software projects (in the same programming language) that have the same functionality**? To answer these questions I first need to answer the three description and classification questions listed below.

- How can the energy consumption of a software project be measured?
- How do we proof if two programs have the same functionality?
- When is a difference in energy consumption big enough to be called a difference?

Based on the results I find when answering the second existence question, I want to look into what this difference is on code level. To do this I want to answer the descriptive/comparative question **what is the difference on code level**? This question is only useful when there is a difference in the energy consumption of software project in the same programming language that have the same functionality. Otherwise we can't link the difference in energy consumption to the difference in the code.

1.1.2 Research method

As method I want to use a controlled experiment. In this experiment I will run different software projects to measure the energy consumption. Here the projects are the variable input, the energy consumption the output and everything else like compiler and energy consumption calculations should be constant. For the first research question the hypothesis is that there is a difference in the energy consumption based on the programming language chosen. The hypothesis of the second research question is that there is a difference in the energy consumption of different software projects (in the same programming language) that have the same functionality.

Data

For the first research question I need programs that have the same functionality, but are written in a different programming language. I want to use library code for this, because they are viewed as well-thought-out solutions to a problem and are a solution to the same problem. Other options are using interview code, student assignments and code from competitions solving math problems. For the second research question I need programs that have the same functionality and are written in the same language. Here we could use the library as a baseline and look at competition code for problems that solve the same problem as the library to compare. I found that libraries are not that useful, because they can use different algorithms to solve the same problem.

When proving that all the projects have the same functionality I will define properties for the input and output and test if these properties hold for all the projects [3].

To proof that the projects have different values for the energy consumption we will preform the independent two sample t-test. This test calculates if two distributions are in the same distribution. This test can only be use if the both distributions are normally or close to normally distributed, have the same size and have the same variance. To check if they are normally distributed we can use the Chi-square test and to check if the variance is the same we can use the f-test.

Measuring energy consumption

When measuring the energy consumption of a project you have to take into account the energy consumption of CPU, memory and disk [4]. The measurements can be done with a hardware or software approach. A hardware approach is more accurate but also more expensive [4]. I want to use a hardware approach, luckily as a student of the UvA I have access to the DAS-4 and the DAS-5. The DAS is a distributed system that can also do energy measurements [5]. You can connect to the DAS using ssh with an username and a password. The nodes that can measure the energy are located at the VU. To measure the energy I need to run a job on the DAS-5 and use the DAS-4 to measure the energy consumption. When running a job I need to specify witch node I want to run the job on, because not all nodes can be measured for the energy consumption. This is done by using the *-w* flag while using the command *srun*. The measuring can then be done by using the *smnpwalk* command and then filtering out the information by using *fgrep Energy*.

Interpret results

When there is a difference in the energy consumption of different software projects in the same programming language I will look at the different projects on code-level. Here I will try to find what is causing a project to have a lower or higher energy consumption. These findings will then be tested by letting myself write two versions, where I think one is written badly regards the energy consumption and one that is written good. I will then run the two version and check if the good version has a lower energy consumption.

1.2 Contributions

Our research makes the following contributions:

1. An energy consumption measurement set-up
2. A data set of software programs with the same functionality
3. Rules for writing good software regards the energy consumption

1.3 Outline

In Chapter 2 we describe the background of this thesis. Chapter **??** describes ... Results are shown in Chapter 6 and discussed in Chapter 7. Chapter 8, contains the work related to this thesis. Finally, we present our concluding remarks in Chapter 9 together with future work.

Chapter 2

Background

This chapter will present the necessary background information for this thesis. First, we define some basic terminology that will be used throughout this thesis. Next, ...

Chapter 3

Energy measurement

When doing energy measurements it is important to not only measure the energy consumption of the CPU, but also of the memory and disk [4]. You can use a hardware method to measure the energy consumption or use a software method to estimate the energy consumption. Using a hardware method is more accurate, but also more expensive [4]. Luckily as a student of the University of Amsterdam (UvA) I have access to the DAS. The DAS is a distributed supercomputer with nodes that have different hardware specifications and some nodes are connected to a PDU, through which we can measure the energy consumption [5].

