

The comparison of the energy consumption of different programming languages and programs

Lukas Koedijk

lukaskoedijk@gmail.com

June 28, 2019, 49 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

1	Introduction	4
1.1	Problem statement	4
1.1.1	Research questions	4
1.1.2	Research method	5
1.2	Contributions	5
1.3	Outline	6
2	Background	7
2.1	Statistics	7
3	Energy measurement	8
4	Data	11
4.1	Programming languages	11
4.2	Gathering data	12
5	Statistical tests	14
5.1	Normal distribution	14
5.2	Same distribution	14
5.3	Correlation	15
5.4	Anomaly detection	15
5.4.1	Input variables	16
5.5	Clustering	16
6	Results	17
6.1	Anomalies	17
6.1.1	Two clusters	17
6.2	Programming Languages	18
6.3	Programs	20
6.3.1	Code level	21
6.4	Additional Findings	21
6.4.1	Hardware	21
6.4.2	Time	21
7	Discussion	23
8	Related work	24
8.1	Finding optimal energy consumption	24
8.2	Estimating energy consumption	25
8.3	Measuring energy consumption	25
8.4	Hardware	26
8.5	Other	26
9	Conclusion	27
9.1	Conclusion	27
9.2	Future work	27
	Bibliography	29

Appendix A Node28	31
Appendix B Node29	37
Appendix C Correlation	43

Chapter 1

Introduction

Currently more and more people are concerned with global warming. Global warming is partly the 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 we miss and are going 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 we 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 we 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 we find when answering the second existence question, we want to look into what this difference is on code level. To do this we need to answer the descriptive/comparative question **what is the difference on code level between software project that are in the same programming language and have the same functionality, but have a difference in the energy consumption?** This question is only useful when there is a difference in the energy consumption of software projects in the same programming language that have the same functionality.

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.

1.1.2 Research method

As method we use a controlled experiment. In this experiment we 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.

Data

For the first research question we need programs that have the same functionality, but are written in a different programming language. Possible resources for such programs are library code, interview code, student assignments and code from competitions solving math problems. We first looked at library code, but found that libraries most of the time don't use the same algorithm. This is a problem because we don't want to compare algorithms, but the way a programmer writes code. Therefore we used the *the computer language benchmark game* [3] as a source for programs that have the same functionality and use the same algorithm for solving the different problems. For the second research question we need programs that have the same functionality and are written in the same language. Here we could also use *the computer language benchmark game* as for most languages there are multiple programs listed.

When proving that all the projects have the same functionality we could have defined properties for the input and output and test if these properties hold for all the projects [4], but *the computer language benchmark game* already makes sure that the programs have the same functionality. Thus we don't have to do this ourselves.

To prove that the projects have different values for the energy consumption we use the one sided Mann Whitney U test. This test has as its hypothesis that two distributions are from the same population.

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 [5]. The measurements can be done with a hardware or software approach. A hardware approach is more accurate but also more expensive [5]. We want to use a hardware approach and luckily as a student of the UvA we have access to the DAS-4 and the DAS-5. The DAS is a distributed system that can also do energy measurements [6]. The nodes that can measure the energy are located at the VU cluster. To measure the energy we need to run a job on the DAS-5 and use the DAS-4 to measure the energy consumption. When running a job we need to specify which node we want to run the job on, because not all nodes can be measured for the energy consumption.

Code level

When there is a difference in the energy consumption of different software projects in the same programming language we will look at the different projects on code-level. Here we will try to find what is causing a project to have a lower or higher energy consumption. There are too many program combinations to look at, thus we need to make a selection. We choose to look at the languages that had the biggest range in energy consumption for all the problem, these languages are Python, Ruby and PHP. Then we can look for every problem to the two programs that differ the most for those three programming languages. We will look at the program combinations side by side and write down the differences that we see. Then we look at these differences and see if some are occurring more often than others. These findings will then be tested by writing our own two versions, where before testing we think one is written badly regards the energy consumption and one that is written good. We then run the two versions and check if the good version has indeed a lower energy consumption using the Mann Whitney U test.

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. Comparison of languages energy consumption

4. 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

2.1 Statistics

Not understanding the terminology of statistics may lead to confusion, therefore here are some basic principles in statistics. When performing a statistical test we have a null hypothesis and an alternative hypothesis. Every test has its specific null and alternative hypotheses and the goal of the test is to reject the null hypothesis. Rejecting the null hypothesis is done by looking at the resulting p -value of the test. The p -value is the chance that the value of the statistical test occurs if the null hypothesis is true on a zero to one scale. We therefore reject the null hypothesis if we think this chance is too low, thus below a certain threshold. This threshold is determined beforehand and is called the *alpha*-value, a common value for *alpha* is 5%. When we cannot reject the null hypothesis, thus the p -value is not below 0.05, it does not necessarily mean that the null hypothesis is true. It could be the case that the test is not powerful enough or your data size is too small.

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 [5]. 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 [5]. 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 [6]. This PDU is from Racktivity and has an accuracy of 1%.

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 *snmpwalk* 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 calculate 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. When looking at the different nodes we see that they all have a different idle power state. To help compare between the nodes and reduce the difference between the measurement moments we removed the idle state from the energy consumption. To do this we measured the idle state one minute before and after a single run of all the programs. The average power was then calculated and subtracted from every measure point. These calculations resulted in the following formula 3.1.

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

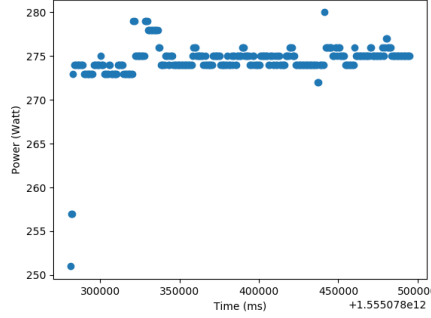


Figure 3.1: An example measurement of the primesOpt program.

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 *node28* are a GPU node with an Nvidia Tesla K20 (with 6 GB onboard memory), an Xeon Phi and a michost and for *node29* are a GPU node with an Nvidia GTX980 (with 4 GB onboard memory) and an TitanX-Pascal.

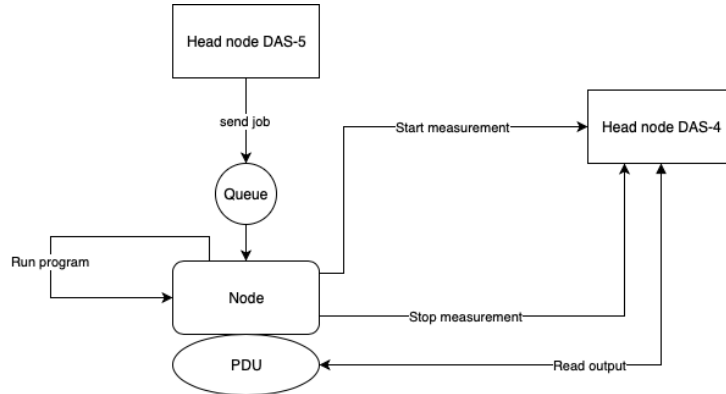


Figure 3.2: The structural overview of how we used the DAS to measure the energy consumption. From the head node of the DAS5 we send jobs to the queue of a certain node we want to measure on. This job will then run on that node when its free. In the job we send a message to the DAS4 to starts its energy measurement, then we run the programs and after that we send a message to the DAS4 to stop the energy measurement. The DAS4 retrieves the energy measurements from the PDU via constant smtp requests.

The DAS-4 uses *snmpwalk* to retrieve the values from 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 than 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: Some 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. Therefore 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 [7]. 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 from 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 [8]. 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 [9]. There are rules which a search engine needs to comply with for it to be used in the calculations and they also look at 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 pages 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 [10]. This method also has favouritism for complex languages, where programmers need to use the tutorials a lot because of the difficulty. Based on data from 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 therefore 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* [3]. 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. Therefore 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 picture of where 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 there 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 [5]. 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 [3]. 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 than the other normality tests [11] and therefore this test was used. The Shapiro-Wilk test calculates a statistic by dividing the summation to the power of two of every point times a coefficient by the summation of every point minus the mean 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 does not follow a normal distribution. For this test we used an *alpha*-value of 0.01, because distributions that are close to normally distributed can still be used in some of the statistical tests. 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. Therefore we need to choose statistical test that don't assume the data to be normally distributed.

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

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 distributions and tests their 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 student's 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 preconditions and therefore 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, with an *alpha*-value of 0.05. If both these tests reject the null hypothesis then we still don't know for certain that the distributions are from the same population.

We also used the two sample Kolmogorov-Smirnov test to find out if two distributions are from the same population. This test also has as its null hypothesis that the two distributions are from the same population and we used an *alpha*-value of 0.05. The Kolmogorov-Smirnov test compares the empirical distribution functions of the two distributions.

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 therefore 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], therefore 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 – 0.2 means slight correlation, 0.2 – 0.4 means low 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 through 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.

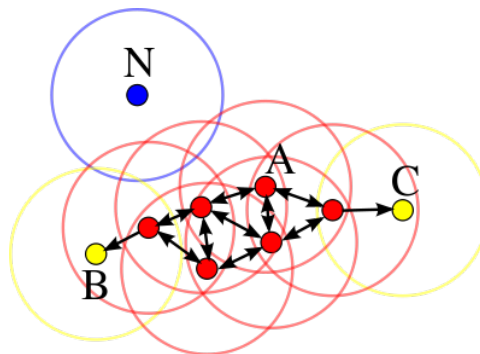


Figure 5.1: The labels according to DBSCAN algorithm with minimum points is four, where the red dots are core points, yellow dots border points and blue dots anomalies. Core points are points that have within a certain distance a minimum amount of points. Border points don't have the minimum amount, but are within reach of a core point. Anomalies don't have the minimum amount of points in a certain area and are not reachable from a core point. (source: <https://en.wikipedia.org/wiki/DBSCAN>)

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. The maximum amount of iterations was left to the default of 300 form the *sklearn* python packages.

