A Comparison Study of Measuring Instruments

Is RAPL Always the Answer?

Project Report

cs-22-pt-9-03

Aalborg University
Computer Science

Department of Computer Science
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
A Comparison Study of Measuring Instruments

Theme:
Programming Technology

Project Period:
Fall Semester 2022

Project Group:
cs-22-pt-9-03

Participant(s):
Mads Hjuler Kusk
Jamie Baldwin Pedersen
Jeppe Jon Holt

Supervisor(s):
Bent Thomsen
Lone Leth Thomsen

Copies: 1

Page Numbers: 129

Date of Completion:
August 10, 2023

Abstract:

Based on observations of how most existing work chose to use Intel's Running Average Power Limit on Linux when measuring the energy consumption of software, this work sets out to explore how other measuring instruments perform, specifically Intel Power Gadget, Microsoft's Energy Estimation Engine and Libre Hardware Monitor on Windows. This is achieved by constructing a framework able to log and save measurements made by different measuring instruments on different test cases targeting different parts of the system. During the analysis, the impact of temperature, battery levels and R3 validation are explored and illustrated. To find the best performing measuring instrument, the correlation between the measuring instruments and the ground truth is found, using a combination of Mann-Whitney U tests and Kendall's Tau coefficient. Through this study, the different measuring instruments all show promising results and reasonable usability.

Preface

This work is written by group cs-22-pt-9-03 during the 3rd semester of the Master of Computer Science as a part of the "Specialization course in Programming Technology" at Aalborg University. The goal of the work is to compare existing software-based measuring instruments able to measure the energy consumption of software, to find the best performing measuring instrument for both Windows and Linux. We would like to express our appreciation to our supervisors, Lone Leth Thomsen and Bent Thomsen from the Department of Computer Science at Aalborg University and Kenneth Knirke from the Department of Electronic Systems at Aalborg University, for their crucial guidance in this project.

Aalborg University, August 10, 2023

Mads Hjuler Kusk
mkusk18@student.aau.dk

Jeppe Jon Holt
jholt18@student.aau.dk

Jamie Jonas Baldwin Pedersen
jjbp18@student.aau.dk

Contents

Chapter 1

Introduction

In recent years there has been an increased focus on emissions and energy consumption. One factor impacting the increased energy consumption is the rise of the computer after the IBM personal computer became a mainstay in most households. The computer can be found in most households in the developed world, and an increasing amount of people have access to a computational device in the developing world, like computers and smartphones.[2].

Because an increasing number of computers exist, they are responsible for a growing percentage of global electricity consumption, where the percentage of energy consumed by computers on a global scale is estimated to be $\approx 6\%$[81]. While hardware has become much more efficient over the years as transistor sizes have become smaller and thus uses less energy, this will not be enough to handle the growth in the market for computational devices[68]. Because of this, a more in-depth understanding of the energy consumption of software is interesting as software could help reduce the energy usage[81]. In recent years several studies of software's energy consumption have been conducted, but it is still a newer field of study. Pereira et al.[65] found that for some test cases the programming languages used, can have a large effect on the energy consumption, even when they are functionally identical, where some languages use over 90x more energy than others. A majority of these studies have been conducted on Linux devices, where energy measurements are performed using the measuring instrument Intel's Running Average Power Limit (RAPL) on Linux. While the studies for Linux are important, the literature does not indicate if the findings for Linux energy consumption translate directly to Windows[65].

Considering the energy consumption of Windows is relevant since Windows has been the most commonly used operating system for personal computers for multiple years. This is still true at the time of writing, and it is, therefore, relevant to know how energy is consumed on these devices as they make up a large percentage of computers worldwide[63]. To monitor energy consumption on Windows, other measuring instruments with similar functionality to RAPL exist, but the actual reliability of these solutions is, to our knowledge, yet to be verified in the literature.

The aim of our work is therefore to compare different measuring instruments and to explore how the solutions for Windows perform. To verify these results, a hardware-based measuring instrument will be utilized as a ground truth and the results will also be compared to RAPL. To conduct this study, four different research questions are formulated. The overall idea is to compare different measuring instruments on the same Device Under Test (DUT), but as one measuring instruments require specific hardware or claim to perform better on certain hardware, this will also be explored. This means

several measuring instruments will measure the energy consumption on several different DUTs to see how well they perform compared to RAPL and a hardware-based measuring instrument. The main contributions of our work are formulated as research questions, defined as the following:

- **RQ1:** How do we systematically measure the energy consumption of a test case on different DUTs and OSs and how do we make the test case measurements comparable?
- **RQ2:** How do existing energy measuring instruments compare to each other?
- **RQ3:** How does the test case measurement compare between Windows and Linux?
- **RQ4:** How does the test case measurement compare between different DUTs?

The overall goal of our work is to compare existing measuring instruments (**RQ2**). This will be done based on operating systems (OS) (**RQ3**) and DUTs (**RQ4**). When comparing measuring instruments, it will be done with different test cases. To achieve this, a systematic way of measuring energy consumption and making them comparable across different DUTs and OSs, as stated in **RQ1** is required.

Our work is motivated by how most research today mainly uses RAPL on Linux[70, 65, 30, 50], and those who do not, have the disclaimer of not knowing how accurate their results are[7, 64, 93]. This is despite claims from Microsoft that their tool, Energy Estimation Engine, (E3) can measure the energy consumption of software with 98% accuracy in certain cases[91]. The main contribution of our work is to create an independent source of how accurate different measuring instruments are.

In chapter 2 a look into existing work will be made, for inspiration. Following this, chapter 3 will aim to answer **RQ1**, by introducing the framework created, and how to make measurements comparable. In chapter 4, the different experiments performed in order to answer **RQ2-4** will be explored, and analyzed in chapter 5. Lastly, the results will be discussed and concluded in chapter 6 and chapter 7 respectively.

Chapter 2

Related Work

In this chapter, a look into existing work will be made. This is done to gain inspiration from already existing work and to find potential pitfalls to avoid. In section 2.1, existing work concerning the energy consumption of software is presented. Thereafter, different approaches to measuring the energy consumption of software, used in existing work will be explored in section 2.2. This is done to find which measuring instruments exist, and which ones are considered state of the art to ultimately decide which measuring instruments should be used in our work. In section 2.3 different works discussing measuring methodology and benchmarks setups will be covered. Finally, in section 2.4 the key aspects of this chapter are summarized.

2.1 Energy Consumption of Different Languages and Constructs

Multiple works focus on the differences in energy consumption of different languages and various language constructs within the same language. One such work is by Lindholt et al.[50], where a comparison was made of four groups of language constructs within C#, including concurrency, casting parameter mechanisms and lambda expressions. This is achieved using 150 micro benchmarks, where further analysis is conducted on each of the constructs to understand the results. Based on the findings, 68 macro benchmarks are created to test the observations on a larger scale and to see how the constructs work in combination with each other.

Another such work is the work by Rasmussen et al.[70], in which another work by Kumar et al.[49] made in Java is analyzed and reproduced in C# to see how the results compare in a new language. The study includes different primitive types like selections, collections, objects inheritance and exceptions, and when comparing the results it is concluded that there can be drastic differences between Java and C# in some situations. Another study by [30] looks into two different implementations of a microservice architecture, one with a shared database connection and one with a database connection per service. The energy consumption of these implementations is analyzed and compared to a monolithic architecture. This study was done on two similar implementations on C# and Java, where the energy consumption is compared, and a significant difference can be found, both when comparing languages, but also between the type of database connection.

There also exist other studies, where more than two programming languages are compared. An example of this is the work by Pereira et al.[65], where 27 of the most popular languages are compared. The languages are compared with each other, but also in groups considering compiled, interpreted and virtual machine languages or object-oriented, functional and imperative languages. The

languages are tested across ten test cases with different loads, and the results show how the language can have an impact on the energy consumption of the program, and also that a faster program is not always greener.

In the work by Najmuddin et al.[58] the energy consumption of different OSs are compared on different aspects, including color schemes, resolutions, brightness, memory management, process management, scheduler tasks, and other modules. The study is based on observations where Linux uses more energy than Windows and aims to find out why. Based on an analysis of 15 existing works, the reason is found to be drivers. Linux is found to lack optimized drivers, as a result of hardware manufacturers' unwillingness to give details about the hardware. This makes it difficult to make optimized drivers for open-source projects, like Linux. They also find the kernel in Linux to be poorly optimized in some versions. There is however a disclaimer about the work by Najmuddin et al.[58], as we do not deem the work to be of the highest quality, mainly due to the language. The work is still included as it combines existing work rather than creating their own experiments. It is the only work which to our knowledge compares the energy consumption of Windows and Linux.

2.2 Different Measuring Instruments

Energy measurements are needed to be able to compare the energy consumption of software. This section aims to cover what methods to measure the energy consumption of software are used in the literature.

In the work by Jagroep et al.[42] different software-based measuring instruments are compared. This work evaluates different measuring instruments through a series of experiments on both idle and full load scenarios where the estimated energy consumption is compared to the actual consumption as reported by hardware measurements. Here it is noted that despite measuring the energy consumption of the same software, the measurements can differ significantly between the different measuring instruments. The study includes 14 different measuring instruments, picked based on four criteria. These include: at least a beta version has to be available, it has to be able to sample at least once per second, it has to run on Linux or Windows and it has to be compatible with their hardware. They do however have some issues when trying to download and set up most of the measuring instruments, and only successfully use two measuring methods, one on Windows 7 and one on Ubuntu 12.04. Both were found to be inaccurate.[42]

The work by Khan et al.[46] aims to analyze RAPL, by comparing this measuring instrument to a hardware-based measuring instrument. In this work, they argue the advantages and disadvantages of RAPL are yet to be fully uncovered, which is what they aim to do. This is achieved on an Intel Skylake and an Intel Haswell CPU, which also uncovers the differences in the performance of RAPL on different hardware, where the newer versions of the CPU are deemed more optimized. The key findings of the work are first of all in regards to the accuracy of RAPL, where they find the correlation between the wall power and the package power from RAPL to be 0.99 for the Haswell machine. In addition to this, the overhead of RAPL is deemed low and negligible through experiments. The overhead of RAPL was found by pinning the test case and RAPL on the same core, and analyzing if the benchmark would slow down depending on the sampling rate of the measurement program. The results showed that with the highest sampling rate, the performance overhead is still less than 2%. The work also analyzes what impact the temperature has on the power consumption measured by RAPL, where the correlation coefficient between this was found to be 0.34 and 0.93 for Skylake and Haswell respectively, meaning

the temperature does measurably impact the power consumption, especially for the older Haswell processor. One big advantage of RAPL, as argued by Khan et al.[46], is the sampling rate of 1.000Hz. This sampling rate is argued to be much higher than most external power meters and enables RAPL to distinguish between different phases within the benchmark. This sampling rate is however also one root of error in RAPL, as the measurements are inconsistent and non-atomic. According to the documentation, RAPL updates every 0.976ms, but this value deviates and is overall found to be closer to 1 ms. Because of this, cases can occur where the energy is measured at a consistent interval and will contain a different number of updates, resulting in spikes in energy consumption. Since RAPL does not give any timestamps on updates, it is difficult to say when updates are made, unless a check is made more often than every 1ms, resulting in some sampling being made without it being an update, but these values are removed. The work[46] also notes things like poor driver support, the inability to measure the energy consumption from a single core, and the static sampling rate as disadvantages with RAPL.[46]

In the study by Hackenberg et al.[32] the accuracies of several different measuring instruments are tested. Here they compare several different hardware-based measurement approaches to RAPL for intel and APM for AMD CPUs. In this work, the impact of the granularity is analyzed, whereas for AC measurements it is found that sampling down to a granularity of 50ms yields more accurate information. This is contrary to common beliefs that large capacitors in PSUs level out the consumption, at least from PSUs in 2013. They also show that a granularity of $1-20$ samples per second is necessary to analyze the consumption of the different phases of an application. They conduct AC measurements with a ZES ZIMMER LMG450, IPMI, and PDU from the power outlet leading into the power supply. The DC measurement is made using a custom setup using a Hall sensor where they only measure the CPU, RAM, and motherboard. For high granularity of 100+ samples per second, the AC techniques cannot sample fast enough. They achieve good results with their DC measurement, but they note that some filtering is needed, as a lot of noise is present in the measurements. They also remark that the success of their measurements is highly dependent on the type of motherboard as different manufacturers use different kinds and sizes of capacitors that could smooth out the measurements. In conclusion, they determine that RAPL and APM could need some improvements, but that RAPL is decently accurate and good enough to be used, while APM is distinctly lacking in certain scenarios. For the hardware measurements, the AC measurements were accurate enough and could provide them with good granularity and their DC setup was surprisingly accurate with a high sampling rate.[32]

Another work comparing different measuring instruments is the work by Fahad et al.[20]. The hardware measurements were made using an HCLWattsUp and then compared with energy predictive models, represented by RAPL and on-chip-sensors for Intel CPUs Manycore Platform Software Stack (MPSS). For the experiments, a high granularity was required to get the energy consumption of the different phases of the test cases. To make the measurements made by the software and hardware-based measuring instruments comparable, the notion of dynamic energy consumption is introduced. This is required as the HCLWattsUp measures the energy consumption of the entire DUT, whereas the software-based measuring instruments only measure the energy consumption of the CPU and RAM. Dynamic energy consumption is an estimation of the test cases energy consumption and is calculated using the following formula:[20]

$$E_D = E_T - (P_S * T_E) \tag{2.1}$$

Where E_D is the dynamic energy consumption, T_E is the duration of the program execution and P_S is the energy consumption when the system is idle. E_T is the total energy consumption of the

system. The dynamic energy consumption will then represent the energy consumption of the test case. The HCLWattsUp meter measurements represent the ground truth of the experiments. The study shows that RAPL generally underreports the energy consumption of the system. The accuracy of RAPL is also tested in different scenarios to see if calibrations are feasible. Calibrations for RAPL and on-chip sensors were generally not possible with their techniques in most scenarios. They provide some recommendations for future works and setup which will be covered in more depth later in **??**.

2.3 Measuring Methodology and Setup

When running experiments on the energy consumption of hardware, many factors can impact the measurements. There are several existing works discussing the setup process and how to minimize such factors. One of which is covered in the work by Mancebo et al.[52] who found that there are three types of problems that occur in the domain of evaluating software's energy consumption.[52]

1. There are inconsistencies in the terminology, with different terms being used for the same concept or even the same term being used for different concepts. This lack of consistency hurts the understanding of the subject.
2. There is no agreed-upon methodology, which makes it difficult to compare and replicate results.
3. Choosing the correct measuring instruments that are fitting for the particular experiment evaluation.

To solve these problems they created a framework called Framework for Energy Efficiency Testing to Improve Environmental Goals of the Software (FEETINGS). FEETINGS consists of three main components which aim to solve these issues. These are the conceptual component, methodological component, and technological component.[52]

The conceptual component provides an extension of the work by García et al.[21] where the Software Measurement Ontology is defined. This extension is called Green Software Measurement Ontology (GSMO) and provides terminology and the corresponding definitions to help authors of papers about energy measurements describe their process with a specified terminology. This also helps readers of papers about energy measurements understand and compare results from different papers. The Methodological component is a process called Green Software Measurement Process (GSMP) which are guidelines to assist with the study design, analysis, and presenting the results. It is presented in seven phases, described as follows by Mancebo et al.[52]

1. *"Scope Definition"*: A requirements specification for evaluation of the results is made and the test cases are chosen.
2. *"Measurement Environment Setting"*: The DUT and measuring instrument are chosen. The baseline measurements should also be acquired in this phase.
3. *"Measurement Environment Preparation"*: Preparation of environment
4. *"Perform the measurements"*: Experiment is conducted and data is recorded.
5. *"Test Case Data analysis"*: The data is processed and analyzed. Here outliers can be found and a sanity check of the results is done.

6. *"Software Entity Data analysis"*: Conclusions about the experiments are started from the analysis in the previous step.

7. *"Reporting the results"*: Here the process and results of the study are documented.

Mancebo et al.[52] claim that the methodological component aids in giving more reliable and consistent measurements which in turn makes them more comparable and replicable.

The technological component consists of two parts. The first part is the Energy Efficiency Tester (EET) which is a hardware-based approach to measuring energy consumption. It is presented in more detail by Mancebo et al. in [51]. EET is a portable measuring instrument, which has three main components: A system microcontroller which is a Mega Arduino Development Board responsible for getting the data. The Mega Arduino Development Board has a set of 9 sensors for measuring the energy consumption and temperature of different components of the DUT. Lastly, a power supply has to be used instead of the DUT's original power supply because the sensors are connected to the power supply.[51] The second part is called ELLIOT, which is a software tool that handles the data provided by EET with the main goal of providing a visual representation that can be used to process, analyze and construct graphs and tables of the data collected by the EET. It can identify potential outliers and calculate different statistical variables.[52] It should be noted that to the extent of our knowledge, the ELLIOT tool is not available to the public and can therefore not be used.

Another work is by Sestoft[78], where a framework to measure the execution time of microbenchmarks is created in an iterative manner where different pitfalls are uncovered. The reason why energy consumption measurements are not straightforward is because of a large number of factors which has been introduced in complex modern systems. This is especially clear on managed execution platforms like the Common Language Infrastructure e.g .Net from Microsoft or the Java Virtual Machine (JVM) where software in an intermediate form is compiled to real machine code at runtime by just-in-time (JIT) compilation, which affects the execution time for a couple of reasons. One reason is the start-up overhead of the JIT or the adaptive optimization of JIT compilation. What the optimization does is that it estimates how many times different parts of the code are executed during run-time. This results in a prioritization, where a lot of time is used to generate optimized code for code executed many times, and code only executed a few times is quickly generated and thus less optimized. The JIT compiler also avoids using a lot of time on code analysis, which can result in cases where the generated code works well in simpler contexts, and in more complex contexts, the performance is not as good. Another factor the programmer cannot control is automatic memory management, which may decide to run the garbage collection during the run of a test case, resulting in unreliable results. The same can be said for the operating system, processors, and memory management systems, where garbage collectors can run, resulting in unreliable results.

In the process of creating the framework for measuring execution time, Sestoft[78] uses a `multiply` method which performs 20 floating point multiplications, an integer bitwise "and" and a conversion from a 32-bit int to a 64-bit double. During the first phases, one observation is that when running the test cases for many iterations, the execution time drops to zero. This happens because the JIT compiler observes that the result of the method is not used for anything, resulting in it skipping the method entirely. When measuring the runtime of the test case, only measuring one execution is deemed too little as the results vary too much. This should rather be multiple runs, and then looking at the average runtime and the standard deviation. These multiple runs are in the work by Sestoft et al.[78] deemed to be until the execution time exceeds 0.25 seconds, as this is long enough to avoid problems with virtual machine startup and clock resolution. In the end, some additional pitfalls are noted, including:[78]

- Shut down as many background services as possible, as this can impact performance
- The generation of logging and debug messages can use more than 90% of execution time, so this has to be disabled
- An IDE uses a lot of CPU time and causes slower execution time because of debugging code, so do not execute through IDEs
- Disable power-saving schemes so the CPU does not reduce its speed during benchmark runs.
- Compile with relevant optimization options so the generated bytecode does not include code to accommodate debugging
- Different implementations of .NET and JVM have different characteristics and different garbage collectors
- Different CPU brands, versions (like desktop or mobile), and hardware (like ram) have different characteristics
- Reflect on results, and if they look slow/fast something might have been overlooked.

Bokhari et al[6] found that when running benchmarks comparing different variants of the same program on Android systems, noise had an impact on the results. This was due to noise coming from several uncontrollable factors such as:[6]

- System software states changing from one restart to another and just changing during a run. This is partly because not all background processes from the system begin immediately after a reboot and additionally some are triggered by battery levels.
- Memory consumption of the Android system due to background processes that cannot be disabled and the Android memory management system.
- The battery voltage. Even though the system was fully charged at the starting point the voltage at the beginning and at the end of an experiment varied from run to run.

To solve this they propose a method called *R3-VALIDATION*[6] which is a rotated round-robin approach to running the test cases (program variants), which ensures more fair execution conditions. In this approach the test cases (A, B, C) are rotated as follows: setup, ABC, ABC, setup, BCA, BCA, setup, CAB, CAB. Here the setup phase is a restart, initialization, and recharge of the system. The amount of times to run a set of test cases between setup phases is based on the battery level. This battery level limit is chosen based on when the battery level would cause noise in the results, which occurs when the energy-saving mechanisms are started. They achieved more consistent system states and memory consumption from this approach and therefore gave more fair results for the different program variants.[6]

When comparing the results gathered from either different DUTs or different measuring instruments different steps need to be taken. In the work by Dongarra et al.[18] the sampling rate is set to the same across all systems, to make the measurements more comparable.

2.4 Key Takeaways

In this section, the major aspects of the related works covered in sections 2.1 to 2.3 are presented.

In section 2.2 the idea of comparing different measuring instruments as Japgroep et al.[42] did was presented. However, only two measuring instruments could be set up to conduct measurements. Those were Joulemeter and Energy Consumption Tools (EC Tools). Joulemeter was last updated in 2011[11] and EC Tools was last updated in 2013[26]. Since the last updates are many years ago we decided against using them as we assumed that they are not maintained. The idea of comparing different measuring instruments as Jagroep et al. did, is however the essence of our work as well. A measuring instrument used in multiple related works is RAPL[46, 32, 20] and is as such relevant to include. However, as shown by Khan et al.[46] RAPL performs differently on different Intel CPU architectures, however, they still deem it to be a viable alternative to hardware-based measurements. A similar conclusion was made in the work by Fahad et al.[20] and Hackenberg et al.[32] where RAPL was shown to be inaccurate compared to the hardware-based measuring instruments. A hardware-based measuring instrument will therefore be used as a ground truth in our work. In sections 2.1 to 2.3 different hardware measurement approaches are used, as can be seen in table 2.1. Another hardware-based measuring instrument seen in the literature is Kill A Watt[87], which is also shown in table 2.1. These will be considered when choosing the hardware-based approach to utilize in our work in subsection 3.1.5 and subsection 3.2.3.

Hardware-based Measuring Instrument	Occurrences
Watts Up Pro	2
Plugwise	1
ZES ZIMMER LMG450	1
Custom setup	1
Kill A Watt	0

Table 2.1: The different hardware-based measuring instruments and how often they are used in the papers described in chapter 2.

Furthermore, Hackenberg et al.[32] provide insight to be considered regarding optimal sample rate depending on the type of measurements.

Another aspect presented by Fahad et al.[20] is the concept of dynamic energy consumption. Dynamic energy enables a comparison between the hardware-based measurements and software-based measurements used in our work, which will be chosen in chapter 3.

In section 2.3 different frameworks and aspects of methodology were presented. In our work, some of the contributions from Mancebo et al. [52] are incorporated to some extent. The incorporated contributions include some of the terminology defined in the GSMO, where some aspects of the GSMO are deemed irrelevant to our use case and will not be used. Some additional terminology is however also needed, which is shown in table A.1. The two other components are the methodological component and the technological components, both of which are not used. The methodological component which is guidelines to assist with the study design is not explicitly followed since we were not aware of it in the initial phase of our work, however, a similar process is used. The technological component consists of a custom hardware-based measuring instrument where the software tool is not available to the public and can therefore not be used.

Sestoft[78] presents several things to consider when running benchmarks. For example how the JIT compilation can affect the execution time of benchmarks and how garbage collection can also be

an uncontrollable factor. Furthermore, a list of potential pitfalls is presented, which are considered when implementing the framework used in our work, as presented in section 2.3.

Bokhari et al.[6] came up with *R3-VALIDATION* to avoid system software states changing from restarting the DUT and to generally improve the fairness of running several different test cases. The ideas of *R3-VALIDATION* will be adapted in subsection 4.1.2 to better fit our use case although the core idea will remain the same.

Finally, it was noted by Dongarra et al.[18] that when comparing results gathered from different DUTs or measuring instruments it is useful to have the same sampling rate across the setups. This will also be implemented if possible within the limits of the different measuring instruments used in our work.

Chapter 3

Method

Based on the knowledge gathered through chapter 2, this chapter will introduce the different components used for the experiments of our work. This will, first of all, be regarding the measuring instruments and DUTs used, which are presented in section 3.1 and section 3.2 respectively. In section 3.3, the experimental setup will be introduced. This will be concerning the test cases, measurement methodology, framework, and database. Lastly, section 3.4 uncovers how the measurements become comparable, as mentioned in **RQ1**.

3.1 Measuring Instruments

The aim of our work is to compare different measuring instruments across different DUTs and OSs, as is covered in **RQ2-4**. In this section, different measuring instruments used in our work will be introduced. When considering measuring approaches, three different kinds exist:

- Hardware-based measurements: System-level physical measurements
- Software-based energy predictive models: A model estimating the energy consumption based on performance monitoring counters.

When selecting measuring instruments, they can be either software-based or hardware-based. When considering software-based measuring instruments, multiple versions exist, and our work will aim to include a variety of measuring instruments. This will include a Linux and Windows version created by Intel, this being RAPL and Intel Power Gadget. RAPL was chosen as chapter 2 found this particular measuring instrument to be widely used in the literature, and Intel Power Gadget was chosen as it is created by the same company. Microsoft has also made its own measuring instrument for Windows, this being E3. This was chosen based on the assumption that their domain knowledge could prove valuable when creating a measuring instrument. Another measuring instrument included is Libre Hardware Monitor (LHM), which represents an open-source solution. LHM was chosen as its implementation was easy to use. An assumption we had here is that all open-source versions will measure energy with a similar method. Lastly, a hardware measuring instrument is also included and will serve as the ground truth.

3.1.1 Intel's Running Average Power Limit (RAPL)

This section about RAPL is inspired by Nielsen et al.[60]. RAPL is a tool that allows for measuring and limiting the energy consumption of different power domains in the processor. RAPL is a software-

based model for estimating energy consumption using hardware performance counters and I/O models through several model-specific registers (MRS) in Intel processors dedicated to RAPL.[75, 92] The accuracy of the energy consumption is high even though it is a software power model[33, 32]. RAPL has in some works been found to be correlated with the power consumption from the wall with a coefficient of 0.99 for some workloads,[46] while other works have found RAPL to tend to underreport and found no correlation with a ground truth measurement.[20] RAPL was introduced with Intel's Sandy Bridge architecture in 2011 and it has since improved with the generations of new chips. In Intel Haswell (2013) the control of the frequencies was improved and along with on-chip voltage regulators, the accuracy of RAPL improved. When Intel Skylake (2015) was released, a new domain was added (PSys) as well as better granularity for the PP0 measurements. Khan et al.[46] found that RAPL is a viable alternative to manual complex power monitors with a low and negligible performance overhead. [46]

The different power domains which are supported are shown in table 3.1. Although there are some limitations, since not all power domains are supported by each processor model e.g. PP1 is only supported on desktop models.

Power Domain	**Description**
CPU package (PKG)	Includes the consumption of the entire socket which is all cores and uncore components which are parts that are not in the core, but are closely connected e.g. caches, memory controller, and integrated graphics.
DRAM	Includes the consumption of the RAM which is integrated into the memory controller.
Power Plane 0 (PP0)	Includes the consumption of all cores. Excluding uncore components.
Power Plane 1 (PP1)	Includes uncore components. Excluding cores.
PSys	Includes the consumption of the system-on-chip (SoC) which along with the processor usually includes a GPU, memory, power management circuits and USB controller. Only available on a smaller subset of models.

Table 3.1: The power domains supported by RAPL

The energy counters in RAPL can be accessed by utilizing model-specific registers (MSRs). By default, these counters are updated every $\approx 1ms$. These MSRs can be accessed on Linux and supplies an interface for doing read operations directly on the CPU.[69]. There are also alternatives to using the MSRs. The Power Capping Framework allows the user to access RAPL by using sysfs, which is a pseudo file system in the Linux kernel[13, 46]. Furthermore, `perf` or `perf_events` are Linux commands available in the Linux kernel which allows to access performance counters.[88, 46]

Another option is the Performance Application Programming Interface (PAPI) library, which is a tool that can be used to measure the performance and provide an interface so the measurement in near real time[82]. Mozilla also has a tool that can print all the available RAPL power estimates[57]

In the literature, different limitations are mentioned with respect to RAPL. This includes for example poor driver support as drivers are only available for the Linux kernel and therefore not for Windows.[46] Work has also been done on how to exploit the information given by the RAPL interface, as it is argued that the security aspect has not been considered when developing RAPL[92]. This is re-

garding how information can be exploited to mount attacks. This has to do with how the fine-grained power consumption of different components of the system can be accessed and read. Using this information, one study used a RAPL-based website fingerprint technique to identify visited webpages with accuracies up to 99% on Chrome and 81% in Tor[92] and in the work by Mantel et al.[54] they can distinguish between different Rivest-Shamir-Adleman (RSA) secret keys only using energy measurements with an accuracy of 99%. To fix this issue, Intel recommends only giving read access to trusted users and enabling Intel Software Guard Extensions (SGX) or using a specific architectural MSR feature. This means the energy information report will be modified, where random energy noise is included in the calculation which creates a small delta between the reported power consumption and actual power consumption. It is however also possible not to enable this, and gain the most accurate readings.[76]

3.1.2 Microsoft's Energy Estimation Engine (E3)

Microsoft's Energy Estimation Engine (E3) is a tool created to monitor the energy consumption of batteries on Windows devices. E3 monitors the DUT at all times to create battery energy usage reports, accessible in the power battery settings under Battery usage.[91, p.43] The reported accuracy of E3 varies depending on whether the DUT has a special energy measurement chip or not. The reported accuracies are presented in [91] can be seen in table 3.2.

	CPU	**Storage**	**Display**	**Network**
Without chip	89%	<70%	<70%	<70%
With chip	98%	98%	98%	98%

Table 3.2: E3 accuracies. For devices with and without Maxim chips

In table 3.2 it can be observed that there is a large difference in the accuracy between devices with and without the chip on the motherboard. The chips referred to here are specifically the MAXIM MAX34407 or the MAXIM MAX34417, present in for example different Microsoft Surface devices.

When comparing E3 to other measuring instruments like RAPL, the primary difference is how E3 is able to estimate the energy consumption for each process running on the DUT, as opposed to RAPL where only the energy consumption of the CPU and RAM are reported. This makes it easier to draw certain conclusions about energy usage from E3 given that the estimations are accurate.

The reports created by E3 will by default contain logs of power usage for the previous 7 days, where each process and their power usage is logged. Each log entry accounts for $1-5$ minutes depending on the actual length of the process execution[14]. Each entry in the report contains many attributes, where some of the relevant attributes to this study can be seen in table 3.3

As of the time of writing to the extent of our knowledge, no study or external entity apart from Microsoft themselves has neither used nor verified the accuracy and reliability of E3 as reported by Microsoft. Because of this, the information about E3 mostly comes from blog posts and presentations by Microsoft and the information is therefore very sparse. An additional limitation is how this software only works on Windows devices with a battery, meaning that E3 is not accessible on Windows desktops.

In order to generate the raw energy report from E3, navigate to the desired destination of the report in an elevated command shell and type `powercfg.exe srumutil`. Upon success, the output should be `Completed with status 0 (0x00000000)`.

Attribute Name	Description
AppId	Unique id for each process and subprocess
TimeStamp	Start time of Measurement
TimeInMs	Duration of the Measurement in Ms
CPUEnergyConsumption	Cpu energy consumption in millijoules
NetworkEnergyConsumption	Network energy consumption in millijoules
SocEnergyConsumption	System-On-Chip energy consumption in milijoules
DiskEnergyConsumption	Disk energy consumption in millijoules
TotalEnergyConsumption	Total energy consumption in millijoules

Table 3.3: The attributes in the E3 report used in this report

3.1.3 Intel Power Gadget

The Intel Power Gadget[40] is a software tool for estimating the power consumption of Intel Core processors from the 2nd to 10th generation, for both Windows and macOS. What this software tool offers, is real-time estimations of the energy consumption in watts for the entire CPU using the energy counters in the processors. The tool also contains a command line version called Powerlog, where the estimated energy consumption systems can be accessed through externally callable APIs.

When measuring energy consumption the average power is measured in watts, cumulative energy in joules, cumulative energy in milliwatt-hours, the temperature in Celsius, and frequency in GHz. When considering the API, the sampling frequency can range from 1 to 1000 milliseconds. Here it is noted a high frequency will bring greater accuracy but worsen the performance of the system and a frequency of 100 milliseconds, which is the default value, is recommended.[85]. When using the Power Gadget API a reference to `EnergyLib64.dll` will make functions available. When using the library, it must first be initialized, by calling `IntelEnergyLibInitialize()`, where the drivers are loaded. In addition to this, the library also contains other methods. Similar to all these methods is how they return a boolean, representing if the call was a success or not. In this study, this tool will only be used for power measuring, despite also having additional features like temperatures or frequency measurements of the processor. All functions can be found in the official documentation[85], where the relevant functions can be found in table 3.4.

Intel Power Gadget and its API have so far been used in studies[7, 64, 93] but, as is noted in the studies, no study exists looking at the accuracy of the tool. In addition to this, it is also noted that the tool only measures the energy consumption of the CPU and not the GPU.

3.1.4 Open Hardware Monitor / Libre Hardware Monitor

Open Hardware Monitor is a free open-source piece of software able to monitor metrics like temperature, fan speeds, voltages, loads, and clock speeds on a computer for both the CPU and GPU. According to the documentation, the software supports most hardware monitoring chips on most motherboards and works for both Intel and AMD chips, in addition to both 32bit and 64bit windows, and any x86-based Linux. When running, the software will publish all data to the Windows Management Instrumentation (WMI), which allows other applications to use the data.[17]

This measuring instrument is, to our knowledge, not mentioned in the literature and the accuracy is not mentioned anywhere in the documentation. A fork of Open Hardware Monitor called Libre

Function	Description
`bool IntelEnergyLibInitialize();`	The library is initialized and a connection is made to the driver
`bool ReadSample();`	Sample data is read from all supported MSRs through the driver
`bool GetPowerData( int iNode, int iMSR, double *pResult, int *nResult);`	The data collected by the most recent `ReadSample()` is returned. The data returned is only for the package iNode specified, from the iMSR MSR specified. pResult represents the data returned, and nResults represents the number of double results in pResult.
`bool StartLog(wchart szFileName);`	Data is collected and written to the file specified by calling `ReadSample()` until `StopLog()`
`bool StopLog();`	All the saved data is written to the file specified in the `StartLog()` call, and stops saving data.