The DAS has a head node where all the users connect to. Here the users can reserve nodes and add jobs to the queue. There are multiple releases of the DAS, currently only DAS-4 and DAS-5 are in use. For measuring the energy we needed to run the programs on a specific set of six nodes on the DAS-5. To retrieve the data from the PDU connected to these six nodes we needed to use the DAS-4 and the *smnpwalk* command. There were two kind of energy measurement values we could retrieve, the current power (Watt) and the energy consumption (kWh) from when the node was plugged-in till now. Both methods have a disadvantage when using it. When using the current power you need to retrieve the power constantly and you loose some accuracy because you don't know what the power does in between two measure points. The method of measuring the energy consumption has the problem of showing a number that is too large, the numbers are in kWh and have three decimal numbers. Thus the lowest decimal shows Watt per hour. We found that for small programs it is not sufficient to only measure in Watt per hour, because of the short run times. We tested this with three programs, a idle program where the only command was sleep, two programs who calculates the 10.000th prime one recursively and one who did this optimized. The results of this test are shown in figure 3.1. Here we see that the difference of 0.001 kWh is a large difference when working with numbers that are of scale 0.005 kWh. There isn't a clear difference between primes optimized and the sleep that takes close to the same amount of seconds as the primes optimized. Because of these results we choose to go with the power measurement method.

	Idle	**Prime**	**PrimeOpt**
Time	8m0.002s	7m58.443s	1m2.394s
Energy	0.033 kWh	0.037 kWh	0.005 kWh
Time	1m2.002s	7m59.611s	1m2.220s
Energy	0.004 kWh	0.037 kWh	0.005 kWh
Time	1m2.002s	7m58.503s	1m2.235s
Energy	0.005 kWh	0.036 kWh	0.004 kWh

Table 3.1: A test done using the energy measurement in kWh with three different programs

When using the power measurement method we get as a result a lot of measure points, where each point has a timestamp and a power value. An example of a measurement and these points are shown in figure 3.1. To calculated the total energy consumed during this program we need to calculate the surface beneath the graph. To do this we calculate the surface between two points and add all surfaces together. When two points do not have the same value we choose the average between the two to use for the surface. These calculation are following formula 3.1.

$$E = \sum (t_{n+1} - t_n) * \frac{p_n + p_{n+1}}{2} \tag{3.1}$$

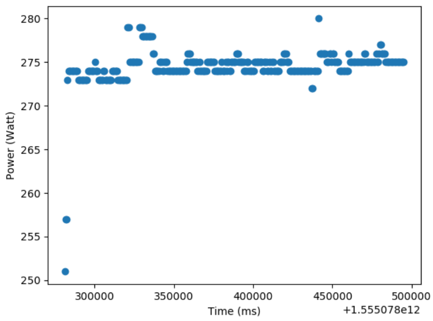

Figure 3.1: The example measurement of the primesOpt program

An overview of the energy measurement set-up can be seen in figure 3.2. Here we can see that from the DAS-5 a job is send to one of the energy measurement nodes. This is done by using the *prun* command and specifying which node to use. The job we send to this node is a bash script that runs all the programs in our data set. In this job we sleep for ten seconds in between the measurements of the programs to give the node time to go back to its idle state. After these ten seconds the measure script on the DAS-4 is started and then the program we want to measure for the energy consumption starts to run on the node. Immediately after the program ends the measure script is stopped. This measure script on the DAS-4 constantly sends a *snmp* message to the PDU and writes the values to an output file. Only six nodes on the DAS-5 are connected to the PDU and all these nodes have different hardware specifications. Therefor we need to separate the measurements to be able to compare the results. Due to the other nodes constantly being occupied, every program was run 27 times on *node28* and 22 times on *node29*. The hardware specifications for *node028* are a GPU node with an Nvidia Tesla K20 (with 6 GB onboard memory), an Xeon Phi and a michost and for *node029* are a GPU node with an Nvidia GTX980 (with 4 GB onboard memory) and an TitanX-Pascal.

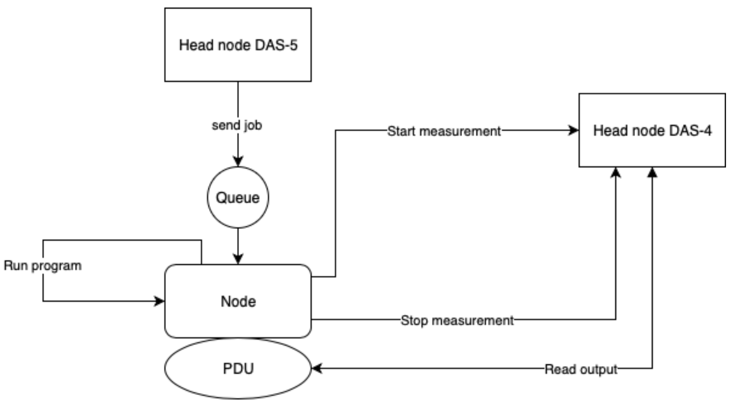

Figure 3.2: The structural overview of how we used the DAS to measure the energy consumption