Chapter 6

Results

6.1 Anomalies

We checked for anomalies using DBSCAN. An example of the results we got from this approach is shown in figure 6.1. There we can see in the top graph the distribution of measurements for a single program on one of the two nodes on the DAS. The red dots are the measurements DBSCAN labelled as an anomaly. In the bottom graph we see from every measurement point its distance to its fourth nearest neighbour. This is sorted and the red dot is the choice of the input variable of DBSCAN called *eps* and is calculated by finding the first valley.

For *node28* there were a total of 345 anomalies from 7263 measurement points, which is roughly 4.8%. In the measurements from *node29* there were a total of 305 anomalies from 5918 measurement points found, which is roughly 5.1%. Even though there are more anomalies found then programs, there was not for every program an anomaly detected. All the anomalies found were removed from further results.

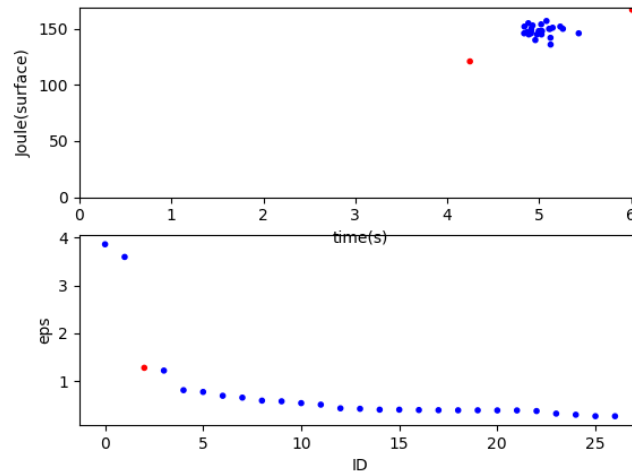


Figure 6.1: In the top graph the distribution of measurements from program java-3 on the Binarytrees problem on *node28* is shown. The bottom graph shows the sorted fourth nearest neighbour graph. The red dots at the top are measurements labelled as an anomaly and at the bottom is the choice for the *eps*-value.

6.1.1 Two clusters

During the processing of the information from the anomaly detection we saw only on *node29* that some programs appeared to have two clusters. We looked further into this and found that where we found two clusters it seemed to follow an ordering based on measurement moment. With clusters based on measurement moment we mean that in the first cluster all measurements were done before a certain

date and in the second cluster after that date. The DBSCAN algorithm found for *node28* seven times two clusters in the measurements for a single program where zero follow measurement moment and 58 times for *node29* where 53 follow measurement moment. At *node29* for all of these 53 programs the null hypothesis that there form the same distribution can be rejected. We then instead of letting a clustering algorithm cluster first and then check if the clusters follow the measurement moment, clustered ourselves based on the measurement moment. We find that for *node28* 218 programs and for *node29* 41 programs the null hypothesis that the two measurement moment clusters are from the same population can't be rejected. Because of the large amount of programs where the clusters were not from the same population at *node29*, we decided to run another clustering algorithm named k-means. We set the amount of clusters the k-means algorithm will search for at two, because we wanted to find these two measurement moment clusters. We then found that there were zero programs that had clusters based on measurement moment for *node28* and 63 programs for *node29*. These 63 programs all rejected the null hypothesis that the two clusters were form the same population. When looking at the programming languages and problems these 63 programs entail, we saw a good representation of the programming languages and different problems. Nothing that would make us suspect some programming language or problem specific cause for this difference. There still could be some other commonality between these programs, but it is too difficult to find this between the 63 programs. Therefor this is were we left it with on *node28* no difference in the measurement moment and on *node29* only a difference in a part of the programs.

6.2 Programming Languages

After all the measurements were done we wanted to compare the programming languages. This was done by filtering out one problem and selecting every measurement of a program that solves this problem. This resulted in a lot of points per language and these are plotted in a box-plot. There are seven problems and thus also seven box-plots. An example of one of these box-plots is shown in figure 6.2. Here the box is from the first quartile till the third quartile and a line at the median. The dots are measurement points laying outside the fences that are located at $Q1 - 1.5(Q3 - Q1)$ and $Q3 + 1.5(Q3 - Q1)$. The whiskers show the range of points that are outside the box but inside the fences. All the graphs are shown in appendix A for *node28* and appendix B for *node29*.

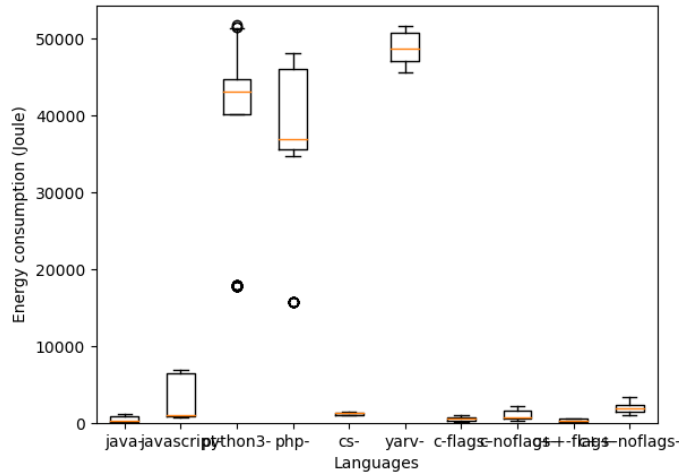


Figure 6.2: The box plot of the different programs in a programming language for the problem Fannkuchredux on *node28*.

In the box-plot we can't always clearly see which programming language is using less energy. To find out what the relationship is between two programming languages we used twice the one-sided Mann Whitney U test. This we calculated for every language with every other language and because of the fact that if one is smaller the other should be larger this table should be inverted around the diagonal zero's, thus a $+$ changes into a $-$. An example of this is shown in table 6.1. In that table a $+$ means that the programming language on the row is performing better than the programming language on the

column, i.e. the programming language on the row consumes less energy. The - means the opposite, a 0 means equal and *unknown* means that both one-sided Mann Whitney U test could not be rejected. All the tables are shown in appendix A for *node28* and appendix B for *node29*.

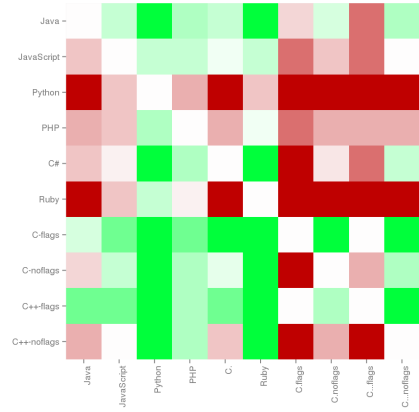
	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	-	+	-	+
JavaScript	-	0	+	+	+	+	-	+	-	+
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	-	-	-	-	-
C#	-	-	+	+	0	+	-	+	-	+
Ruby	-	-	+	+	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	Unknown	+
C-noflags	-	-	+	+	-	+	-	0	-	+
C++-flags	+	+	+	+	+	+	Unknown	+	0	+
C++-noflags	-	-	+	+	-	+	-	-	-	0

Table 6.1: The comparison of the different languages for the Nbody problem on *node28*. A + means that the language on the row has a lower energy consumption then the language on the column, the opposite for -, and the *Unknown* means that we could not reject the null hypothesis.

Using these tables we can see which language is performing better compared to others, but only for a single problem at a time. To give a total overview of problems and programming languages we calculated a score for every programming language combination. An one was added when there was a plus, an one was subtracted when there was a minus and nothing was added or subtracted in the case of a zero or unknown. With the use of these scores we made a heatmap, where green means a high score and red a low. This heatmap is shown in figure 6.3. Here the programming languages with a lot of green in their row, or a lot of red in their column, are performing better than most programming languages. We also wanted to see which programming language combination resulted the most in an *unknown*, thus which combination could not reject the null hypothesis the most. For this we also created a heatmap where a one was added for every *unknown* found. This resulted in figure 6.4.

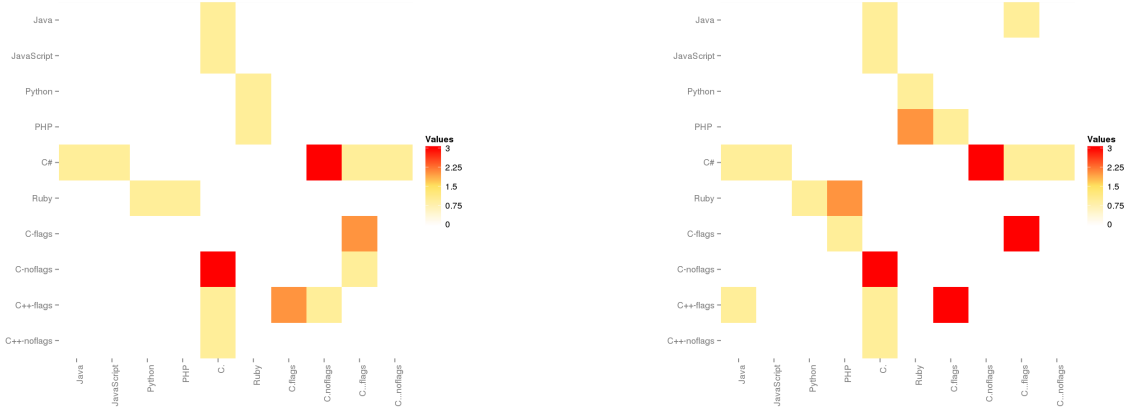


(a) The heatmap of *node28* of the relationship between the programming languages for all problems combined. A green box means that the language on the row has a lower energy consumption over all the programs than the language in the column, red is the reverse.



(b) The heatmap of *node29* of the relationship between the programming languages for all problems combined. A green box means that the language on the row has a lower energy consumption over all the programs than the language in the column, red is the reverse.

Figure 6.3



(a) The heatmap of *node28* of the the amount of times we could not reject the null hypothesis for the relationship between the programming languages for all problems.

(b) The heatmap of *node29* of the the amount of times we could not reject the null hypothesis for the relationship between the programming languages for all problems.