Table 3.4: Function prototypes of relevant functions for Intel Power Gadget.

Hardware Monitor (LHM) is available on Github[24]. From here a version without a GUI is used to minimize the energy consumption from the measuring instrument itself.

3.1.5 Hardware Measurements

Hardware-based measuring instruments are often from the literature seen as the most accurate way to measure the energy consumption of a system and have been used to compare the software-based measuring instruments[20]. The hardware-based measuring instruments are usually using some electrical probe, but the exact make and model are not consistent in the literature, which can be seen in table 2.1. To determine what hardware measuring instrument to use, the MoSCoW Method[90] has been used as shown below, where the criteria are gathered from the literature presented inchapter 2.

- Must have
  - Accuracy: The measurement should be as least as accurate as the Watts Up Pro's (1.5%)[20].
  - Frequency: The measurements should be as frequent as the Watts Up Pro's (1Hz)[20].
  - Safety: as the measurements could involve high voltages safety is a concern.[79].
- Should have
  - Ease of use: It should not be difficult to set up or obtain logging from the device. In the sense that it should not take too long to set up and it should be reproducible, for people without extensive knowledge in the electrical field.
- Could have

  - Availability: It should be available for a private consumer to promote reproducibility.
  - Price: It should be affordable for a private consumer to promote reproducibility.

- Won't have
  - DC measuring: When measuring DC, measurements are from each component within the system, which requires more experience.

When considering the hardware-based measuring instruments presented in chapter 2, one possibility is the WattsUpPro but this device is not in production anymore, whereas the other devices used in chapter 2 do not meet the criteria. The Plugwise has a reported accuracy of 5%[66], which is not accurate enough since a higher degree of accuracy is prioritized in our work. The ZES ZIMMER LMG450 is extremely accurate at about 0.11%[32], but could not be used because of the steep price point of around €6800, and this exceeds our budget. We were not able to find a Kill A Watt[89] that fits the European wall socket, therefore it is not a suitable choice for our use case. None of the devices fits the exact needs of our work, therefore a custom setup has to be made. It also has to measure in AC, as DC measurements are beyond our experience within the area. AC measurements on the wire from the wall socket to the DUT provide measurements for the whole system, which is desired for a ground truth measurement. The setup will involve some type of electrical probe able to measure the current of the energy going into the power supply of the DUT. The probe will then be connected to a logging device, chosen in subsection 3.2.3.

3.1.6 Comparison

After the introduction of different existing measuring instruments, a comparison can be made. Firstly, when considering the four measuring instruments it can be seen that most of them provide the measurements of the same components which is ideal for comparing their measurements. The most commonly used software-based measuring instrument is RAPL, but this measuring instrument is limited to Linux. When considering the Windows measuring instruments, none of them has any credible sources documenting their accuracies. One interesting tool is E3, as it estimates the power of all processes running on the computer individually, and how a Maxim chip can, according to Microsoft, bring the accuracy up to 98%.[91] E3 does however only work on devices with a battery, as opposed to LHM and Intel Power Gadget. When comparing the latter two, they seem very similar, where one big advantage of LHM is that it is open-source, so any uncertainty of how measurements are made, could be verified by looking at the actual implementation, and the implementation can be altered to fit the purpose. The advantage of Intel's Power Gadget and LHM is that they have an implemented interface, which makes it easier to access the relevant data. Opposed to this we have E3, where a log needs to be reset, and a log energy report needs to be parsed, which is not ideal. We have chosen to use RAPL, Microsoft's E3, Intel's Power Power Gadget, and LHM.

3.2 Hardware

In this section, the different hardware required for the experiments will be introduced. This will, first of all, be the DUTs, but also the different hardware required for the experiments. subsection 3.2.1 will give an overview of what hardware is required for the experiments, before going into more depth.

3.2.1 Setup

As was covered in section 3.1, four different measuring instruments were chosen, based on different criteria. Because these measuring instruments require different operating systems, the setup will be

for both Windows and Linux. Additionally, E3 has some specific demands, regarding under which circumstances the measuring instruments perform best[67, 14]. E3 first of all requires the DUT to have a battery and second of all claims to measure most accurately when the DUT has a MAXIM chip. In order to obtain a ground truth using a hardware measuring instrument, a DUT without a battery is also required, as this makes it possible to measure the energy consumption from the power outlet. Because of this, three different DUTs were chosen, to test all these cases:

- A DUT with a battery and a MAXIM chip
- A DUT with a battery and no MAXIM chip
- A DUT with no battery and no maxim chip

Furthermore, E3 also claimed to work best when the charger is disabled[67], which is an additional requirement when answering **RQ1**. When considering how to disable/enable the charger of the DUT, smart plugs are a good solution for this. In order to compare the performance of the different software-based measuring instruments, a hardware solution is also required as a ground truth. This hardware solution should be able to log to a computer so that the measurements can be analyzed.

3.2.2 DUTs

Following the introduction to what kind of DUTs are required for our work in subsection 3.2.1, this section will introduce the DUTs able to meet the requirements. Before going into depth with each DUT, some overall comments can be made about them. One requirement for the DUTs is regarding the OSs, here the versions of both Linux and Windows should be the same across all DUTs, to make the results more comparable. The specific versions were the most recent update when the experiments were executed in November 2022. Each DUT was in addition to this running on a fresh install of each OS, to limit unnecessary background processes. Following this, the three DUTs can be introduced.

Surface Book: The first DUT was the Surface Book, first released in October 2015, where the specifications can be seen in table 3.5. This DUT was the most difficult one to find, as this is the one with a MAXIM chip. The reason why this DUT was difficult to find, was because the MAXIM chip is not mentioned anywhere in the documentation for this Surface Book, or any newer versions of any Surface devices. Because of this, it was a challenge to ensure the DUT had the chip. The Surface Book was chosen as a teardown was found[1], where the chip was pointed out.

Surface Book	
Processor:	Intel Core i5-6300U 2.4 GHz (Intel Skylake)
GPU:	Intel HD Graphics 520
Memory:	DDR3 8GB
Disk:	SAMSUNG MZFLV128HCGR-000MV 128GB
Ubuntu version:	Ubuntu 22.04.1 LTS
Linux kernel:	Linux 5.15.0-52-generic
Windows version:	Windows 10 build: 19045.2251

Table 3.5: The specifications for the Microsoft Surface Book

[1] `https://www.ifixit.com/Teardown/Microsoft+Surface+Book+Teardown/51972`

Surface Pro 4: The Surface Pro 4 represents the DUT with a battery, but without a MAXIM chip, and was chosen based on what was available at the University. The DUT was released in October of 2015, meaning it is from the same generation as the Surface Book, but with an upgraded CPU and more RAM, as can be seen in table 3.6.

Surface Pro 4	
Processor:	Intel i7-6650U CPU 2.20GHz (Intel Skylake)
GPU:	Intel iris Graphics 540
Memory:	DDR3 16GB
Disk:	SAMSUNG MZFLV256HCHP-000MV 256GB
Ubuntu version:	Ubuntu 22.04.1 LTS
Linux kernel:	Linux 5.15.0-53-generic
Windows version:	Windows 10 build: 19045.2251

Table 3.6: The specifications for the Microsoft Surface Pro 4

Workstation: The workstation represents the DUT without a battery, making it possible to measure the energy consumption through a hardware measuring instrument. This DUT was also chosen based on what was available at the University and came with a external GPU, which was removed for comparability as neither of the other DUTs had an external GPU.

Workstation	
Processor:	Intel i7-8700 CPU 3.20GHz (Intel Coffee Lake)
GPU:	NVIDIA GeForce GTX 1060 6gb (removed)
Memory:	DDR4 16GB
Disk:	Samsung SSD 970 EVO Plus 1TB
Motherboard:	TUF B360M-PLUS GAMING
Power Supply Unit	Cooler Master RS-500-ACAB-B1
Ubuntu version:	Ubuntu 22.04.1 LTS
Linux kernel:	Linux 5.15.0-52-generic
Windows version:	Windows 10 build: 19045.2251

Table 3.7: The specifications for the Workstation

3.2.3 Equipment

Following the introduction to the different DUTs, the different hardware required to conduct the experiments will be covered. This will be done both in terms of how they work individually, and also how they work together.

AC Current Clamp MN60: This section will cover the electrical probe we found that meets the criteria from subsection 3.1.5. For the measurements, an AC clamp was used, specifically the MN60 created by Chauvin Arnoux[9]. The MN60 is available from multiple vendors, at around €400. The output of the MN60 is a continuous analog signal, meaning that it does not have a sampling rate as the sensor responds immediately to the environment[1]. The MN60 sampling rate is controlled entirely by the device used to read the analog signal from it[1]. To read the signal the current clamp has to be connected to an oscilloscope, with a BNC connector[4]. The clamp itself is connected to the electrical

phase of the wire leading to the power supply for the DUT. An illustration of the usage of the MN60 together with an oscilloscope can be seen in fig. 3.1

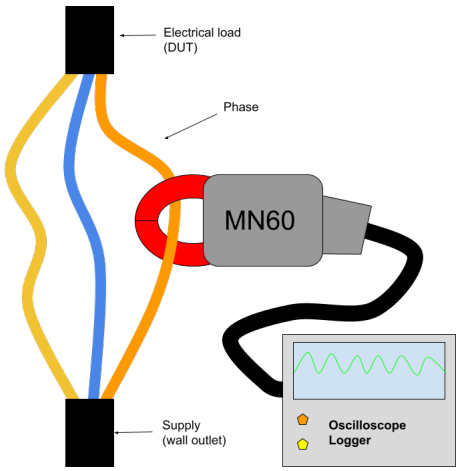


Figure 3.1: Here the MN60 current clamp can be seen connected onto the phase of the wire into the DUT

The output from the current clamp is in mv with a specific conversion rate to a current root mean square (RMS) value, which together with the wall socket specification, this being 230 volts in the EU[79], can be used to calculate the exact amount of joules flowing through the wire at any point in time. The sampling frequency of the current clamps is dependent on the oscilloscope used and the accuracy of the clamp is claimed to be < 1%[4]. As can be seen in 3.1 there is no exposed wiring, that could be accidentally touched or create a short circuit. Thus making it safe especially when measuring mains voltage.

Analog Discovery 2: The Analog Discovery 2 created by Digilent[15] is an electronics multitool that can be connected to any computer via USB and controlled using the proprietary software WaveForm. The Analog Discovery 2 has multiple uses for example it can be used as a Logic analyzer, a variable power supply, and an oscilloscope among other things as described in[16]. An advantage of the Analog Discovery 2 over other oscilloscopes is that it is made to be controlled via a computer, which makes it much easier to log measurements directly from the clamp, as opposed to a traditional oscilloscope. Using a scripting language provided through WaveForm, it is possible to automate interactions between the clamp and Analog Discovery 2. This makes it possible to automate when the measurements should be saved. The measurements will through the Analog Discovery 2 have a frequency of 10Hz. This enables easy and continuous measurements and logging. The WaveForm software has thorough documentation regarding the best practices of how to use the software, making it easy to measure the energy consumption of the DUT without a battery. WaveForm is available for both X86 and ARM, 32-bit and 64-bit chips for Mac, Linux, and Windows making it platform portable.[16]

Raspberry Pi 4: The Raspberry Pi 4[71] is the fourth iteration of the popular single-board computer. The Raspberry Pi runs an OS called Raspbian which is an adapted version of the Debian Linux OS. Raspbian is a 32-bit operating system and the Raspberry Pi 4 has a 64-bit ARM CPU.[71] Because of the portability of the WaveForm software, it is possible to install it on the Raspberry Pi making it a dedicated measuring tool for energy consumption. Since the Raspberry Pi 4 can access the internet, it can process the measurements and upload the results as needed. With the scripting available for the WaveForm software it is possible to make a fully automatic measuring setup. Given these properties, a Raspberry Pi 4 will be used as the system starting, stopping, and processing data from the clamp.

CloudFree EU smart Plugs: When running the experiments, two CloudFree EU Smart Plugs[10] were acquired, to enable and disable the chargers of the two DUTs with batteries. This was done as it is claimed that E3 works best when the DUT is not charging while it is measuring the energy consumption, as was covered in subsection 3.1.2. The Plugs are using the open-source firmware TASMOTA[25], which enables control via `http` requests directly to the plug, avoiding any third-party applications. The plugs also have power measurement capabilities, but the exact accuracies of these are not reported anywhere on the CloudFree website, and will not be subject to testing in our work. The plug can sustain an electrical load of 3680 watts which is enough for the laptops used in these experiments.[10]

3.3 Experimental Setup

After the introduction to the measuring instruments and hardware in section 3.1 and section 3.2 respectively, the experimental setup will be introduced. This will first of all be with regards to the test cases considering both language and implementation. Another aspect is how the experiments will be conducted, how the energy consumption of the test cases will be measured, and how it will be saved.

3.3.1 Languages

When considering what languages to use in the implementation of the test cases, different approaches have been seen. One such approach is from the work by Pereira et al.[65], where 27 different languages were chosen, to compare the energy consumption of similar implementations in different languages. The aim in our work is however a bit different, where instead measuring instruments are compared, as is mentioned in **RQ2**. Because of this, the specific language is not expected to make a huge difference, which is why C# is chosen, as this is the preferred language of the authors. When publishing both test cases and the framework presented in subsection 3.3.4 it will be done to a folder, with the default settings seen in table 3.8.

Name	Setting
Configuration	Release | Any CPU
Target Framework	net6.0.9
Deployment Mode	Framework-dependent
Target Runtime	Portable

Table 3.8: Publishing settings for Visual Studio

3.3.2 Test Cases

After the language has been decided, another aspect to consider is which test cases make sense to run during the experiments. Based on what was found in chapter 2, two common repositories are used, either Rosetta Code[73] or the Computer Language Benchmark Game[83]. From Rosetta and the Computer Language Benchmark Game, different test cases are chosen in literature, depending on what part of the DUT the test case should target. In the end, the choices were made based on the work by Lima et al.[29] and Koedijk et al.[47], where the benchmarks tested either the CPU, memory, synchronization, or IO. The choices for the experiments conducted in our work can be seen in table 3.9. When choosing the specific implementation of each test case, it was done based on speed, where the fastest implementation was chosen for the newest version of C#. Unsafe solutions were also disregarded, as we consider the other implementations to be more realistic.

Source	Name	Target
Benchmark game	Binary Trees	Memory
	Fannkuch-Redux	CPU
	Nbody	CPU
	Fasta	CPU, Memory, Synchonization and IO

Table 3.9: The benchmarks chosen for the experiments.

3.3.3 Measurement Methodology

When running the experiments, different types of measuring instruments are used, as is presented in section 3.1. Representing hardware measurements, a clamp was found in subsection 3.2.3, which was able to fit the needs presented in our work, in subsection 3.1.5.

The ground truth as measured by the clamp will be compared against different software-based measuring instruments. The software-based measuring instruments will in our work be RAPL, Intel Power Gadget, LHM, and E3 found in section 3.1. Here RAPL represents the state of the art measuring instrument, based on the literature covered in chapter 2. One limit of RAPL is however how this measuring instrument is limited to Linux, which is why Intel Power Gadget, LHM, and E3 were chosen as additional measuring instruments, as these all work on Windows. Between the different measuring instruments, RAPL, Intel Power Gadget, and LHM are similar, as they measure the entire energy consumption of the CPU. To isolate the energy consumption of the test case, the notion of dynamic energy is introduced, as presented by Fahad et al.[20], covered in section 2.3, which requires the idle energy consumption. Opposed to this, E3 is different as it is able to estimate the energy consumption of each process running on the CPU.

When conducting the experiments aiming to compare the different measuring instruments, some specefic control over the DUT is required. This control is based on a combination of what was presented in section 2.3 by Sestoft[78] and Fahad et al.[20], and are as following:

- The DUT is reserved exclusively for the experiments, this is to prevent deviations in the results.
- The networking will be disabled on the machines to ensure that these do not affect the results.
- The processes and the temperature in degrees Celsius of the machines will be measured before and after each experiment.
- After each batch [2], the DUT will be restarted and its setup phase is performed before continuing to the next batch.
- The memory of the test cases will not exceed the DUTs main memory to avoid memory swapping
- Some baseline power usage will be established by measurements before and after the experiments
- The generation of logging and debug messages can use more than 90% of execution time, so this has to be disabled

[2] A set of test cases executed in sequence with a cooling of periods, as presented in table A.1

- Disable power-saving schemes so the CPU does not reduce its speed during benchmark runs.
- Reflect on results, and if they look slow/fast something might have been overlooked.

Each batch will be tested on every DUT with each type of OS, these being Windows and Linux. Each DUT will have a slightly different setup depending on the hardware and the OS of the system. Some unnecessary background processes will be disabled to ensure as little noise in the measurements as possible[78], the specific process disabled can be seen in table 3.10. These processes were picked based on observing which processes were executing in the background and picking out the irrelevant processes to be disabled. It can be noted that there are no disabled background processes for Linux, which was the case because no unnecessary background processes were observed. A few additional things to note here are that all DUTs are run on a fresh install of the OS to limit the number of background processes, and for DUTs with a battery, the energy mode was set to performance. The reason why performance was chosen was that the different DUTs had a different number of modes. The Surface Book has three modes, while the Surface Pro 4 has four. Performance was therefore chosen to make the DUTs comparable, and get the best performance. A difference between the DUTs is whether they are desktops or laptops, whereas on the desktop it is possible to perform system-level physical measurements. Because the clamp measures the energy consumption of the entire system and not just the CPU, it is important to limit the power fluctuation related to other components of the DUT, as this can impact the result. This power fluctuation is limited by setting the fan speed to max on the desktop and removing the external GPU. For the laptops, this is not an issue as the measurements do not include the whole system.

Windows	Linux
AsusUpdateCheck	
AsusDownLoadLicense	
msedge	
OneDrive	
GitHubDesktop	
Microsoft.Photos	
SkypeApp	
SkypeBackgroundHost	

Table 3.10: The background processes which are disabled when running a test case

3.3.4 Framework

In order to answer the first part of **RQ1**, a framework is introduced. This framework should be able to systematically measure the energy consumption of test cases on different DUTs and OSs. The measurements should be conducted in a rotated round-robin fashion using R3-Validation.

The first thing to consider, is the measuring instruments and test cases, as both of these need to be implemented in a generic fashion, to be able to alter between them during runtime. Because of this, both are implemented using interfaces, as this was concluded not to impact performance for Java in the work by Sestoft[78], where it is assumed a similar conclusion would be made for C#. When considering the measuring instrument, the implementation can be seen in listing 3.1, where the `IMeasuringInstrument` has four methods implemented. The `Start` and `Stop`, start and stop the measuring instruments, where the input `date` in `Start` is used to name the file in which the results are saved. `GetName` is used to get the name of the current measuring instrument, also related to

saving the data correctly. Lastly, `ParseCsv` is used to transform the data saved for a measuring instrument for one measurement, on a given time, into a `DtoRawData`, this being the format accepted by the database, which will be introduced in subsection 3.3.5.

```
1 public interface IMeasurementInstrument
2 {
3     public string GetName();
4     public void Start(DateTime date);
5     public void Stop();
6     public DtoRawData ParseCsv(string path,
7         int experimentId,
8         DateTime startTime);
9 }
```

Listing 3.1: The interface used to implement the measuring instruments

Next up, the implementation for the test case can be seen in listing 3.2, where the `ITestCase` implements four methods. Here the `GetLanguage` and `GetName` return the language the test case is implemented in, and the name of it respectively, which is used when saving the results. The `GetProgram` returns a `DtoTestCase` object containing data about the test case. The `DtoTestCase` is saved in the database, and the last method `Run` executes the test case.

```
1 public interface ITestCase
2 {
3     public DtoTestCase GetProgram();
4     public string GetLanguage();
5     public void Run();
6     public string GetName();
7 }
```

Listing 3.2: The interface used to implement the test cases

To get a better context of how the `IMeasuringInstrument` and `ITestCase` is used in the framework, pseudo code illustrating this can be seen in listing 3.3. `ExecuteAsync` will in line 5 wait until a stable condition is reached, where a stable condition is defined as a condition where the battery is charged to a certain level, and the CPU is below some temperature. Each time `ExecuteAsync` is executed, one test case will be used, as expressed in line 8. In lines 10-12 there are some conditionals regarding whether the framework should keep running the test case, or the DUT should restart. This will be decided based on three conditions, the first one being the method `ShouldStopMeasurement()`. This method will return `true` if the temperature is above some limit, or the battery levels are too low. `ismMasurementValid` ensures the last measurement was valid, and `EnoughEntires` ensures the method will not run forever, by restarting the DUT every time each measuring instrument has measured a test case a set number of times. On each iteration of the `While`-loop, a new measuring instrument is chosen, as can be seen in line 14, where `GetNextMeasuringInstrument` will take the next measuring instrument, based on which one was used in the last iteration. On the first iteration, the measuring instrument starting is based on which measuring instrument started last time the DUT was running, which is saved in the database, as covered in subsection 3.3.5. In line 18, the measurement is performed, and before and after this, different dependencies are initialized or removed. This includes for example a connection to the database, which is not required during the measurements. Lastly, in lines 25-26, the results are saved, and the DUT is restarted.

```
1  public async Task ExecuteAsync()
2  {
3      CreateNonexistingFolders();
4
5      await WaitTillStableState();
6      var isExpValid = true;
7
8      var tc = GetTestCase();
9
10     while (!ShouldStopExperiment()
11             && isExperimentValid
12             && !EnoughEntires())
13     {
14         var mi = GetNextMeasuringInstrument();
15
16         RemoveDependencies();
17
18         isExpValid = await RunMeasurement(mi, tc);
19
20         InitializeDependencies();
21
22         await Task.Delay(MinutesBetweenExperiments);
23     }
24
25     await SaveProfilers();
26     RestartComputer();
27 }
```

Listing 3.3: The method handeling dependencies and the validity of the results

To get a better understanding of how the measurements are performed, pseudo-code for `RunMeasurement` from line 18 in listing 3.3 can be seen in listing 3.4. This method will count how many times the test case is executed during the given duration and handle measurements of temperatures and battery levels before and after the measurements in lines 4, 6, and 19 respectively. In line 7 the WiFi/ethernet is disabled before the measurements are performed, line 9 triggers the garbage collection, to ensure all removed dependencies are cleaned up, and in line 11 the stopwatch and measuring instrument will be started. One thing to consider here is the order in which the measuring instrument and the stopwatch are started, since if the stopwatch is started first, it will also measure the time it took for the measuring instrument to start and stop, and if the measuring instrument is started first, the energy consumption of starting and stopping the stopwatch will be included. In our work, the stopwatch will be started first, as this was the setup in the work by Pereira et al.[65], where both versions of either starting the stopwatch or measuring instrument first were tested, and the impact was concluded to be insignificant. In addition to this, the impact will become even small if the test case executes multiple times between the start and stop of the measuring instrument. In line 13, the `ITestCase` will execute, until some predefined number of minutes has passed. When the measurement is done, the WiFi/ethernet will be enabled again in line 22, and the result will be handled.

```
1  public bool RunMeasurement(IMeasuringInstrument mi, ITestCase tc)
2  {
3      var stopwatch = new Stopwatch();
4      var counter = 0;
5
6      var (inTemp, inBat) = InitializeExperiment();
7      await DisableWifi();
8
9      RunGarbageCollection();
10
11     var startTime = StartTimeAndProfiler();
12
13     counter = tc.Run();
14
15     var (stopTime, duration) = StopTimeAndProfiler();
16
17     await EnableWifiAndDependencies();
18
19     var (endTemp, endBat) = GetEndMeasurements();
20     var experimentId = await EndExperiment();
21
22     return await HandleResultsIfValid();
23 }
24
```

Listing 3.4: The method running the test case

3.3.5 SQL

When considering existing work as presented in chapter 2, most existing work save data in comma-separated files(CSV)[47, 65]. This will however not be done in our work, as we argue SQL will improve the workflow and the ability to analyze the data later. The ability to analyze the data later in a convenient manner is argued to be important, as a lot of data will be recorded from the different DUTs, measuring instruments, and test cases. Relational data like this, where one test case, DUT, and measuring instrument can be used in zero or more experiments are an ideal use case for SQL, where it enables the user to process the data later in an optimized manner[80].

In this section, the notion of a configuration will also be introduced, as this is an important part of the experiments. A configuration will in this case save under which circumstances the experiments were executed, including what temperature and battery limits were used when reaching a stable condition and when to stop running the experiments in the framework, as introduced in subsection 3.3.4.

In this case, the test case, measuring instrument, DUT, and configuration will be contained in their own tables, to avoid replicated data, where each table will have its respective primary key presented as an auto-incremented integer. An integer is chosen, as it is easy to generate the next primary key and the primary key will not be related to the data, thus it will never be changed. The primary key of a row, in a table, will also be used by other tables when referring to that specific row, as a foreign key.

When performing a measurement, it will refer to a test case, a DUT, a configuration, and a measuring instrument through foreign keys, in addition to the data saved. The data will in this case both

include the temperature, battery level, and energy consumption. The energy consumption will be saved from each measurement as both the entire energy consumption from one measurement and a time series representing how the energy consumption evolved. The temperature and battery levels at the start and end of the measurements will be represented as zero to many relationships. The structure of the data can be seen in fig. 3.2 and is used for two things during the experiments. The relevant components for capturing results from fig. 3.2 are as following:

- Test case: Represents a test program through a unique ID, name, and which language it was implemented in.
- Measuring instrument: Represents a measuring instrument through a unique ID and name
- DUT: Represents a DUT through a unique ID, name, operating system, and version[3].
- RawData: Represents the output from the measuring instrument. This is represented through a unique ID, a measurement ID, a value, and the date of execution. The value here is a serialized object representing the data, as the different measuring instruments had different formats in their output. The raw data will represent the sum over the entire measurement, meaning how much energy was consumed as a single value.
- TimeSeries: Also represents the output from the measuring instrument in a similar manner as in RawData. The difference is how the values will be a time series during the entire measurement, with some interval between the data points. This can be used to analyze during which parts of the test case the most energy is consumed.
- Configuration: Represents what configuration was used for a measurement. Here min/max temp and battery represent the limits the system had to exceed before the results were no longer useful, and a system restart and cooldown period are required. Between represents the cooldown period between two test case runs in minutes, and duration specifies the minimum duration the test case had to run for, denoted in minutes. This is also represented through a unique ID.
- Environment: Represents what conditions the DUT was under at the start and end of the experiments, including battery levels and temperatures. The environment is represented through a unique ID, an experiment ID, a time, a type, a name, and a value. Here the name is the sensor name e.g. 'CORE #1', and the type is more general like 'CpuTemperature'.
- Measurement: The measurement ties all other tables together, where one measurement is represented with a unique ID, configuration ID, measuring instrument ID, DUT ID, test case ID and start/end times. This enables one measurement to several rows in Environment etc. In addition to this, runs represent how many times the test case was executed during the duration specified in the Configuration, and iteration represents how many times the test case has been measured at the given point in time, by the given measuring instrument since the last restart of the DUT. This is relevant for the R3-Validation, where for the first ABC, the iteration will be 1 for all measuring instruments, the second BCA will be 2, etc.

One thing of focus when creating the database, was to make it as generic as possible. This is why the Environment table is as it is, to facilitate different kinds of Environment measurements. This can also be seen when considering the test case, where it is noted what language it is implemented in, to

[3]The version is to separate different versions of the framework. Meaning each time a change is made to the framework, the version increments.

make it possible to include different languages, if it is deemed important for the experiments.

The second use case for the database is to save the state of the framework upon a restart of the DUT. This is relevant for R3-Validation, where after each restart, a new measuring instrument needs to start. The relevant tables for this are as follows:

- Test case: Same definition as before
- DUT: Same definition as before
- Run: Represents what measuring instrument was the first measuring instrument last time an experiment was run on for a given test case on a given DUT. This is represented using a unique run ID, a test case ID, and a DUT ID. In addition to this is the value, which represents the different measuring instruments used in the experiment, and which one started last time.

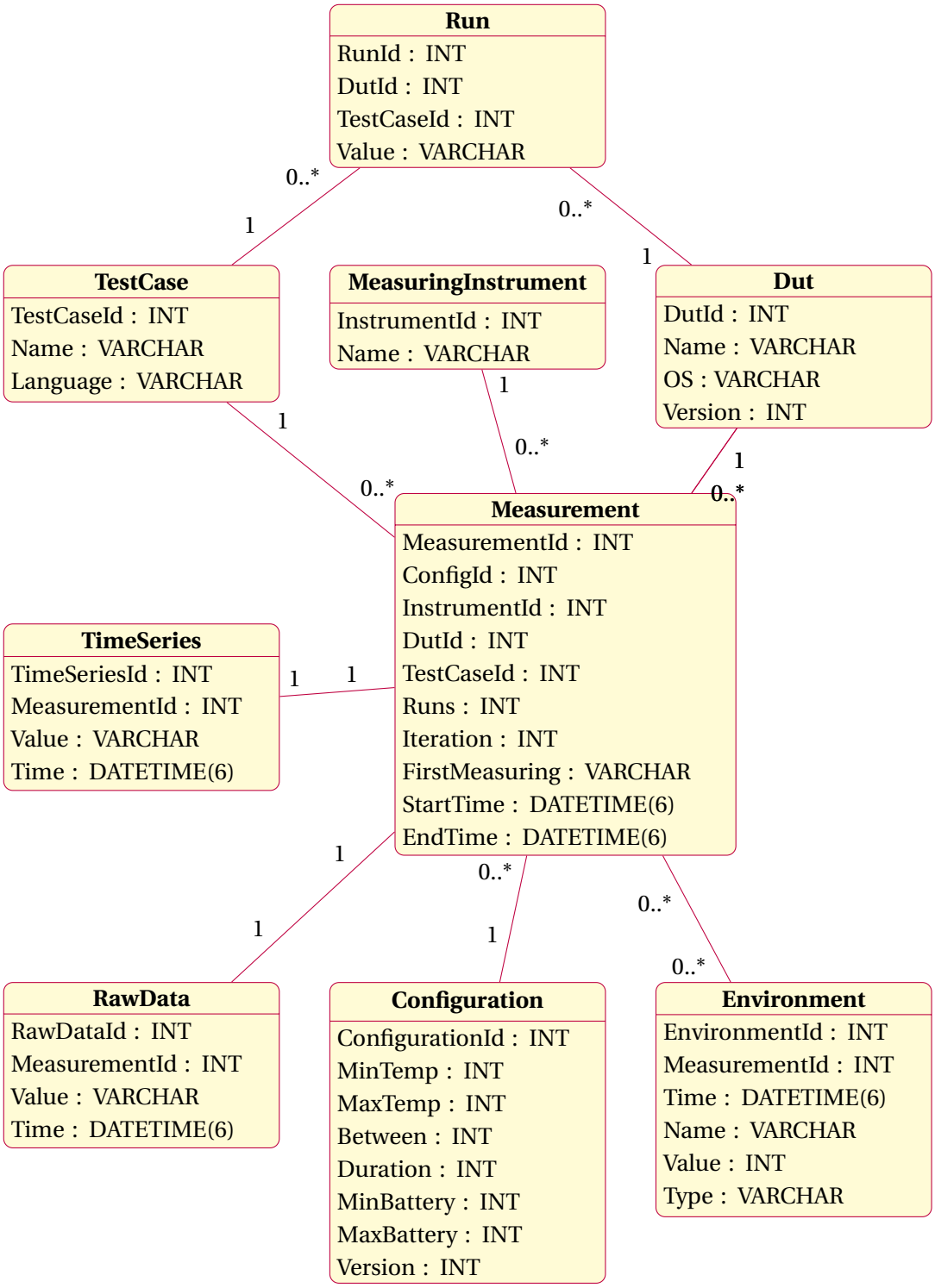


Figure 3.2: An UML diagram representing the tables in the SQL database

3.4 Statistical methods

In order to answer the second part of **RQ1** which is about making measurements comparable, this section will aim to introduce different statistical models able to do this. The selection of the various statistical methods is partly inspired by Koedijk et al.[47] and Fahad et al.[20].

3.4.1 Distribution

In this section, the methods for statistically comparing the distribution of the different measurements will be elaborated on and explained.

Shapiro-Wilk When analyzing data to find correlations, one of the most common assumptions about the data is whether it is normally distributed or not[44]. To test if a distribution is normally distributed a normalcy test can be used. While there are multiple different normalcy tests, the Shapiro-Wilk test is generally more powerful than the others that are commonly used.[72] Because of this, the Shapiro-Wilk test will be used to check for normal distributions. The formula for the Shapiro-Wilk test is:

$$W = \frac{(\sum a_i x_i)^2}{\sum (x_i - \bar{x})^2} \tag{3.1}$$

The null hypnosis H_0 for the test is[72]:

The sample comes from a normally-distributed population

To find whether H_0 can be rejected, the p_{value} is used. The p_{Value} or probability value represents the probability of achieving the same results under H_0 of a statistical test. α represents the statistical significance, where a common threshold is 0.05[86]. If the p_{Value} is less than α, it can reject H_0. Using the p_{value} is commonplace in statistical testing[29], and it is also used for the rest of the tests in this section unless specified otherwise. All tests are conducted with an p of 0.05.

T-Test If the H_0 for the Shapiro-Wilk test could not be rejected it is assumed that the distribution is normal or close enough not to be significant.

In the case of a normal distribution, a student's t-test can be utilized to check if there is a statistically significant difference between the two samples. When using a t-test, the following conditions have to be met:[44]:

- Both samples have to follow a normal Distribution.
- The samples are independent of each other.
- Both the samples should also have approximately the same variance.

The null hypothesis H_0 for the t-test is as follows:

There is no statistically significant difference between the samples

The t-test is performed using two values the t_{value} and the p_{value} and the formula for the calculation of the t value can be seen in:

$$t_{Value} = \frac{|\bar{x_1} - \bar{x_2}|}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{3.2}$$

Mann-Whitney U test If the distributions are not normally distributed another test will have to be made instead of the student t-test. A non-parametric alternative to the student T-Test is the Mann-Whitney U Test[44]. There are some important differences between the two methods. The t-test calculates the difference in the mean between two samples, while the Mann-Whitney U test checks if there is a difference in the rank sum of the samples.[53]

The idea behind the rank sum is that each data point is given a rank, and this rank is based on its position in a sorted order between all of the other data points. The basic requirement for using the Mann-Whitney U test is to have ordinal variables. Ordinal variables are variables that can be ordered and sorted. The null hypothesis H_0 for the Mann-Whitney U test is defined as:

In the population, the sum of the rankings in the two groups does not differ.