The DAS-4 uses *snmpwalk* to retrieve the values form the PDU. The time it takes to retrieve the data from the PDU using *snmpwalk* is not constant. This means that we need to take into account that we have some loss of information and in some cases even too few measure points. When there are less then 30 energy measure points for a program that specific programs will run again until it has enough measure points. In figure 3.3 the average time between two measure points is shown alongside the standard deviation, the maximum and minimum time difference. There we can see that the average time between two measures is really fast, but the largest time period between two measures is really long.

```
The mean of the time difference: 0.102455347255
The standart deviation of the time difference: 0.0593552869984
The maximum time difference: 11.888
The minimum time difference: 0.067
```

Figure 3.3: The statistics of the time between two energy measurement in seconds

Chapter 4

Data

4.1 Programming languages

To find data we first need to decide which programming languages to choose for testing. We did this by looking into what the most popular and most used programming languages are. For this question it depends who you ask which result you get. Therefor four sources were used to determine which programming language to use. The four sources are indeed, git-hub, TIOBE and PYPL.

Indeed is a job search site. They looked at the percentage of jobs with a programming language in their name in the tech software category [6]. Thus the more job offering for a programming language the more popular that programming language is. The problem with this method is that job offerings do not show how many people work with a programming language, but only which programming language has a shortage in programmers. Based on data form September 2018 the top ten according to this method is Java, JavaScript, HTML, Python, C#, C++, XML, Ruby, PHP and Perl.

Git-hub is a version control system where multiple programmers collaborate in a project. They looked at the amount of pull requests made for that language [7]. The thought was that the language that programmers work a lot with on git is the most popular, but this is based on only the public repositories. In the first quarter of 2019 the top ten according to this method is JavaScript, Python, Java, Go, C++, Ruby, PHP, TypeScript, C# and C.

TIOBE is a software quality company. They looked at the amount of hits they got when searching "[Language] programming" on a lot of different search engines [8]. There are rules which a search engine needs to comply with for it to be used in the calculations and they also look a what type of hits they find to determine whether or not to use it in the calculations. The pitfall of this method is the favouritism for complex languages. When a language is more difficult to understand, more page of tutorials are needed and more questions about this language will be asked. As of April 2019 the top 10 according to this method is Java, C, C++, Python, Visual Basic .NET, C#, JavaScript, SQL, PHP and Assembly language.

PYPL index stands for the PopularitY Programming Language index. They look at how many times a language tutorial is searched [9]. This method also has favouritism for complex languages, where programmers need to use the tutorials a lot because of the difficulty. Based on data form April 2019 the top ten according to this method is Python, Java, JavaScript, C#, PHP, C/C++, R, Objective-C, Swift and Matlab.

When languages are in all the four top tens they were labelled as popular and these are the languages that are gonna be investigated, thus the languages Java, JavaScript, Python, C#, C++ and PHP. We also choose to investigate C and Ruby. C because it was in three of the four top tens and it seemed interesting to see the difference in the variations of C like C++ and C#. Ruby was in two of the top tens, but also 13th according to TIOBE and 12th in the PYPL index. Thus Ruby was close to be in all the top tens and therefor also chosen to be investigated.

4.2 Gathering data

To be able to compare different programs they need to have the same functionality. A source for programs that have the same functionality is *the computer language benchmark game* [10]. This benchmark game compares different programs and languages based on their speed, memory usage, zipped program size and CPU usage. They have ten different problems with a lot of programs from different languages. Everyone can submit a program if it holds to the two requirements. The requirements are that the program has the correct output and that it uses the same algorithm. This is important because we want to compare the way of writing a program and the difference in programming language, but not the difference in algorithm used. For every program used they also have the compiling steps listed and for every problem the correct output.

All the programs for the ten problems in our seven programming languages were downloaded. These programs were then tested to see if they could compile, run and have the correct output. This was all done on the DAS5 to make sure there were no local dependencies. All the programs that weren't compiling, gave a run error or gave the wrong output were excluded from the data set. There were three problems, Knucleotide, Pidigits and Regexredux, that for different programming languages use a partly different algorithm because of different library implementations. Therefor we excluded these problems from the data set. The problem Mandelbrot didn't have a working version of the JavaScript implementation, but this problem will still be used and we will just have an empty spot in our results.