Figure 6.4

6.3 Programs

When comparing programs that are written in one language and solve the same problem, we can also use a box-plot. An example of such a box plot is shown in figure 6.5. Here we can't always see which program is performing better concerning the energy consumption. Therefor we also used the one-sided Mann Whitney U test twice here. For some combination of two programs we weren't able to reject both the Mann Whitney U tests. All these program combinations and their p -values are listed in table A.8 for *node28* and in table B.8 for *node29* in respectively appendix A and B. There were a total of 24 program combinations on *node28* and 59 on *node29* where the null hypotheses could not be rejected. We tried to also use Kolmogorov-Smirnov test to see if we would get less cases of uncertainty. This resulted in a total of 33 program combinations for *node28* and 61 for *node29*. But when we look at the overlap we see that there are only 24 program combinations for *node28* and 50 for *node29*. Thus this second test removed the uncertainty of nine program combinations on *node29*.

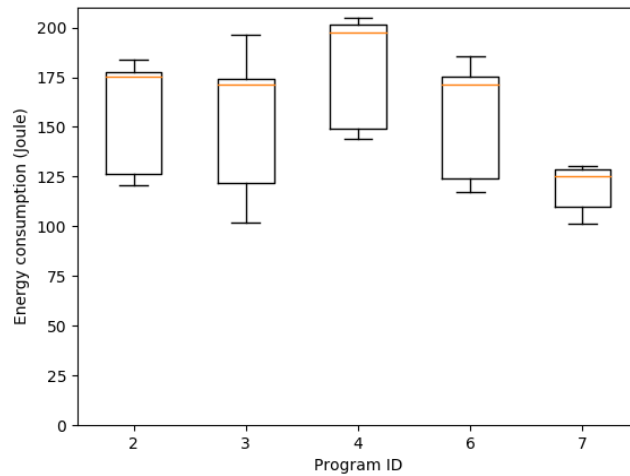


Figure 6.5: The box-plot for the programming language Java comparing the measurements of different programs that solve the Binarytrees problem and are measured on *node28*.

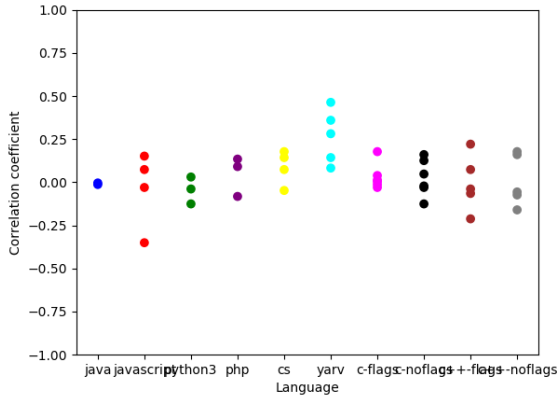
6.3.1 Code level

When looking at programs with a difference in the energy consumption from a code level perspective we find that.....

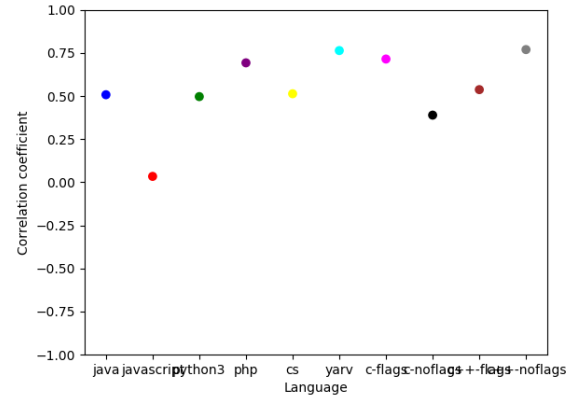
6.4 Additional Findings

6.4.1 Hardware

Our measurements were run on two different nodes, which had different hardware specifics. To find out how big of an influence the node is we wanted to compare the measurements between the two node for a single program. We first used the one-sided Mann Whitney U test and found that for two programs we couldn't reject the null hypothesis that their from the same distribution. For most programs their energy consumption depends on which node they were run on. To see if only the height is different, but still follow the same trend we calculated the correlation. This correlation was calculated between the different programs and an example of the results is shown in figure 6.6a. In this figure we have positive and negative numbers, this only shows the direction of the correlation. Thus the correlation number 0.7 and -0.7 have the same correlation strength. All the other graphs of the other problems are shown in appendix C. In all these graphs we see that most scores are low. Because the distribution of the programs are small we decided to check the correlation of all the measurements in one language for one problem. These are also all listed in C and an example is show here in figure 6.6b. There we see that in most cases the Kendall correlation score is higher.



(a) The Kendall correlation score for every single program that solves the Fasta problem.



(b) The Kendall correlation score for every programming language that solves the Fasta problem.

Figure 6.6

6.4.2 Time

Because of some statements in related work, we also wanted to look at the relationship between the energy consumption and the run time. We plotted every measurement from both the nodes in the same graph, where on the x-axis is the run time and on the y-axis the energy consumption. This graph is shown in figure 6.7. Looking at this graph we see multiple lines in the relationship. We also see that a lower run time does not always mean that less energy is consumed. When calculating the Kendall correlation between the energy consumption and the run time we get 0.82, which means high correlation according to the Guilford scale.

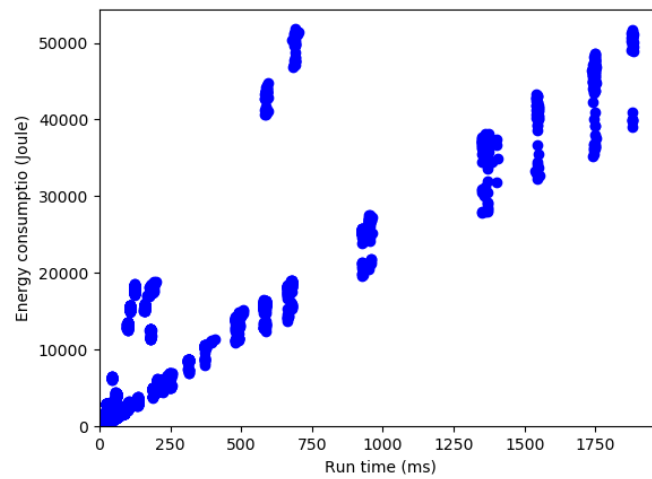


Figure 6.7: All the measurements from both the measurement nodes.

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 we made the table 8.1 where we are comparing different related 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.

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
[5]	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]	Controlled experiment	I/O application tasks	Estimating energy consumption of I/O tasks
[24]	Empirical Study	Six commonly used refactorings	Finding impact of refactorings on energy consumption
[25]	Controlled experiment	Programming languages	Rank programming languages based on speed, memory usage and energy consumption
[26]	Case Study	Multi-core processor scheduling	Efficient workload partitioning
[27]	Case Study	Cache storage management algorithms	Power aware cache management
[28]	Case Study	Java applications	Framework to automate decision-making support regarding energy consumption
[29]	Case Study	Software process	Two level green software model

Table 8.1: Overview of related research

8.1 Finding optimal energy consumption

The paper [2] 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 were 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.

The paper [20] 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 than 12% of the resources an exhaustive 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%.

8.2 Estimating energy consumption

The paper [5] 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 where the only difference is the time steps and that the optimized code is faster.

The paper [21] 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. They found that the ISS function is less accurate when the value of N increases and that the SRA function is not accurate for small values of N. Here N is the input value of the function that is tested for its energy consumption.

STILL NEED TO REREAD AND ADD HOW THEY ESTIMATE The paper [22] 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 want 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.

The paper [23] 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 in energy consumption 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. Therefore 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, 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.

8.3 Measuring energy consumption

The paper [24] addresses the lack of information about the energy impact of code refactorings. They did this by testing the energy impact of 197 projects when 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.

The paper [25] 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.

8.4 Hardware

The paper [26] proposes an algorithm to make 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.

The paper [27] 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 than the LRU algorithm. They also find that the cache policy write-back can use 20% less energy than write-through.

8.5 Other

The paper [28] 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 than the other input. They showed that by letting their framework choose which library to use they reduced their energy consumption by 17%.

The paper [29] 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 discuss the relationship between the two levels.

Chapter 9

Conclusion

9.1 Conclusion

(1) Is there a difference in the energy consumption of software projects in different programming languages that have the same functionality

- Hypothesis: There is no difference in the energy consumption of software projects in different programming languages that have the same functionality.
- Alternative hypothesis: There is a difference in the energy consumption of software projects in different programming languages that have the same functionality.
- Method: Using twice the one sided Mann Whitney U test where in both cases the null hypothesis is that the distributions are the same but the alternative hypothesis is that one distribution is greater or less.
- Result: For *node28* in the most cases, 294 of 306 cases, we can reject the null hypothesis, but in 12 cases we can not. This is roughly 4% and in these cases we cannot reject the null hypothesis. Which does not mean that they are not different, maybe the test is not powerful enough. For *node29* the numbers change a bit, there we cannot reject the null hypothesis in 30 of the 306 cases. Which is roughly 10%.
- Conclusion: For most cases we reject the null hypothesis and accept the alternative hypothesis. Thus for most cases there is a difference in the energy consumption of software projects in different programming languages that have the same functionality.

(2) Is there a difference in energy consumption of different software projects (in the same programming language) that have the same functionality

- Hypothesis: There is no difference in energy consumption of different software projects, in the same programming language, that have the same functionality.
- Alternative hypothesis: There is a difference in energy consumption of different software projects, in the same programming language, that have the same functionality.
- Method: Compare programs if they have the same language and solve the same problem. Perform twice the one sided Mann Whitney U test where in both cases the null hypothesis is that the distributions are the same but the alternative hypothesis is that one distribution is greater or less.
- Result: We cannot reject the null hypothesis in 24 of 479 (roughly 5%) cases for the *node28* and 50 of 479 (roughly 10%) cases for the *node29*.
- Conclusion: We for most cases reject the null hypothesis and accept the alternative hypothesis. Thus in most cases there is a difference in the energy consumption of software projects that are in the same programming language and solve the same problem.