Doing the calculations require a series of calculations where the initial is finding U_1 and U_2. The actual calculations for each group consist of the number of cases in the group n_1 and rank sum T_1. These are then used in the formula:

$$U_1 = n_1 * n_2 + \frac{n_1 * (n_1 + 1)}{2} - T_1 \tag{3.3}$$

$$U_2 = n_1 * n_2 + \frac{n_2 * (n_2 + 1)}{2} - T_2 \tag{3.4}$$

The smallest of these two will then be used in the test as $U = min(U_1, U_2)$. The next calculations cover the calculations of φU and σU. φU is the expected value of U, while σU is the standard error of U.

$$\varphi U = \frac{n_1 * n_2}{2} \tag{3.5}$$

$$\sigma U = \sqrt{\frac{n_1 * n_2 * (n_1 + n_2 + 1)}{12}} \tag{3.6}$$

Then calculate the value z.

$$z = \frac{U - \varphi U}{\sigma U} \tag{3.7}$$

using z to find the p-value to check if H_0 can be rejected, again the α is set to 0.05.

Kolmogorov-Smirnov test Comes in two different variations the one-sample and two-sample Kolmogorov-Smirnov test[55]. The one-sample test is also known as the Kolmogorov-Smirnov normalcy test(KS1), where it is tested how close to the normal distribution a sample is, much like the Shapiro-Wilk test. The two-sample Kolmogorov-Smirnov test(KS2) is, however, a bit different as instead of checking if a sample is normally distributed it instead checks if the sample is from the same underlying distribution as some other provided sample. Thus KS2 is used here to determine if the two samples are from the same distribution. To calculate the KS2 the following series of calculations are needed[55].

$$D_{n,m} = max(|F_1(x) - F_2(x)|)$$

where:

$$F_1 = CDF(sample1, x)$$

$$F_2 = CDF(sample2, x)$$

where CDF (cumulative distribution function) assumes a sorted list of observations, counts the number of observations below x, and then divides with the total number of samples. The KS2 depends on a parameter en, which can be calculated using the following formula:

$$en = \frac{(m * n)}{(m + n)}$$

where n is the size of sample 1 and m is the size of sample 2. The H_0 for KS2: *The two samples are from the same distribution.* H_0 can be rejected in the case of $D_{n,m} > c(\alpha)\sqrt{en}$[55]

3.4.2 Correlation

Correlation is a statistical measurement informing about the relationship between two samples[27]. The correlation can be positive or negative. However, a strong correlation does not mean causation, just that it statically seems to be a relationship. Commonly 0 means there is no correlation between the two samples. A positive value means that the two samples are positively correlated, which means they seem to grow together. A negative value inversely means as one shrinks while the other grows.

Pearson correlation coefficient The Pearson correlation coefficient is a frequently used method for finding correlations in linear relationships[3]. The strength of the correlation spans between $-1 \ldots 1$, where the directions are given by the polarity sign. Depending on the context, different strengths are considered strong or weak. Some assumptions have to be true about the data for us to utilize the Pearson correlation coefficient:

- The data must be quantitative
- The two samples must be normally distributed
- There must be no outliers in the data
- The relationship between the two must be linear

If all of these assumptions are true for the data then the Pearson correlation coefficient can be calculated with the formula:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \tag{3.8}$$

However, if the data does not follow the conditions, another method of calculating the correlation has to be used.

Kendall Rank correlation coefficient There are two common choices when doing non-parametric rank correlation, which can be used on non-normally distributed data. These are Spearman's rank correlation coefficient and Kendall Rank correlation coefficient also referred to as Kendall's Tau coefficient[34]. While both can be used in the same scenarios there are some differences between them. Both are ranked with the same procedure that Mann-Whitney U tests are. The Spearman rank seems to be the most popular way of measuring correlations, but some argue that Kendall's Tau coefficient is better in most cases [23]. Some of the major differences between the two are the effect of outliers. Spearman's rank is largely affected by a few outliers in the data while Kendall's Tau coefficient in comparison seems more robust. Because of this the choice to utilize Kendall's Tau coefficient seems the most natural and is what will be continued with. The results from Kendall's Tau coefficient can be

interpreted the same way as from Pearson, the range is from −1 to 1. The formula for Kendall's Tau coefficient is:

$$\frac{C-D}{C+D} \tag{3.9}$$

C is concordant pairs and D is discordant. A concordant pair is ordered, while discordant is disordered pairs[45]. To evaluate the correlations the scale presented by Guildford in [31, p. 219] can be used.

Values	Label
$< .20$	Slight; almost negligible relationship
$.20 - .40$	Low correlation; definite but small relationship
$.40 - .70$	Moderate correlation; substantial relationship
$.70 - .90$	High Correlation; marked relationship
$.90 - 1$	Very high correlation; very dependable relationship

Table 3.11: The values for the scale presented by Guildford in [31, p. 219]

This is the scale that will be worked with to evaluate the correlations of the experiments.

3.4.3 Anomalies

Anomaly detection is important when using the data, as outliers can influence the results from the experiments. When an anomaly is detected it will be removed from the data to avoid corrupting the results.

DBScan To find and remove these anomalies the DBScan algorithm will be used. The DBScan algorithm is a clustering algorithm used to cluster data into groups, but it can also be used for density-based abnormality detection. The pseudocode for the DBScan algorithm is from Ester et al.[19] and is shown here:

```
DBSCAN(D[4], eps[5], MinPts[6])
{
    C = 0;
    foreach unvisited point p in dataset D
    {
        mark P as visited;
        NeighborPts = regionQuery(P, eps);
        if(sizeof(neighborPts) < MinPts)
        {
            mark P as NOISE;
        }else
        {
            C = next cluster;
            expandCluster(P, NeighborPts, C, eps, MinPts);
        }
    }
}
expandCluster(P, NeighborPts, C, eps, MinPts){
    add P to cluster C;
    foreach( point P' in NeighborPts)
    {
        if(P' is not visited)
        {
            mark P' as visited;
            NeighborPts' = regionQuery(P', eps);
            if(sizeof(NeighborPts')>= MinPts)
            {
                NeighborPts = NeighborPts joined with NeighborPts';
            }
        }
        if(P' is not yet member of any cluster)
        {
            add P' to cluster C
        }
    }

}
regionQuery(P, esp)
{
    return all points within P's eps-neighborhood (including P)
}
```

The algorithm iterates through each point and counts how many points are located within some radius of the point, where if this count exceeds the threshold (`MinPts`), it will be labeled as a core point. Points not exceeding the limit, but are still located inside the radius of a core point are instead labeled border points. The points with no label are outliers or anomalies. The variable assignment of `esp` and `MinPts` will be done via the following procedures. According to Ester et al[19] a good `MinPts` value for 2-dimensional data is 4. For the `esp` it can be calculated by finding the distance to the nearest `MinPts` are found. These distances are then sorted and plotted on a graph, where the line breaks would be a

[4] D is the dataset
[5] esp is a radius around a point
[6] MinPts is the minimum amount of points found in the radius of a point to be a core point

good value of `esp`.

3.4.4 Sample Size Formulas

It is important to get enough measurements for each test case such that the test case measurements reflect a representative mean and standard deviation. Therefore an approach to determine a sufficient amount is needed. In other words, the required sample size needs to be determined. There are several approaches used in the literature. For example, some papers use what seems like an arbitrary number of measurements[65, 47, 22]. Another method is to base the number of measurements on how many times the experiment can be run within a time-frame[78]. In this report, a formula will be used to calculate the number of measurements needed i.e. the sample size. One formula or family of formulas is Cochran's formula.[12, 48, 41], which gives the minimum number of required observations for performance metrics within a specified standard deviation. However, there are several versions of the formula. The first one in eq. (3.10)[7] is for categorical data when the population is large[41]:

$$n_0 = \frac{Z^2 * p * q}{e^2} \tag{3.10}$$

Where

- n_0 is the number of measurements (sample size)
- Z is the abscissa of the normal curve which removes an area α at the tails of the distribution. (Z-score) Where $1 - \alpha$ is the desired confidence level.
- p and q is an estimate of variance, where p is the estimated proportion of an attribute in the data and q is $1 - p$
- e is desired level of precision (acceptable margin of error)

Then there is a correction step for when the sample size is larger than 5% of the population. As well as for smaller populations a smaller sample size is necessary.[41, 48] The equation is shown in eq. (3.11).

$$n = \frac{n_0}{1 + \frac{(n_0 - 1)}{N}} \tag{3.11}$$

Where n is the new sample size and N population size.
Furthermore, there is a simplified formula called Yamane's formula, shown in eq. (3.12)[41].

$$n = \frac{N}{1 - N(e)^2} \tag{3.12}$$

Lastly, there is another version of Cochran's formula shown in eq. (3.13), Which is for continuous data.

$$n_0 = \frac{Z^2 * \sigma^2}{e^2} \tag{3.13}$$

Where instead of pq we have σ which is the standard deviation of the data. The correction formula can also be used in combination with this formula.

[7] The terminology used by Cochran is different to the one utilized in this report

It should still be considered if categorical variables will play an important role in the data analysis, then eq. (3.10) should be used[48]. For our data eq. (3.13) seems to most accurately fit the description and is therefore chosen. The correction formula eq. (3.11) is not needed since we, in theory, have an infinite population, while in practice it is of course limited by time. With Cochran's formula given a desired margin of error, desired confidence level, and an estimate of the standard deviation a sample size can be calculated. So these variables are determined as follows.

The confidence level is how little the results must deviate. A confidence level of 95% would represent 95% of the data points would match 95% of the times. If a confidence level of 95% is desired and confidence level = $1-\alpha$ then $\alpha = 0.05$. Kotrlik et al.[48] found that a margin of error of 5% is commonly chosen and is found acceptable for categorical data, but for continuous data, a 3% margin of error is found acceptable[48]. Since the variables measured are continuous data a margin of error of 3% is chosen.

The Z-score reflects how many standard deviations a measurement is from the mean. An approximate score can be found in a Z-table when the desired confidence level or α has been chosen. The most commonly used α is 0.05 or 0.01[48, 47], which is why 0.05 is chosen in this work, which gives a confidence level of 95%. From the Z-table the estimated Z-value is then 1.96.

Lastly, an estimate of the standard deviation is also needed, which is not available. Cochran listed four methods for estimating the standard deviation in case it is not initially available:

1. The measurements are taken in two steps. The first step is used to determine how many further measurements are required in step two, based on the standard deviation in the first set of measurements.

2. Utilizing results of a pilot study

3. Utilizing results from a similar study

4. Come up with a guess assisted with logical-mathematical results.

Method one and two are both based on the same principle of getting an initial smaller amount of measurements. Method three is to the extent of our knowledge not feasible since no results from a study similar enough are available. Method four requires more knowledge than we possess to give a qualified estimate. Therefore method one is chosen. Now for step one when making a small number of measurements it is required to know how many measurements to acquire. The Central Limit Theorem says that the mean of a small number of samples will become close to the mean of the overall population in correlation with an increase in the sample size. Ross found that at least 30 samples are enough for the central limit theorem to hold[74]. This means the distributions of the sample means are close to normally distributed.

From the initial sample, an estimated standard deviation of each parameter measured can be acquired. Then the minimum number of measurements required can be calculated for each parameter. Whereafter the largest of the values are chosen and all of the experiments are run that number of times to get the minimum required measurements.

Chapter 4

Experiments

After answering **RQ1**, this section will cover the experiments conducted in order to answer **RQ2-4** in sections 4.2 and 4.3. The first section will cover some initial experiments conducted, to test the performance of the different DUTs and to gain a better understanding of E3.

In this chapter, different abbreviations are introduced for labels in the different graphs and plots. These can be seen in table 4.1 and will be used from this point on in our work.

Name	Abbreviation
Intel Power Gadget	IPG
Libre Hardware Monitor	LHM
Surface Pro 4	SP4
Surface Book	SB
Workstation	WRK

Table 4.1: Abbreviations introduced for showing results

4.1 Initial Experiments

In this section, some initial experiments will be introduced and analyzed. This will be done to address some of our issues with E3. Since there is no documentation, we conduct these experiments in an attempt to get more insight into how E3 works. This will not be done for Intel power Gadget, RAPL and LHM as no issues has been observed when creating the framework. Secondly, the configuration introduced in subsection 3.3.5 needs some values for what ranges of battery percentages and temperatures are ideal for the experiment to run in.

4.1.1 E3 Experiments

The following experiments with E3 address some of the discoveries made in the initial testing, which contradicts the statements in the available sources[67, 14, 91].

This experiment was conducted based on observations where there were deviations from the expected output in the log file according to the sources and the actual output. The output of E3 is a log file, where each row in this file represents the measurement of one application in a given state. It is claimed that an application needs to run for $1-5$ minutes before the application will be added to this

file. Based on observations this is not true, as the lowest measurements were observed to be as low as 0 and 1.554 seconds. The highest value found in the test is 902 minutes, which also does not match the claim. Another aspect is that E3 counts the same application as different measurements depending on its status. Status in the context of E3 could for example be `App.Status=Focus`, `App.Status=Visible`, `App.Status=Minimized` and `App.Status=NotUnique`, and these will all be counted as different measurements in E3. The different statuses in E3 mean the following:

- `App.Status=Focus`: The application is in focus
- `App.Status=Visible`: When the application is not in focus, but still visible on the screen
- `App.Status=Minimized`: When the application is minimized
- `App.Status=NotUnique`: Background processes that do not have a user interface

The description of the different states of applications in E3 is not from any documentation but is rather based on observation. Because of the uncertainty about how E3 functions it was deemed necessary to conduct some further experiments to determine when a process is included in the measurements and when it is not. To carry out these experiments different scenarios were formulated, which will be covered now.

Order of measurement

- Scenario 1: A process is started before the measurements and ended after the measurements
- Scenario 2: A process is started during the measurements and ended after the measurements
- Scenario 3: A process is started before the measurements and ended during the measurements

Expectations The first three scenarios are designed to see when measurements are recorded by E3. The initial expectation is that E3 uses the start and exit timestamp for the measured process. Because of this, expectations are that scenarios 1 and 2 would not record a process but that scenario 3 would.

Findings Scenarios 1 and 2 were both included in the file contrary to our expectations, where scenario 1 resulted in several measurements of the process, and similar results were obtained from scenario 2. In scenario 3, the process is not found, which is also contrary to our expectations. These results point to E3 instead of using the start and end times instead take several snapshots of the process over some time. Exactly how frequent these snapshots are will be tested later.

State Change

- Scenario 4: Several process "states" are swapped between with increasing intervals
- Scenario 5: Several process "states" are swapped between with a fixed interval
- Scenario 6: Change The "state" of a single process during measurements

Expectations Scenarios 4 and 5 are meant to see how E3 handles changes in the process state during the measurements. Scenario 4 attempts to see how granular a new state is measured. While scenario 5 uses the lowest measurement found in scenario 4 to check for consistency. The expectation is that each state change will be measured, until a certain threshold where the change will no longer be registered as the swapping is more frequent than E3's sampling. Scenario 6 is designed to test something similar to scenario 4 but instead uses the same process to see how that changes the results.

Findings For scenarios 4 and 5, swapping between the states is only recorded at the first occurrence, and then the results for further state changes are aggregated together. The speed of the change does not hinder E3's measurements contrary to our expectations and does not create a measurement for each state change. Scenario 6 had the same results as 4 and 5 but provided some insight. Each application instance has the same id in E3 and cannot be differentiated based on it, but the execution time for each instance is carried over from state to state, so using this, two identical processes, can be identified since the time reported by E3 is the total execution at the time of collection.

Instances

- Scenario 7: A process opened and restarted several times during the measurements
- Scenario 8: Several instances of the same application

Expectations Scenario 7 is designed to see how E3 looks at multiple starts and shutdowns during measurements. The expectation is that each instance will be a separate measurement. Scenario 8 is designed to see how E3 handles multiple concurrent instances of the process with the same process id the expectation here is that the measurements will be merged into one big measurement.

Findings In scenario 7 a process opening and closing within 1 second seem to not always get picked up by E3, but the instances that do get picked up seem to be aggregates of the instances recorded. This could indicate that E3 might not be able to tell the difference between the instances if they are opened and closed fast enough but still knows that they did execute. In scenario 8 several instances of the same application are not immediately identifiable in E3 since they have the same id, but by looking at the execution time in each recording the different instances can be identified.

Measurement resolution

- Scenario 9: Taking several measurements with different durations

Expectations Scenario 9 is designed to test if E3 uses a per-application sampling rate or if it has a global sampling rate and exactly how frequent it is. The expectation for Scenario 9 is that measurements with durations less than 1 minute will be inconsistent as we expect a global sampling around once every minute.

Findings Scenario 9 confirmed our understanding of E3 as it showed that a new snapshot was taken exactly at the start of a minute.

Recommended usage From the experiments, several aspects of E3 has been uncovered, and an understanding of its inner working has become deep enough to utilize for further experimentation. The important aspects that were learned are summarized below:

- E3 takes the measurements in snapshots, which contain every active process on the system
- If a program is active during the snapshot it will be included, unaffected by process start, but it has to be active at the end of the snapshot
- Change state will only be recorded once every snapshot
- The separate measures of the same instance in different states are linked

- Each snapshot is taken at the start of every minute

Taking these discoveries into account a process for recommended usage can be created. Our process for using E3 is the following: Await the start of a new snapshot and then execute the program repeatedly until another snapshot is taken. Because a snapshot is taken each minute, the test case will be executed for one minute. The energy for the test case is then calculated by going through the measurement experiment id and summing each of their energy usage to get the dynamic energy usage for the whole test case. There must be some time between the snapshot for the next measurement to not include the same process twice, our findings suggest that two minutes is adequate for isolating the measurements. The code for running the experiment and the data can be found on github[1]

4.1.2 Experiment Representation

After the introduction to the framework in subsection 3.3.4, the format of the data in subsection 3.3.5, the different measuring instruments, DUTs, and test cases in sections 3.1 and 3.2 and subsection 3.3.2, respectively, the experiments can now be introduced. This will be done in the context of the different terms related to the experiments found in table A.1.

When running the experiments, it will be done based on different experiment configurations. These configurations will be different DUTs, test cases, and measuring instruments. Here the same computer running either Windows or Linux will count as different DUTs. In the framework, a test case will run for a set minimum period, before switching to the next configuration, and based on what was found in subsection 4.1.1, this period will be one minute since that is required to use E3. In this period of one minute, the total energy consumption will be captured, in addition to how many times the test case was executed, which means an average energy consumption per test case run can be calculated. After this one minute of execution, there is an optional cooldown period, which is required for E3. However, since E3 is not used in the initial experiment a cooldown period is not used. This was chosen as no argument was found as to why it should be used or what the value should be. However, there is an indirect cooldown period of about ≈ 30 seconds as the measurements are sent to the database and the setup phase is entered to ensure background processes and WiFi is turned off.

During the one minute, the test case will execute n times, resulting in n samples, according to table A.1. When the test case has been executed for one minute, this will be called a measurement. When the same configuration has been executed multiple times, it is a test case measurement according to table A.1.

As was introduced in subsection 3.4.4, Cochran's formula will be used to calculate how many measurements are required, for the results to be within a pre-defined standard deviation. Before this value is chosen, an arbitrary value of 120 is used. Meaning that we run for one minute 120 times. If based on Cochran's formula the values are not deemed sufficient, additional experiments will be executed.

In addition to this, R3 validation[6] will also be utilized in an adapted manner. The idea from the original article was to compare different implementations of the same test case, to see which version performed best. The idea was to switch between which implementation was the first to run after a restart, to make the comparison fairer. In our work, it is a bit different, where different measuring instruments will be compared. When running the experiment, with three measuring instruments A, B, and C and test cast T and DUT D, D will execute T m times in a row. The first time A will measure

[1] https://github.com/KuskIV/BiksPower

the energy consumption, then B, etc. When D has reached a state where it needs to restart (after m measurements), it will save which measuring instrument was used first. When D has restarted, it will again execute T, but this time B will measure first, then C, and so on. This will continue until the measuring instruments have 120 measurements each. Therefore, a value is also needed for how many measurements m should be made for each measuring instrument between restarts. This is done to ensure there will be multiple measurements where a test case is run with a measuring instrument for the first time after a restart. Here $m = 30$ was chosen, as this would result in each configuration having four measurements, where it was the first measuring instrument on a specific test case to run, which is important to avoid measurements for specific configurations being affected by variables in changing system states.

4.1.3 Expected Energy Consumption

In this section, the expected energy consumption of the different components within the DUTs will be covered. The values and calculations will be covered, in addition to the rationale behind them. This will be achieved by looking at the expected range of energy consumption of each component individually, this will then provide a lower and upper limit for the consumption of the DUT.

When referring to the energy consumption of computer components, a common term is Thermal Design Power (TDP). In the work by Hennessy[35], TDP is defined as a representation of the average power a processor will draw, when all cores are active and under a high complexity workload. While under peak workload the processor can reach around 1.5 times more than its TDP. The lower power limit of a CPU is disputed, but it is approximated To be around $10-25$W for most modern CPUs[77].

Workstation: The first DUT is the workstations, and the energy consumption for the different components can be seen in table 4.2.

	Workstation				
	RAM[8]	**CPU**[39]	**SSD**[84]	**Motherboard**[77]	**Fans**
Minimum consumption	$2*1.5$W	10W	$2*0.2$W	15W	
Average consumption	$2*2.25$W	65W	$2*6$W	37.5W	
Maximum consumption	$2*3$W	97.5W	$2*9$W	60W	$2*2.4$W

Table 4.2: The energy consumption for the different components in the workstation

The power supply does not have a specific energy consumption but instead has an energy efficiency rating. The energy efficiency rating represents how much of the energy going into the power supply, is used by the system. The power supply used in the workstation has an 80-Plus Gold certification, representing an efficiency of 90% at a load of 20%, an efficiency of 92% at a load of 50%, and an efficiency of 89% at a load of 100%. This is highly relevant for the clamp measurements as they are measured before going into the power supply and will thus be around 8-11% higher than the actual system consumption.

Using these values lower and upper bounds for the expected energy consumption can be created for the DUTs.

Software-based measuring instruments: The different software approaches measure the CPU and RAM individually so the minimum and maximum values for the CPU and RAM are the values the measurements should be between. Given the test cases will not put the DUT under a significant load, the measurements would be expected to be close to the following:

$$CPU : 10W < x < 65W$$
$$Ram : 1.5W < y < 2.25W$$

Where x and y represent the measured energy consumption of the CPU and Ram respectively.

Clamp measurements: The clamp measurements include the energy consumption of the entire system. To calculate the lower and upper limits to the energy consumption, the values found in table 4.2 are summarized. Based on our observations, the CPU never utilizes more than 40%. Because of this, and as there is no GPU in the DUT, the efficiency of the power supply is set to the minimum efficiency for the calculations. The lower bound is thus calculated based on:

$$Min = MinCPU, MinBoard, MinRam, AvgSSD, CPUFan, CaseFan \tag{4.1}$$

Based on Min, the $LowerCase$ can be calculated, representing the minimum energy consumption of the DUT. This is calculated as:

$$LowerCase = \left(\sum_{m \in Min} m \right) * \left(1 + \left(\frac{1}{MinEff} \right) \right) = 35.75W \tag{4.2}$$

A thing to note here is how $avgSSD$ was used instead of $minSSD$. This decision is based on observations, where the SSD is used by the OS all the time. Next up, the average case can be calculated. Here, the equation is the same, but the values change. They are as follows:

$$Avg = AvgCPU, AvgBoard, AvgRam, AvgSSD, CPUFan, CaseFan \tag{4.3}$$

And $AverageCase$ is as follows:

$$AverageCase = \left(\sum_{a \in Avg} a \right) * \left(1 + \left(\frac{1}{MinEff} \right) \right) = 119.6W \tag{4.4}$$

The measurements from the clamp are expected to be between the $LowerCase$ and $AverageCase$.

Surface Book: As the Surface Book is a laptop and is running on a battery, the clamp measurement setup will not be used on this DUT. Thus only the CPU and RAM will be subject to the measurements. The energy consumption of the components of this DUT can be seen in table 4.3

$$CPU : 7.5W < x < 14W$$
$$Ram : 2W < y < 2.5W$$

Where x and y represent the measured energy consumption of the CPU and Ram respectively.

Surface Pro 4: The Surface Pro 4 is also a laptop and is hardware wise similar to the Surface Book. They are similar in terms of the RAM, and the CPU is from the same generation, but the CPU in the Surface Pro 4 is slightly better with a higher turbo clock speed and bigger L3 cache. The energy consumption for the Surface Pro 4 can be seen in table 4.4, and the upper and lower bounds are as follows:

Surface Book		
	RAM[8]	**CPU** [61]
Minimum consumption	2W	7.5W
Average consumption	2.5W	14W
Maximum consumption	3W	21W

Table 4.3: The energy consumption for the different components in the Surface Book

$$CPU : 9.5W < x < 15W$$
$$Ram : 2W < y < 2.5W$$

Where x and y represent the measured energy consumption of the CPU and Ram respectively.

Surface Pro 4		
	RAM[8]	**CPU**[38]
Minimum consumption	$1.5 * 2W$	9.5W
Average consumption	$2.25 * 2W$	15W
Maximum consumption	$3 * 2$	22.5W

Table 4.4: The energy consumption for the different components in the Surface Pro 4

s

These should be considered rough estimates for what the DUTs can consume, values both above and below these estimates might still be valid, but require some further investigation if the deviation is significant.

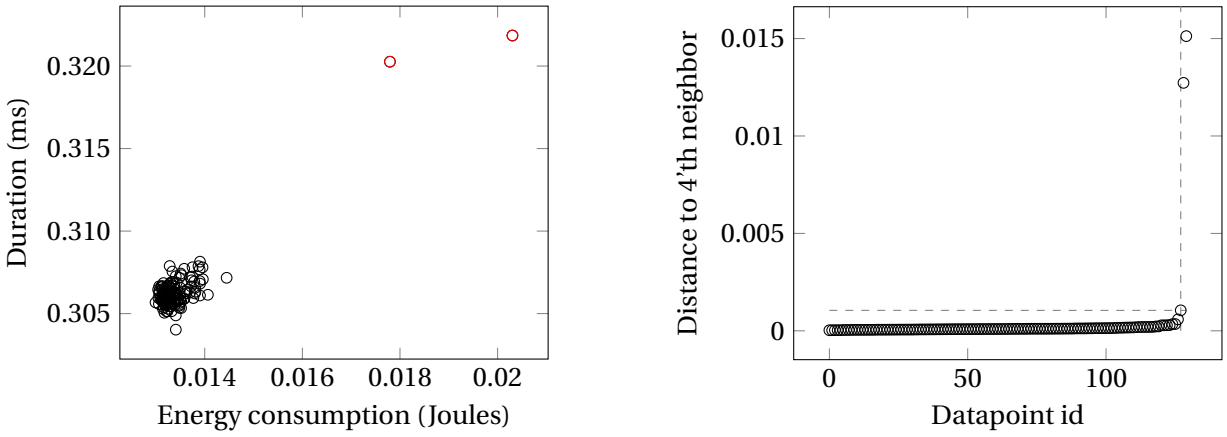
4.1.4 Anomalies Detection

Anomaly detection was found to be an important tool to avoid corrupted results. For this, the DBSCAN algorithm was found in subsection 3.4.3 to be the algorithm of choice to find anomalies in the data.

An example of a test case measurement for the workstation is shown in fig. 4.1. The example is of the test case Fasta measured by LHM, but outliers have been detected and removed for all test case measurements.

The algorithm DBSCAN used to find outliers is described in the work by Ester et al.[19]. To use this algorithm, the value of two parameters needs to be decided, this being `Eps` and `MinPts`. to do this, a `k-dist` function is defined, where k represents the distance from a point to its k'th neighbor. This function is applied to all data points, where each data point represents a measurement. The output from applying the `k-dist` to all measurements is plotted in descending order. This is called the sorted k-dist graph and can be seen in fig. 4.1b. The next step is to choose a threshold point, which is the last point in the first "valley" on the graph. All points with a higher k-dist value will then be considered to be noise. The `MinPts` will in this case be set to `k`, and `Eps` will be the y value of `k-dist(p)`. When finding this "valley", it is noted in the work by Ester et al.[19] to be very difficult to find the "valley" automatically, but it is however easy for a user to see it. Because of this, the process of finding the "valley" is done manually for all cases. In this example, the first valley is marked in fig. 4.1b with a

dotted line, where the Eps parameter is set to 0.00105. Based on this, the outliers can be found and removed. This is illustrated in fig. 4.1a, where the outliers are marked in red.



(a) The outliers (red) based on the 4-dist sorted graph

(b) A 4-dist sorted graph

Figure 4.1: An example of how outliers are found for on the workstation for Fasta measured by HardwareMonitor

4.1.5 Temperature and Battery Experiment

One thing to consider before running the experiments is to determine some values for the configuration. This includes at what range of temperature and what range of battery charge the DUT can be within without degrading performance. The reason why the experiments are not executed on all levels of battery charge is based on the work by Bokhari et al.[6] where experiments were only run when the battery charge was above a certain limit, as the DUT would otherwise enter power saving mode. Because of this, the hypothesis will be that a similar observation can be made on laptops. Another parameter that can make the DUT underperform is the temperature, as was noted in the work by Lindholt et al.[50]. For this experiment, some expectations will first be presented, before the results will be discussed. When considering how the temperature and battery levels are measured, it will be done through the OS on Linux. This is however not possible on Windows. Therefore, a software solution is needed. One software solution with this capability is LHM which is chosen in our work. When considering the different measuring instruments, LHM was not the only one able to measure the temperature. LHM was however chosen for all temperature measurements in Windows to ensure all measurements were done in the same way to make the results more comparable.

Expectations: Based on these two parameters, the assumption would be that there is some lower limit on the battery where the CPU is underclocked, and some upper limit for the temperature where the CPU will thermal throttle, resulting in a worse performance. For the temperature this is based on the work by Khan et al.[46], where a correlation between temperature and performance was found for RAPL. To find these values, the test cases will be executed on all DUTs, and the battery and temperature will be measured before and after running the test case.

Issues: When running the experiments to determine the temperature and battery, some different issues came up. These were as follows:

- Intel Power Gadget: This measuring instrument would sometimes cause the framework to crash. This occurred more frequently on some DUTs, but without any observable pattern. The errors were most likely caused by the Intel Power Gadget API, implemented in c++, which meant the

error could not be caught within C#, despite sources claiming this[36]. We were unable to replicate the error inside an IDE, and the error would close the terminal window down immediately, which meant it was impossible to see or log the specific error. To solve this, different versions of Intel Power Gadget were tested, but without success.

- LHM: This measuring instrument reached a limit of how often it could sample. LHM is implemented through the official GitHub[2] but threw exceptions if values were read more than every 500ms.
- Startup: When reading certain values and disabling the WiFi, the framework was required to run with administrator privileges on Windows. It was however not possible to start the framework automatically with administrator privileges in Windows, despite attempting to disable notifications about changes on the computer.
- Battery level: One of the DUTs was unable to charge fully up. We believe this is because it is an old battery as it is common for batteries to degrade over time. When running batteryreport on Windows it reports the capacity of the battery has decreased to ≈ 87 of the design capacity. Which is close to the max battery percentage we are able to get.
- No battery: During the experiments, there were issues with the battery-powered DUTs running out of power, and shutting down. This could be related to the DUTs being older devices and thus shutting down before reaching 0% battery.

To solve this, a few measures were taken:

- Background script: To handle cases where the framework crashed without any warning, a script was introduced on Windows. This script would run every 20 minutes, to ensure the framework was running. If it was not, it would restart the DUT. This script was implemented in PowerShell and would affect the energy measurements, but as the energy consumption of the script is consistent across all measurements, it should be removed from the energy measurements when using dynamic energy.
- AutoHotKey: To solve the issue of the framework not being able to run on startup, an AutoHotKey script was used. This would execute the framework one minute after the DUT was started, to ensure the DUT was in a stable condition. After the script has been executed once, it would shut down, thus not affecting the energy consumption during the experiment.
- LHM: Because of the issues with LHM, the sampling rate was raised to every 500ms, which was found to be stable.
- Battery level: Because of the issues both with low and high battery levels, some limits were set before the experiments were run. This was done to ensure the DUT would not stop prematurely because a DUT would charge forever, or would shut down. These values were set similarly across all DUTs, with a battery, to make them more comparable. The DUT facing issues with charging to 100% could charge to almost 90%. Because of this, 80% was chosen, to ensure it would work. For the lower limit, some different values were tested, where the issues occurred at both 5%, 10%, 20%, and 30%. Because of this, 40% was chosen as the battery level the DUT could not go below.

[2] https://github.com/LibreHardwareMonitor/LibreHardwareMonitor

A thing to note is that both scripts described are used in all experiments on Windows devices. Following this, the experiments were conducted.