Before running all the programs some needed to be compiled first. The compilation step of languages that have a separated compilation step were not included in the energy measuring. The reason for this is that a finished program is compiled once and then could be used multiple times. This does mean that the languages JavaScript, Python, PHP and Ruby have a bit of a disadvantage, because they don't have a separate compilation step. The compiler can nowadays do a lot of optimizations of the code. During this research we want to see the result of user decisions on the energy consumption. Therefore besides testing the programs with the flags used on the language benchmark game we also tested with as few flags as possible. This means that we removed all the optimization flags except the ones needed for the compilation. Also the compiler version of the different languages is important. To give a good pictures of were we stand today we need to use the most recent stable version and also the most commonly used one. Unfortunately it isn't that easy to update the language versions on the DAS. Therefore I used the versions that were already on their and these programming language versions are listed alongside the compiler used in table 4.1.

Languages	**Compiler**	**Version**
Java	javac	1.8.0_161
JavaScript	node	6.12.3
Python	python	3.4.5
PHP	php	5.4.16
C	gcc	6.3.0
C++	gcc	6.3.0
C#	mcs	5.10.1.20
Ruby	ruby	2.0.0

Table 4.1: All the different compilers and versions of the programming languages used

This all resulted in a large data set of 202 programs with 67 programs that had to run twice, once with and once without flags. Running all these programs takes about eight hours to complete. Unfortunately we were only allowed to use a node on the DAS for 30 minutes to run one script during working hours. This means that we had to run it at night and in the weekend, which limited the amount of data points that were measured. Of course running programs more often would give more accurate results, but this was just too time consuming.

Some programs were too fast to get a good amount of energy measurement points. To solve this problem those programs were run for multiple times during one measurement and then afterwards their energy consumption was divided by the amount of times the programs was run. Another problem that occurred was that retrieving a measure point takes a variable amount of time. This caused some programs to still not have a good amount of energy measurement points. Therefor I decided to set a minimum amount of 30 of measurement points needed in order for it to be used.

In the previous chapter we found that the energy consumption of a program is the sum of the CPU, memory and disk energy consumption [4]. To get a good view on what influences the energy consumption, our problems need to be diverse when it comes to these three categories. The problems that we looked into are called Binarytrees, Fannkuchredux, Fasta, Mandelbrot, Nbody, Revcomp and Spectralnorm. For the Binarytrees problem a lot of trees are allocated and deallocated in memory and thus this is a memory intensive task. The fannkuchredux problem does a lot of calculations on all permutations and is thus CPU intensive. The Fasta problem creates and saves a large DNA sequence and is memory intensive. For the Mandelbrot problem a large bitmap is saved and thus is it memory intensive. The Nbody problem models the orbit of Jovian planets and is CPU intensive. The Revcomp problem reads a DNA sequences line by line, transforms them and writes the result to output. Therefor the Revcomp problem is disk intensive. For the Spectralnorm problem a lot of calculations are done on a large matrix and is thus CPU intensive. A more extensive explanation of the problems can be found on the computer language benchmark game website [10]. An overview of category and problem is shown in table 4.2.

Category	**Problems**
CPU	Fannkuchredux, Nbody, Spectralnorm
Memory	Binarytrees, Fasta, Mandelbrot
Disk	Revcomp

Table 4.2: The job intensive categories the different problems are in

Chapter 5

Statistical tests

5.1 Normal distribution

When choosing a statistical test we have to take into consideration the preconditions for using the test. The most common precondition is that the data follows a normal distribution. The Shapiro-Wilk test has more power then the other normality tests [11] and therefor this test was used. The Shapiro-Wilk test calculates a statistic by dividing the the summation to the power of two of every point times a coefficient by the summation of every point minus the mean to to the power of two [12]. This formula is shown in equation 5.1. The null-hypothesis of the Shapiro-Wilk test is that the data is normally distributed. This means that when the null-hypothesis gets rejected the data not a normal distribution follows. When testing if the energy measurements from a single program on a single node follow the normal distribution using the Shapiro-Wilk test we get that not all programs measurements are normally distributed. From the 269 different programs 183 were not normally distributed on node029 and 38 on node028. Therefor we need to choose statistical test that don't assume the data to be normally distributed.

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.1}$$

5.2 Same distribution

When comparing different programs, that are in the same language and have the same functionality, we need to find out if there is a significant difference concerning the energy consumption. This can be tested by looking at the two different distributions and testing if they are from the same population. A statistical test that compares two distribution and test there equality is called the Mann Whitney U test [13]. Here the null hypothesis is that the distributions are from the same population. The Mann Whitney U test looks if the chance that a random variable from the first distribution is greater than a random variable from the second distribution. When this chance is 50%, then the two different distributions belong to the same population. Another test that we could have used was the students t-test. This test however has the preconditions that the means of the two distributions should follow the normal distribution and the variance of the two distributions should be equal. Our data doesn't match these precondition and therefor we chose to use the Mann Whitney U test.