9.2 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] I. Gouy, *The computer language benchmarks game*, Accessed: 26-04-2019. [Online]. Available: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/>.
- [4] T. Mens and T. Tourwé, “A survey of software refactoring”, *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.
- [5] 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.
- [6] 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.
- [7] 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/>.
- [8] 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.
- [9] TIOBE, *TIOBE index for april 2019*, Accessed: 16-04-2019, Apr. 2019. [Online]. Available: <https://www.tiobe.com/tiobe-index/>.
- [10] P. Carbonnelle, *PYPL popularity of programming language*, Accessed: 16-04-2019, Apr. 2019. [Online]. Available: <http://pypl.github.io/PYPL.html>.
- [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. P. Guilford, “Fundamental statistics in psychology and education”, 1942.
- [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. Oprea, 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. 1541005, 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] 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.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.

Appendix A

Node28

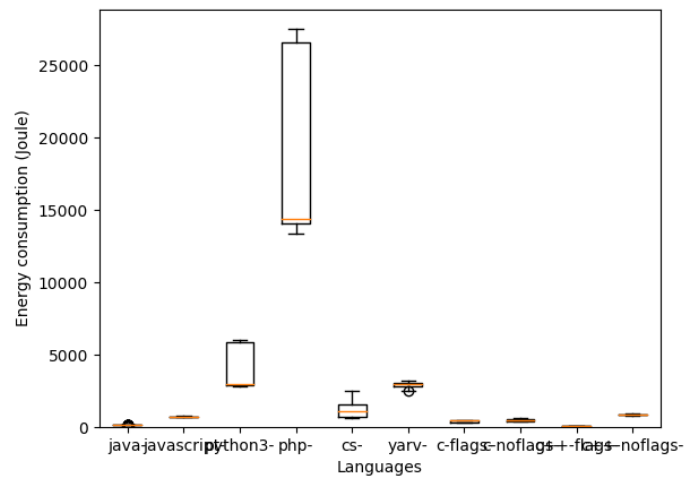


Figure A.1: The box plot of the different programs in a programming language for the problem Binarytrees on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	+	+	-	+
JavaScript	-	0	+	+	+	+	-	-	-	+
Python	-	-	0	+	-	-	-	-	-	-
PHP	-	-	-	0	-	-	-	-	-	-
C#	-	-	+	+	0	+	-	-	-	-
Ruby	-	-	+	+	-	0	-	-	-	-
C-flags	-	+	+	+	+	+	0	+	-	+
C-noflags	-	+	+	+	+	+	-	0	-	+
C++-flags	+	+	+	+	+	+	+	+	0	+
C++-noflags	-	-	+	+	+	+	-	-	-	0

Table A.1: The comparison of the different languages for the Binarytrees problem on *node28*. A *+* means that the language on the row has a lower energy consumption then the language on the column, the opposite for *-*.

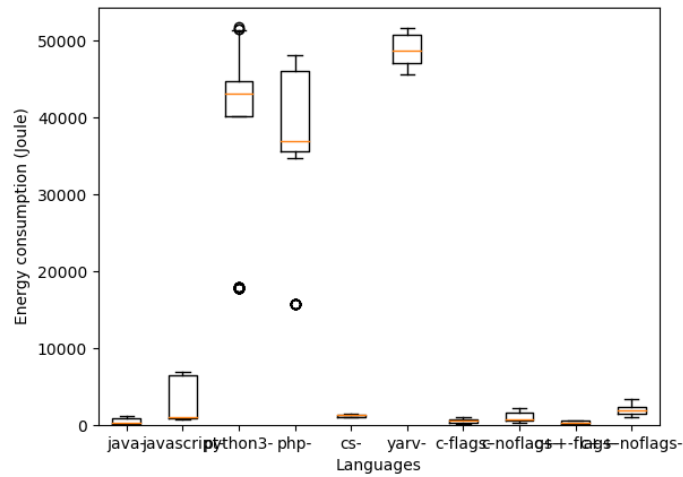


Figure A.2: The box plot of the different programs in a programming language for the problem Fannkuchredux on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	+	+	+	+
JavaScript	-	0	+	+	+	+	-	-	-	+
Python	-	-	0	-	-	+	-	-	-	-
PHP	-	-	+	0	-	+	-	-	-	-
C#	-	-	+	+	0	+	-	-	-	+
Ruby	-	-	-	-	-	0	-	-	-	-
C-flags	-	+	+	+	+	+	0	+	-	+
C-noflags	-	+	+	+	+	+	-	0	-	+
C++-flags	-	+	+	+	+	+	+	+	0	+
C++-noflags	-	-	+	+	-	+	-	-	-	0

Table A.2: The comparison of the different languages for the Fannkuchredux problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$.

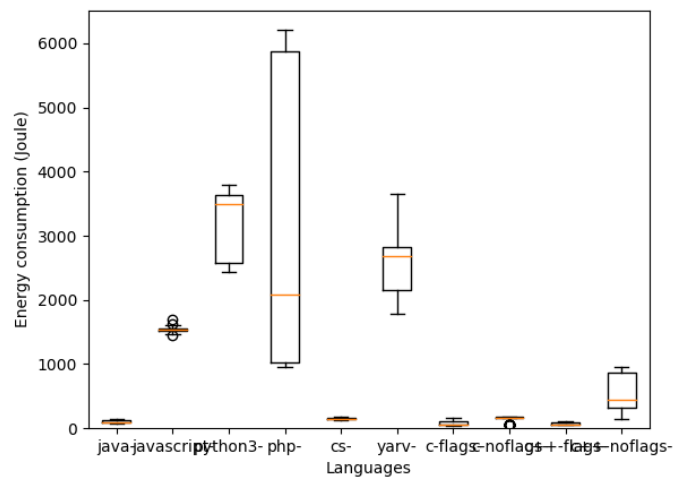


Figure A.3: The box plot of the different programs in a programming language for the problem Fasta on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	-	+	-	+
JavaScript	-	0	+	+	-	+	-	-	-	-
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	+	-	-	-	-
C#	-	+	+	+	0	+	-	+	-	+
Ruby	-	-	+	-	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	+	+
C-noflags	-	+	+	+	-	+	-	0	-	+
C++-flags	+	+	+	+	+	+	-	+	0	+
C++-noflags	-	+	+	+	-	+	-	-	-	0

Table A.3: The comparison of the different languages for the Fasta problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$.

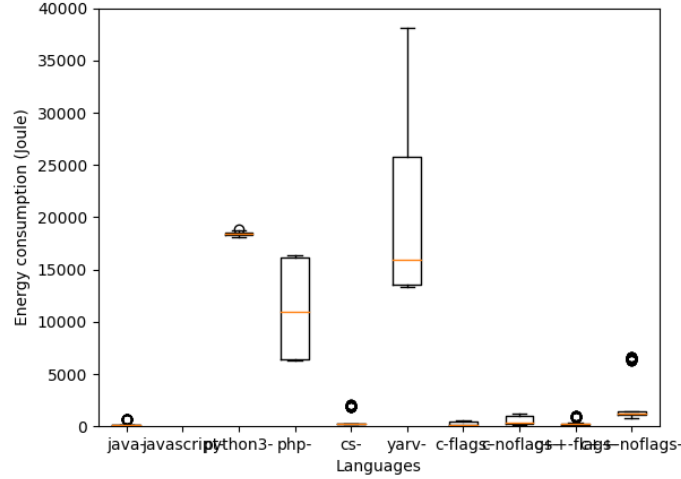


Figure A.4: The box plot of the different programs in a programming language for the problem Mandelbrot on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0		+	+	+	+	-	+	-	+
JavaScript										
Python	-		0	-	-	Unknown	-	-	-	-
PHP	-		+	0	-	+	-	-	-	-
C#	-		+	+	0	+	-	Unknown	-	+
Ruby	-		Unknown	-	-	0	-	-	-	-
C-flags	+		+	+	+	+	0	+	Unknown	+
C-noflags	-		+	+	Unknown	+	-	0	-	+
C++-flags	+		+	+	+	+	Unknown	+	0	+
C++-noflags	-		+	+	-	+	-	-	-	0

Table A.4: The comparison of the different languages for the Mandelbrot problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

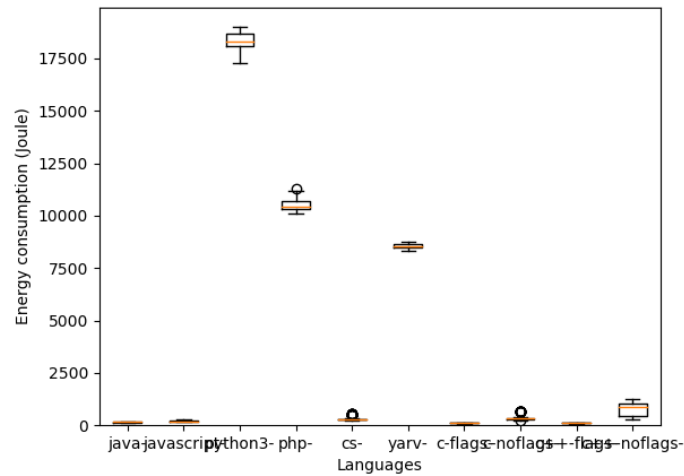


Figure A.5: The box plot of the different programs in a programming language for the problem Nbody on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	-	+	-	+
JavaScript	-	0	+	+	+	+	-	+	-	+
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	-	-	-	-	-
C#	-	-	+	+	0	+	-	+	-	+
Ruby	-	-	+	+	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	Unknown	+
C-noflags	-	-	+	+	-	+	-	0	-	+
C++-flags	+	+	+	+	+	+	Unknown	+	0	+
C++-noflags	-	-	+	+	-	+	-	-	-	0

Table A.5: The comparison of the different languages for the Nbody problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