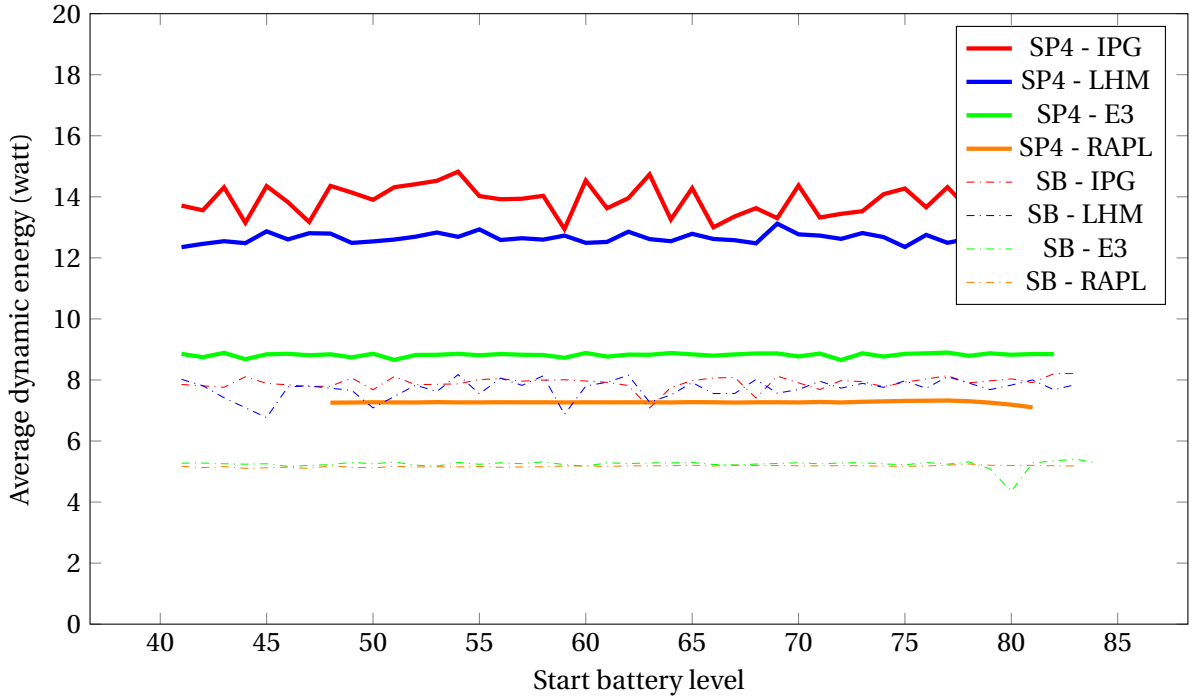


Figure 4.2: A graph illustrating the energy consumption of Cores for test case FannkuchRedux with regards to the battey level of the DUT (without outliers)

Battery level: When considering the impact of the battery level on the performance, FannkuchRedux was arbitrarily picked to illustrate this and can be seen in fig. 4.2. Only one test case is chosen in this case, as similar conclusions can be made on the other test cases, which can be found in appendix A.7. When looking at the graph, a slight decrease in energy consumption can be observed when the battery level increases, like Intel Power Gadget for Surface Pro 4. The overall conclusion for the impact of energy is however concluded to be negligible when the battery is above 40%. Based on this, the experiments will be executed with a lower limit of 40% and an upper limit of 80%.

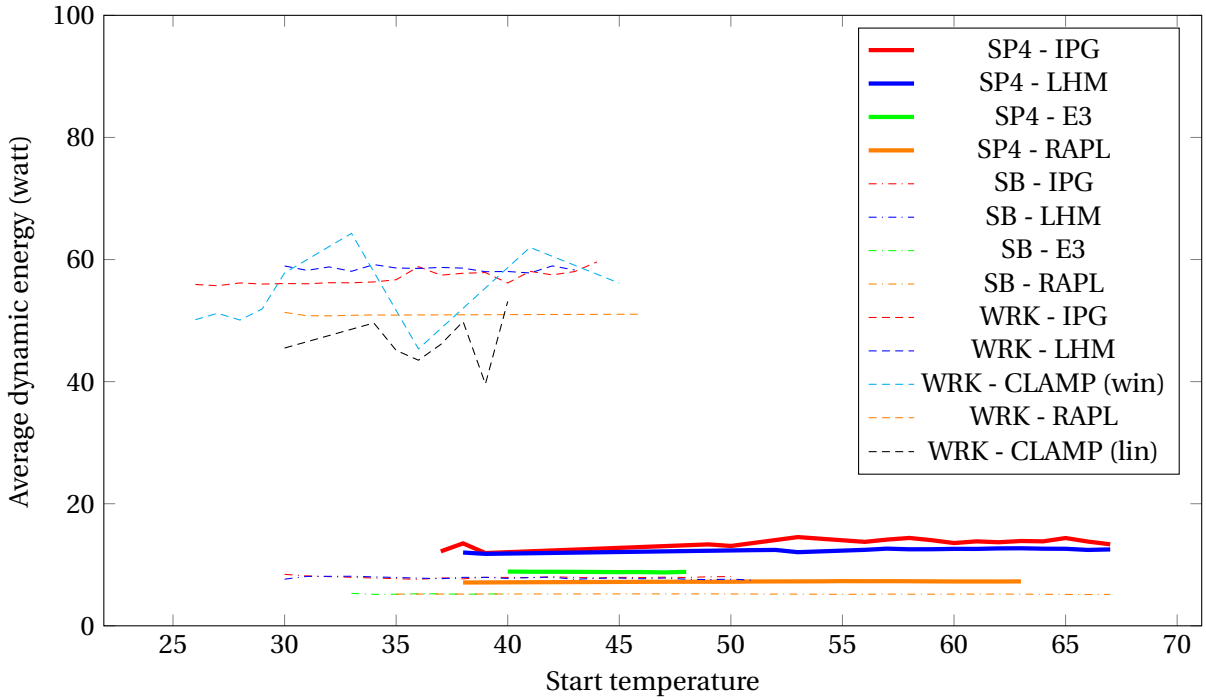


Figure 4.3: A graph illustrating the energy consumption of Cores for test case FannkuchRedux with regards to the temperature of the DUT (without outliers)

Temperature: When looking at the impact of the temperature on the energy performance, FannkuchRedux was chosen again to illustrate this, as can be observed in fig. 4.3. The other test cases can be found in appendix A.8, where a similar pattern in the data can be observed. When considering the graph in fig. 4.3, a few things can be observed. The temperatures for example never exceed 65 degrees Celsius on any of the DUTs, and the highest temperature for the workstation is even lower. This is partly because the workstation had full fan speed during all experiments, which was not possible in the laptops. Based on the expectations, an increase in power was expected to be observed when the temperature increased, which did not occur. When looking at the results, no impact can be observed, which is why no limits are set for the temperature going forward.

4.1.6 R3 Validation

In our work, an adapted version of R3 validation is used, based on the work by Bokhari et al.[6]. The data presented in this section is based on the three measuring instruments Clamp (on Windows), LHM, and IPG. When looking at the results for the different DUTs, similar observations can be made, which is one reason only the results for the workstation can be observed in figs. 4.4 to 4.7. The results for the other DUTs can be found in appendix A.6.

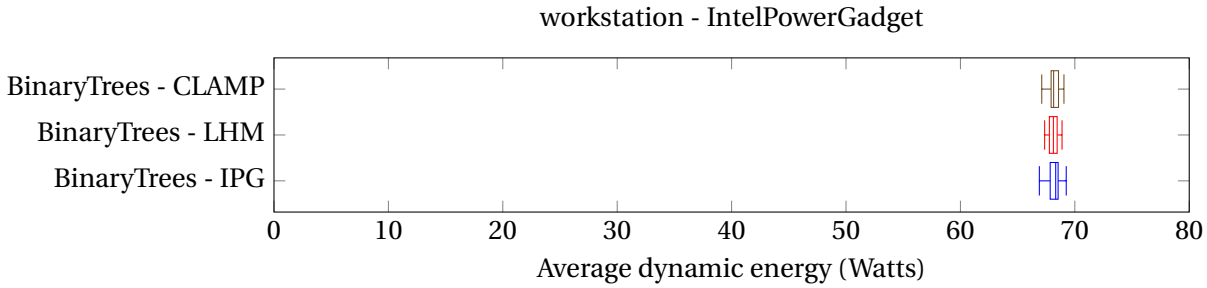


Figure 4.4: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT workstation and test case BinaryTrees, where the impact of the first profiler can be seen (without outliers)

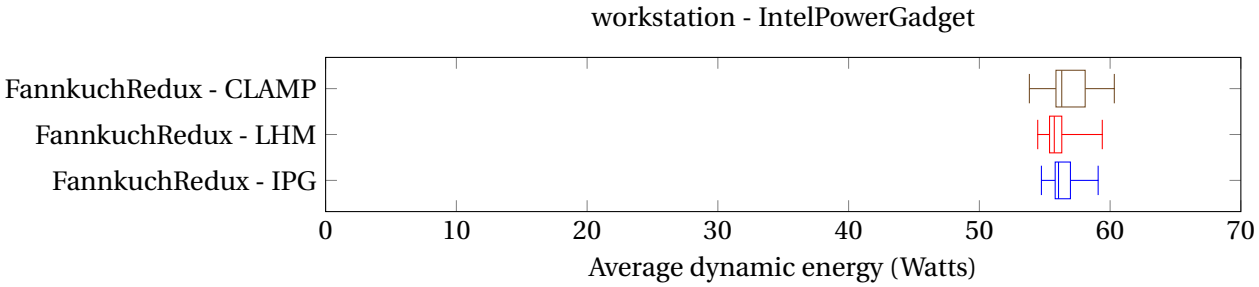


Figure 4.5: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT workstation and test case FannkuchRedux, where the impact of the first profiler can be seen (without outliers)

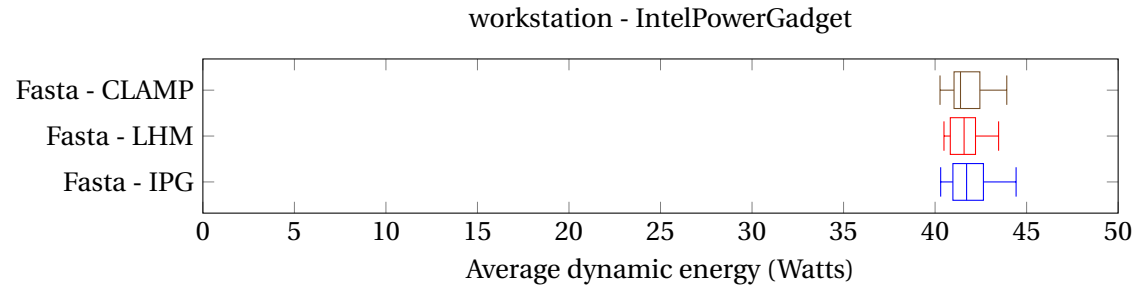


Figure 4.6: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT workstation and test case Fasta, where the impact of the first profiler can be seen (without outliers)

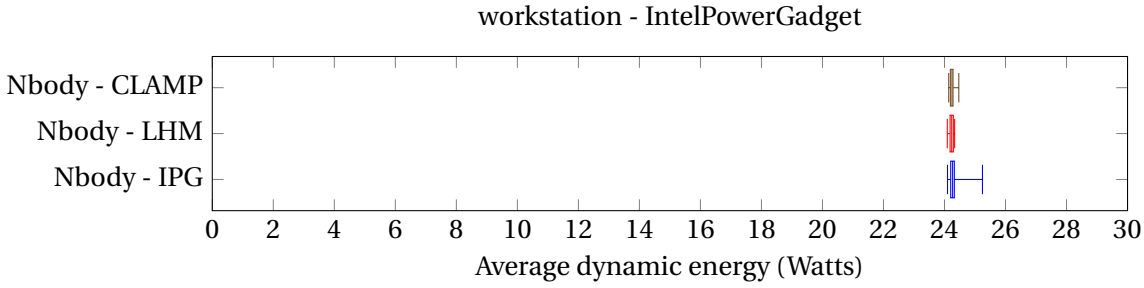


Figure 4.7: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT workstation and test case Nbody, where the impact of the first profiler can be seen (without outliers)

Expectations: R3 validation was chosen in our work because Bokhari et al.[6] observed that the order in which the test cases were executed, affected the results. This was argued to be the case because the state of the DUT changed over time, where different background processes were executed at different times and would use a different amount of processor power. This could evoke garbage collection, which would impact the measured energy consumption. Because of this, it was deemed unfair to execute the test cases in the same order, which resulted in the R3 way of switching the order upon a restart. The work by Bokhari et al. was however done on an android phone, whereas our work is on Windows and Linux, where we expect background processes to have less of an impact due to the hardware, in general, being faster and having more memory on computers than on phones. Because of this, the impact of R3 is expected to be limited.

Problems: During the experiments regarding the impact of R3 validation, E3 was not fully working in our framework yet and was therefore excluded from the experiments. However, a conclusion was made based on iterating over Clamp, LHM, and IPG. An issue occurred because of the problems with IPG, as covered in subsection 4.1.5, which caused the DUTs to crash. The crashes occurred especially for the Surface Book, less for the Surface Pro 4, and rarely for the workstation, which is another reason why the workstation was chosen to illustrate the impact of R3-validation. Because the Surface Book crashed, most of the restarts of this DUT are a result of the background script, rather than the framework. This means the distribution of which measuring instrument started, is not equal.

Conclusion: When considering the results for the workstation, as can be seen in figs. 4.4 to 4.7. The energy consumption measured by IPG can be seen, where the measurements are the average dynamic energy in watts. The label for each plot does not represent which test case was performing the measurement, but rather which measuring instrument was the first to measure on DUT startup, in addition to which test case was used. This means, for each test case, there are three plots as three measuring instruments were used on the workstation on Windows. What can be observed in figs. 4.4 to 4.7 is a smaller impact than expected, where the R3 validation can be concluded to have no impact on the measurements at all. This means the assumption about fewer background processes in Windows is true, at least on a fresh install of Windows. Because of this, R3 validation will be disregarded, and not used anymore through the experiments. Because of this conclusion, the measurements made by E3 will be run independently, without iterating over measuring instruments.

4.1.7 Sample Size

After the initial experiment phase, an estimation of the standard deviation for each configuration can be calculated. The standard deviation is calculated for each test case measurement. In the initial experiment, 120 measurements were conducted for each configuration. Using Cochran's formula as shown in eq. (3.13) we can calculate how many measurements are required. In subsection 3.4.4 we aimed to use a confidence level of 95% and a margin of error of 3%, however, the required amount of measurements using Cochran's formula was a low number. Therefore, we tried with stricter requirements than what we initially aimed for in subsection 3.4.4. The new, stricter requirements are a confidence level of 99%, which gives a Z-value of 2.58 and a margin of error of 1%. We calculated the required measurements for each configuration and the highest output was 3.05 with anomalies and 2.36 having removed anomalies. Thus if we include all anomalies and round up, the maximum required measurements are 4. Therefore since we have 120 measurements, running further measurements is not necessary. As to why such a low number of measurements is required, it is due to the standard deviation being low since each measurement in reality consists of thousands of samples.

When calculating how many measurements were required for the different test cases, one outlier was found, this being the idle test case. During the one minute of execution, the DUT would sleep for 30 seconds twice, meaning only two samples were made during one minute. When calculating how many measurements were required, this resulted in thousands of samples. In an attempt to avoid having to run this test case thousands of times, the sleep duration of the idle test case was set to 2ms, resulting in more samples for each measurement. The result of this was that Cochran's formula required 55 measurements for this test case using a confidence level of 95% and a margin of error of 0.03. Given that we had more than 55 measurements for all test cases, no additional measurements were deemed necessary.

4.1.8 Control Experiment

The last part of the initial experiments will be a control experiment. This will be done before the rest of the experiments can be conducted, to see how the energy measurements in our work, compared to other works. This control experiment will compare the energy measurements from the framework implemented in our work, to the results in the work by Pereira et al.[65] on the same test case. It should however be noted that the exact implementation of the different test cases might not be the same, as our work and the work by Pereira et al.[65] both picked the fastest implementation of each test case at the time of writing. Given the work by Pereira et al.[65] is from 2017, these will most likely be different when compared to the fastest implementation in 2022. Our work and the work by Pereira et al.[65] also differs on other areas, including:

- Temperature: The work by Pereira et al.[65] does not consider the temperature when running the test cases, which could impact the results.
- Software: When comparing RAPL in 2017 to RAPL in 2022, the performance of the measuring instrument could have changed as a result of updates to the software. The impact of these updates to RAPL are unknown.
- Hardware: The DUT in the work by Pereira et al.[65] had 16GB of ram and a Haswell Intel(R) Core(TM) i5-4460 CPU, which is an older CPU also compared to all DUT's used in our work.

When considering the results in the work by Pereira et al.[65], the energy consumption of a given test case is given in joules and a duration in seconds. When comparing the energy consumption against the measurements in our work, it is done so by calculating the average energy consumption in joules, given the time by Pereira[65], e.i. if Fasta was reported to use 50 joules in 10 seconds and the measurements from our work were 25 joules for 5 seconds, the energy consumption will be scaled to have the same time, i.e. $25J * 2 = 50$. Following this, it is time to present the expectations.

Expectations: The CPUs used in our work are all newer than the one used in the work by Pereira et al.[65], However, only the workstation has a higher clock frequency so we expect it to be able to execute the test case more times during the same period. Because of this, higher energy consumption is expected, at least for the workstation. For the two Surface devices, both CPUs have a lower clock frequency of 2.4 and 2.2GHz against 3.2GHz. Because of this, lower energy consumption is expected for both surface devices.

Results: The results from the control experiments can be seen in table 4.5, where RAPL measurements from our work are presented. In table 4.5 there are five columns, where the two first shows the time and energy consumption as reported by Pereira et al.[65], and the last three are the energy

	Time	Pereira	Workstation	Surface Pro 4	Surface Book
Binary Trees	10797ms	189.74j	613.87j (3.24)	86.68j (0.45)	55.69j (0.29)
FannkuchRedux	10840ms	399.33j	551.34j (1.38)	78.78j (0.19)	56.09j (0.14)
Fasta	1549ms	45.35j	58.28j (1.29)	12.10j (0.26)	8.40j (0.18)

Table 4.5: A comparison of the RAPL energy measurements reported by our work against the work by Pereira et al.[65]

consumption from the DUTs in our work, over the time reported by Pereira et al.[65]. When looking at the three columns for the DUTs of our work, two values can be seen, where the first is the energy consumption, and the second is how the energy consumption compares against the energy consumption of our work. An example of this could be the number 3.24 for Binary Trees on the workstations, which means that the energy consumption in our work is 3.24 times higher compared to the number reported by Pereira et al.[65]. As a result, a few observations can be made. This is first of all how the workstation always has a higher energy consumption and the surface devices have a lower energy consumption, where this is as expected. When considering each test case, Binary Trees is for all DUTs the test case deviating the most from the measurements made by Pereira et al.[65]. This could be related to what the test case targets, where the Binary Trees target memory as can be seen in table 3.9, but exactly why, is a subject for potential future work. When comparing the energy measurements, it is difficult to say if the measurements made in our work are more or less correct compared to the numbers provided by Pereira et al.[65] as the hardware and software have changed a lot since 2017.

4.2 Experiment #1

Following the initial experiments presented in section 4.1 the experiments can now be conducted. This will be done based on the findings of section 4.1. This means the measurements will execute for one minute as a result of E3, R3-validation will be disregarded, the temperature will not have an upper limit and the battery limits are set between 40-80%. A few scripts were also introduced, including a script to ensure the framework is running and a script to start the framework on startup on Windows.

This experiment is conducted to answer the research questions presented in chapter 1. With these research questions in mind, comparisons will be made between measuring instruments, OSs, and DUTs.

For the initial experiments, the experiments were run with 120 measurements for each unique test case measurement. This was done with the hope of getting a wide variety of temperatures, and battery levels as well as calculating an estimated standard deviation to use to calculate how many measurements would be required according to Cochran's formula. Based on what has been found so far, no additional measurements are required and no new requirements have been found which would require any changes to the current state of the framework. Because of this, no new measurements are made, and the data used to analyze this experiment will be the same as in the initial experiment.

The next step is thus to present and analyze the data in more depth, from this experiment, which will be done in chapter 5. Based on observations, one additional experiment will however be conducted, as will be covered in section 4.3.

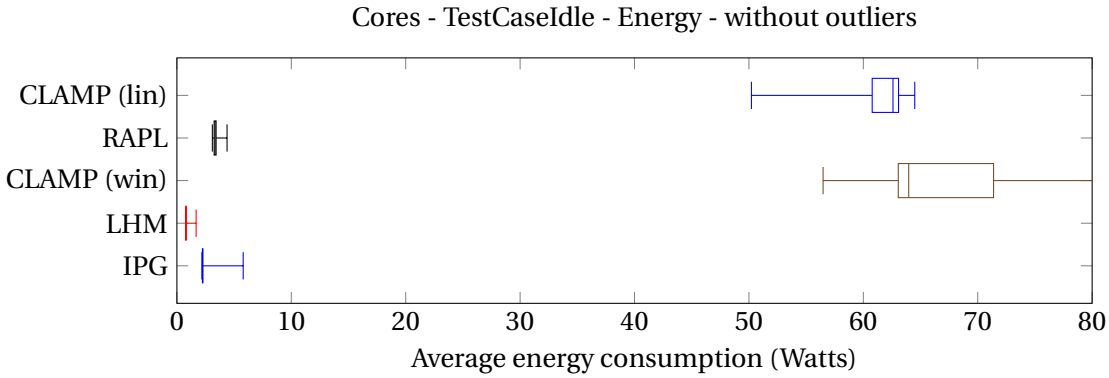


Figure 4.8: A comparison of Cores energy consumption for test case TestCaseIdle for the workstation, (without outliers)

4.3 Experiment #2

During Experiment #1 in section 4.2, a few observations differed from our expectations. This was first of all regarding Linux having a lower energy consumption than Windows when comparing the clamp measurements, and second of all that the energy consumption of the idle test case is lower compared to the expected energy consumption of the different DUTs covered in subsection 4.1.3. Because of this, a second experiment was introduced to look into this. One difference between the first and second experiments is how many test case measurements were made. In the first experiment, 120 was made, but this was found in subsection 4.1.7 to be too many. In this experiment, only 20 measurements were made for all test cases except the idle case, where 55 measurements were made.

This experiment will aim to uncover why the idle test case had a lower energy consumption than expected. The expectation is that the low energy consumption measured on the idle test case also impacts the dynamic energy consumption on the different OSs, resulting in higher energy consumption for Windows.

An example of the idle test case energy consumption being lower than expected can be observed in fig. 4.8. Here the idle test case's median energy consumption is reported by Intel Power Gadget to be $2.24W$ on the workstation, where the expectations were between $10W$ and $65W$. Because of this, some further investigation into the results was conducted. In terms of why the measurements were so low, a few things could have caused it, which will be covered now.

4.3.1 Implementation of Test Case Idle

One thing which could potentially cause the low energy consumption of the idle test case is the implementation. The implementation of the idle test case was using `Thread.Sleep()` to represent a DUT doing nothing. An alternative way of representing a DUT doing nothing was to use `Task.Delay()`. The difference between these two implementations is that `Thread.Sleep()` will block the current thread[3], where `Task.Delay()` will not block the current thread[4]. An experiment was conducted where the energy consumption was measured when `Task.Delay()` was used instead, to see if the error was related to the implementation.

[3]https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread.sleep?view=net-7.0
[4]https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.task.delay?view=net-7.0

Expectation: The expectation in this experiment is to see no difference in the energy consumption between the two implementations of the idle case. This is expected as the DUT will not be doing anything, thus blocking/not blocking the thread is expected to have a limited impact.

Results: The results of this experiment showed that the measured energy consumption is the same as when using `Thread.Sleep()` and `Task.Delay()`, as was expected. Because of this, the implementation will keep using `Thread.Sleep()` for the remainder of our work.

CPU-States

Another possible cause of the issue of lower-than-expected measurements could be related to hardware. In the work by Fahad et al.[20] they experimented with disabling Core-states (C-states) on the CPU. These states include performance states (P-states) and C-states[56], where the P-states provide a way to change the frequency and voltage of the CPU. Here P0 represents max performance and higher values of P underclock the CPU. The C-states become relevant when the CPU is doing little to no work, where certain parts of the CPU will be turned off, resulting in reduced power consumption.

Expectation: When considering the intuition, purpose, and function of the different Power States, it shows that they could potentially affect the power consumption of the DUT and could, depending on the circumstances, change the outcome of the experiments. This would cause the results of the experiments to be incorrect if the Power-States are not in the same state during all test cases. To get further information about the different states a CPU can be in, see appendix A.2

Results: To test whether the C-states were causing low energy consumption, they were disabled through the BIOS. This was only possible on the workstation as the Surface devices had very limited options available in their respective BIOS. Running the experiments again with the C-states disable seemed to have little to no effect on the measurements. Looking more into the BIOS we discovered that the TUF B360M-PLUS GAMING motherboard had three modes, performance mode, Max power saving mode, and automatic. Max power and performance mode would each change multiple BIOS settings where automatic would switch between performance and max power mode. The reason why just disabling the C-states did not impact the results, is expected to be a result of additional settings also needing to be changed. The specific changes made by max power and performance mode can be seen in table 4.6.

When performance mode is enabled, a comparison of the energy measurements from the different measuring instruments can be seen in fig. 4.9. For the workstation, the limits for the energy consumption are set to be between 10 and 65 for the CPU and between 28.2 and 113 for the entire system, as presented in subsection 4.1.3. Both values are relevant in this case as the software-based measuring instruments only measure the CPU while the clamp measures the entire system. According to fig. 4.9, all measuring instruments report an energy consumption between these limits for Windows, where Intel Power Gadget is 25.9, LHM is 23.57 and the clamp is 107.74. For Linux however, the measurements from the first and second experiments remain the same. This could indicate that Linux overwrites the C-states of the CPU, but this is a subject for future work.

The results show increased energy consumption for both hardware and software-based measuring instruments for Windows. This shows the power-saving abilities of the C-states built into the motherboard and CPU. Given the different test cases, the only test case expected to enter another C-state than C0 is the idle test case. Because of this, it makes sense to disable the C-states. Some sources mention it might be possible to control the C-states through the OS[5, 28], but this was not prioritized and it is therefore a subject for future work. Given that it is only on the workstation that we can disable

	Performance Mode	**Max Power-Saving Mode**	**Default (Auto)**
Intell(R) SpeedStep	Disabled	Enabled	Auto
Long Duration Package Power Limit	4095	Auto	Auto
Package Power Time Window	127	Auto	Auto
Short Duration Power Limit	4095	Auto	Auto
CPU Core/Cache Current Limit	255.50	Auto	Auto
PCI Express-Native Power Management	255.50	Enabled	255.50
Native ASPM	Disabled	Enabled	Disabled
PCH DMI ASPM	Disabled	L0sL1	Disabled
ASPM	Disabled	L0sL1	Disabled
DMI Link ASPM Control	Disabled	L0sL1	Disabled
PEG - ASPM	Disabled	ASPM L0sL1	Disabled
Intel(R) Speed-Shift Technology	Disabled	Enabled	Enabled
CPU C-states	Disabled	Enabled	Auto
Package C State Limit	CO/C1	C10	C10
RC6(Render Standby)	Disabled	Enabeld	Auto
Aggressive LPM support	Disabled	Enabled	Enabled

Table 4.6: These are the different BIOS setting that change based on which Performance mode is selected

the C-States through the bios, a second experiment will be conducted with the C-states disabled for this DUT only.

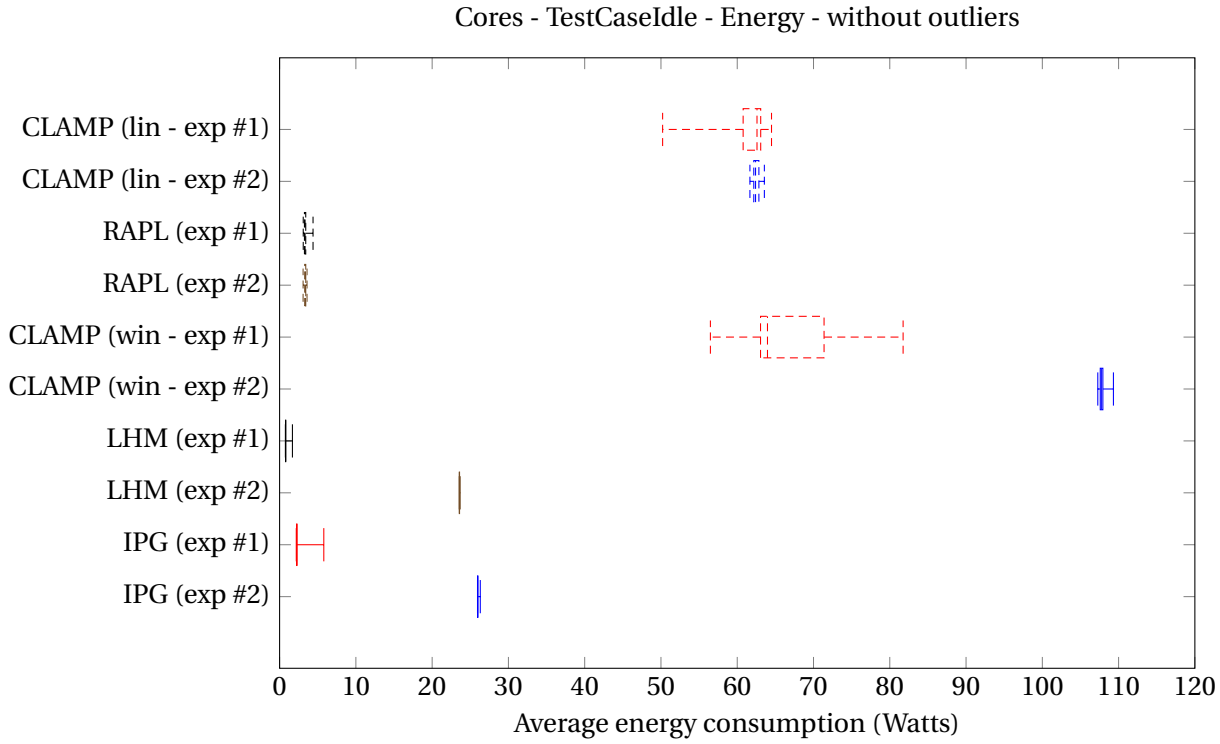


Figure 4.9: A comparison of Cores energy consumption for test case TestCaseIdle for the workstation, experiment #2 (without outliers)

Chapter 5

Results

In this chapter, the results will be analyzed and discussed. This will be done to answer **RQ2-4**, by comparing measuring instruments, DUTs, and OSs.

5.1 Statistical Analysis of Experiment #1

In this section, the various statistical methods covered in subsections 3.4.1 and 3.4.2 will be used to analyze the data collected from experiment #1, with an overall goal of finding which measuring instruments are correlated with each other. This analysis will first of all aim to check if the data is normally distributed as it has an effect on which statistical methods can be used in the later stages of the analysis.

5.1.1 Normal Distribution

The Shapiro-Wilk test will be used to test if the data is normally distributed, this is important as it affects which statistical methods can be applied to the test case measurements correctly when finding correlations as covered in subsection 3.4.1.

Expectations: For the Shapiro-Wilk test the expectation is that the majority of the test case measurements will be normally distributed. This assumption is primarily based on the findings by Koedijk et al.[47].

Result: The results for this experiment can be seen in appendix A.15. The values in this table are normally distributed if they are smaller than p, where $p = 0.05$[86]. Based on this, most of our data is normally distributed, with a few exceptions e.g. E3 on TestCaseIdle. Because not all of our data is normally distributed, the Mann-Whitney U Test is used and not the T-test. This is in line with what we expected based on the literature.

	TestCaseIdle	**BinaryTrees**	**FannkuchRedux**	**Nbody**	**Fasta**
IntelPowerGadget	0.0004	**0.3685**	0.0007	0.0	**0.0809**
HardwareMonitor	0.0	0.0033	**0.088**	0.0	0.0002
E3	**0.9307**	**0.2229**	0.0	**0.0966**	0.0002
RAPL	0.0152	0.0311	0.0	0.0	0.0007

5.1.2 Independence Test

It was found in section 5.1.1, that some of our data were not normally distributed which is why the Mann-Whitney U Test will be used when testing if the test case measurements are independent of each other. The null hypothesis H_0 for the Mann-Whitney U test is defined as:

> *"In the population, the sum of the rankings in the two groups does not differ"*[53]

Expectations: The expectation is that H_0 can be rejected in all cases as each of the measurements is obtained independently from each other and should not influence each other as found in subsection 4.1.6. The expected values will therefore be close to zero.

Results: The results from the FannkuchRedux test case can be seen in fig. 5.1, and the results for the remaining test cases can be found in appendix A.15. As can be seen in fig. 5.1, the majority of the values are below $p = 0.05$. When $p < 0.05$ the null hypothesis is rejected, which it is only with a few exceptions, including SP4 LHM and SP4 IPG. This aligns with the expectations that the samples are independent of each other in the majority of cases.