There are two versions of the Mann Whitney U test, the one-sided and the two-sided test. For the two-sided Mann Whitney U test the alternative hypothesis is that the distributions are from the same population, but if you want to know the direction of this comparison you need to use the one-sided Mann Whitney test [14]. Which means that the alternative hypothesis for the one-sided Mann Whitney U test is that the first distribution is stochastically larger than the second distribution [14]. Because rejecting the null hypothesis holds more power than not rejecting the null hypothesis, a one-sided Mann Whitney U test was used twice. We tested if the first distribution was stochastically larger than the second distribution and if the second distribution was stochastically larger than the first. If both these tests reject the null hypothesis then we know that the distributions are from the same population.

5.3 Correlation

A commonly known method for calculating the correlation is called the Pearson coefficient. The Pearson coefficient uses the covariance of two variables to calculate its correlation score [15]. For this method the following assumptions should hold, the correlation is linear and the data follows a normal distribution. Our data however does not meet these assumptions for all programs and therefor we needed to look at a different method. The Kendall Tau coefficient calculates a ranked correlation coefficient and does not assume that the data follows a normal distribution [16], therefor this method was used. The Kendall Tau method looks at how many pairs of points follow the same order. For example points (x_i, y_i) and (x_j, y_j) follow the same order if $x_i > x_j$ and $y_i > y_j$ or if $x_i < x_j$ and $y_i < y_j$. Because every pair needs to be checked this could get computational heavy for large number of points, luckily that is not the case in this research project.

The coefficient is in the range of one to minus one, where at zero there is no correlation and at one or minus one there is. But what about the scores in between? According to the standard Guilford scale the range $0 - 0.4$ means no correlation, $0.4 - 0.7$ means moderate correlation, $0.7 - 0.9$ means high correlation and $0.9 - 1$ means very high correlation [17]. Because the minus in the coefficient shows which direction the correlation is in, we can use the absolute value to determine in which range of correlation a negative number is.

5.4 Anomaly detection

An anomaly, also called outlier, is a point that does not behave like the rest of the points in the results. These anomalies can be detected by using a clustering algorithm like DBSCAN [18]. The DBSCAN algorithm goes though every point and counts how many other points are in a predefined area around the point. If there are more or equal to the minimum amount of points needed in this area then this point is labelled as a core point. Points that are not core points but are in the predefined area around a core point are labelled border points, all the other points are anomalies [19]. This is visualized in figure 5.1, where all red dots are core points, yellow dots border points and blue dots anomalies. The minimum amount of points needed around a point and the predefined area around a point are the input variables of this algorithm, which is the disadvantage of this method.

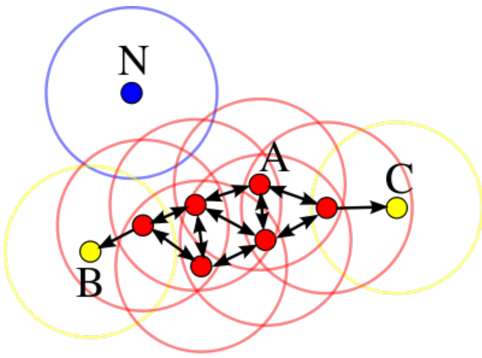

Figure 5.1: The labels according to DBSCAN algorithm with minimum points is four, where the red dots are core points, yellow dots border points and blue dots anomalies

5.4.1 Input variables

According to [19] we can set the minimum amount of points a core point needs in his area to four for two dimensional data. However for the area we need to calculate the radius, also called the eps variable. This eps is calculated by first calculating the distance to the fourth nearest neighbour for every point. These distances are then sorted and plotted. The value for eps can then be read from the graph looking at the first valley [19]. In our implementation we calculated this eps automatically for every single program. To retrieve the first valley we looked at the differences between the slopes around a point. The point with the biggest difference in slopes was in the valley and was used as our eps variable for that specific program.

These two parameters have a big influences on how the clusters are divided and which points are labelled as an anomaly. When the minimum amount of points gets smaller more clusters will be formed and when it gets larger less clusters will be formed, assuming eps stays the same. When eps gets smaller more points will be labelled as anomalies and when eps get larger less points will be labelled as anomalies, assuming the minimum points parameter stays the same.