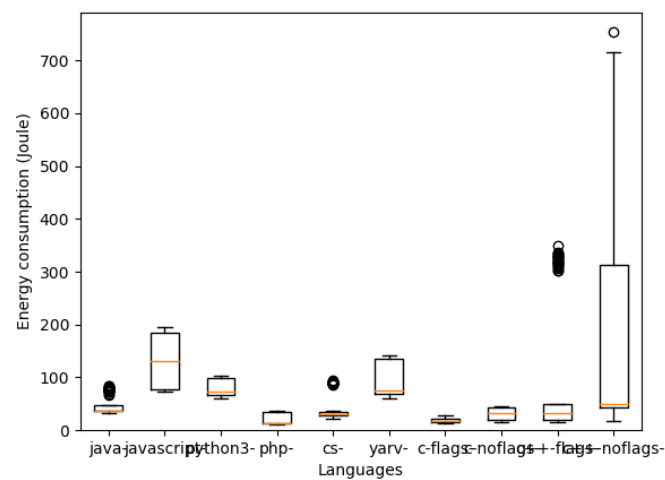


Figure A.6: The box plot of the different programs in a programming language for the problem Revcomp on *node28*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	-	-	+	-	-	-	+
JavaScript	-	0	-	-	-	-	-	-	-	-
Python	-	+	0	-	-	+	-	-	-	-
PHP	+	+	+	0	+	+	+	+	+	+
C#	+	+	+	-	0	+	-	Unknown	Unknown	+
Ruby	-	+	-	-	-	0	-	-	-	-
C-flags	+	+	+	-	+	+	0	+	+	+
C-noflags	+	+	+	-	Unknown	+	-	0	Unknown	+
C++-flags	+	+	+	-	Unknown	+	-	Unknown	0	+
C++-noflags	-	+	+	-	-	+	-	-	-	0

Table A.6: The comparison of the different languages for the Revcomp problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

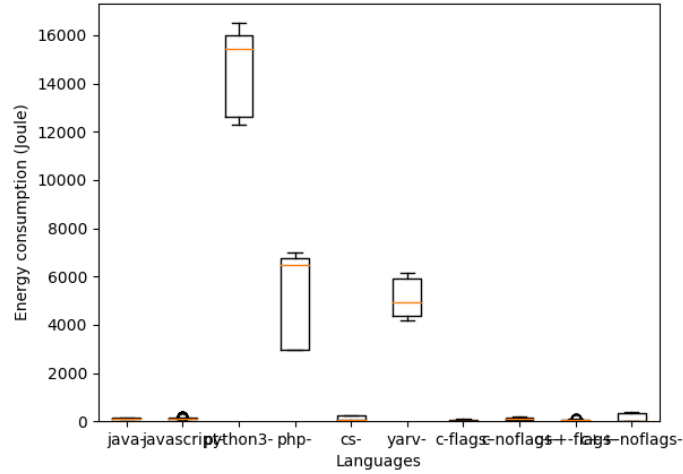


Figure A.7: The box plot of the different programs in a programming language for the problem Spectralnorm on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	-	+	+	Unknown	+	-	-	-	-
JavaScript	+	0	+	+	Unknown	+	-	-	-	-
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	Unknown	-	-	-	-
C#	Unknown	Unknown	+	+	0	+	-	Unknown	-	Unknown
Ruby	-	-	+	Unknown	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	+	+
C-noflags	+	+	+	+	Unknown	+	-	0	-	-
C++-flags	+	+	+	+	+	+	-	+	0	+
C++-noflags	+	+	+	+	Unknown	+	-	+	-	0

Table A.7: The comparison of the different languages for the Spectralnorm problem on *node28*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

Program 1	Program 2	p-less	p-greater
java-3.problem0	java-6.problem0	0.266	0.740
javascript-1.problem2	javascript-2.problem2	0.829	0.176
javascript-1.problem2	javascript-3.problem2	0.413	0.594
javascript-2.problem2	javascript-3.problem2	0.197	0.808
javascript-1.problem6	javascript-3.problem6	0.532	0.475
javascript-1.problem6	javascript-5.problem6	0.272	0.734
javascript-3.problem6	javascript-5.problem6	0.243	0.763
python3-2.problem3	python3-5.problem3	0.488	0.520
cs-3.problem3	cs-4.problem3	0.088	0.915
cs-3.problem4	cs-5.problem4	0.622	0.385
cs-4.problem4	cs-6.problem4	0.493	0.515
c-noflags-1.problem2	c-noflags-2.problem2	0.882	0.122
c-flags-2.problem4	c-flags-3.problem4	0.230	0.776
c-noflags-1.problem4	c-noflags-6.problem4	0.218	0.787
c-flags-3.problem5	c-flags-6.problem5	0.175	0.830
c-noflags-4.problem5	c-noflags-5.problem5	0.090	0.912
c++-flags-1.problem0	c++-flags-8.problem0	0.571	0.436
c++-noflags-1.problem0	c++-noflags-3.problem0	0.883	0.121
c++-noflags-1.problem0	c++-noflags-8.problem0	0.874	0.130
c++-noflags-3.problem0	c++-noflags-8.problem0	0.453	0.555
c++-flags-1.problem2	c++-flags-2.problem2	0.317	0.690
c++-flags-3.problem4	c++-flags-8.problem4	0.168	0.837
c++-flags-4.problem4	c++-flags-6.problem4	0.225	0.781
c++-noflags-5.problem6	c++-noflags-6.problem6	0.374	0.633

Table A.8: The programs result from *node28* where the null hypothesis that they are from the same distribution could not be reject for the Mann Whitney U one-sided test less and bigger.

Appendix B

Node29

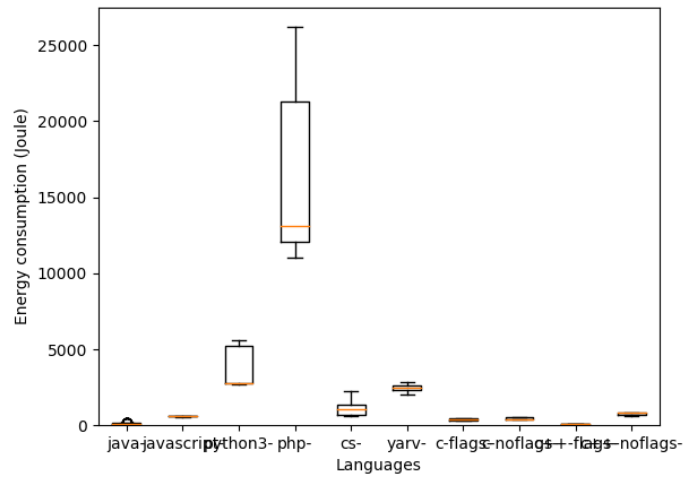


Figure B.1: The box plot of the different programs in a programming language for the problem Binarytrees on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	+	+	-	+
JavaScript	-	0	+	+	+	+	-	-	-	+
Python	-	-	0	+	-	-	-	-	-	-
PHP	-	-	-	0	-	-	-	-	-	-
C#	-	-	+	+	0	+	-	-	-	-
Ruby	-	-	+	+	-	0	-	-	-	-
C-flags	-	+	+	+	+	+	0	+	-	+
C-noflags	-	+	+	+	+	+	-	0	-	+
C++-flags	+	+	+	+	+	+	+	+	0	+
C++-noflags	-	-	+	+	+	+	-	-	-	0

Table B.1: The comparison of the different languages for the Binarytrees problem on *node29*. A *+* means that the language on the row has a lower energy consumption then the language on the column, the opposite for *-*.

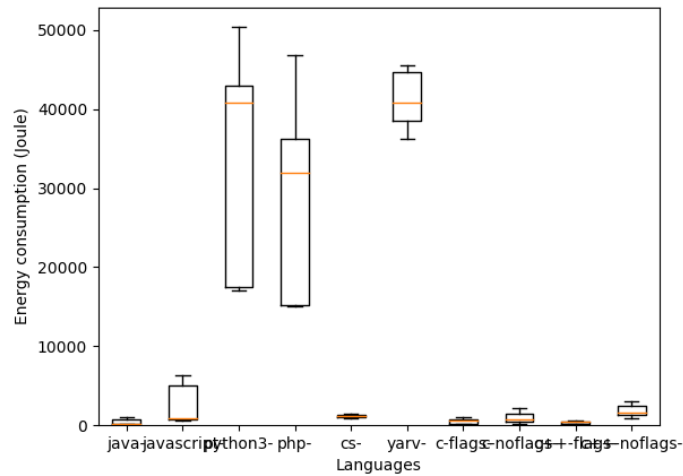


Figure B.2: The box plot of the different programs in a programming language for the problem Fannkuchredux on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	+	+	Unknown	+
JavaScript	-	0	+	+	+	+	-	-	-	+
Python	-	-	0	-	-	Unknown	-	-	-	-
PHP	-	-	+	0	-	+	-	-	-	-
C#	-	-	+	+	0	+	-	-	-	+
Ruby	-	-	Unknown	-	-	0	-	-	-	-
C-flags	-	+	+	+	+	+	0	+	-	+
C-noflags	-	+	+	+	+	+	-	0	-	+
C++-flags	Unknown	+	+	+	+	+	+	+	0	+
C++-noflags	-	-	+	+	-	+	-	-	-	0

Table B.2: The comparison of the different languages for the Fannkuchredux problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$.