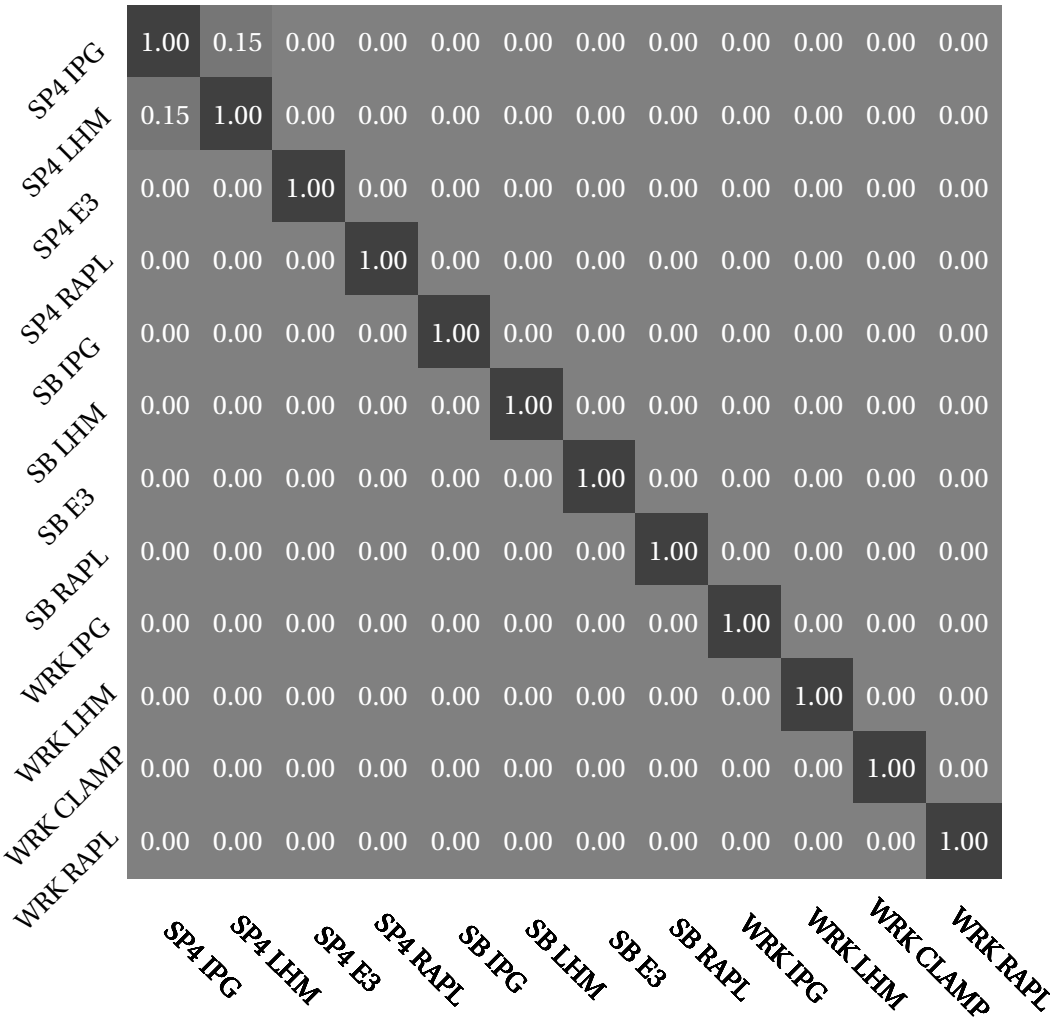


Figure 5.1: The results for the FannkuchRedux on the Mann Whitney U Test can be seen here. The Range is 0 – 1

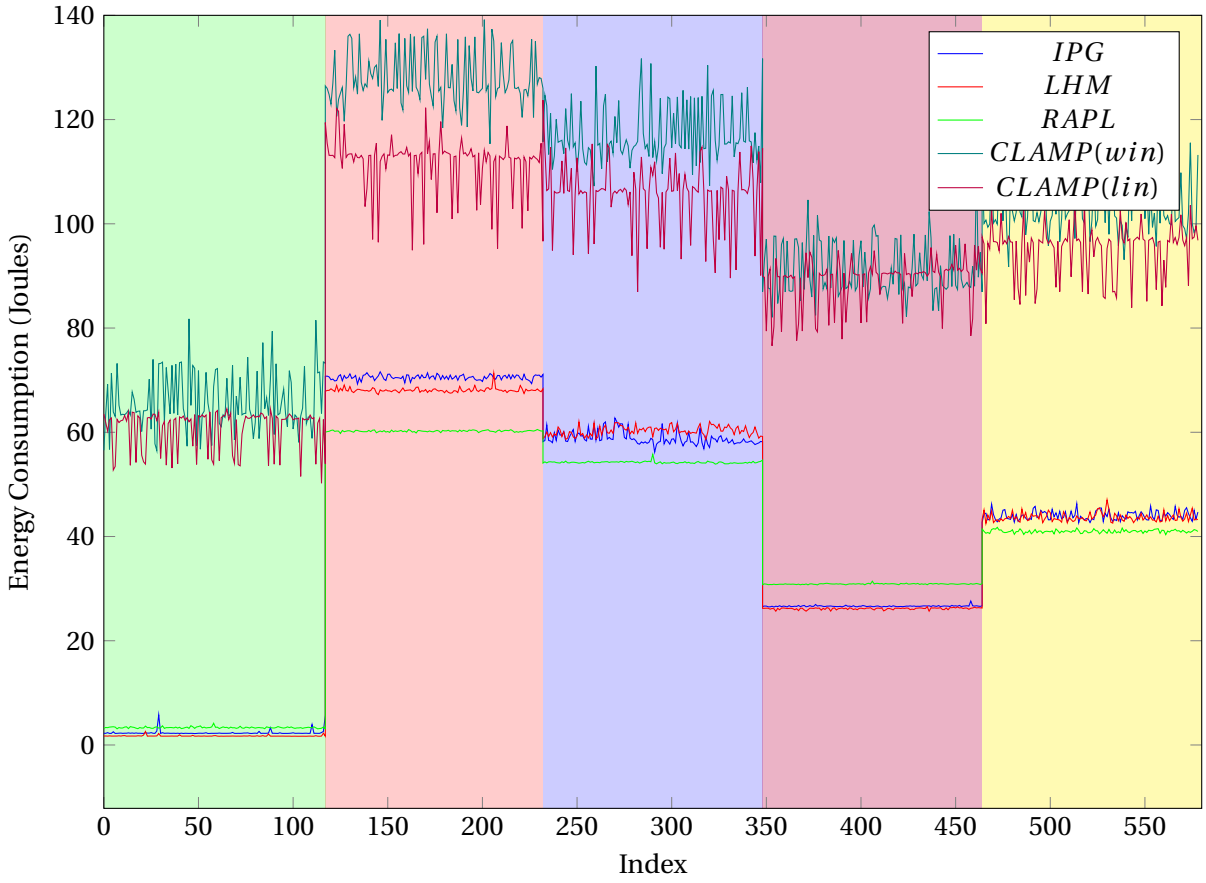


Figure 5.2: Here an example of how the corrolation is calculated can be seen where the backgrounds indicate the specefic TestCase. Green = "TestCaseIdle", Red = "Binary Tree", Blue = "Fannkuch Redux", Purple = "Nbody", Yellow = "Fasta"

5.1.3 Correlation

In this section, the correlation between the data will be calculated. To do this Kendall's Tau coefficient[45] described in section 5.1.1 will be used, as our data is not normally distributed. When testing the test case measurements from experiment #1 with Kendall's Tau coefficient, the following data is processed in the following manner. The test case measurements for each of the test cases were combined for each DUT and sorted based on the test case, where an example of this can be seen in fig. 5.2.

Expectations: The expectations are that all of the measuring instruments will be positively correlated with each other. The exact correlation is difficult to predict, but a positive correlation is expected. When calculating correlation higher correlations would generally be more indicative of valid measurements, as it shows the measuring instruments agreeing.

Result: The correlation between DUTs and measuring instruments can be observed in figs. 5.3, 5.4a and 5.4b for the workstation Surface Book and Surface Pro 4 respectively.

These results match our expectations as all of the correlations are positive. A more in-depth analysis of these results will be conducted in the following paragraphs, based on **RQ2-4**.

RQ2: When comparing the measuring instrument with each other, the correlations coefficients found in section 5.1.3 is used and when evaluating the results, the scale presented by Guildford in [31, p. 219] is used. The values for the scale can be found in table 5.1.

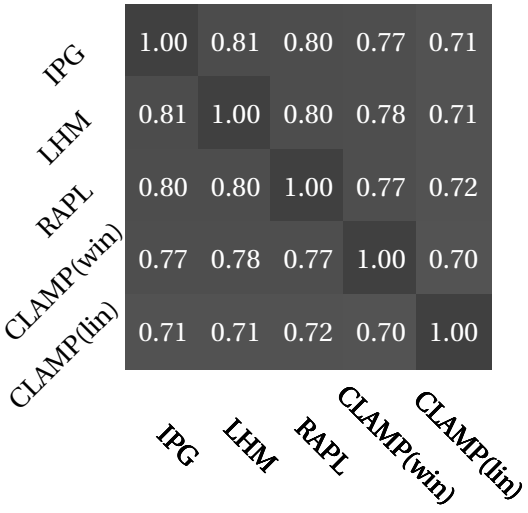
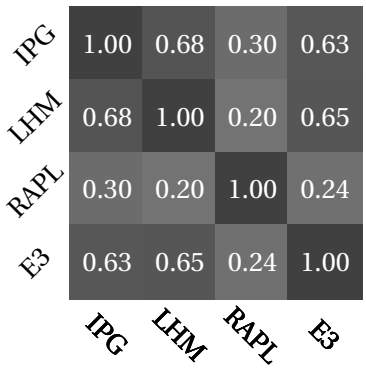
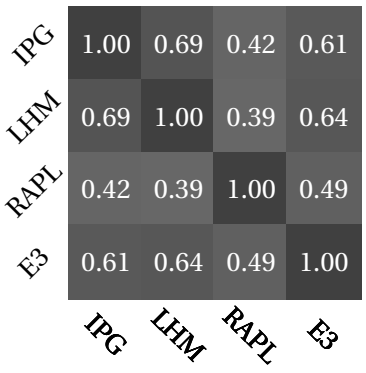


Figure 5.3: This heat map represents Correlation coefficients between the different measuring instruments -1 to 1 on the Workstation



(a) This heat map represents Correlation coefficients between the different measuring instruments -1 to 1 on the Surface Book



(b) This heat map represents Correlation coefficients between the different measuring instruments -1 to 1 on the Surface Pro 4

Figure 5.4

Values	Label
< .20	Slight; almost negligible relationship
.20 – .40	Low correlation; definite but small relationship
.40 – .70	Moderate correlation; substantial relationship
.70 – .90	High Correlation; marked relationship
.90 – 1	Very high correlation; very dependable relationship

Table 5.1: The values for the scale presented by Guildford in [31, p. 219]

Using this scale we evaluate the correlation between the different measuring instruments. Looking across the different DUTs we calculate the average correlation between each of the measuring instruments. This can only be done for the measuring instruments IPG, LHM, and RAPL, as these are the only measuring instruments used on all DUTs. When comparing the correlation between the different measuring instruments across all DUTs, the average correlation is as follows:

$$IPG|LHM = (0.68 + 0.81 + 0.69)/3 = 0.726$$

$$IPG|RAPL = (0.30 + 0.42 + 0.80)/3 = 0.506$$

$$LHM|RAPL = (0.80 + 0.39 + 0.21)/3 = 0.466$$

Next up, the average correlation between the different software-based measuring instruments against the clamp is calculated. This is done for only the workstation, as the clamp was only used on this DUT.

$$IPG|Clamp = (0.77 + 0.71)/2 = 0.74$$

$$LHM|Clamp = (0.78 + 0.71)/2 = 0.745$$

$$RAPL|Clamp = (0.77 + 0.72)/2 = 0.745$$

Similarly, the average correlation between the different measuring instruments from the DUTs with a battery is calculated, to see how they are correlated with E3.

$$IPG|E3 = (0.63 + 0.61)/2 = 0.62$$

$$LHM|E3 = (0.65 + 0.64)/2 = 0.645$$

$$RAPL|E3 = (0.24 + 0.49)/2 = 0.365$$

Looking at these numbers and evaluating them on the Guildford scale, we see that IPG | LHM, IPG | Clamp, and RAPL | Clamp all are highly correlated, while LHM | RAPL, LHM | RAPL, IPG | E3, and LHM | E3 has substantial relationships. The lowest correlation between the measurements was found with RAPL | E3 which is a low correlation.

RQ3: In regards to the different OSs and how these affect the results, it can be seen that there are differences between the measurements conducted on Linux and Windows. When looking at the coefficients for RAPL and the clamp on Linux, it can be observed that they are generally less correlated with the other measuring instruments on Windows. This can be seen when calculating the average coefficients for each of the instruments across the different DUTs.

- **Windows**

  - IPG: 0.642
  - LHM: 0.635
  - Clamp: 0.755
  - E3: 0.543

- **Linux**
  - Clamp: 0.71
  - RAPL: 0.513

Looking at these numbers it can be seen that RAPL and the hardware measurements on Linux are generally less correlated with the rest of the measuring instruments. This does make sense as they are different OSs and would be expected to have different power consumption.

RQ4 : Similarly comparing the different DUTs it is found that there are significant differences in how correlated the measuring instruments are on each DUT.

- **Workstation**
  - IPG, LHM, Clamp, RAPL: 0.757
- **Surface 4 Pro**
  - IPG, LHM, E3, RAP: 0.540
- **Surface Book**
  - IPG, LHM, E3, RAP: 0.450

When comparing the different DUTs it can be observed that the measuring instruments are more correlated with each other on the Workstation than on the two laptops. When comparing the two laptops it can be seen that the Surface Book has significantly lower coefficients than the Surface 4 Pro. Looking closely at the numbers they are very similar in all cases except for RAPL, where the Surface Book performance is worse.

- **Surface Book**
  - RAPL: 0.246
- **Surface Pro 4**
  - RAPL: 0.433

5.2 Iterations

In this section, the energy consumption over time will be analyzed. This will be done by plotting the average dynamic energy consumption for each iteration after a restart. The results will be analyzed with respect to **RQ2-4**, where DUTs, test cases, and OSs will be compared for the different measuring instruments. Before this, some expectations will be presented.

5.2.1 Expectations

The expectation for this experiment is to see a similar energy consumption when executing the same test case multiple times between a restart. This expectation is primarily based on the limited effect of R3 validation, as was shown in subsection 4.1.6.

5.2.2 Results

To demonstrate the results, FannkuchRedux and BinaryTrees were chosen as these illustrate the findings well, whereas the other test cases can be found in appendix A.13. When discussing our results, it will be done based on **RQ2-4** one at a time. Before going into depth with these, some overall comments will be made. It can first of all be noted how in figs. 5.5 and 5.6, some DUTs do not have values on the entire x-axis. This is the case for both DUTs with a battery, meaning they reach the lower limit of 40% battery before executing the test cases 30 times. It can also be seen how energy consumption does not increase over time, which is in line with our expectations.

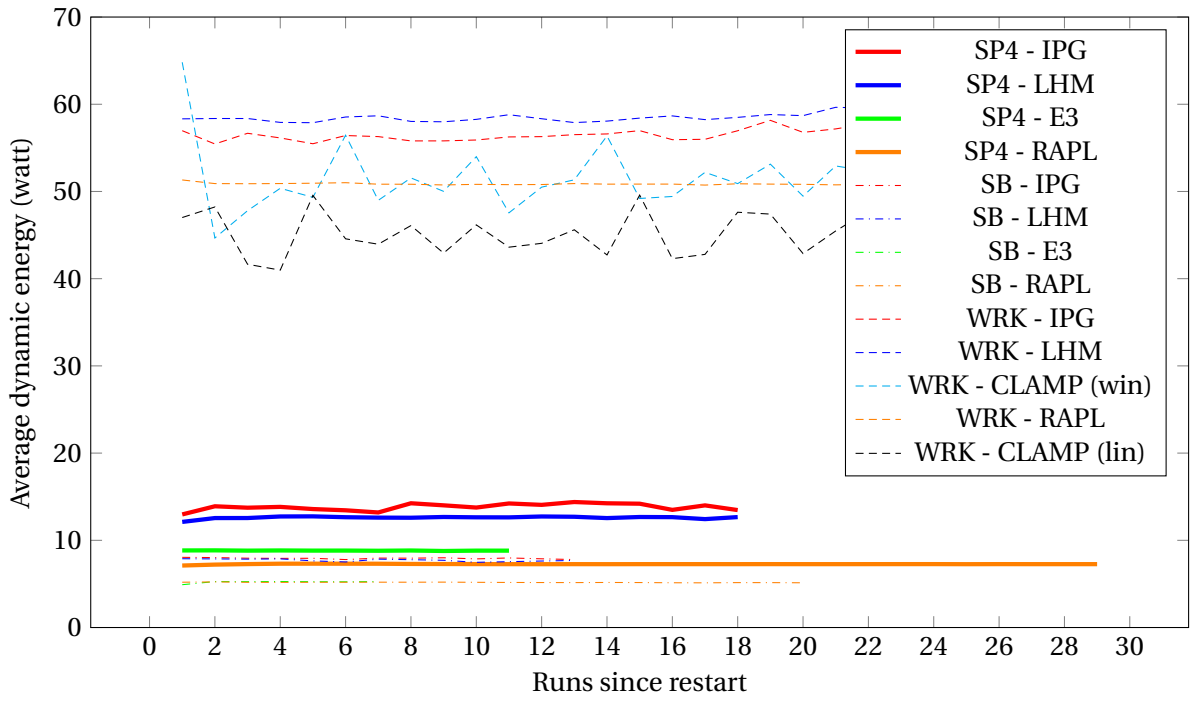


Figure 5.5: A graph illustrating the energy consumption of Cores for test case FannkuchRedux with regards to how long ago the DUT was restarted (without outliers)

RQ2: The second research question aims to compare the different measuring instruments. When comparing the measuring instruments, some overall comments can be made for all DUTs. One observation is how IPG and LHM seem to have similar measurements across all runs after a restart. When comparing IPG and LHM against E3, they are similar in some test cases, but not in all of them. In FannkuchRedux in fig. 5.5, E3's measurements are lower compared to IPG and LHM, but for BinaryTrees in fig. 5.6, E3's measurements are similar to IPG and LHM. When considering RAPL measurements, these are in most cases lower compared to IPG and LHM measurements, there are however exceptions, one being for BinaryTrees in fig. 5.6 for the Surface Book. When considering the clamp, the Windows measurements are higher than the Linux measurements, where both have measurements lower than IPG and LHM.

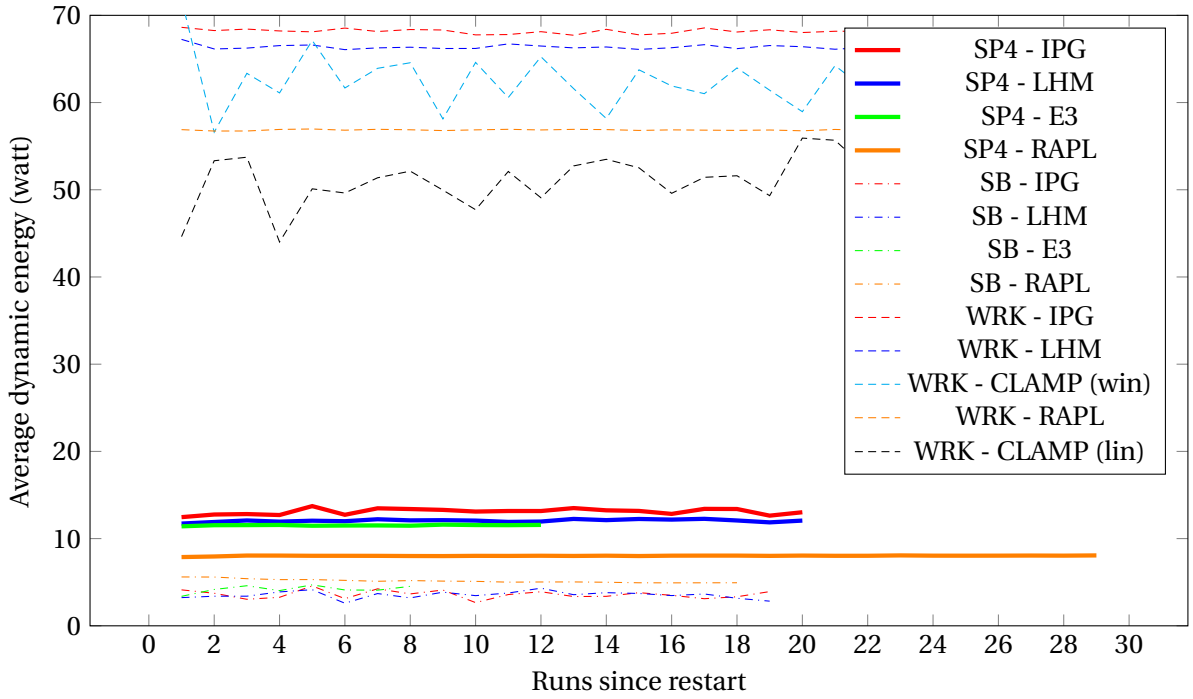


Figure 5.6: A graph illustrating the energy consumption of Cores for test case BinaryTrees with regards to how long ago the DUT was restarted (without outliers)

RQ3: The third research question considers the measurements between the different OSs. When comparing Linux and Windows, the overall pattern is higher measurements from Windows, with a few outliers. One outlier is RAPL for BinaryTrees, for the Surface Book in fig. 5.6. This observation is based both on software-based measuring instruments, but also the clamp, where the clamp represents the ground truth. Because of this, the results show that Windows has a higher energy consumption. When looking at the workstation, the clamp measurements for Linux are lower than RAPL, and for Windows, they are lower than IPG and LHM.

RQ4: The last research question aims to compare the DUTs. When comparing the DUTs, one difference is regarding energy consumption. Here the workstation will in all cases have the highest energy consumption, which can be observed in figs. 5.5 and 5.6. Of the two laptops, the Surface Pro 4 has a higher energy consumption than the Surface Book. A key difference between the laptops is the MAXIM chip on the Surface Book which, according to Microsoft meant E3 would have an accuracy of 98%[91], as was covered in subsection 3.1.2. Based on the results of this experiment, it is difficult to find any pattern suggesting the measurements has improved.

5.3 Comparison

In this section, a comparison of the measurements performed on the different DUTs will be made. This will be done in a manner where all test case measurements will be summed to a boxplot. When analyzing the results, this will be done by looking at each of **RQ2-4** one at a time. But first, some expectations.

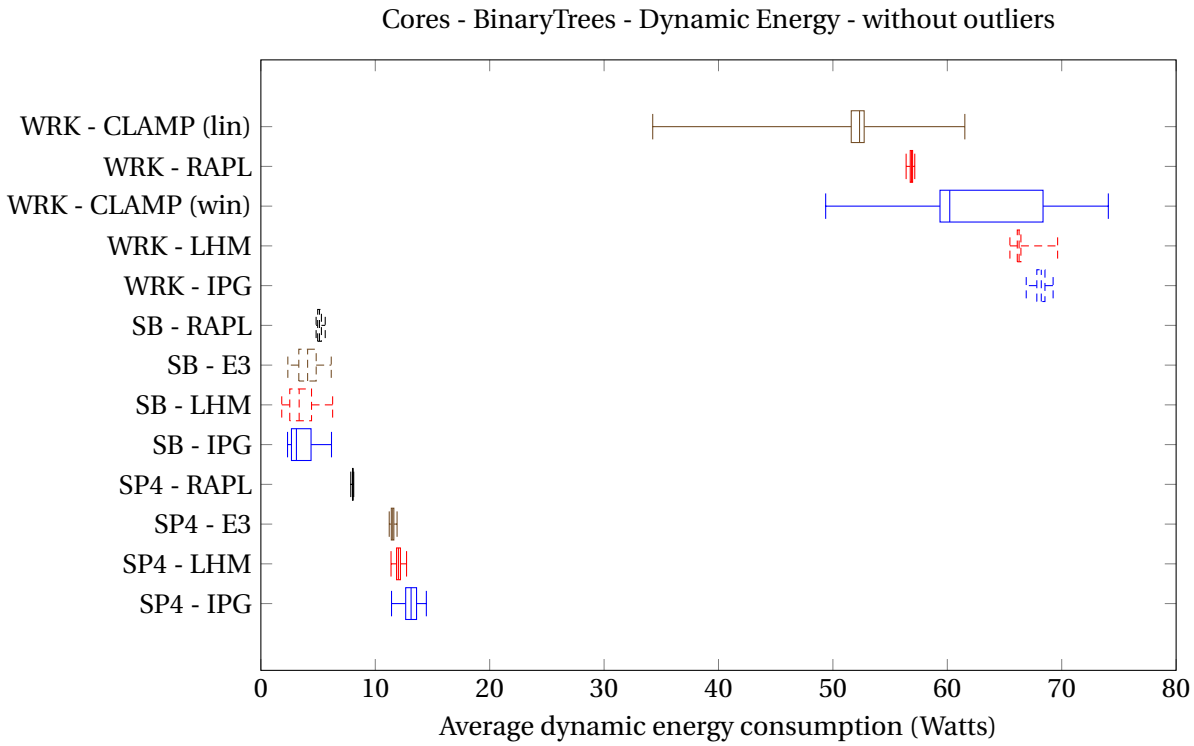


Figure 5.7: A comparison of Cores dynamic energy consumption for test case BinaryTrees for all DUTs and OSs (without outliers)

5.3.1 Expectations:

Based on what was seen in section 5.2, similar observations are expected. This will include a clamp with a high standard deviation compared to the different software-based measuring instruments, and cases where IPG and LHM measurements will appear similar. For RAPL, a low standard deviation is expected, in addition to lower measured energy consumption in most cases compared to the other measuring instruments. E3 measurements are expected to be somewhere between RAPL and the measurements by IPG and LHM.

5.3.2 Results

When presenting the results, this will be done based on BinaryTrees and Nbody, as these test cases illustrate the different tendencies in the test cases well. The results from the additional test cases can be found in appendices A.9 to A.12. Before going into the different research questions, some overall observations can be made. This is first of all regarding the standard deviation of the different measuring instruments, as seen for BinaryTrees and Nbody in figs. 5.7 and 5.8 respectively. The standard deviation for the clamp can be observed to be very high, compared to the different software-based measuring instruments.

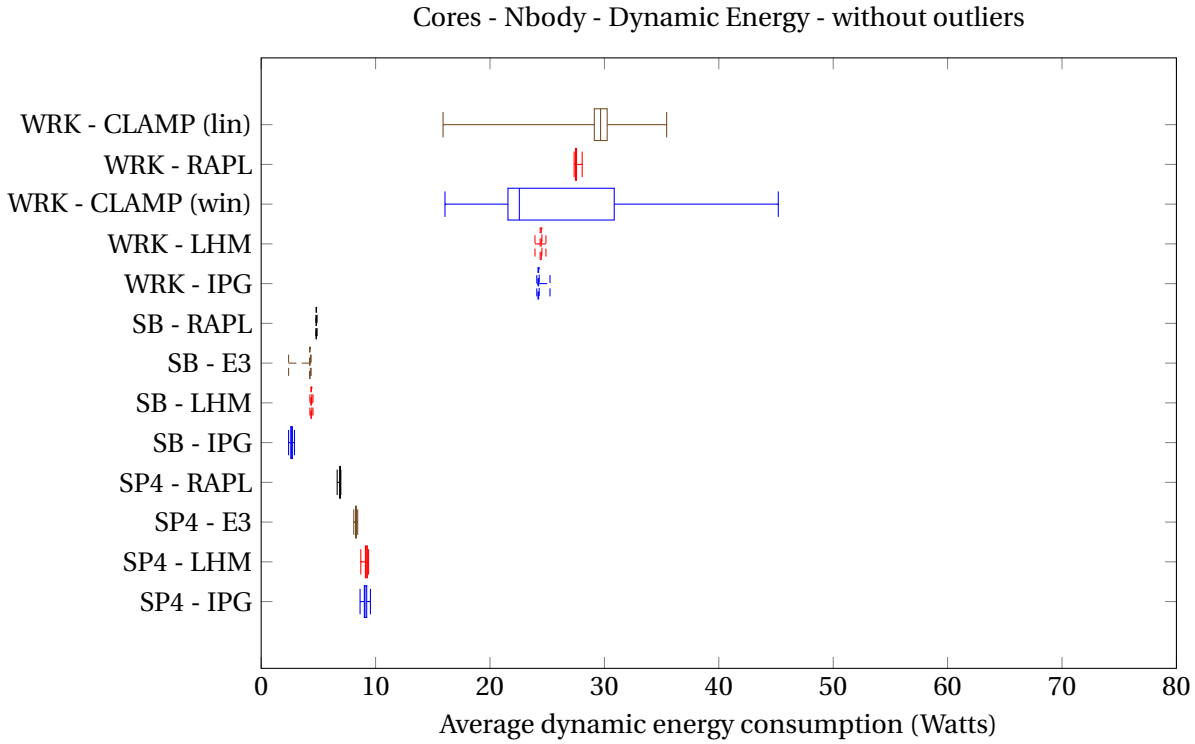


Figure 5.8: A comparison of Cores dynamic energy consumption for test case Nbody for all DUTs and OSs (without outliers)

RQ2: in **RQ2**, the different measuring instruments are compared across the different test cases. Here, BinaryTrees and Nbody are illustrated in figs. 5.7 and 5.8 respectively. When comparing the different measuring instruments, many observations are similar to what was found in section 5.2. Here IPG and LHM measurements can be observed to be very similar for all test cases. When comparing against RAPL, RAPL will in most cases measure lower energy consumption and will in all cases have a lower standard deviation. When comparing the clamp on Windows and Linux, it can be observed that the energy consumption on Linux is lower and deviates less compared to Windows, which can be observed across all test cases. When considering E3 measurements, they are for both the Surface Book and the Surface Pro 4 somewhere between the RAPL measurements and IPG and LHM measurements, but with a lower standard deviation than IPG and LHM in most cases.

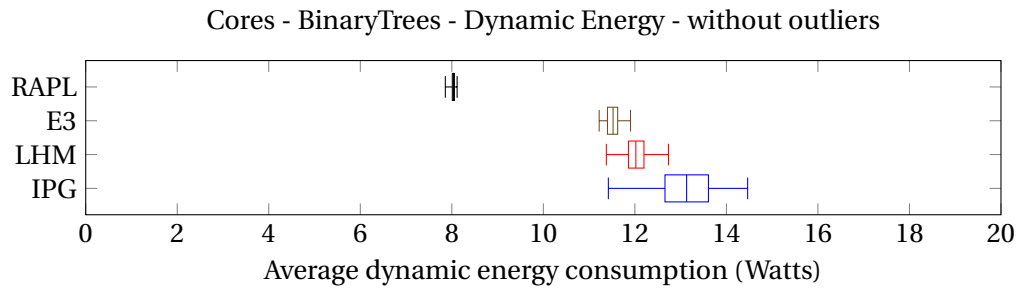


Figure 5.9: A comparison of Cores dynamic energy consumption for test case BinaryTrees for the Surface4Pro, (without outliers)

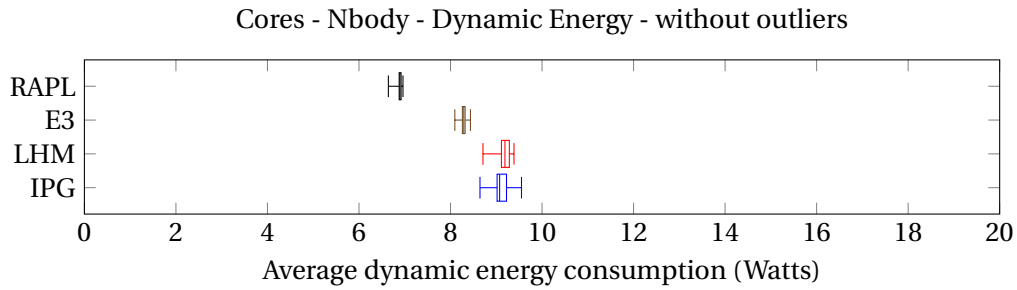


Figure 5.10: A comparison of Cores dynamic energy consumption for test case Nbody for the Surface4Pro, (without outliers)

RQ3: This research question considers the different OSs. When comparing the OSs, figs. 5.7 and 5.8 are used again. Two differences can be found, standard deviation and highest energy consumption. When considering the standard deviation, there is more uncertainty for Windows, as can be seen for BinaryTrees in fig. 5.7. In other cases, however, they are very similar, as can be observed for Nbody in fig. 5.8. When comparing the measuring instruments for Windows, E3 will in most cases have a lower standard deviation compared to IPG and LHM. When considering the energy consumption between the OSs, Windows has a higher energy consumption for most test cases and measuring instruments.

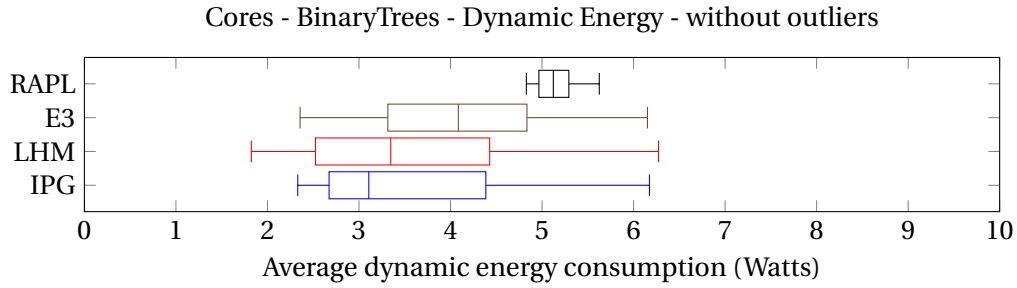


Figure 5.11: A comparison of Cores dynamic energy consumption for test case BinaryTrees for the SurfaceBook, (without outliers)

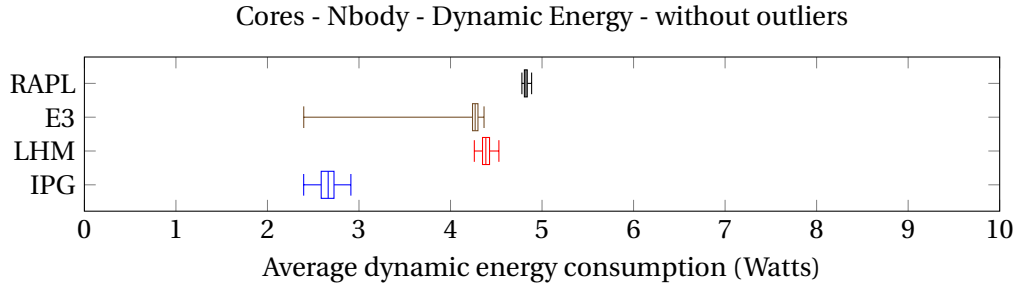


Figure 5.12: A comparison of Cores dynamic energy consumption for test case Nbody for the SurfaceBook, (without outliers)

RQ4: Lastly, **RQ4** considers the energy consumption between the different DUTs. For this research question, additional tables for the different DUTs will be introduced to get a more in-depth look at the measurements. The first DUT is the workstation, where the measurements for BinaryTrees and Nbody can be found in figs. 5.13 and 5.14. For the workstation, RAPL will in all cases measure a lower energy consumption except for Nbody. Nbody is also the only test case where the clamp for windows has a lower energy consumption than the clamp on Linux and Nbody is the test case with the lowest energy consumption for all measuring instruments, where BinaryTrees had the highest. The next DUT is the

Surface Pro 4, where test case BinaryTrees and Nbody is found in figs. 5.9 and 5.10 respectively. When considering the Surface Pro 4, RAPL will in all cases except Fasta report the lowest energy consumption. Given the x-axis in figs. 5.9 and 5.10 is $0-20$ rather than $0-80$ seen so far, the difference between IPG and LHM can be seen more clearly. Here IPG will in two cases have higher measurements, and more uncertainty compared to LHM. When considering measurements made by E3, they will in some cases be more similar to RAPL, like in Fasta, but in other cases be more similar to IPG and LHM like for BinaryTrees. The last DUT is the Surface Book, where the measurements for BinaryTrees and Nbody can be seen in figs. 5.11 and 5.12. One interesting observation for this DUT is the high uncertainty for all measuring instruments when measuring BinaryTrees compared to all other test cases. The Surface Book is also the only DUT where RAPL reports a higher energy consumption compared to the Windows measuring instruments, for both Nbody, BinaryTrees, and Fasta.