5.5 Clustering

When trying to find clusters in our data we were searching for a specific amount of clusters. Because the amount of clusters was known before clustering the k-means clustering algorithm was used. This algorithm begins by randomly assigning means for the amount of clusters specified. Then all the points are passed and divided into a cluster based on which mean has the shortest euclidean distance. After this the mean of the clusters is calculated and these will be the new means. This will iterate till a local maximum is found or the maximum amount of iterations has passed.

Chapter 6

Results

In this chapter, we present the results of our experiment.

table with languages table with programs averages figure per problem example figure of a problem with different programs in same language anomalies same distribution

without flags, takes more energy

For binarytrees: php cosumes the most, python ruby and csharp also relatively bad For fannkuchredux: php python ruby really bad, javascript also not so good For fasta: php python ruby javascript not good For mandelbrot: python php ruby not good For nbody: python php ruby not good For revcomp: not one that really stands out, though c++-noflags has a huge range thus potentially really bad For spectralnorm: python least good, php ruby also not good

Overall php, python and ruby not good conform the energy consumption

Chapter 7

Discussion

In this chapter, we discuss the results of our experiment(s) on ...

Finding 1: Highlight like this an important finding of your analysis of the results.

Refer to Finding 1.
Risks: -compiler maybe expects people to program stupidly

Chapter 8

Related work

To get an overview of the related research I made a table, table 8.1, comparing the different papers. Most research is about reducing the energy consumption of a specific piece of hardware, for example scheduling on a multi-core processor. There are also some papers about reducing the energy consumption in the software and the decision making process. There is also some research done on the energy consumption of software, but their research goal was to estimate the energy consumption. Therefore what I miss and want to look into is to research if there is a good way of writing code regards the energy consumption.

Papers	Type of Research	Unit of Analysis	Goal
[2]	Controlled Experiment	Deployment strategies, releases and use case scenarios	Finding optimal energy consumption
[20]	Case Study & Controlled Experiment	HPC bag of task applications	Finding optimal energy consumption
[4]	Case Study	Small functions	Estimating energy consumption
[21]	Case Study & Controlled Experiment	Small functions	Estimating energy consumption
[22]	Case Study	Task on complex micro-architectures	Estimating worst-case energy consumption
[23]	Empirical Study	Six commonly used refactorings	Finding impact of refactorings on energy consumption
[24]	Controlled experiment	Programming languages	Rank programming languages based on speed, memory usage and energy consumption
[25]	Case Study	Multi-core processor scheduling	Efficient workload partitioning
[26]	Case Study	Cache storage management algorithms	Power aware cache management
[27]	Case Study	Java applications	Framework to automate decision-making support regarding energy consumption
[28]	Case Study	Software process	Two level green software model

Table 8.1: Overview of related research

8.0.1 Estimating energy impact of software releases and deployment strategies: the KPMG case study [2]

This paper looks at the impact of releases and deployment strategies of a software product on the energy consumption. They used a controlled experiment where the variables they changed where deployment strategies, releases and use case scenarios. The variables they measured during their tests were power consumption and execution time. They saw that both the releases and deployment strategies impacted the energy consumption and that this impact was influenced by which use case scenario they used. Therefore they concluded that there is no absolute optimal option for releases and deployment strategies with respect to energy consumption. They also found that the execution time had a bigger impact on the energy consumption than the power consumption, because of the low variability in power consumption.

8.0.2 The Impact of Source Code in Software on Power Consumption [4]

This paper looks at different techniques to measure the power consumption. Then they propose a model to measure the power consumption and they used this model in their implementation named *Tool to*

Estimate Energy Consumption (TEEC). They test their implementation against a power meter, but they do not mention how accurate their implementation is. The figure they use at the validation is also not clear, they just state that it shows the same behaviour as TEEC. They find that the power consumption of unoptimized code is higher and has a longer execution time than the optimized code. They do not mention it but looking at their graphs, the unoptimized and optimized code seem to have the same peaks only on different time steps for the optimized code is faster.

8.0.3 Energy Consumption Analysis of Programs based on XMOS ISA-Level Models [21]

This paper estimates the energy consumption by developing a model which can be applied at instruction set simulation level. This was done by designing a translation from instruction set architecture code to Horn- clause representation and this model is called in the paper *Instruction Set Simulation* (ISS). They also use the CiaoPP general resource analysis framework, which is low level, to model the energy consumption. They named it *Static Resource Analysis* (SRA) in the paper. In their experiments they only use small functions to test and the results were compared to a mathematical equation. The found that the ISS function is less accurate when the value of N increases and that the SRA function is only not accurate for small value of N. Here N is the input value of the function that is tested for its energy consumption.