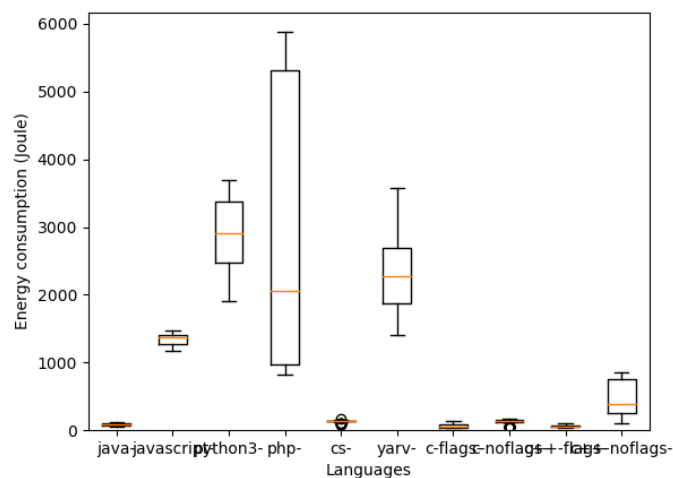


Figure B.3: The box plot of the different programs in a programming language for the problem Fasta on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	-	+	-	+
JavaScript	-	0	+	+	-	+	-	-	-	-
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	Unknown	-	-	-	-
C#	-	+	+	+	0	+	-	Unknown	-	+
Ruby	-	-	+	Unknown	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	Unknown	+
C-noflags	-	+	+	+	Unknown	+	-	0	-	+
C++-flags	+	+	+	+	+	+	Unknown	+	0	+
C++-noflags	-	+	+	+	-	+	-	-	-	0

Table B.3: The comparison of the different languages for the Fasta problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

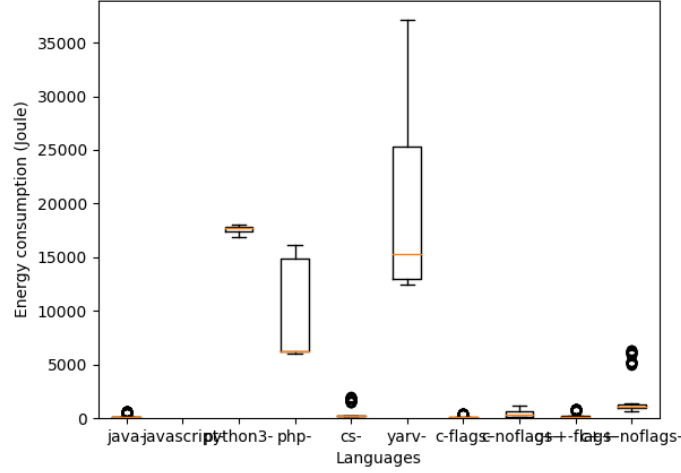


Figure B.4: The box plot of the different programs in a programming language for the problem Mandelbrot on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0		+	+	+	+	-	+	-	+
JavaScript										
Python	-		0	-	-	-	-	-	-	-
PHP	-		+	0	-	+	-	-	-	-
C#	-		+	+	0	+	-	Unknown	-	+
Ruby	-		+	-	-	0	-	-	-	-
C-flags	+		+	+	+	+	0	+	Unknown	+
C-noflags	-		+	+	Unknown	+	-	0	-	+
C++-flags	+		+	+	+	+	Unknown	+	0	+
C++-noflags	-		+	+	-	+	-	-	-	0

Table B.4: The comparison of the different languages for the Mandelbrot problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

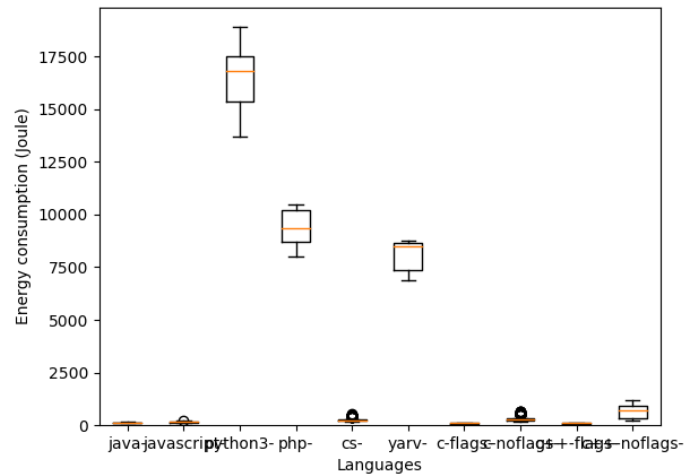


Figure B.5: The box plot of the different programs in a programming language for the problem Nbody on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	+	+	+	-	+	-	+
JavaScript	-	0	+	+	+	+	-	+	-	+
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	-	-	-	-	-
C#	-	-	+	+	0	+	-	+	-	+
Ruby	-	-	+	+	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	Unknown	+
C-noflags	-	-	+	+	-	+	-	0	-	+
C++-flags	+	+	+	+	+	+	Unknown	+	0	+
C++-noflags	-	-	+	+	-	+	-	-	-	0

Table B.5: The comparison of the different languages for the Nbody problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

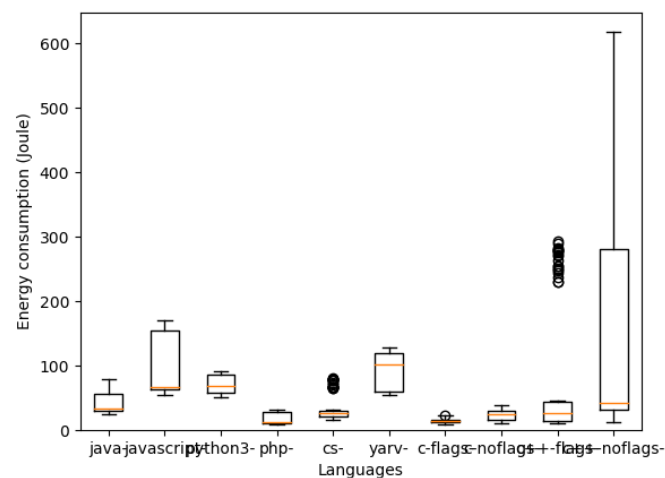


Figure B.6: The box plot of the different programs in a programming language for the problem Revcomp on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	+	+	-	-	+	-	-	-	+
JavaScript	-	0	-	-	-	-	-	-	-	-
Python	-	+	0	-	-	+	-	-	-	-
PHP	+	+	+	0	+	+	Unknown	+	+	+
C#	+	+	+	-	0	+	-	-	Unknown	+
Ruby	-	+	-	-	-	0	-	-	-	-
C-flags	+	+	+	Unknown	+	+	0	+	+	+
C-noflags	+	+	+	-	+	+	-	0	+	+
C++-flags	+	+	+	-	Unknown	+	-	-	0	+
C++-noflags	-	+	+	-	-	+	-	-	-	0

Table B.6: The comparison of the different languages for the Revcomp problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

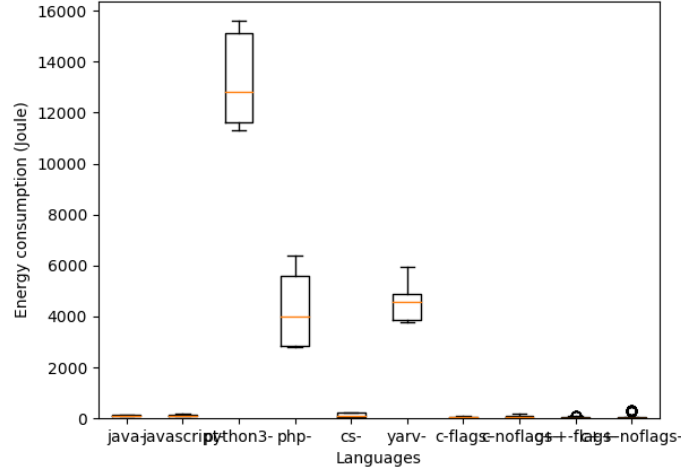


Figure B.7: The box plot of the different programs in a programming language for the problem Spectralnorm on *node29*.

	Java	JavaScript	Python	PHP	C#	Ruby	C-flags	C-noflags	C++-flags	C++-noflags
Java	0	-	+	+	Unknown	+	-	-	-	-
JavaScript	+	0	+	+	Unknown	+	-	-	-	-
Python	-	-	0	-	-	-	-	-	-	-
PHP	-	-	+	0	-	Unknown	-	-	-	-
C#	Unknown	Unknown	+	+	0	+	-	Unknown	-	Unknown
Ruby	-	-	+	Unknown	-	0	-	-	-	-
C-flags	+	+	+	+	+	+	0	+	+	+
C-noflags	+	+	+	+	Unknown	+	-	0	-	-
C++-flags	+	+	+	+	+	+	-	+	0	+
C++-noflags	+	+	+	+	Unknown	+	-	+	-	0

Table B.7: The comparison of the different languages for the Spectralnorm problem on *node29*. A $+$ means that the language on the row has a lower energy consumption then the language on the column, the opposite for $-$, and the *Unknown* means that we could not reject the null hypothesis.