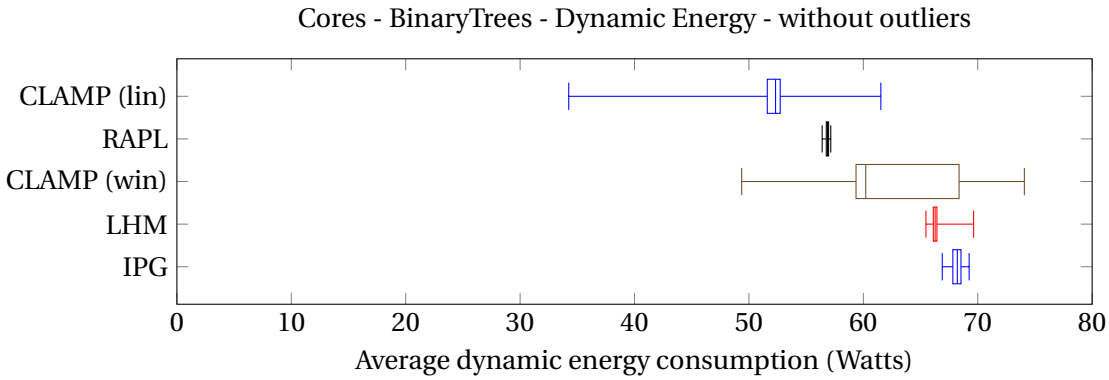


Figure 5.13: A comparison of Cores dynamic energy consumption for test case BinaryTrees for the workstation, (without outliers)

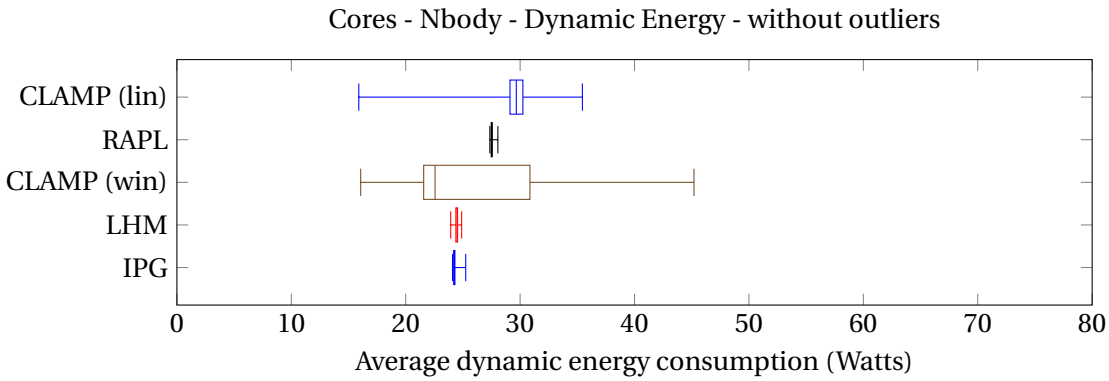


Figure 5.14: A comparison of Cores dynamic energy consumption for test case Nbody for the workstation, (without outliers)

5.4 Time Series

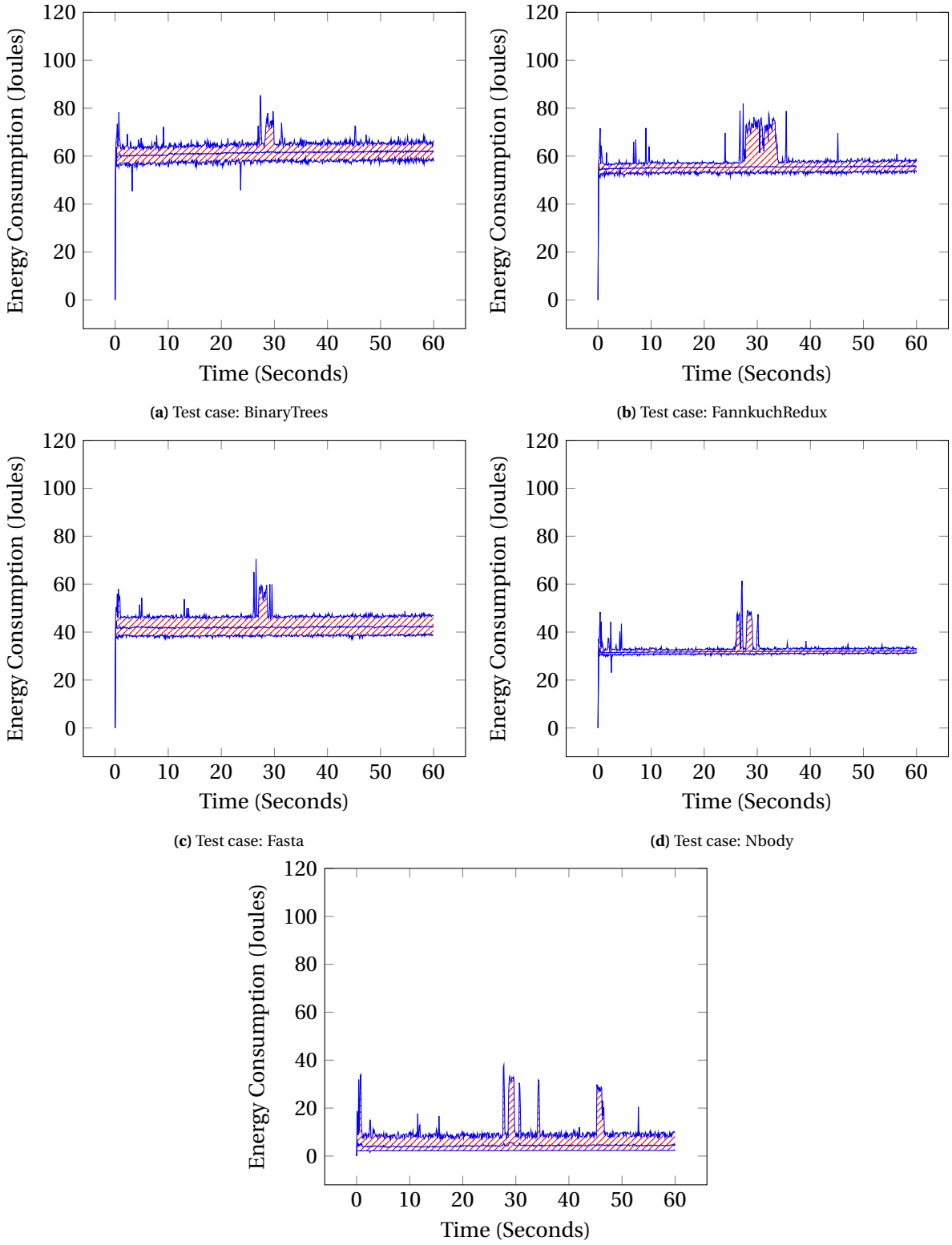
In this section, we present some of the time series graphs containing the average energy consumption in joules at a specific time stamp over 120 measurements. These graphs are not showing dynamic energy consumption, but rather the raw data. Not all of the graphs will be shown, because of the vast amount of graphs that would be, however, some additional graphs are shown in appendix A. This section is split up into different parts that focus on either the cases, the DUT, or the measuring instrument. They will each include our expectations and then the actual results will be presented.

5.4.1 Comparing the Test Cases

In this subsection, light will be shed on the different test cases.

Expectations: When looking at the different test cases with their real measured energy consumption and not dynamic energy consumption, we expect that the energy consumption of TestCaseIdle is a lot less than the other test cases. Furthermore, the energy consumption should be consistent as there should not be many background processes that can interfere with the energy consumption. For BinaryTrees and Fasta there could potentially be some visible spikes due to garbage collection since those test cases are memory-heavy.

Results: In fig. 5.15 some time series graphs are shown where the DUT is the workstation and the measuring instrument is RAPL. Each graph is an average of all the measurements of a given configuration. In fig. 5.15e TestCaseIdle is shown. What can be seen, is a lower average than the other test cases as we expected. However, there are some spikes, which were not expected. We can also observe that BinaryTrees consumes the most energy, followed by FannkuckRedux, Fasta, Nbody, and finally TestCaseIdle. Nbody has a lower variance than the other test cases, but like the others, it does have some spikes. Since these graphs show an average over the measurements, the spikes are consistent over many measurements. Notably, all of the test cases in fig. 5.15 have a spike at around ≈ 28 seconds into a test case run. This is an interesting observation, in an attempt to figure out why this occurs a look at a second configuration is done. With the same DUT, but using IPG as shown in fig. A.1 the spike at around ≈ 28 does not appear. However, it could have something to do with the measuring instrument and not the DUT so a third configuration is looked at, where the Surface Pro 4 and RAPL are used, as shown in fig. A.2. Here the spikes also do not appear. The cause of the spike is as such currently unknown, but we speculated that there is some process that would start at the same interval in each run. To further investigate this is future work. Another observation which appears in many configurations is that the measurement will start with a measurement close to ≈ 0, this could be due to the measuring instrument being started by our framework one line of code before the test case. As such the measuring instrument could potentially conduct a measurement before the test case is executed. Therefore in theory it could be a measurement of the DUT being idle. However, looking at fig. 5.15e there is also an initial measurement that is lower than the remaining measurements. This trend is consistent with all the RAPL measurements on TestCaseIdle as can be seen in fig. A.3, however, it is not present on the other measuring instruments for TestCaseIdle as can be seen for IPG in fig. A.4.



(a) Test case: BinaryTrees

(b) Test case: FannkuchRedux

(c) Test case: Fasta

(d) Test case: Nbody

(e) Test case: TestCaseIdle

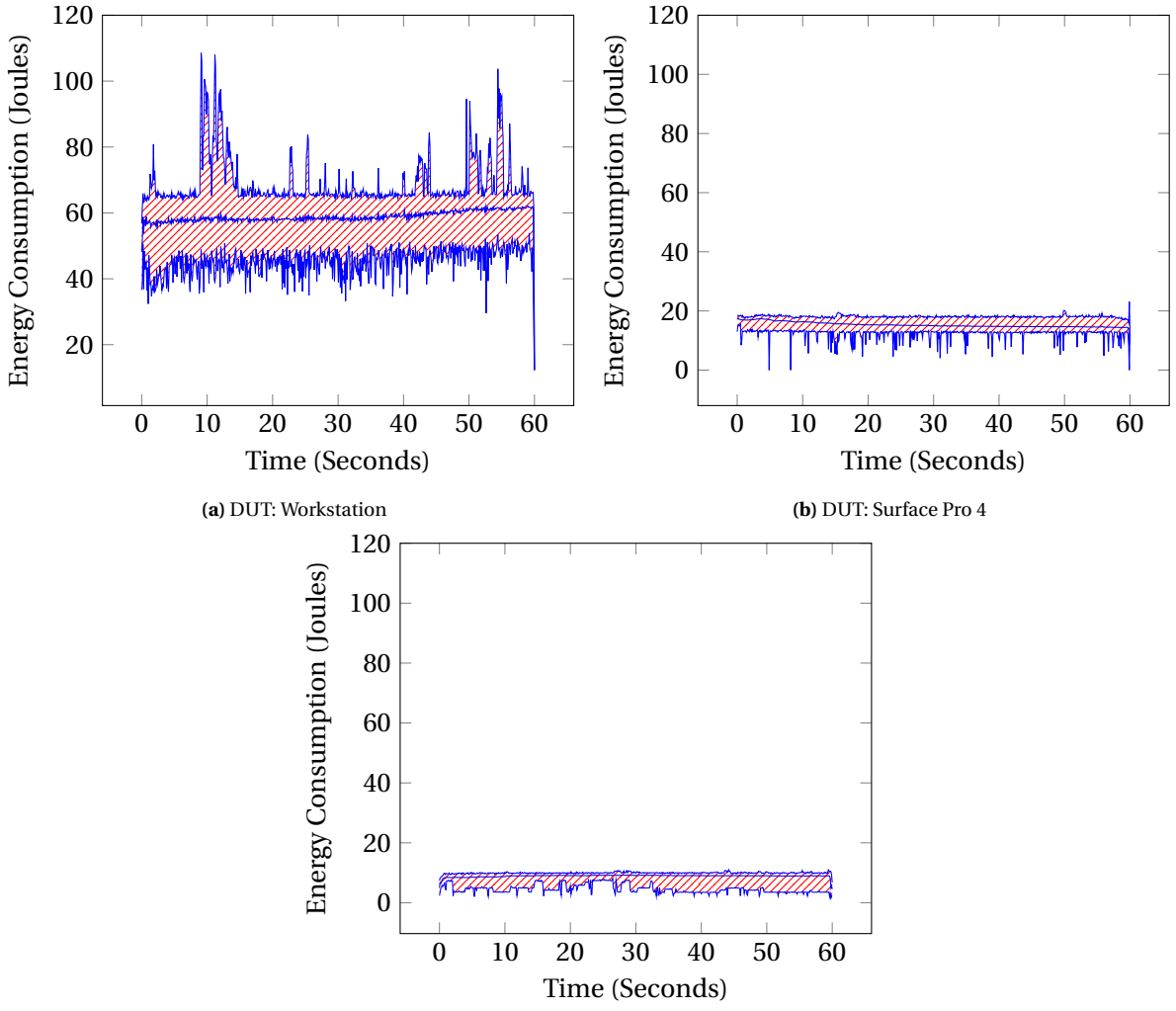
Figure 5.15: The energy consumption of all the test cases on the DUT: workstation with measuring instrument: RAPL. The lines represent the minimum, maximum and average energy consumption

5.4.2 Comparing the DUTs

After looking at the different test cases, this subsection will compare the difference between the DUTs. Therefore the test case and the measuring instrument will be kept consistent.

Expectation: We expect that the workstation has a higher energy consumption than the laptops since the CPU has a higher TDP. Regarding the laptops, it is expected they perform similarly since the hardware specifications are very similar.

Results: As can be seen in fig. 5.16 the Surface Book and Surface Pro 4 look similar as expected. However, the Surface Pro 4 consumes $\approx 5-10$ more joules than the Surface Book. Although, they are similar in regards to variance except that the Surface Pro 4 has lower valleys. When looking at fig. 5.16a which is the workstation it is clear that it consumes notably more energy than the other systems as expected. It also has more variance with more peaks and valleys. Interestingly the workstation shows a trend with a slight climb in the average energy consumption as the test case goes on.



(a) DUT: Workstation

(b) DUT: Surface Pro 4

(c) DUT: Surface Book

Figure 5.16: FannkuchRedux, measured by IntelPowerGadget, with the lines representing the minimum, maximum and average energy consumption

Regarding energy consumption, we can see that the Surface Book uses the least, followed by the

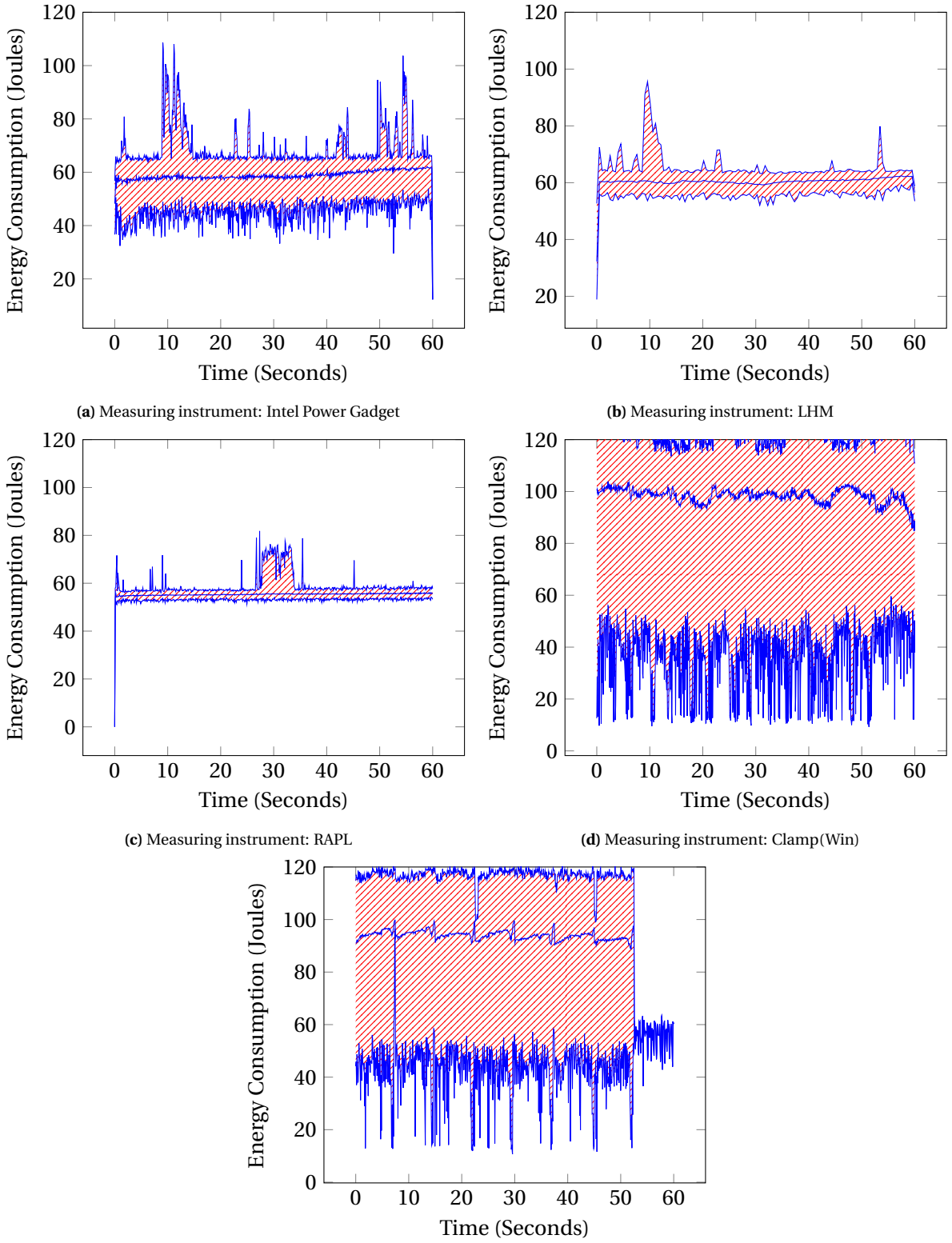
Surface Pro 4 and then the Workstation. If we look at the specifications for each DUT shown in tables 3.5 and 3.6. Here the two laptops' CPUs have the same TDP ($15W$), however, following the trend seen in the graphs, the Surface Book's CPU's configurable TDP-down ($7.5W$) is lower than the Surface Pro 4's($9.5W$). However, the workstation's CPU's TDP is much higher at $65W$, which also shows in the energy consumption.

5.4.3 Comparing the Measuring Instruments

Finally, after comparing the different DUTs, this section will proceed with a comparison of the different measuring instruments.

Expectation: We expect the clamp measurements to have a higher energy consumption than the software-based measuring instruments, due to the clamp measurements measure the whole system. The software-based measuring approaches should be very similar. However, we expect that the Linux measurements are slightly lower than Windows due to Linux being a more lightweight OS.

Results: On fig. 5.17 the graphs for test case FannkuchRedux on the workstations can be observed. The workstation was chosen since this DUT is the only one with clamp measurements as well as software-based measuring instruments. When considering figs. 5.17a and 5.17b, which are the two software-based measuring instruments on Windows, there is a consistent average, although with a slight upwards trend throughout the runtime. The average is very similar as expected, although with IPG there is more variance and the peaks and valleys are further from the average. Notable both measuring instruments have a peak at $\approx$ 10 seconds. Looking at fig. 5.17c, which is RAPL, the energy consumption is slightly lower, and the slight upward trend is not present. When looking at the clamp measurements shown in figs. 5.17d and 5.17e there is a higher energy consumption as expected since the clamp measurements measure the system as a whole and not just the CPU. Furthermore, the average is not as straight as a line as the software-based measurements. For the clamp on Windows as shown in fig. 5.17d the average becomes more varied in the final 20 seconds of the measurement. In fig. 5.17e there is a pattern where there is a small drop followed by a sharp peak followed by a sharp drop, where it then slowly rises until the pattern starts again. The pattern observed in subsection 5.4.1 regarding the initial measurement being ≈ 0 is also noticeable here, however only on LHM and RAPL. Furthermore, there is missing data, which occured as the test cases were not executed for exactly one minute, but at least one minute.



(a) Measuring instrument: Intel Power Gadget

(b) Measuring instrument: LHM

(c) Measuring instrument: RAPL

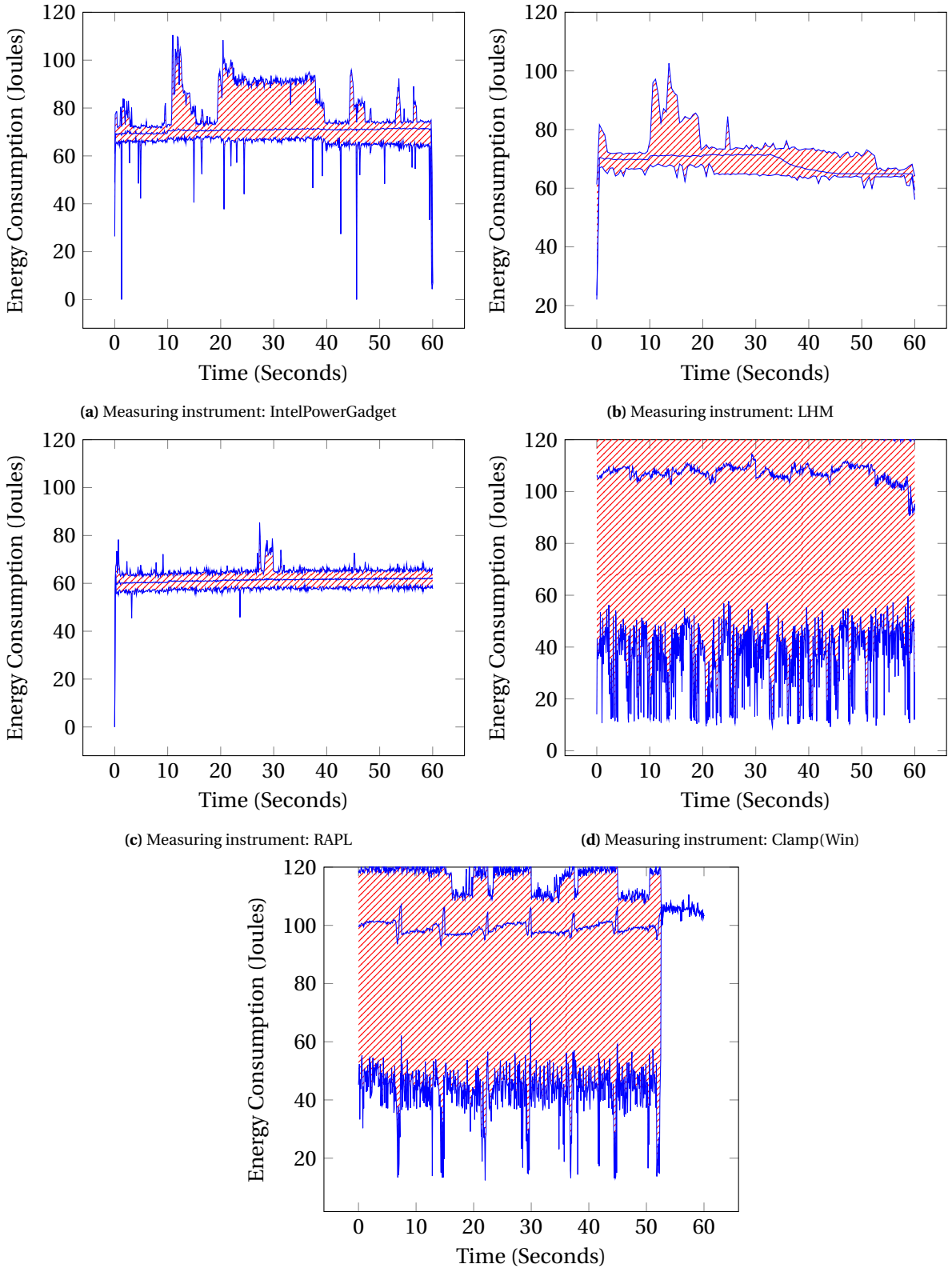
(d) Measuring instrument: Clamp(Win)

(e) Measuring instrument: Clamp (Lin)

Figure 5.17: FannkuchRedux, on the workstation measured by the different measuring instrumentst, with the lines representing the minimum, maximum and average energy consumption

To gain more insight into the differences between the different measuring instruments a look at a different configuration is useful. Therefore in fig. A.5. This figure shows the test case BinaryTrees on

the workstation. Previously we saw that the measuring instruments IPG and LHM showed a consistent average, but with a slight upwards trend in energy consumption. This is not as obvious in fig. A.5a and in fig. A.5b there is a trend downwards. They both also have a peak at around the same time step as in fig. 5.17 However, another thing that can be observed is that IPG is noisier in this configuration as well. Regarding the clamp measurements in fig. A.5d there is also a drop off in energy consumption towards the end of the measurement. When looking at fig. A.5e a very similar pattern occurs although with a bit more variance.



(a) Measuring instrument: IntelPowerGadget

(b) Measuring instrument: LHM

(c) Measuring instrument: RAPL

(d) Measuring instrument: Clamp(Win)

(e) Measuring instrument: Clamp(Lin)

Figure 5.18: BinaryTress on the workstation measured by the different measuring instrumentst, with the lines representing the minimum, maximum and average energy consumption

	TestCaseIdle	BinaryTrees	FannkuchRedux	Nbody	Fasta
IntelPowerGadget	0.0	**0.9103**	**0.1293**	0.0002	**0.8291**
HardwareMonitor	0.0213	**0.1345**	0.0492	**0.3209**	0.0
Clamp Win	0.0034	0.0023	0.012	**0.8143**	**0.5335**
RAPL	**0.1899**	**0.5744**	0.0015	**0.9437**	**0.0518**
Clamp Lin	**0.4601**	0.0004	0.0	**0.1006**	0.0002

Table 5.2: P values for the normal distribution for the Workstation in Ex2

5.5 Statistical Analysis of Experiment #2

In this section, the results from experiment #2 will be analyzed. The expectation from this analysis is to see similar observations to what was found in section 5.1, possibly with some deviations as a result of the disabled C-State.

5.5.1 Normal Distribution

The first part of the analysis will be analyzing the distribution for this experiment, using the Shapiro-Wilk test[72].

Expectations: The expectation is to find a similar conclusion as was found when analyzing the first experiment in section 5.1. In section 5.1 the data was found to be normally distributed in some cases, but not all.

Results: The results from the Shapiro Wilk test can be seen in table 5.2, it shows that a lot of our data is not normally distributed, where the data from this experiment is generally further away from being normally distributed than in section 5.1. One reason why this experiment is further away from being normally distributed could be a result of fewer measurements compared to the first experiment.

5.5.2 Independence Test

It was found in section 5.5.1, that the data for this experiment was not normally distributed. Because of this, the Mann-Whitney U Test[53] will be used to test independence equal to the method used in section 5.1.2.

Expectations: For the Mann-Whitney U test we would expect very similar results to the results from experiment #1, where the null hypotheses can be rejected in most of the cases. This is because the changes between experiment #1 and experiment #2 would not change the independence of the data, so should not change the result remarkably.

Results: The results from the Mann-Whitney U test on the test case FannkuckRedux can be seen here in fig. 5.19, and the results from the rest of the test cases can be found in appendix A.16. In fig. 5.19 a similar tendency to the first experiment can be observed. This tendency is that we can reject H_0 in most cases. When comparing this analysis to the one conducted in the first experiment, we find more cases where we cannot reject H_0 in this experiment. This could be a result of fewer measurements compared to the first experiment, meaning outliers has a larger effect.

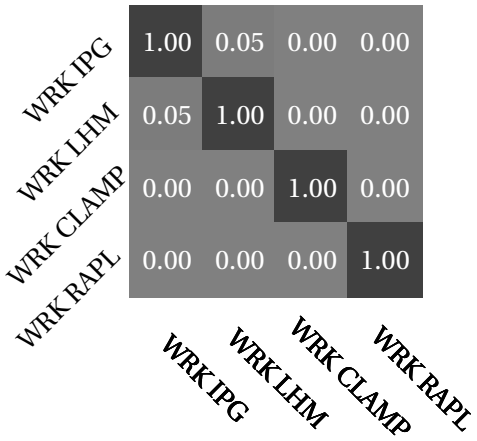


Figure 5.19: Here the results for the FannkuchRedux on the Mann Whitney U Test can be seen. The Range in $0-1$

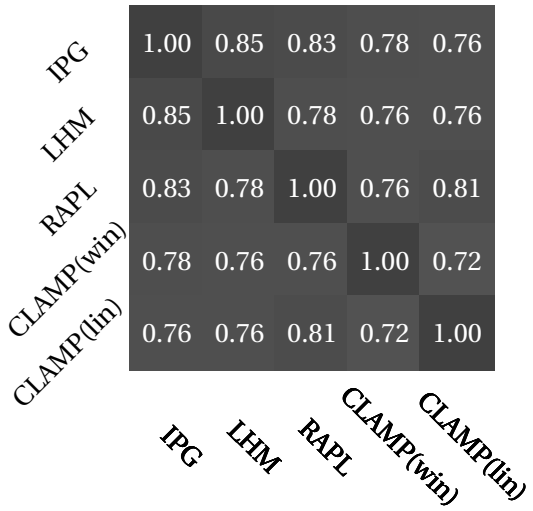


Figure 5.20: This heat map represents Correlation coefficients between the different measuring instruments -1 to 1 on the Workstation

5.5.3 Correlation

In this section, the correlation between the different measuring instruments will be conducted for the second experiment. Kendall's Tau coefficient[45] will again be used as it was in section 5.1.3

Expectations: The expectations for the correlations when the C-states are disabled will either be a similar or higher correlation. If the similarity increases, it would be expected to be because the uncertainty of the C-states has been removed.

Results: The calculated Correlations for this experiment can be seen in fig. 5.20. These results look very similar to the similarities obtained from the first experiment in section 5.1.3. The average correlation from experiments on the workstation is as follows:

$$\text{AvgCoefEx1} = (0.81 + 0.80 + 0.77 + 0.71 + 0.80 + 0.78 + 0.71 + 0.77 + 0.72 + 0.70)/10 = 0.757$$

$$\text{AvgCoefEx2} = (0.85 + 0.83 + 0.78 + 0.76 + 0.78 + 0.76 + 0.76 + 0.76 + 0.81 + 0.72)/10 = 0.781$$

These coefficients show higher correlations between the different DUTs and their measuring instruments. Now, the coefficients will be analyzed and utilized to answer our research questions.

RQ2: When comparing the measuring instruments to each other the average correlation for each instrument can be calculated. These will then be compared with the results from section 5.1, to see what effect disabling the C-States have had on the measurements.

- **Experiment #1**
  - $IPG = 0.772$
  - $LHM = 0.775$
  - $RAPL = 0.725$
  - $CLAMP(win) = 0.755$
  - $CLAMP(lin) = 0.710$
- **Experiment #2**
  - $IPG = 0.805$
  - $LHM = 0.787$
  - $RAPL = 0.795$
  - $CLAMP(win) = 0.755$
  - $CLAMP(lin) = 762$

When looking at the average correlation for the measuring instruments across the two experiments, the second experiment generally had larger coefficients than the first experiment, showing a higher correlation for the second experiment. Looking at these numbers using the Guildford scale, it can be seen how the correlation does not increase enough to change evaluation.

RQ3: When comparing the results across OSs, some differences can be observed when compared to the first experiment. The OS benefitting the most in terms of correlation coefficients when disabling the C-Stats is Linux, as the correlation between RAPL and the other measuring instruments increased. This is mostly dues to an increased correlation between RAPL and the clamp on Linux, which increased from 0.72 to 0.81. Overall it seems that RAPL is the measuring instrument most closely correlated with the hardware measurements while IPG is the most correlated measuring instrument for Windows.

5.6 Experiment #2: Disabled C-States

Next, up, experiment #2 will be covered, where the DUT was in performance mode. This was done to attempt to get a more realistic idle case, resulting in a more realistic dynamic energy consumption of the other test cases.

5.6.1 Expectations

The expectation for this experiment is a higher energy consumption on all test cases, especially the idle test case. This increased energy consumption for the test cases will most likely result in a decreased dynamic energy consumption, as we expect the energy consumption for the idle test case to increase more than the other test cases. In addition to this, the expectation will also be a lower standard deviation, especially for the clamp.

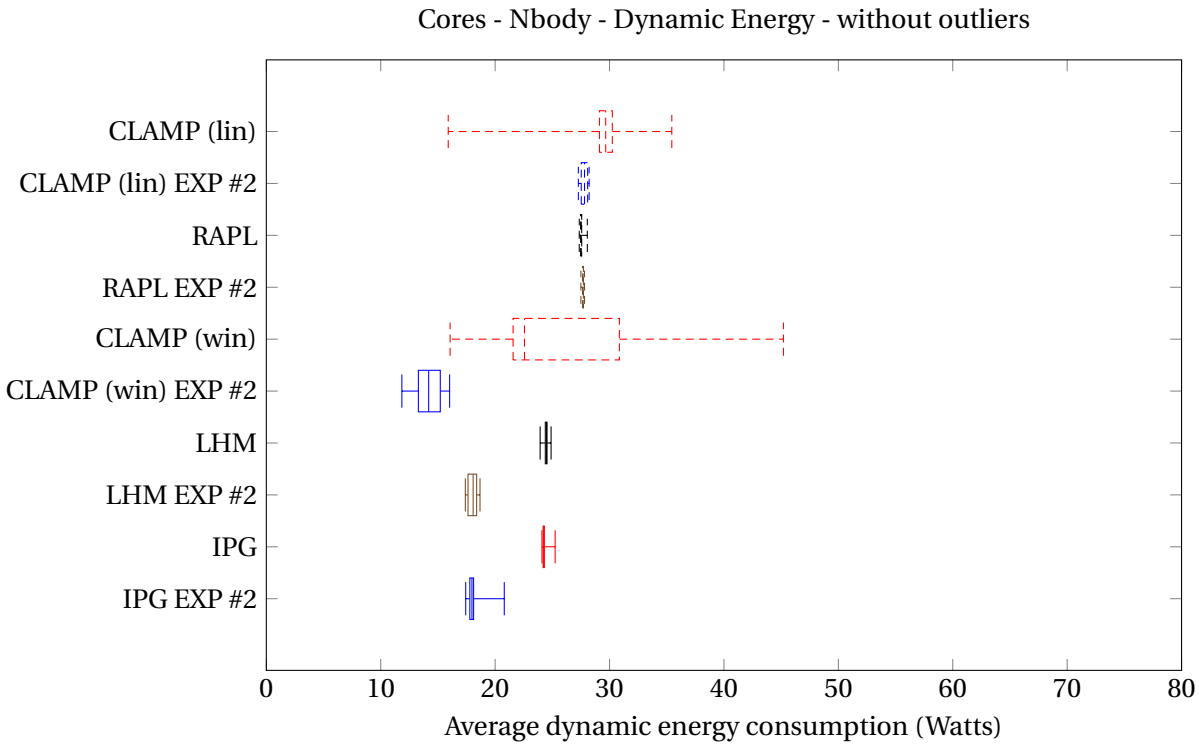


Figure 5.21: A comparison of Cores dynamic energy consumption for test case Nbody for the workstation, experiment #2 (without outliers)

5.6.2 Result

When considering the result, the test case Nbody can be seen in fig. 5.21, and for the other test cases in appendix A.14. In this graph, the measurements from the first and second experiments can be observed and compared. The results will be discussed through **RQ2** and **3**.