8.0.4 E-BATS: ENERGY-AWARE SCHEDULING FOR BAG-OF-TASK APPLICATIONS IN HPC CLUSTERS [20]

This paper looks at the scheduling of bags of task application in High performance Computing (HPC). They delved into the trade-off between energy consumption and performance by finding a optimal point for both variables. This was calculated by designing a dynamic Hill Climbing algorithm. Their algorithm uses less then 12% of the resources an exhausted search would use to find a majority of points close to the optimal for the trade-off in 10 of the 12 scenarios. They validated their algorithm by implementing it and running a wide range of HPC bag of task applications. They found that the estimations of their algorithm have an error below 5%.

8.0.5 SEEDS: A Software Engineer's Energy-Optimization Decision Support Framework [27]

This paper implements a framework that automatically optimizes the energy consumption of a Java software project. The framework does this by running multiple different given options and testing which option consumes the least amount of energy. Thus as input the framework needs a list of possible changes. Because the framework needs possible changes we don't know if the resulting code is the most energy efficient version, only that it is more energy efficient then the input. They showed that by letting their framework chose which library to use they reduced their energy consumption by 17%.

8.0.6 How Do Code Refactorings Affect Energy Usage? [23]

This paper addresses the lack of information about the energy impact of code refactorings. They did this by testing the energy impact of 197 projects when the using six commonly used refactorings. From this test they found that refactorings can influence the energy consumption. Also they find that one refactoring does not necessarily have the same influence on the energy consumption when used with different projects.

8.0.7 Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management [26]

This paper tries multiple algorithms for storage cache management to decrease the energy consumption. One algorithm is an offline greedy algorithm and they also propose an online algorithm. They evaluate their algorithms by simulating a complete storage system, enhancing the DiskSim simulator. Their greedy algorithm results in 16% less energy used then the LRU algorithm. They also find that the cache policy write-back can use 20% less energy then write-through.

8.0.8 A Green Model for Sustainable Software Engineering [28]

This paper makes a two level green software model. The first level is about making the software process more energy efficient. This new process is a hybrid of the sequential, iterative, and agile development processes. The second level is about the role software tools can have on improving the energy efficiency of software. They also discus the relationship between the two levels.

8.0.9 Estimating the Worst-Case Energy Consumption of Embedded Software [22]

This paper estimates the worst-case energy consumption of a task on complex micro-architectures. This is important for battery-operated embedded devices, where we don't what the battery to drain empty before a critical task is completed. They test their result against a commonly used benchmark and they find that their values have at most 33% difference with the benchmark.

8.0.10 Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core [25]

This paper proposes an algorithm to makes sure all cores in a multi-core processor have the same workload. This is reducing energy consumption because multiple single core processors consume more energy.

8.0.11 Energy Efficiency across Programming Languages [24]

This paper tries to find a connection between the speed, memory usage and energy consumption of a programming language. They do this by choosing the fastest implementation of the exact same algorithm, defined in the computer language benchmarks game, in different programming languages. From these programs they measured the execution time, memory usage and energy consumption. They used Intel's Running Average Power Limit (RAPL) tool to measure the energy consumption and for the memory usage and execution speed they used the *time* command available in Unix-based systems. They find that a faster programming language does not necessarily have a lower energy consumption and memory usage does not relate to energy consumption. A big problem with this paper is that in their threads to validity paragraph they defend their implementation instead of stating what could be wrong with their implementation. My main problem with this paper is that they compare languages based on the fastest solution in some competition and these are not comparable. Because there can be languages where the fastest solution given in the competition is not the fastest at all.

8.0.12 Accurately Simulating Energy Consumption of I/O-intensive Scientific Workflows [29]

This paper looks at two applications that are I/O heavy. Different tasks were run for these applications and the energy consumption was measured. Another variable in their experiments was the amount of cores used. They compared the energy values measured with a commonly used estimation scheme for the energy consumption which only looks at the CPU utilization. They noticed a difference and the correlation between power consumption and CPU utilization was close to zero. The reason for this was that the estimation scheme didn't factor in the energy consumption of I/O operations. Therefor they came up with a scheme that factors in the CPU utilization and the I/O operations. This scheme used values from the tests to put in different values in the formula and was tested against the two applications and they found a small error. This is an issue for me, because you expect the data you used to create a model to fit the model. A better approach would have been to use one application for calculating the values and the other for the validation.

Chapter 9

Conclusion

9.1 Future work

Acknowledgements

If so inclined, thank people. Kees Verstoep

Bibliography

[1] W. Van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ict electricity consumption from 2007 to 2012", *Computer Communications*, vol. 50, pp. 64–76, 2014.