Program 1	Program 2	p-less	p-greater	K-S test
java-2.problem0	java-3.problem0	0.937	0.066	Unknown
java-2.problem0	java-6.problem0	0.820	0.186	Unknown
java-3.problem0	java-6.problem0	0.164	0.842	Unknown
java-2.problem3	java-4.problem3	0.514	0.495	Unknown
java-2.problem4	java-3.problem4	0.864	0.142	Unknown
java-4.problem5	java-5.problem5	0.732	0.276	Unknown
java-5.problem5	java-6.problem5	0.915	0.089	Unknown
javascript-1.problem2	javascript-2.problem2	0.174	0.832	Unknown
javascript-1.problem2	javascript-3.problem2	0.106	0.898	Unknown
javascript-2.problem2	javascript-3.problem2	0.305	0.704	Unknown
javascript-3.problem2	javascript-4.problem2	0.120	0.884	Unknown
javascript-1.problem4	javascript-4.problem4	0.060	0.943	Unknown
javascript-1.problem4	javascript-5.problem4	0.053	0.950	Unknown
javascript-4.problem4	javascript-5.problem4	0.401	0.609	Unknown
javascript-1.problem6	javascript-3.problem6	0.658	0.352	Unknown
javascript-1.problem6	javascript-5.problem6	0.628	0.382	Unknown
javascript-3.problem6	javascript-5.problem6	0.495	0.516	Unknown
cs-1.problem1	cs-3.problem1	0.609	0.400	Rejected
cs-1.problem1	cs-4.problem1	0.606	0.404	Rejected
cs-2.problem1	cs-5.problem1	0.793	0.214	Rejected
cs-2.problem1	cs-6.problem1	0.909	0.095	Rejected
cs-1.problem3	cs-4.problem3	0.756	0.252	Unknown
cs-3.problem3	cs-4.problem3	0.372	0.638	Unknown
cs-3.problem3	cs-6.problem3	0.877	0.128	Rejected
cs-4.problem3	cs-6.problem3	0.855	0.151	Unknown
cs-2.problem4	cs-3.problem4	0.952	0.050	Unknown
cs-3.problem4	cs-5.problem4	0.882	0.123	Unknown
cs-4.problem4	cs-6.problem4	0.430	0.580	Unknown
cs-4.problem4	cs-8.problem4	0.945	0.058	Unknown
cs-6.problem4	cs-8.problem4	0.952	0.050	Unknown
cs-1.problem5	cs-5.problem5	0.907	0.097	Unknown
cs-2.problem5	cs-5.problem5	0.157	0.849	Unknown
yarv-1.problem1	yarv-2.problem1	0.825	0.190	Unknown
c-flags-2.problem2	c-flags-5.problem2	0.292	0.716	Rejected
c-noflags-1.problem2	c-noflags-2.problem2	0.525	0.485	Unknown
c-noflags-1.problem2	c-noflags-5.problem2	0.200	0.806	Unknown
c-noflags-1.problem2	c-noflags-7.problem2	0.582	0.427	Unknown
c-noflags-2.problem2	c-noflags-5.problem2	0.224	0.783	Unknown
c-noflags-2.problem2	c-noflags-7.problem2	0.690	0.319	Unknown
c-noflags-5.problem2	c-noflags-7.problem2	0.861	0.145	Unknown
c-flags-2.problem4	c-flags-3.problem4	0.657	0.352	Unknown
c-flags-2.problem4	c-flags-6.problem4	0.470	0.540	Unknown
c-flags-3.problem4	c-flags-6.problem4	0.311	0.698	Unknown
c-noflags-1.problem4	c-noflags-6.problem4	0.410	0.599	Unknown
c-flags-3.problem5	c-flags-6.problem5	0.278	0.730	Unknown
c-noflags-4.problem5	c-noflags-5.problem5	0.300	0.708	Unknown
c-noflags-4.problem6	c-noflags-5.problem6	0.401	0.609	Rejected
c++-noflags-1.problem0	c++-noflags-3.problem0	0.340	0.669	Unknown
c++-noflags-1.problem0	c++-noflags-8.problem0	0.212	0.795	Unknown
c++-noflags-3.problem0	c++-noflags-8.problem0	0.398	0.612	Unknown
c++-flags-4.problem1	c++-flags-6.problem1	0.244	0.763	Rejected
c++-flags-1.problem2	c++-flags-2.problem2	0.676	0.333	Unknown
c++-flags-1.problem2	c++-flags-6.problem2	0.903	0.102	Rejected
c++-flags-2.problem3	c++-flags-5.problem3	0.790	0.217	Unknown
c++-noflags-8.problem3	c++-noflags-9.problem3	0.272	0.736	Unknown
c++-flags-3.problem4	c++-flags-8.problem4	0.642	0.367	Unknown
c++-flags-4.problem4	c++-flags-6.problem4	0.473	0.538	Unknown
c++-noflags-1.problem4	c++-noflags-6.problem4	0.277	0.731	Unknown
c++-noflags-7.problem4	c++-noflags-8.problem4	0.910	0.094	Unknown

Table B.8: The programs result from *node29* where the null hypothesis that they are from the same distribution could not be reject for the Mann Whitney U one-sided test less and bigger.

Appendix C

Correlation

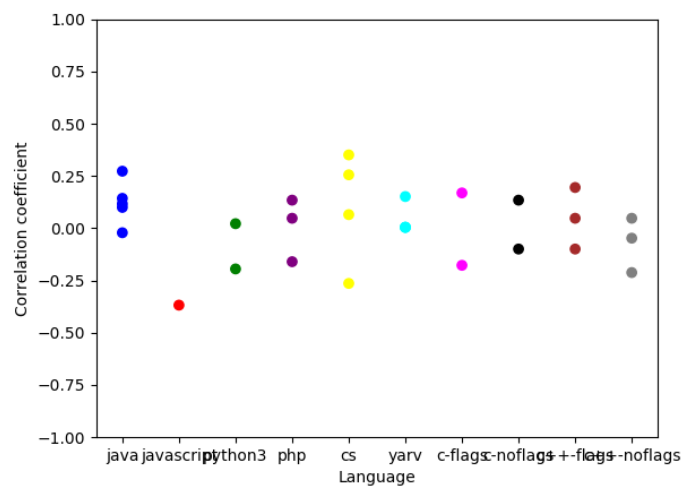


Figure C.1: The Kendall correlation score for every single program that solves the Binarytrees problem.

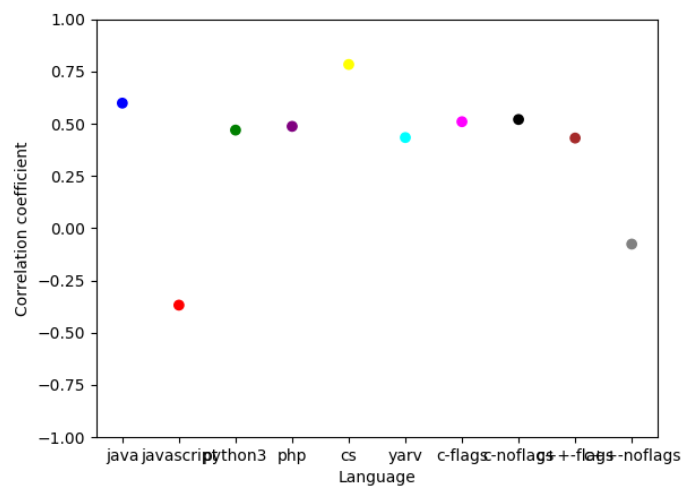


Figure C.2: The Kendall correlation score for every programming language that solves the Binarytrees problem.

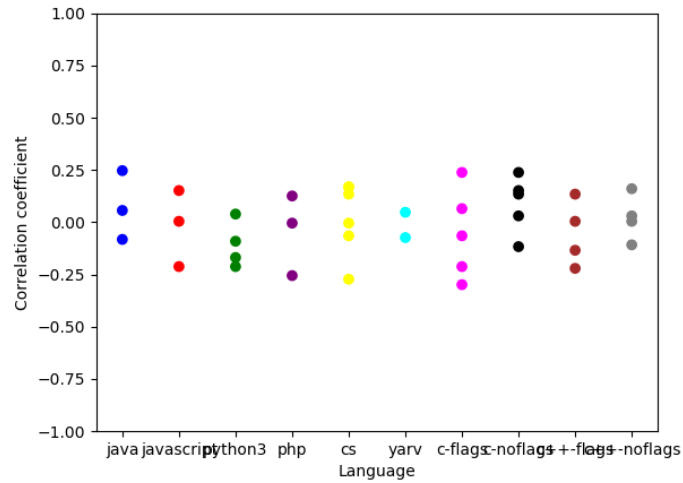


Figure C.3: The Kendall correlation score for every single program that solves the Fannkuchredux problem.

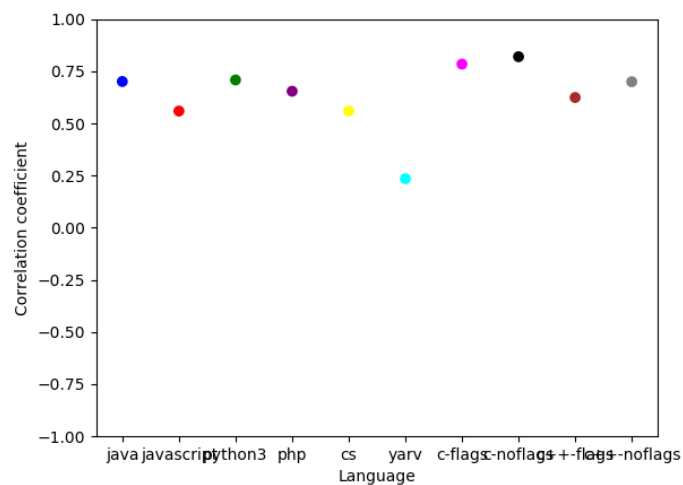


Figure C.4: The Kendall correlation score for every programming language that solves the Fannkuchredux problem.

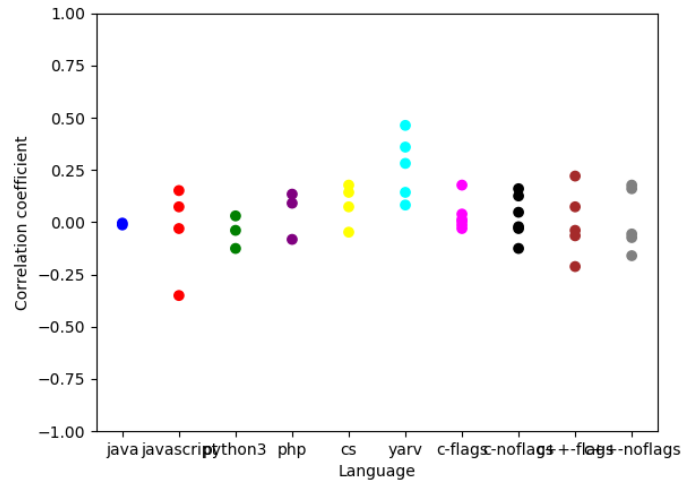


Figure C.5: The Kendall correlation score for every single program that solves the Fasta problem.

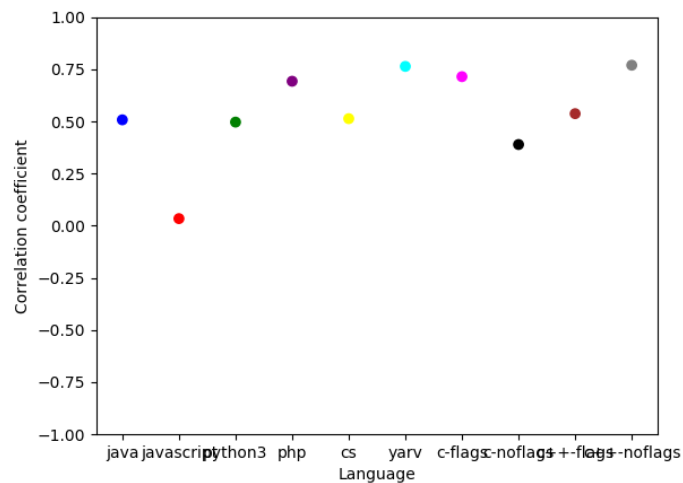


Figure C.6: The Kendall correlation score for every programming language that solves the Fasta problem.