RQ2: When comparing based on the different measuring instruments, the patterns and orders of the profilers have not changed, but the dynamic energy consumption has. This is especially for the clamp on Windows, where the variance has decreased in all cases except for BinaryTrees, where it increased. IPG and LHM are still similar and still measure higher values compared to the clamp on Windows. When comparing IPG and LHM to RAPL, RAPL now reports a higher energy consumption in all cases, as a very limited effect on the reported measurements when disabling the C-states is observed for RAPL.

RQ3: When comparing Windows and Linux, the difference between an enabled and disabled C-state can be observed for Nbody in fig. 5.21 and all other test cases in appendix A.14. The effect for Linux in general is limited, especially on RAPL. When considering the clamp on Linux it has improved in some cases. For Nbody, the min/max values are now very close to the 25th and 75th percentile, but the dynamic energy consumption is still close to the same value, for Fasta, a very limited effect is observed. On Windows, it is different, where the dynamic energy for all measuring instruments has decreased. This is especially clear for the clamp on Windows, where before the variance was the highest of all measuring instruments.

5.7 Summary

Based on the comparison between the measuring instruments of this first experiment, it is difficult to find a clear conclusion. This is because patterns across the different DUTs are difficult to find, and the lack of consistency across the DUTs. Some things are however clear:

- IPG and LHM are in most cases similar. This could be due to both measuring instruments measuring similarly. An observation was made on how LHM made measurements, where it made system calls to a process called RAPL. It could be interesting to see if a similar call is made by IPG, but this is a subject for future work.
- The clamp measurements deviate the most, especially on windows. This could be due to the OS having more background processes, or us not disabling enough. Linux still has outliers, but most measurements are within the 25th and 75th percentile.
- RAPL measurements deviate less compared to the other measuring instruments.

When considering the different measuring instruments, no consistent pattern could be found when considering which measuring instrument would measure the highest, second highest, and lowest measurements. This could be due to the different targets the test case focuses on, as was presented in table 3.9, where some of the measuring instruments might measure certain operations or certain loads better. We were however unable to find such patterns.

Chapter 6

Discussion

In the following a discussion of the measuring instruments and the results presented in chapter 5 will be illuminated.

6.1 Measuring instruments

The first section of the discussion will cover the different measuring instruments used. This will be done by first discussing them independently, in terms of ease of use, etc., before all measuring instruments will be compared.

Open Hardware Monitor / Libre Hardware Monitor: One issue with LHM is the sampling rate. It was found in section 2.4 that using the same sampling rate for the measuring instruments is beneficial when attempting to compare them. However, during some initial testing with LHM, it was found that using a sampling rate of more than 2 Hz would frequently produce errors. Therefore a frequency of 2 Hz is used during the experiments with LHM, whereas the other measuring instruments use a frequency of 10 Hz. We decided to not limit the other measuring instruments to the maximum frequency of LHM and instead accept that it could be a disadvantage with LHM. Besides this and a lack of documentation, no other issues came up when importing and using the library.

Intel Power Gadget: In terms of using measuring instruments, Intel Power Gadget is the most promising. This tool had official documentation and an API making it very user-friendly. The issues came when executing on the Surface Book, where it crashed for unknown reasons. The exact extent of why these crashes occur and what devices it occurs on is unknown but is something to explore in the future.

E3: This measuring instrument was based on the numbers provided by Microsoft, presented in subsection 3.1.2, very promising. However, when using the tool, some issues arose. This was first of all regarding the use case. For the other measuring instruments, they could either be started and stopped on demand, or the energy consumption could be read in real-time. This was different for E3, where you could start it, but since it only performed a measurement once a minute, it was vital to know when E3 did this. The sampling was important to know about, to be sure the energy consumption read was for the whole duration of one minute. Since no official documentation was found this was a difficult task and meant we had to rely on blog posts but, the material about how E3 worked was still limited. The claims made by Microsoft regarding the MAXIM chip which was the reason why the Surface Book was used, are also 7 years old by the time of writing this and the chip does not seem to exist in any Surface devices since 2016. We speculated that they either do not work as great as Microsoft claimed or

because they were too expensive to implement. For E3, the measurements were in most cases close to those made by IPG and LHM, which indicated that it performed similarly compared to the other measuring instruments. Given that E3 could isolate the energy consumption of each process running on the DUT, it could have been interesting to see how well this isolation was. An experiment could have been conducted where E3 measurements on an idle DUT were compared to E3 measurements when the DUT did some background work. However, this was not prioritized and is therefore a subject for future work. For E3, it is noted in subsection 3.1.2 how E3 claims to work best when it is not connected to a charger. It would be interesting to try and compare E3 measurements with and without a charger connected, to see how this would affect the results. This is however also a subject for future work.

Clamp: When considering the results from the first experiment presented in chapter 5, the clamp was observed to have a high deviation compared to the other measuring instruments. The issue with the clamp compared to the other measuring instruments is how it measures the energy consumption of the entire system compared and not just the CPU. Because of this, many factors can impact the measurements. To minimize the variability the fan speed was set to maximum and the external GPU was removed. Furthermore, dynamic energy consumption was implemented to isolate the energy consumption from specifically running a test case by removing the energy consumption of the fans, etc. As mentioned in subsection 3.2.3, the MN60 is an analog current sensor, which is different from digital sensors, and this difference could account for the deviations seen in the results. Analog sensors output a continuous measurement, while digital sensors have a discrete output and the sensor performs processing of the data.[62] Another thing that could help explain the deviations is the power supply of the workstation, there are at least two reasons why this could be the case. In the power supply, there are several capacitors that change how the computer uses energy as they could help flatten out any spikes in energy consumption, but this would mostly be a problem with smaller measurements and not the minute long ones we conduct[32]. The second and more likely reason, in our opinion, is the unknown efficiencies of the power supply. In subsection 4.1.3 we covered power supply efficiency, and that the only tested efficiency is 20%,50% and 100% load, and that these efficiencies did not have a linear relationship. This means that at other loads the efficiency could fluctuate, and affect the measurements.

RAPL: This measuring instrument was similar to LHM and IPG in terms of usability. It also had some documentation, which was useful. RAPL has the advantage of not being required to install anything, it just works when installing Linux. In other works[46], one thing noted when using RAPL, is how the energy consumption register will overflow approximately every 52 minutes, depending on the hardware. These overflows, resulting in a reset of the register were however not observed in our data, which could be because the DUT was restarted more frequently than an overflow. This was however not explored and is a problem for future work.

Summary: In our work, five different measuring instruments have been used and if the use case is to measure the energy consumption of software, most of them worked as expected. The only exception to this was E3, as it required the test case to run for one minute, but it was on the other hand the only measuring instrument able to separate the different processes running, rather than only presenting the energy consumption of the entire CPU.

6.2 Findings

This part of the discussion is about the different results covered in chapter 5.

Energy consumption of Windows: When comparing measurements made on Windows and Linux on the clamp, different conclusions can be made in experiments #1 and #2. In #1, the energy consumption was in most cases higher on Windows, as can be seen in section 5.2, where the energy consumption over time was analyzed. This is the opposite of what was found by Najmuddin et al.[58]. Because of this, the second experiment was conducted, where the C-states were disabled, as was covered in section 5.6. In this experiment, the dynamic energy consumption of Windows decreased, while it remained the same for Linux, which shows two things. This first of all means we have the same conclusion as Najmuddin et al.[58], but it also shows how the conclusion made by Najmuddin et al.[58] could be correct. The conclusion stated that the energy consumption on Linux is higher compared to Windows because of a lack of well-written drivers for Linux and based on the results of our work, it seems like Linux is unable to utilize the different C-states within a CPU from the motherboard. For Linux, they appear to be set through the OS, as the measurements remained the same when they were disabled or enabled. Some improvements were however seen on Linux for the second experiment, as the clamp had a much lower standard deviation.

Experiment #1: In the first experiment, a few observations were made resulting in the second experiment. This was because of very low observed energy consumption for the idle test case and higher energy consumption on windows compared to Linux. The low energy consumption is different from the expected energy consumption presented in subsection 4.1.3, and the lower energy consumption on Linux did not match the findings by Najmuddin et al.[58] The question is therefore whether the results from the first experiments should be disregarded. In this case, it is argued that the measurements can still be used, but only for comparisons within the same OS. The measurements will still give an indication of which profiler provides the highest measurement and how close they are to a ground truth etc. The measurements can however not be used to compare measurements from different OSs, as Experiment #2 illustrated how disabling the C-states affected the Windows and Linux differently, as was covered in section 5.6. Experiment #1 can therefore be concluded to be useful for finding patterns, but it should be noted the measurements are too high based on Experiment #2.

Experiment #2: In the second experiment, the C-state was disabled to get a more realistic idle case. In this context, it can be discussed what can be, and cannot be considered realistic. For this, we consider what the idle case is used for. The idle case measurements were subtracted from the measurements for the other test cases to isolate the energy consumption for only the test case. Because of this, we needed all test cases to be in the same C-state as the idle test case, as other C-states would impact energy consumption. Through observations, energy measurements also changed from experiment #1 to experiment #2, indicating the idle test case was not the only test case impacted by the C-states. This is a result of the different test cases having different workloads on the CPU. It could be argued that a more realistic setting would be to show the actual energy consumption, and let the CPU enter the C-states it finds fitting for the workload. However, in our work, we valued the dynamic energy consumption higher, where an idle energy consumption is required, as all measuring instruments except E3 measure the energy consumption for either the entire CPU or DUT. And for the dynamic energy to be calculated, a static C-state is required.

6.3 Setup

In this section, different aspects of how the experimental setup was made will be discussed.

Test cases based on time or iterations: In our work, the test cases are executed based on time rather than runs. In the literature, it is common to use runs as seen in[65, 47, 22]. One work running test cases based on time is the work by Sestoft[78], where a duration of 0.25s is chosen to avoid problems with the virtual machine and the clock resolution. Another argument for using time made by Sestoft[78] is when considering test cases of different sizes, where 100 million runs might be too time-consuming for larger test cases, but not smaller test cases. In our work, we chose to use time because of E3, where the test cases have to run for one minute for E3 to detect the test case. A side effect of running based on time is that the different test cases will not run for an equal amount of iterations. This potential issue is addressed in our work by running Cochran's formula on all test cases, thereby ensuring all test cases have enough measurements.

Test cases: In our work, the test cases used were chosen based on what is generally used in research[47, 29]. The argument for using such test cases is to make our work more comparable to ensure everything is implemented correctly. Questions can however be raised concerning how well such test cases represent a real-life application. Because of this, it could have been interesting to test the different measuring instruments on larger applications, which could be a potential future work. In our work, all test cases are implemented in C#, as the focus is not to compare the energy consumption of any specific language, but rather to compare the measuring instruments. It could however have been interesting to test an even wider range of test cases, also including different languages. This could be interesting as when comparing the different measuring instruments in section 5.2 some overall patterns could be observed, but there were always exceptions, like how RAPL only in some cases reported a higher energy consumption compared to IPG and LHM. Additional test cases could help to find patterns if, for example, RAPL was better/worse when measuring certain kinds of operations. This is however a task for future work.

Background processes: In our work, different processes were disabled, in order to limit their effect on the measurements. These processes include the ones presented in table 3.10 and Windows update. One thing to note here is how no background processes are disabled for Linux, as we were unable to find any processes which made sense to disable. For Windows, eight background processes were located and stopped upon startup for the DUT, but for both OSs, it can be argued more time could have been spent trying to limit the activity of background processes. This was based on the expectation that a fresh install of both OSs would mean they would have a limited amount of background processes, which is a subject for future work.

Temperature: For the temperature, there was an expectation that when the CPU reached some temperature, an effect would be observable in the results. This did however not happen, as was discussed in subsection 4.1.5. The reason for this, as was also mentioned in subsection 4.1.5, was most likely due to the test cases never stressing the CPU enough to reach temperatures beyond 65 degrees celsius. As a result of this, no temperature limits were set when running the experiments. It could however have been interesting to see what impact temperatures above 65 degrees celsius had on the results. Given the temperatures never increased above 65 for the test cases used, a stress test could have been used for the experiment instead. The stress test would be expected to bring the CPU to higher temperatures, as it will bring the load of the CPU to 100%, and since the effect of an overheated CPU would be expected to be similar for all test cases, it would be fine that the stress test is used instead of the test cases. This is however a subject for future work.

Battery: Given some issues on the surface devices as discussed in subsection 4.1.5, the battery limit were set between $40-80\%$. Within these limits, no visible effect of the battery limit could be found. The expectation for this experiment covered in subsection 4.1.5 was to find some lower limit, where the energy consumption would change. This would occur as the DUT would enter a power saving mode when some battery percentage is reached. This power saving mode would most likely cause the OS to stop some background processes and underclock the CPU. When a CPU is underclocked, the test case would run fewer samples during a measurement. The energy consumption of one sample when the DUT is in power saving mode is expected to be different compared to when the DUT is not in power saving mode, but this is a subject for future work. For both surface devices, the power mode is set to maximum, where it could have been interesting to see how the results would change if the power saving mode would be different, this is however a subject for future works.

Chapter 7

Conclusion

In this work we have tested measuring instruments across both OSs and DUTs and our contribution is a comparison of these measuring instruments, as the literature has primarily focused on RAPL. In the following, we will summarize the answers to the research question presented in chapter 1. These research questions were made based on observations of how most existing works use the Linux tool RAPL, where our work's main goal was to compare other existing software-based measuring instruments for Windows against RAPL. All measurements were also compared to a hardware measurement tool, serving as a ground truth. To conduct the experiments we created a framework in C# which could automatically run the measuring instruments and test cases on both Windows and Linux. The framework would run on the DUT's startup, where background processes would be stopped and WiFi disabled to limit interference in the measurements. When measuring the energy consumption, test cases would execute for one minute, while the framework would start and stop the different measuring instruments in a rotated round-robin fashion using R3-validation, where between each test case the WiFi would be enabled so the measurements could be uploaded to an SQL database. The framework also had a set number of runs it would execute before restarting the DUT and then continuing to iterate over the four different software-based measuring instruments and the hardware-based measuring instrument as well as the five different test cases. Cochran's formula was then used to calculate how many measurements we needed to ensure our desired margin of error of 3% and confidence level of 95% on our results. This framework ran on three different DUTs, where one of them was a workstation without a battery and the remaining two were laptops. One of the laptops had a MAXIM chip which was claimed to improve the measurement capabilities of E3. This claim was however not reflected in the results. To allow for comparison between DUTs we used dynamic energy consumption, which allowed us to isolate the energy consumption for the test case only, rather than from the entire CPU. Then analyzing our results the correlation of the measurements is calculated to discover if they follow the same trends when measuring energy consumption.

In experiment #1 we observed a higher or similar standard deviation for the measurement on Windows compared to Linux. We also saw that Windows in the majority of cases had a higher actual and dynamic energy consumption compared to Linux, which did not align with our expectations. It was also observed that the energy consumption of the idle test case was lower than we expected. Because of this, a second experiment was conducted, where the C-states were disabled on the CPU of the workstation. In this experiment, the energy consumption in the idle test case increased, and the dynamic energy consumption of Windows decreased, reflecting our expectations from experiment #1.

Between the different DUTs, all measuring instruments reported a higher energy consumption on the workstation. For the laptops, the Surface Pro 4 had a higher energy consumption compared to

the Surface Book, reported by all measuring instruments, only with a few exceptions. One difference between the laptops was the MAXIM chip on the Surface Book, which supposedly should make E3 more accurate. The E3 measurement for the Surface Book does have a .02 and .01 higher correlation with IPG and LHM respectively, which could mean that E3 is more accurate on the Surface Book if IPG and LHM are accurate. The Surface Book had a lower correlation between the RAPL measurements and the other software-based measuring instruments than the Surface Pro 4, it is however unclear why.

For the measuring instruments, we found that LHM and IPG are very similar in terms of measurements as well as being highly correlated with each other. RAPL in the majority of cases measured a lower energy consumption than LHM and IPG and it also had a lower standard deviation, with a few exceptions. E3 in most cases reports an energy consumption between RAPL and LHM/IPG. The primary issue with E3 includes requirements of how the DUT should have a battery and the inability to start and stop the measuring instrument on demand, making the E3 more tedious to utilize for our purpose. In experiment #1, the highest correlation with the clamp on the same OS as the software-based measuring instrument was LHM and IPG with a .01 difference. Compared to this, RAPL had a slightly lower correlation, which suggests that LHM and IPG are more accurate than RAPL. In experiment #2 RAPL had the highest correlation with the clamp, indicating it is more accurate for our test cases. It should however be noted that both IPG and LHM are also highly correlated, while E3 has a low to moderate correlation. The higher correlation between LHM and IPG suggests similarities in terms of how they access energy consumption from the CPU. When comparing IPG and LHM, both had issues, where IPG crashed on some DUTs and LHM had a lower sampling rate. Opposed to this there is RAPL, which overall seems like a more integrated measuring instrument.

Chapter 8

Future Work

In future work, different aspects covered in both chapter 5 and chapter 6 could be addressed by some additional experiments. These aspects will be split into three categories, including hardware, measuring instruments, and test cases, and addressed one at a time.

8.1 Hardware

The hardware chosen in our work included three different DUTs with two Microsoft Surface devices and one workstation. Both Surface devices are from 2015, whereas the CPU from the workstation was released in 2017. To get a better understanding of how the hardware affects the results it could be interesting to include some additional hardware. This could include both newer and older hardware, to see if some measuring instruments work better or worse on different versions of hardware. It could also be very interesting to get hardware measurements on a DUT with a battery, to compare E3 against. In the current state of our work, it is difficult to say anything about how accurate E3 is compared to IPG, RAPL, and LHM. The hardware measurements could be obtained either by measuring the DC energy consumption directly on the CPU, or by removing the battery, and thus using the clamp on the charger.

The idea of including DC measuring in the experiments could also be interesting on the workstation, as this would mean measurements would only include the energy consumption from the CPU and not the other components of the DUT.

8.2 Measuring Instruments

When considering the measuring instruments, two things can be considered. This is first of all regarding adding new measuring instruments, or more in-depth analysis of the ones included in our work.

When considering adding new measuring instruments, one promising one was brought to our attention by our supervisors. This measuring instrument is a Windows implementation of RAPL[1], which to our knowledge is yet to be the subject of an analysis of the accuracy of its measurements.

[1] https://github.com/hubblo-org/windows-rapl-driver

When considering additional experiments on the measuring instruments included in our work, a few were mentioned in chapter 6. This could first of all be concerning E3's ability to measure and isolate the energy consumption of a test case when some workload is running in the background. An experiment could also be to look at the effect of the charger, by measuring the energy consumption of test cases, when the DUT is charging compared to when it is not. In the work by Khan et al.[46], an analysis of the overhead of RAPL was conducted, by pinning the test case and framework to the same core of the CPU. Here the overhead of RAPL is found to be less than 2%, where a similar analysis of the other measuring instruments could be interesting.

8.3 Test Cases

In our work, four test cases are included from the Computer Language Benchmark Game. Based on the results, it was difficult to find a pattern with regards to which measuring instruments always measured highest, seconds highest, etc. This could mean the different measuring instruments perform differently on different workloads or operations. Because of this, it could be interesting to include more test cases. This could be a combination of more test cases from the Computer Language Benchmark Game, larger applications representing a real-life application, or test cases only performing a single operation.

Our work only considers C#, where the test cases chosen for future work, could include some additional languages. This could be used to extend the analysis of the different measuring instruments.

8.4 Use Windows Measuring Instruments

The idea of this project was to create a source about the accuracy of measuring instruments on Windows. In this work, RAPL is concluded to be the best measuring instrument with a correlation coefficient between RAPL and the clamp of 0.81, where IPG and LHM were 0.78 and 0.76 in fig. 5.20 respectively. Given the small difference between the measuring instruments on Windows and Linux, it could be interesting to reproduce some tools made for Linux, on Windows.

One such example could be the Continuous Integration Pipeline for the MSTest made by Joergensen et al.[43]. The idea of this pipeline was to run methods similarly to running a unit test, but instead of either passing or failing, the methods are run multiple times, where a distribution of the energy consumption is found. The methods are run multiple times, as a single run is found to be insufficient given the impact of the JIT compiler and background processes.

Another work we could reproduce in Windows is the work by Nielsen et al.[59]. In this work, an extension for VS code is made to help developers optimize code with respect to energy consumption. The energy consumption is estimated using a static analysis using either machine learning, energy models, or by measuring the energy consumption of the CPU using RAPL and then isolating the energy consumption of the code. The static analysis was in this work performed using an interpreter on the Common Intermediate Language (CIL), where the different instructions are analyzed. The instructions can be analyzed by a machine learning model, or by an energy model where each instruction is then mapped to an energy consumption. Given that the energy consumption is shown to be different between Linux and Windows in this work, both the machine learning model and the energy model should be run on energy measurements made on Windows, by either LHM or IPG. For the best performing models in the work by Nielsen et al.[59], which were nonlinear models like random forest, the

performance was between −7.5% and 9.2% from the ground truth, where this would be the score to beat on Windows.

Bibliography

[1] Anant Agarwal and Jeffrey Lang. *Foundations of analog and digital electronic circuits*. Elsevier, 2005.

[2] Thomas Alsop. *Share of households with a computer at home in developing countries from 2005 to 2019*. https://www.statista.com/statistics/748564/developing-countries-households-with-computer/. 2021.

[3] Richard A Armstrong. "Should Pearson's correlation coefficient be avoided?" In: *Ophthalmic and Physiological Optics* 39.5 (2019), pp. 316–327.

[4] Chauvin Arnoux. *AC current clamps user's manual*. 2013.

[5] Mete Balc. *A Minimum Complete Tutorial of CPU Power Management, C-states and P-states*. https://metebalci.com/blog/a-minimum-complete-tutorial-of-cpu-power-management-c-states-and-p-states/. 2018.

[6] Mahmoud A. Bokhari, Brad Alexander, and Markus Wagner. "Towards Rigorous Validation of Energy Optimisation Experiments". In: *CoRR* abs/2004.04500 (2020). arXiv: 2004.04500. URL: https://arxiv.org/abs/2004.04500.

[7] Bobby R. Bruce, Justyna Petke, and Mark Harman. "Reducing Energy Consumption Using Genetic Improvement". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015).

[8] buildecomputers. *Power Consumption of PC Components in Watts*. https://www.buildcomputers.net/power-consumption-of-pc-components.html.

[9] ChauvinArnoux. *MN60*. https://catalog.chauvin-arnoux.com/fr_en/mn60.html. Accessed: 28/11/2022.

[10] CloudFree. *CloudFree EU Smart Plug*. https://cloudfree.shop/product/cloudfree-eu-smart-plug/. Accessed: 28/11/2022.

[11] *cnet*. https://download.cnet.com/Microsoft-Joulemeter/3000-2086_4-75578519.html, Accessed: 24/11/2022.

[12] William G. Cochran. *Sampling Techniques, 3rd edition*. John Wiley & sons, 1977. ISBN: 0-471-16240-X.

[13] The kernel development community. *Power Capping Framework - The Linux Kernel documentation*. URL: https://www.kernel.org/doc/html/latest/power/powercap/powercap.html. 28/09/2022.

[14] *Defrag Tools 157 - Energy Estimation Engine E3*. https://learn.microsoft.com/en-us/shows/defrag-tools/157-energy-estimation-engine-e3. Accessed: 23/09/2022.

[15] Digilent. *Analog Discovery 2: 100MS/s USB Oscilloscope, Logic Analyzer and Variable Power Supply*. https://digilent.com/shop/analog-discovery-2-100ms-s-usb-oscilloscope-logic-analyzer-and-variable-power-supply/. Accessed: 28/11/2022.

[16] Digilent. *Analog Discovery 2 Reference Manual*. 2015.

[17] *Documentation*. https://openhardwaremonitor.org/documentation/. Accessed: 23/09/2022.

[18] Jack Dongarra et al. "Energy Footprint of Advanced Dense Numerical Linear Algebra Using Tile Algorithms on Multicore Architectures". In: *2012 Second International Conference on Cloud and Green Computing*. 2012, pp. 274–281. DOI: 10.1109/CGC.2012.113.

[19] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[20] Muhammad Fahad et al. "A comparative study of methods for measurement of energy of computing". In: *Energies* 12.11 (2019), p. 2204.

[21] Félix García et al. "Towards a consistent terminology for software measurement". In: *Information and Software Technology* 48.8 (2006), pp. 631–644. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2005.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0950584905001011.

[22] Stefanos Georgiou and Diomidis Spinellis. "Energy-Delay investigation of Remote Inter-Process communication technologies". In: *Journal of Systems and Software* 162 (2020), p. 110506. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.110506. URL: https://www.sciencedirect.com/science/article/pii/S0164121219302808.

[23] Andrew R Gilpin. "Table for conversion of Kendall's Tau to Spearman's Rho within the context of measures of magnitude of effect for meta-analysis". In: *Educational and psychological measurement* 53.1 (1993), pp. 87–92.

[24] *GitHub*. https://github.com/LibreHardwareMonitor/LibreHardwareMonitor. Accessed: 21/11/2022.

[25] *GitHub*. https://github.com/arendst/Tasmota, Accessed: 21/11/2022.

[26] *GitHub*. https://github.com/cupertino/ectools, Accessed: 24/11/2022.

[27] Keith R Godfrey. "Correlation methods". In: *Automatica* 16.5 (1980), pp. 527–534.

[28] golinuxhub. *What are the CPU c-states? How to check and monitor the CPU c-state usage in Linux per CPU and core?* https://www.golinuxhub.com/2018/06/what-cpu-c-states-check-cpu-core-linux/. 2018.

[29] Sander Greenland et al. "Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations". In: *European journal of epidemiology* 31.4 (2016), pp. 337–350.

[30] Kristian Gregersen and Daniél Nielsen. "Energy Consumption of Software Architectures - A comparison between microservice- and monolithic architectures in C sharp and Java". In: Aalborg, Jylland, Denmark, 2022. URL: https://projekter.aau.dk/projekter/da/studentthesis/energy-consumption-of-software-architectures--a-comparison-between-microservice-and-monolithic-architectures-in-c-and-java(7f1db56d-ebab-4a6b-832a-249e03b02669).html.

[31] Joy Paul Guilford. "Fundamental statistics in psychology and education". In: (1950).

[32] Daniel Hackenberg et al. "Power measurement techniques on standard compute nodes: A quantitative comparison". In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2013, pp. 194–204. DOI: 10.1109/ISPASS.2013.6557170.

[33] Marcus Hähnel et al. "Measuring Energy Consumption for Short Code Paths Using RAPL". In: *SIGMETRICS Perform. Eval. Rev.* 40.3 (2012), 13–17. ISSN: 0163-5999. DOI: 10.1145/2425248.2425252. URL: https://doi.org/10.1145/2425248.2425252.

[34] Aaron K Han. “Non-parametric analysis of a generalized regression model: the maximum rank correlation estimator”. In: *Journal of Econometrics* 35.2-3 (1987), pp. 303–316.

[35] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[36] *How to catch a non-CLS exception*. https://learn.microsoft.com/en-us/dotnet/csharp/how-to/how-to-catch-a-non-cls-exception. Accessed: 02/12/2022.

[37] Intel. *https://www.intel.com/content/dam/develop/external/us/en/documents/green-hill-sw-20-185393.pdf*. https://www.intel.com/content/dam/develop/external/us/en/documents/green-hill-sw-20-185393.pdf. 2011.

[38] Intel. *Intel Core i7-6650U Processor*. https://www.intel.com/content/www/us/en/products/sku/91497/intel-core-i76650u-processor-4m-cache-up-to-3-40-ghz/specifications.html.

[39] Intel. *Intel Core i7-8700 Processor*. https://ark.intel.com/content/www/us/en/ark/products/126686/intel-core-i78700-processor-12m-cache-up-to-4-60-ghz.html.

[40] *Intel Power Gadget Official documentation*. https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html. Accessed: 22/09/2022.

[41] Glenn D Israel. “Determining sample size”. In: (1992).

[42] Erik Jagroep et al. “Profiling Energy Profilers”. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC ’15. Association for Computing Machinery, 2015, 2198–2203. ISBN: 9781450331968. DOI: 10.1145/2695664.2695825. URL: https://doi.org/10.1145/2695664.2695825.

[43] Philip Bengston Joergensen and Jonas Bylling Andersen. “Energy measurements of tests: Exploring the energy consumption of tests in C# using Intel RAPL”. In: (2022). Accessed: 06/01/2023.

[44] Amandeep Kaur and Robin Kumar. “Comparative analysis of parametric and non-parametric tests”. In: *Journal of computer and mathematical sciences* 6.6 (2015), pp. 336–342.

[45] Maurice G Kendall. “A new measure of rank correlation”. In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[46] Kashif Nizam Khan et al. “RAPL in Action: Experiences in Using RAPL for Power Measurements”. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3.2 (2018). ISSN: 2376-3639. DOI: 10.1145/3177754. URL: https://doi.org/10.1145/3177754.

[47] Lukas Koedijk and Ana Oprescu. “Finding Significant Differences in the Energy Consumption when Comparing Programming Languages and Programs”. In: *2022 International Conference on ICT for Sustainability (ICT4S)*. 2022, pp. 1–12. DOI: 10.1109/ICT4S55073.2022.00012.

[48] JWKJW Kotrlik and CCHCC Higgins. “Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research”. In: *Information technology, learning, and performance journal* 19.1 (2001), p. 43.

[49] Mohit Kumar, Youhuizi Li, and Weisong Shi. “Energy consumption in Java: An early experience”. In: *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. 2017, pp. 1–8. DOI: 10.1109/IGCC.2017.8323579.

[50] Rasmus Lindholt, Aleksander Nielsen, and Kasper Jepsen. “Analyzing C sharp Energy Efficiency of Concurrency and Language Construct Combinations”. In: Aalborg, Jylland, Denmark, 2022. URL: https://projekter.aau.dk/projekter/da/studentthesis/analyzing-c-energy-efficiency-of-concurrency-and-language-construct-combinations(e984ad65-cc08-4394-8d0f-8bd3cbfe1173).html.

[51] Javier Mancebo et al. “EET: A Device to Support the Measurement of Software Consumption”. In: *2018 IEEE/ACM 6th International Workshop on Green And Sustainable Software (GREENS)*. 2018, pp. 16–22. URL: https://ieeexplore.ieee.org/document/8449823.

[52] Javier Mancebo et al. “FEETINGS: Framework for Energy Efficiency Testing to Improve Environmental Goal of the Software”. In: *Sustainable Computing: Informatics and Systems* 30 (2021), p. 100558. ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2021.100558. URL: https://www.sciencedirect.com/science/article/pii/S2210537921000494.

[53] Henry B Mann and Donald R Whitney. “On a test of whether one of two random variables is stochastically larger than the other”. In: *The annals of mathematical statistics* (1947), pp. 50–60.

[54] Heiko Mantel et al. “How Secure Is Green IT? The Case of Software-Based Energy Side Channels”. In: *Computer Security*. Ed. by Javier Lopez, Jianying Zhou, and Miguel Soriano. Cham: Springer International Publishing, 2018, pp. 218–239. ISBN: 978-3-319-99073-6.

[55] Jr Massey and Frank J. “The Kolmogorov-Smirnov test for goodness of fit”. In: *Journal of the American statistical Association* 46.253 (1951), pp. 68–78.

[56] Microsoft. *P-states and C-States*. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/xperf/p-states-and-c-states. 2018.

[57] Mozilla. *tools/power/rapl*. URL: https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html. 29/09/2022.

[58] Sayed Najmuddin, Zablihullah Atal, and Riaz Ahmad Ziar. “Comparative Analysis of Power Consumption of the Linux and its Distribution Operating Systems vs Windows and Mac Operating Systems.” In: *Kardan Journal of Engineering and Technology* 3.1 (Dec. 2021), pp. 96–109. DOI: 10.31841/KJET.2022.21. URL: https://doi.org/10.31841/KJET.2022.21.

[59] Casper Susgaard Nielsen, Jacog Noerhave, and Anne Ejsing. “IDE Extension for Reasoning About Energy Consumption”. In: (2021). Accessed: 06/01/2023.

[60] Daniél Nielsen et al. “Energy Benchmarking With Doom: Utilizing video game source ports for marcobenchmarking”. In: Aalborg, Jylland, Denmark, 2021. URL: https://projekter.aau.dk/projekter/da/studentthesis/energy-benchmarking-with-doom(54556b3b-5728-493c-8b11-6ac67a4da5fb).html.

[61] Notebookcheck. *Intel Core i5-6300U*. https://www.notebookcheck.net/Intel-Core-i5-6300U-Notebook-Processor.149433.0.html.

[62] Dominic O’Donnell. *Whats the Difference Between Analog and Digital Sensors?* = https://sensorex.com/2022 vs-digital-sensors/, 2022.