[2] R. Verdecchia, G. Procaccianti, I. Malavolta, P. Lago, and J. Koedijk, "Estimating energy impact of software releases and deployment strategies: The kpmg case study", in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, IEEE Press, 2017, pp. 257–266. DOI: `10.1109/ESEM.2017.39`.

[3] T. Mens and T. Tourwé, "A survey of software refactoring", *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[4] H. Acar, G. Alptekin, J.-P. Gelas, and P. Ghodous, "The impact of source code in software on power consumption", *International Journal of Electronic Business Management*, vol. 14, pp. 42–52, 2016.

[5] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term", *Computer*, vol. 49, no. 5, pp. 54–63, 2016.

[6] A. D. Rayome, *The 10 most in-demand programming languages of 2018*, Accessed: 16-04-2019, Sep. 2018. [Online]. Available: `https://www.techrepublic.com/article/the-10-most-in-demand-programming-languages-of-2018/`.

[7] C. Zapponi, *Githut 2.0 - a small place to discover languages in github*, Accessed: 16-04-2019, Apr. 2019. [Online]. Available: `https://madnight.github.io/githut/#/pull_requests/2019/1`.

[8] TIOBE, *TIOBE index for april 2019*, Accessed: 16-04-2019, Apr. 2019. [Online]. Available: `https://www.tiobe.com/tiobe-index/`.

[9] P. Carbonnelle, *PYPL popularity of programming language*, Accessed: 16-04-2019, Apr. 2019. [Online]. Available: `http://pypl.github.io/PYPL.html`.

[10] I. Gouy, *The computer language benchmarks game*, Accessed: 26-04-2019. [Online]. Available: `https://benchmarksgame-team.pages.debian.net/benchmarksgame/`.

[11] N. M. Razali, Y. B. Wah, *et al.*, "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests", *Journal of statistical modeling and analytics*, vol. 2, no. 1, pp. 21–33, 2011.

[12] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)", *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[13] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other", *The annals of mathematical statistics*, pp. 50–60, 1947.

[14] N. Nachar *et al.*, "The mann-whitney u: A test for assessing whether two independent samples come from the same distribution", *Tutorials in quantitative Methods for Psychology*, vol. 4, no. 1, pp. 13–20, 2008.

[15] E. S. Pearson, "Some notes on sampling tests with two variables", *Biometrika*, pp. 337–360, 1929.

[16] M. G. Kendall, "A new measure of rank correlation", *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[17] J. Sandler, "Jp guilford: Fundamental statistics in psychology and education (book review)", *The International Journal of Psycho-Analysis*, vol. 38, p. 432, 1957.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey", *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.", in *Kdd*, vol. 96, 1996, pp. 226–231.

[20] A. V. Filip, A.-M. Oprescu, S. Costache, and T. Kielmann, "E-bats: Energy-aware scheduling for bag-of-task applications in hpc clusters", *Parallel Processing Letters*, vol. 25, no. 03, p. 1 541 005, 2015.

[21] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. Lopez-Garcia, N. Grech, M. V. Hermenegildo, and K. Eder, "Energy consumption analysis of programs based on xmos isa-level models", in *International Symposium on Logic-Based Program Synthesis and Transformation*, Springer, 2013, pp. 72–90.

[22] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the worst-case energy consumption of embedded software", in *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, IEEE, 2006, pp. 81–90.

[23] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2014, p. 36.

[24] 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*, ACM, 2017, pp. 256–267.

[25] M. Zakarya, N. Dilawar, M. A. Khattak, and M. Hayat, "Energy efficient workload balancing algorithm for real-time tasks over multi-core", *World Applied Sciences Journal*, vol. 22, no. 10, pp. 1431–1439, 2013.

[26] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing energy consumption of disk storage using power-aware cache management", in *Software, IEE Proceedings-*, IEEE, 2004, pp. 118–118.

[27] I. Manotas, L. Pollock, and J. Clause, "Seeds: A software engineer's energy-optimization decision support framework", in *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 503–514.

[28] S. S. Mahmoud and I. Ahmad, "A green model for sustainable software engineering", *International Journal of Software Engineering and Its Applications*, vol. 7, no. 4, pp. 55–74, 2013.

[29] R. Ferreira da Silva, A.-C. Orgerie, H. Casanova, R. Tanaka, E. Deelman, and F. Suter, "Accurately simulating energy consumption of I/O-intensive scientific workflows", in *2019 International Conference on Computational Science (ICCS)*, 2019.

Appendix A

Non-crucial information