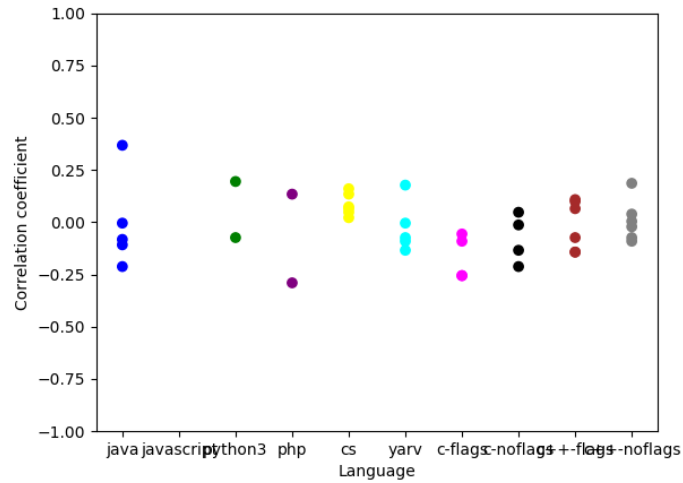


Figure C.7: The Kendall correlation score for every single program that solves the Mandelbrot problem.

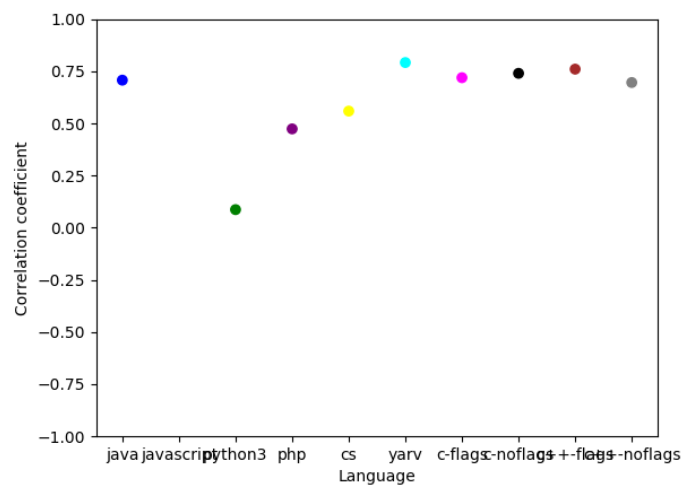


Figure C.8: The Kendall correlation score for every programming language that solves the Mandelbrot problem.

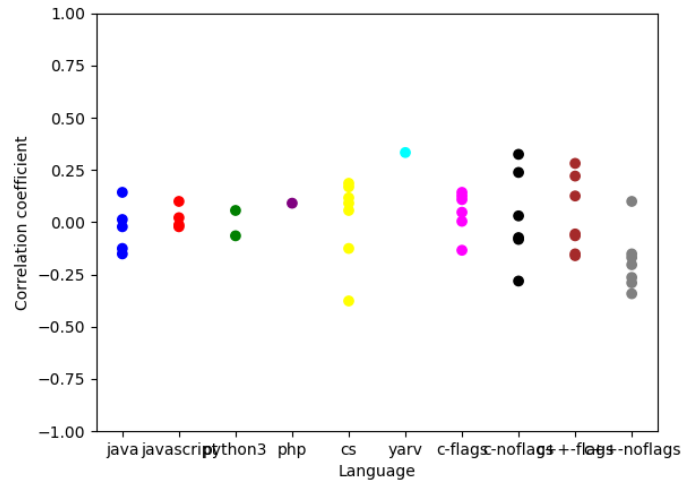


Figure C.9: The Kendall correlation score for every single program that solves the Nbody problem.

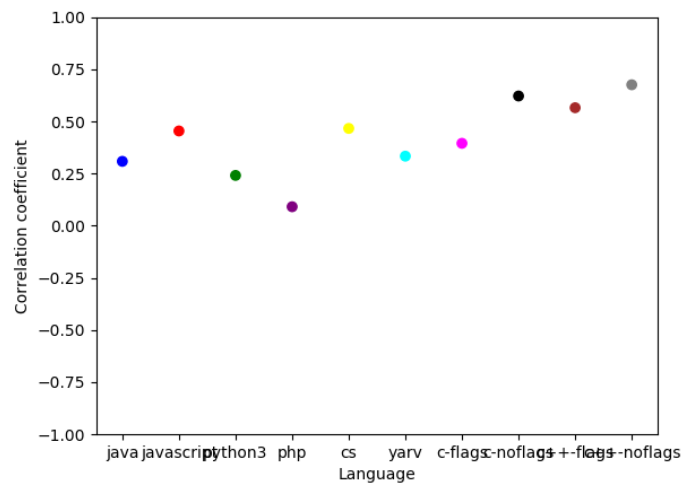


Figure C.10: The Kendall correlation score for every programming language that solves the Nbody problem.

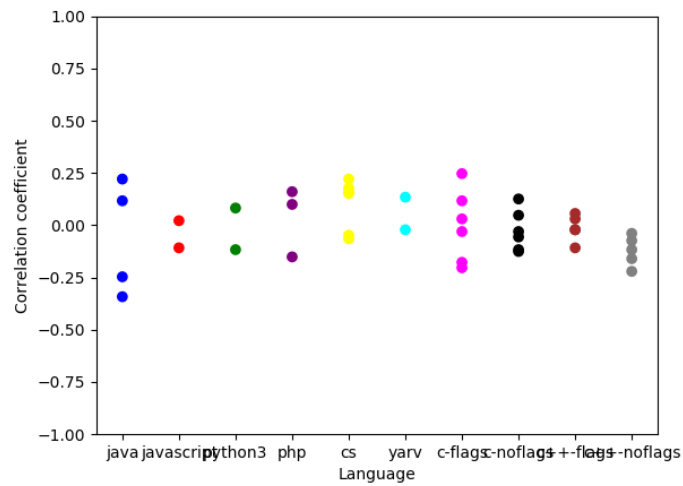


Figure C.11: The Kendall correlation score for every single program that solves the Revcomp problem.

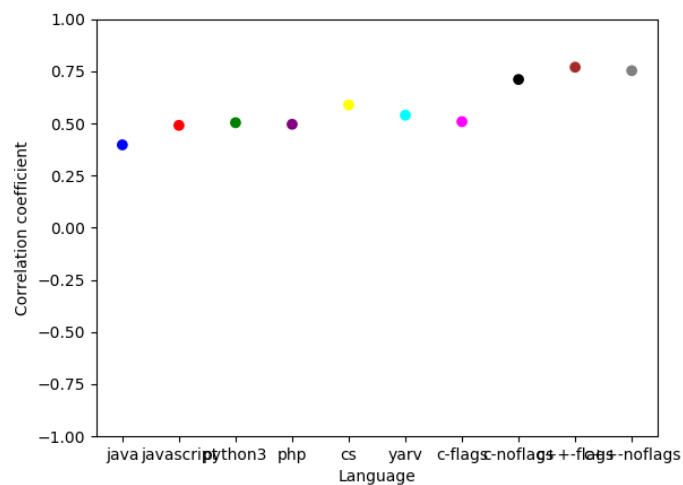


Figure C.12: The Kendall correlation score for every programming language that solves the Revcomp problem.

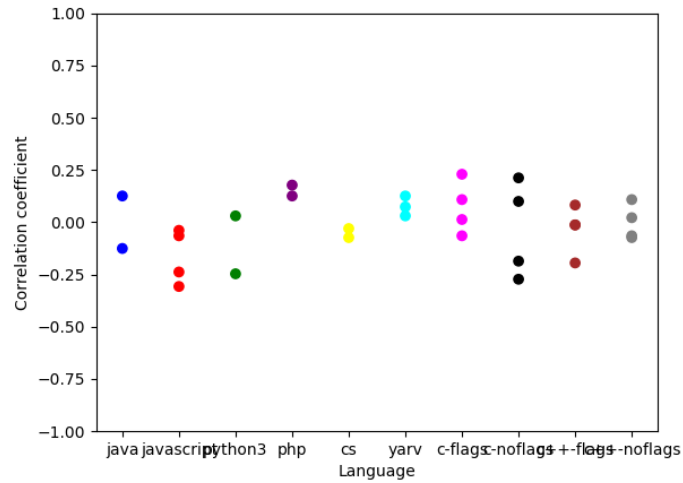


Figure C.13: The Kendall correlation score for every single program that solves the Spectralnorm problem.

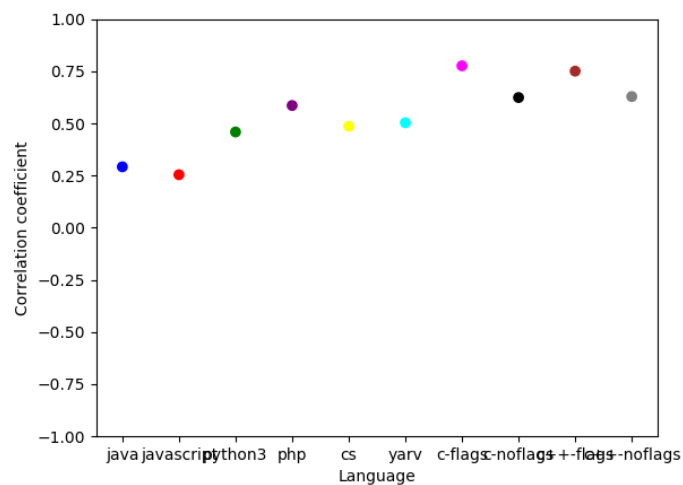


Figure C.14: The Kendall correlation score for every programming language that solves the Spectralnorm problem.