[63] *Operating System Market Share Worldwide*. https://gs.statcounter.com/os-market-share. 2022.

[64] Muhammed Ozturk. “Tuning Stacked Auto-encoders for Energy Consumption Prediction: A Case Study”. In: *International Journal of Information Technology and Computer Science* 2 (Feb. 2019), pp. 1–8. DOI: 10.5815/ijitcs.2019.02.01.

[65] Rui Pereira et al. “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*. SLE 2017. Association for Computing Machinery, 2017, 256–267. ISBN: 9781450355254. DOI: 10.1145/3136014.3136031. URL: https://doi.org/10.1145/3136014.3136031.

[66] Plugwise. *Plugwise*. https://www.plugwise.com/product/plug/?lang=en-+. Accessed: 02/12/2022.

[67] *Powercfg command-line options*. https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/powercfg-command-line-options. Accessed: 23/09/2022. 2021.

[68] Giuseppe Procaccianti et al. "Profiling power consumption on desktop computer systems". In: *International conference on information and communication on technology*. Springer. 2011, pp. 110–123.

[69] The Linux man-pages project. *msr(4)*. URL: https://man7.org/linux/man-pages/man4/msr.4.html. 23/09/2022.

[70] Lasse Rasmussen, Milton Lindof, and Soeren Christensen. "Benchmarking C sharp for Energy Consumption: Energy Implications of Different Language Constructs and Collections". In: Aalborg, Jylland, Denmark, 2022. URL: https://projekter.aau.dk/projekter/da/studentthesis/benchmarking-c-for-energy-consumption(5821c977-8df4-45de-8f11-113708747048).html.

[71] Raspberry Pi. *Raspberry Pi 4*. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed: 28/11/2022.

[72] Nornadiah Mohd Razali, Yap Bee Wah, et al. "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests". In: *Journal of statistical modeling and analytics* 2.1 (2011), pp. 21–33.

[73] *Rosetta Code*. URL: https://rosettacode.org/wiki/Rosetta_Code. 20/10/2022.

[74] Sheldon M. Ross. *INTRODUCTORY STATISTICS, 3rd edition*. Elsevier Inc, 2010. ISBN: 978-0-12-374388-6.

[75] Efraim Rotem et al. "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge". In: *IEEE Micro* 32.2 (2012), pp. 20–27. DOI: 10.1109/MM.2012.12.

[76] *Running Average Power Limit Energy Reporting*. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. Accessed: 10/03/2022.

[77] servethehome. *The truth about CPU power consumption*. https://forums.servethehome.com/index.php?threads/the-truth-about-cpu-power-consumption.16/. 2010.

[78] Peter Sestoft. "Microbenchmarks in Java and C#". In: *Lecture Notes, September* (2013).

[79] Sikkerhedsstyrelsen. *Sådan kender du forskel på forskellige stikkontakter*. https://www.sik.dk/privat/goer-det-sikkert/el/brug-elprodukter-sikkert/saadan-kender-du-forskel-paa-forskellige-stikkontakter,

[80] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company, 2020. ISBN: 9780078022159. URL: https://www.db-book.com/db7/index.html.

[81] Pavel Somavat, Vinod Namboodiri, et al. "Energy consumption of personal computing including portable communication devices". In: *Journal of Green Engineering* 1.4 (2011), pp. 447–475.

[82] Dan Terpstra et al. "Collecting Performance Data with PAPI-C". In: *Tools for High Performance Computing 2009*. Ed. by Matthias S. Müller et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173. ISBN: 978-3-642-11261-4.

[83] *The Computer Language 22.05 Benchmarks Game*. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html. 20/10/2022.

[84] tomshardware. *Samsung 970 EVO Plus 1 TB*. https://www.techpowerup.com/ssd-specs/samsung-970-evo-plus-1-tb.d61. 2019.

[85] *Using the Intel Power Gadget 3.0 API on Windows*. https://www.intel.com/content/www/us/en/developer/articles/training/using-the-intel-power-gadget-30-api-on-windows.html. Accessed: 22/09/2022.

[86] Ronald L Wasserstein, Allen L Schirm, and Nicole A Lazar. *Moving to a world beyond "p< 0.05"*. 2019.

[87] Vincent M Weaver et al. "Measuring energy and power with PAPI". In: *2012 41st international conference on parallel processing workshops*. IEEE. 2012, pp. 262–268.

[88] Perf wiki. *Main Page*. URL: https://perf.wiki.kernel.org/index.php/Main_Page. 28/09/2022.

[89] Wikipedia. *Kill A Watt*. https://en.wikipedia.org/wiki/Kill_A_Watt. Accessed: 02/12/2022.

[90] Wikipedia. *MoSCoW Method*. https://en.wikipedia.org/wiki/MoSCoW_method. Accessed: 02/12/2022.

[91] *WinHEC 2015 12/4/2017*. https://slideplayer.com/slide/12250080/. Accessed: 23/09/2022.

[92] Zhenkai Zhang et al. "Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels". In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. ASIA CCS '21. Virtual Event, Hong Kong: Association for Computing Machinery, 2021, 162–175. ISBN: 9781450382878. DOI: 10.1145/3433210.3437517. URL: https://doi.org/10.1145/3433210.3437517.

[93] Hüseyin Ünlü and Yeliz Yesilada. "Transcoding web pages via stylesheets and scripts for saving energy on the client". In: *Software: Practice and Experience* 52.4 (2022), pp. 984–1003. DOI: https://doi.org/10.1002/spe.3046. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3046. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3046.

Appendix A

Appendix

A.1 Terminology table

Term	Definition	Source
Software entity	Software that is to be characterized by measuring its attributes.	FEETINGS[52]
Software entity class	The collection of all the entities that satisfy the determined objective.	FEETINGS[52]
Test Case	A representation of functionality of the software entity to be measured.	FEETINGS[52]
Test Case Measurement	A set of energy consumption measurements of all the runs in a test case.	FEETINGS[52]
Measurement	A set of energy consumption samples from a single test case run.	FEETINGS[52]
Samples	Each energy consumption record taken by a measuring instrument.	FEETINGS[52]
Device Under Test (DUT)	A device where the software entity to be measured is run.	FEETINGS[52]
Measuring Instrument	A method used to make energy consumption measurements.	FEETINGS[52]
Setup	A defined step of procedures executes at DUT startup.	R3[6]
Batch	A set of test cases executed in sequence with a cool of periods.	NEW
Configuration	A combinations of a DUT, measuring instrument and test case, temperature and battery.	NEW

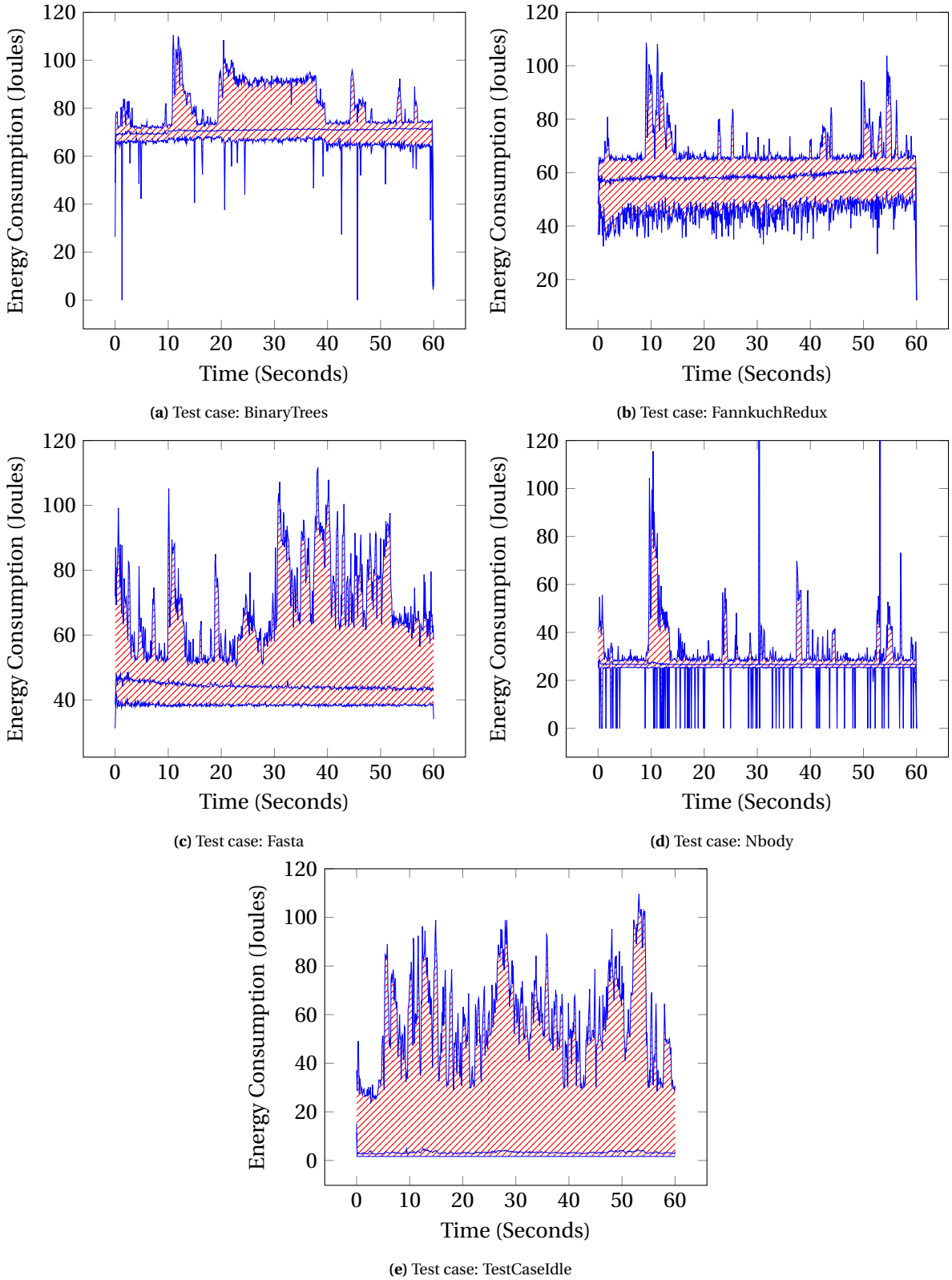
Table A.1: Terminology used throughout our work.

A.2 CPU-states

What is presented here is largely based on information from Intel[37] and some sources that convey the material in a more presentable manner [5, 28]. The C-states manage how the system consumes energy, C0 is the normal operation of a working computer under load. Each incremental C-State shuts more of the CPU down until at C10 it is nearly completely shut down. Different CPUs and motherboards could however support a different amount of c-states. The same idea applies to CC-States(Core C-states), PC-States(Package C-States) in addition to the Thread C-States and Hyper-Thread C-States, but the information on these is very sparse. Some CPUs also have enhanced C-States (C1E) able to shut more of the CPU down, but not enough to be the next C-State. The P-States are only used during C0, where they control the frequency of the CPU under load to better manage its energy usage. S-States (Sleep State) controls how the system is using energy, but on a larger scale as it controls if the system is sleeping or not. Every C-States occur within S0, while increments define deeper states of sleep such as Sleep and Hibernation. The G-States (Global-States) define the overall state of the system such as G0 being a working computer where S0, C-States and P-States can occur and G3 when the DUT is completely shut down. The expectation based on this is that since the problem only seemed to occur during the idle experiments we suspected it had something to do with the C-states.

A.3 Comparing Time Series for the Test Cases

The analysis of the timeseries of the different DUTs, test cases and measuring instruments, as covered in section 5.4.



(a) Test case: BinaryTrees

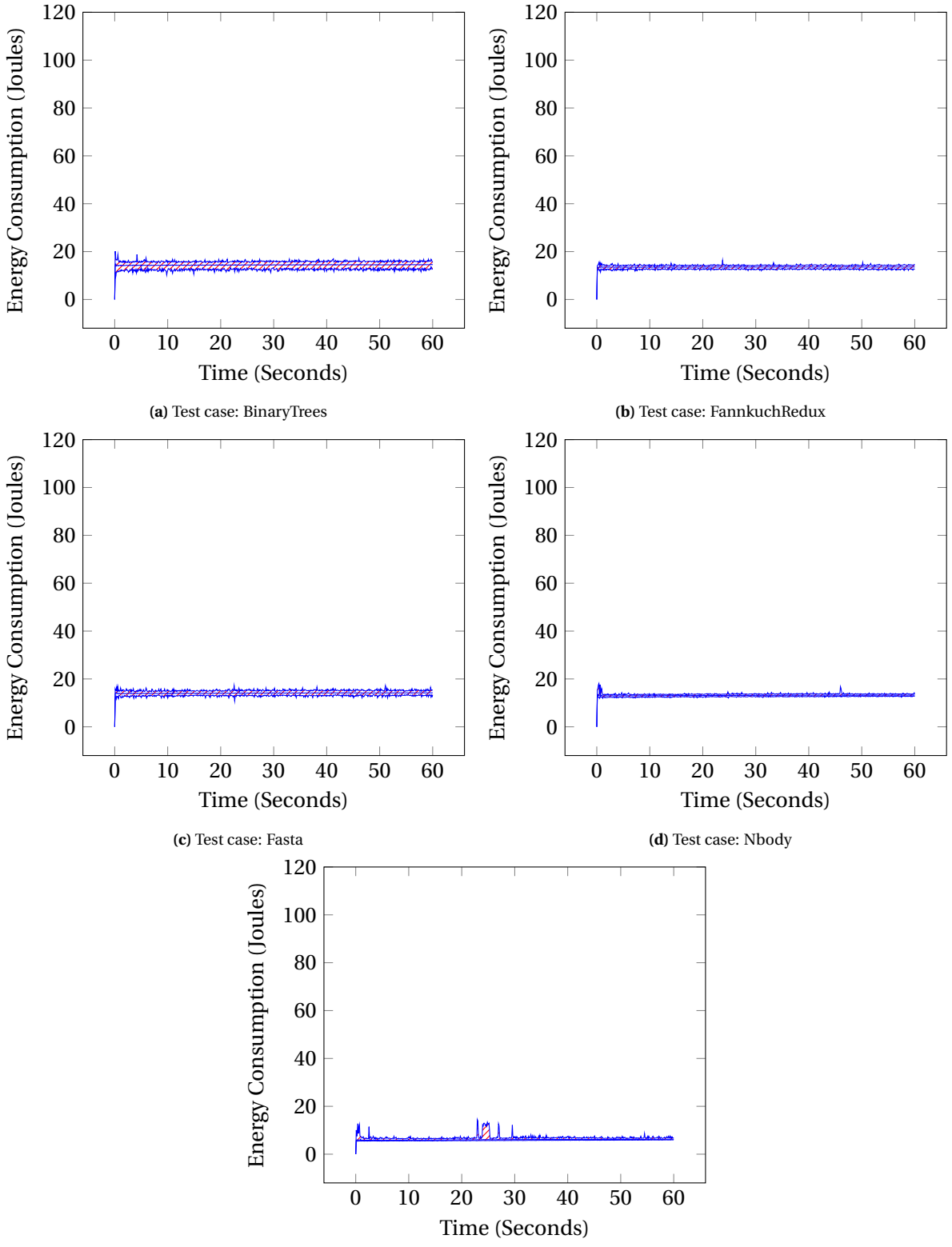
(b) Test case: FannkuchRedux

(c) Test case: Fasta

(d) Test case: Nbody

(e) Test case: TestCaseIdle

Figure A.1: The energy consumption of all the test cases on the DUT: workstation with measuring instrument: Intel Power Gadget. The lines represent the minimum, maximum and average energy consumption



(a) Test case: BinaryTrees

(b) Test case: FannkuchRedux

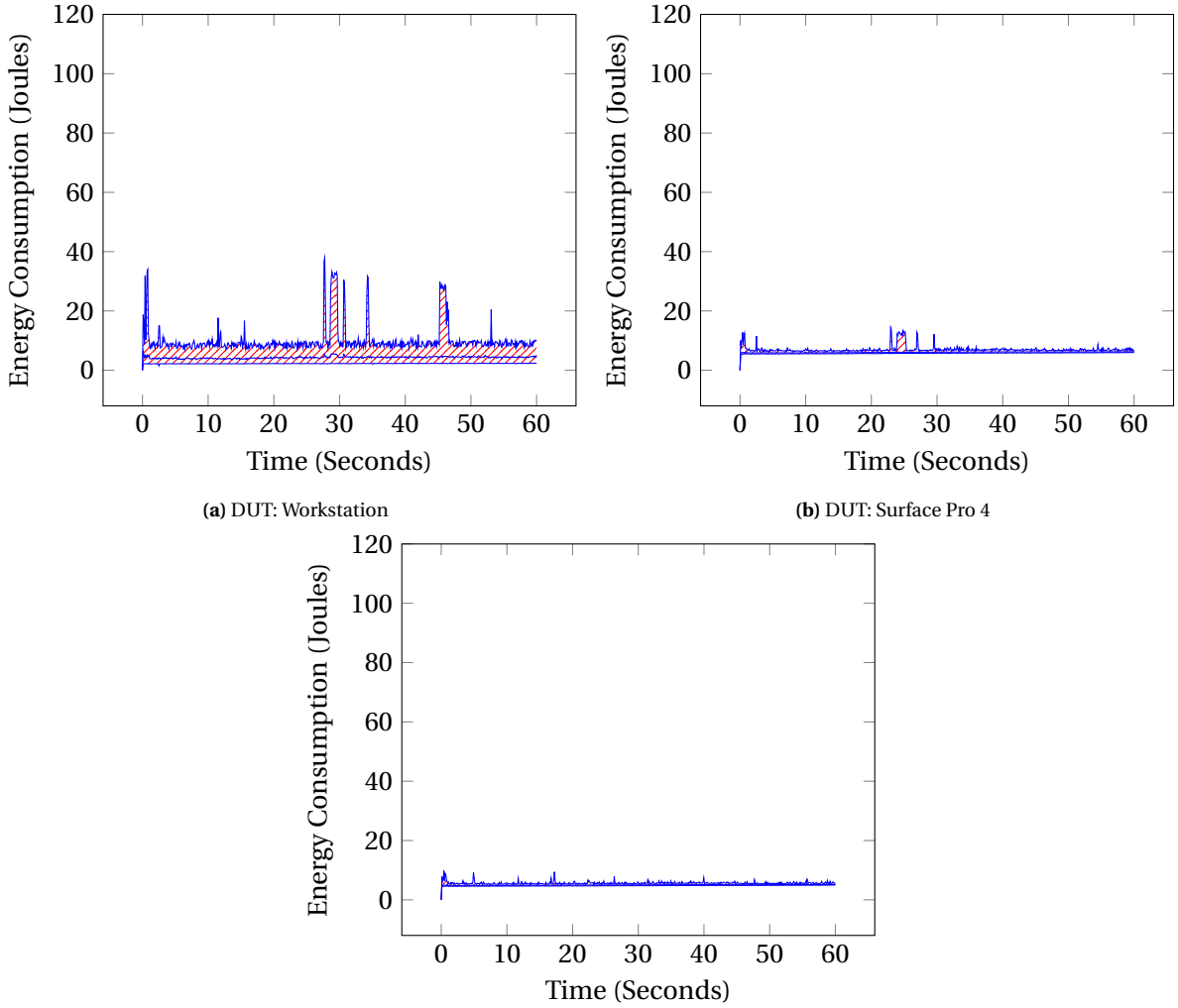
(c) Test case: Fasta

(d) Test case: Nbody

(e) Test case: TestCaseIdle

Figure A.2: The energy consumption of all the test cases on the DUT: Surface Pro 4 with measuring instrument: RAPL. The lines represent the minimum, maximum and average energy consumption

A.4 Comparing Time Series for the DUTs

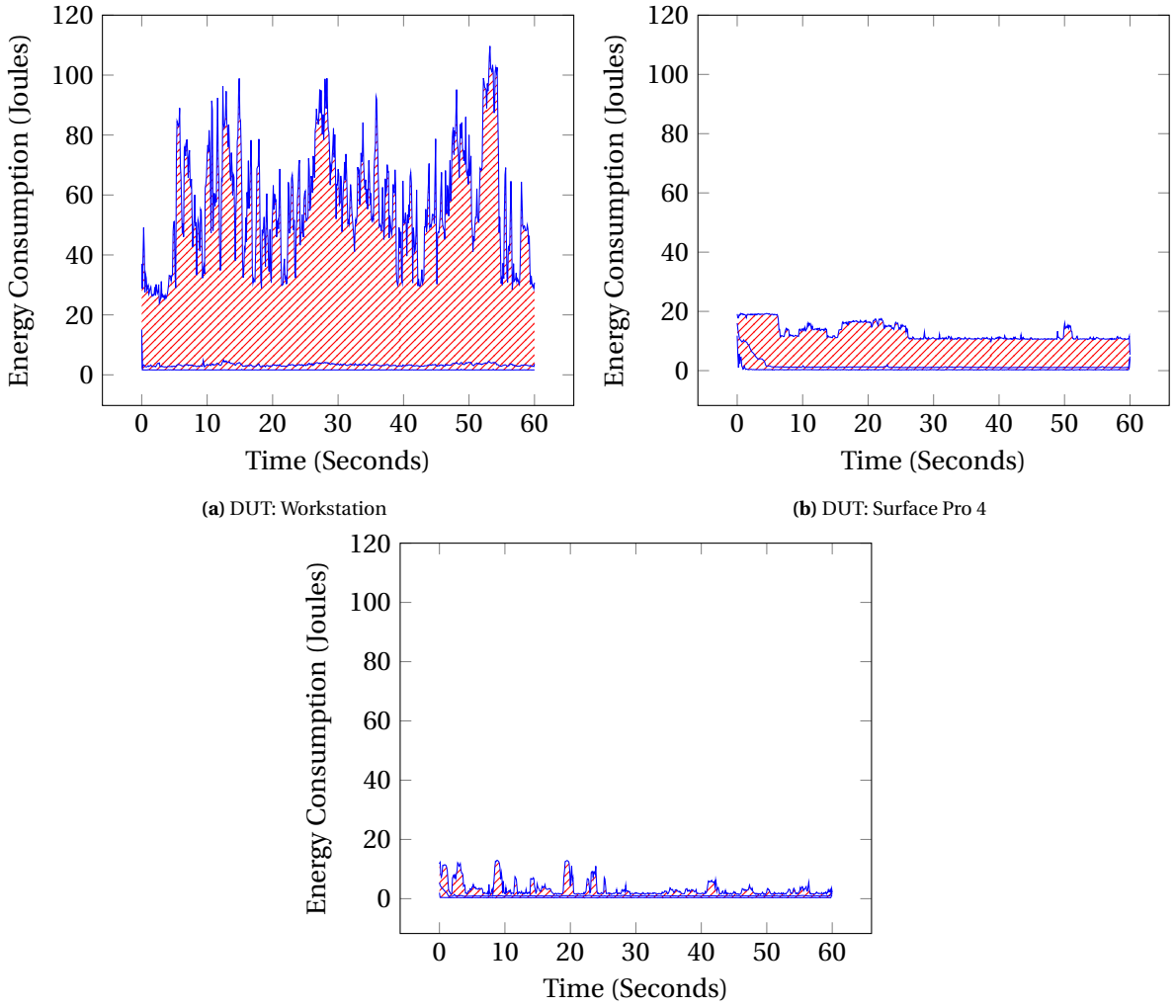


(a) DUT: Workstation

(b) DUT: Surface Pro 4

(c) DUT: Surface Book

Figure A.3: TestCaseIdle, measured by RAPL, with the lines representing the minimum, maximum and average energy consumption



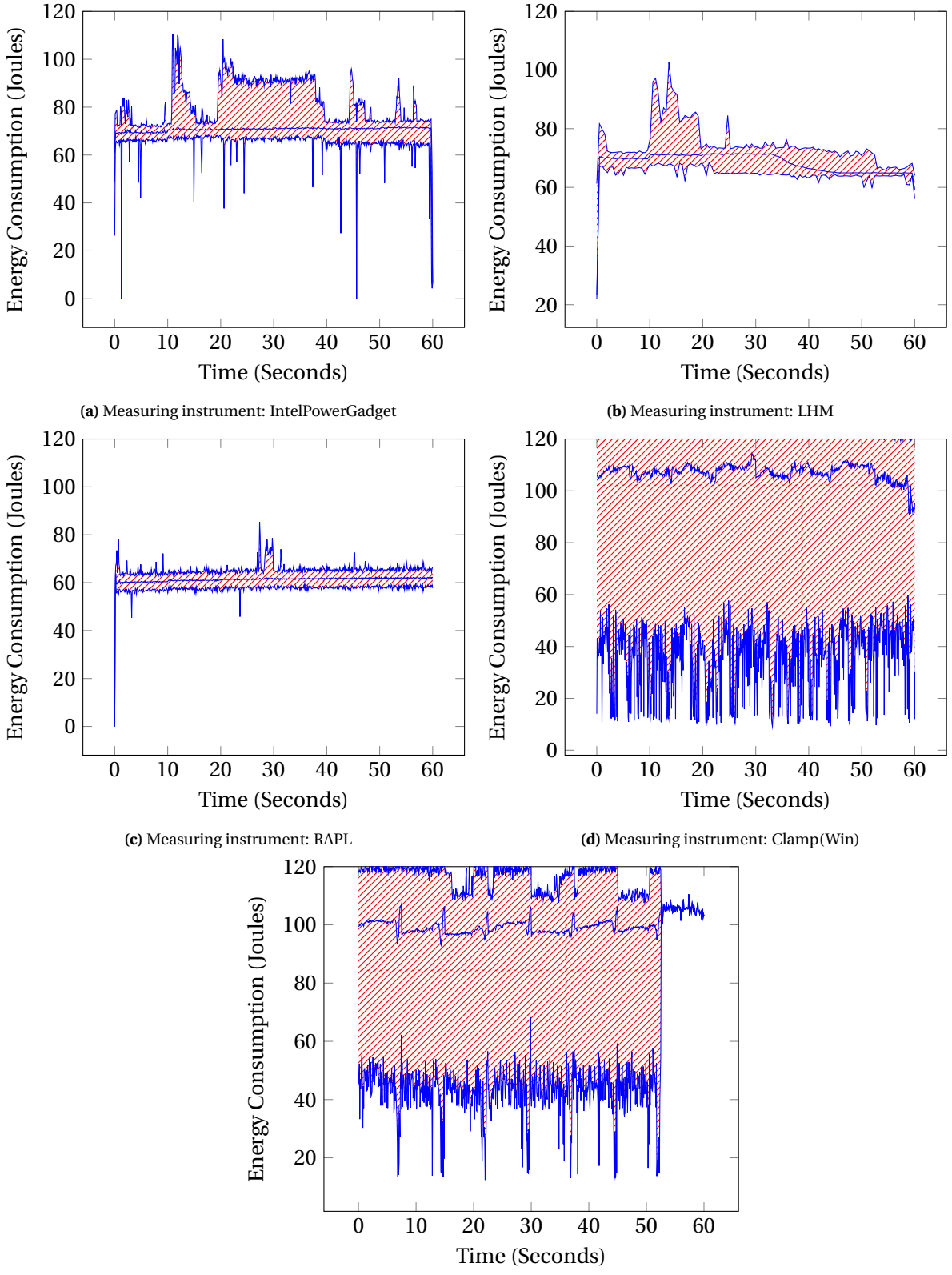
(a) DUT: Workstation

(b) DUT: Surface Pro 4

(c) DUT: Surface Book

Figure A.4: TestCaseIdle, measured by Intel Power Gadget, with the lines representing the minimum, maximum and average energy consumption

A.5 Comparing Time Series for the Measuring Instruments



(a) Measuring instrument: IntelPowerGadget

(b) Measuring instrument: LHM

(c) Measuring instrument: RAPL

(d) Measuring instrument: Clamp(Win)

(e) Measuring instrument: Clamp(Lin)

Figure A.5: BinaryTress on the workstation measured by the different measuring instrumentst, with the lines representing the minimum, maximum and average energy consumption

A.6 R3 Validation for the Three DUT's

Results from the experiment about R3 validation found in subsection 4.1.6.

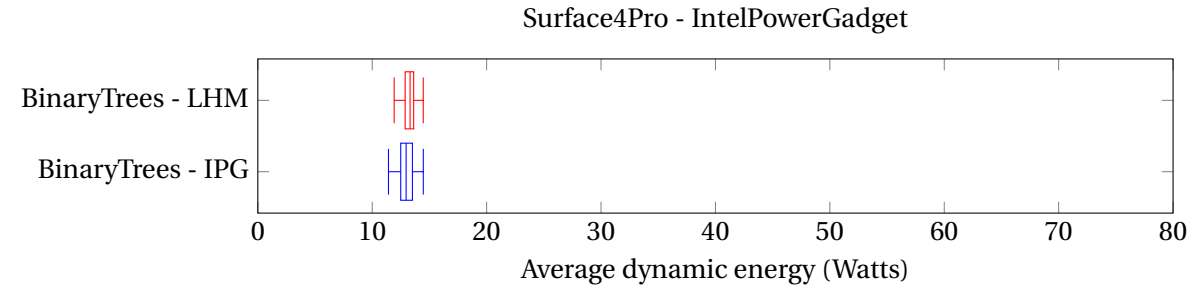


Figure A.6: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT Surface4Pro and test case BinaryTrees, where the impact of the first profiler can be seen (without outliers)

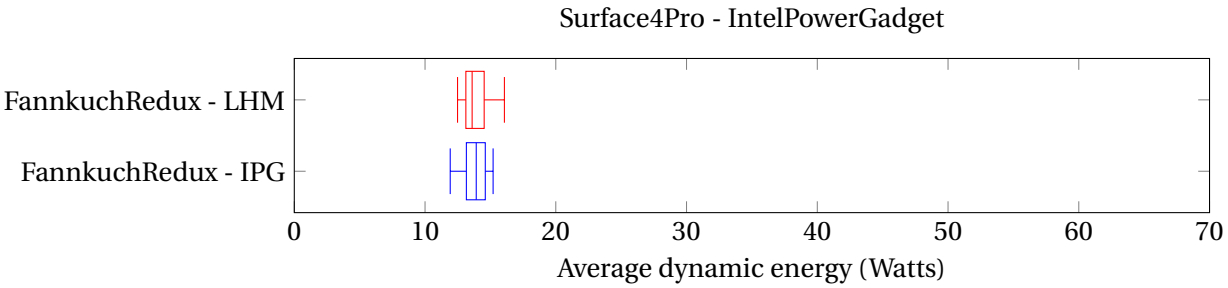


Figure A.7: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT Surface4Pro and test case FannkuchRedux, where the impact of the first profiler can be seen (without outliers)

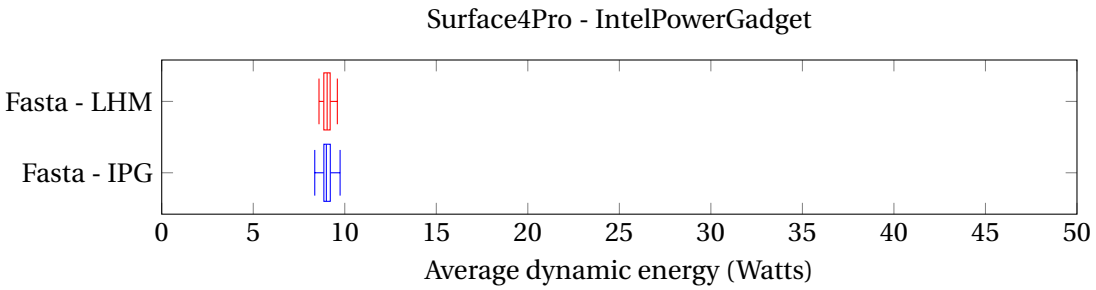


Figure A.8: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT Surface4Pro and test case Fasta, where the impact of the first profiler can be seen (without outliers)

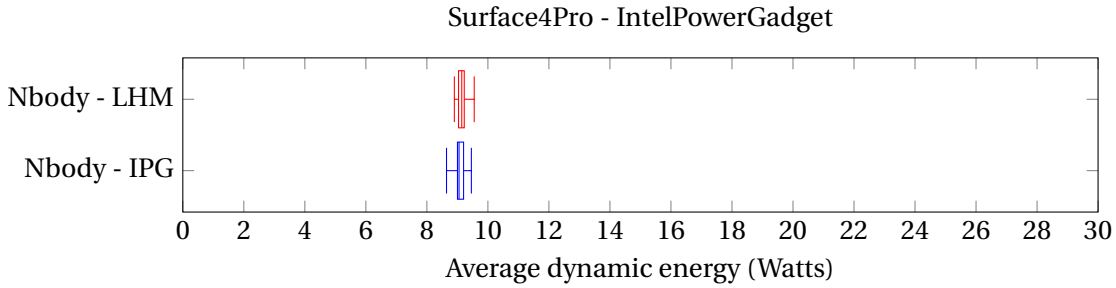


Figure A.9: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT Surface4Pro and test case Nbody, where the impact of the first profiler can be seen (without outliers)

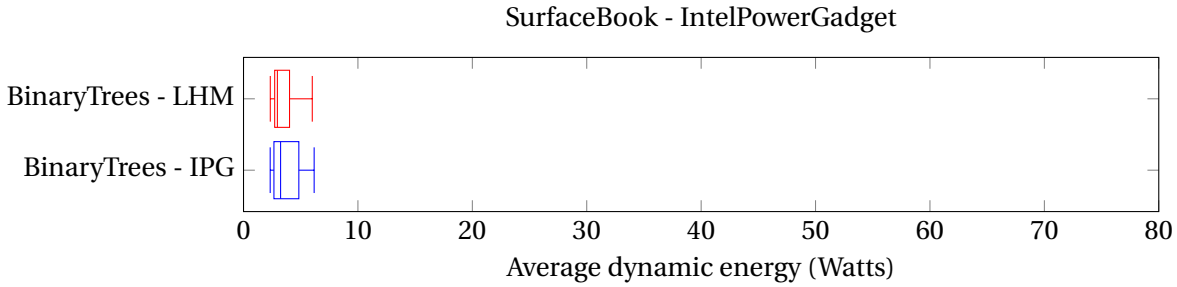


Figure A.10: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT SurfaceBook and test case BinaryTrees, where the impact of the first profiler can be seen (without outliers)

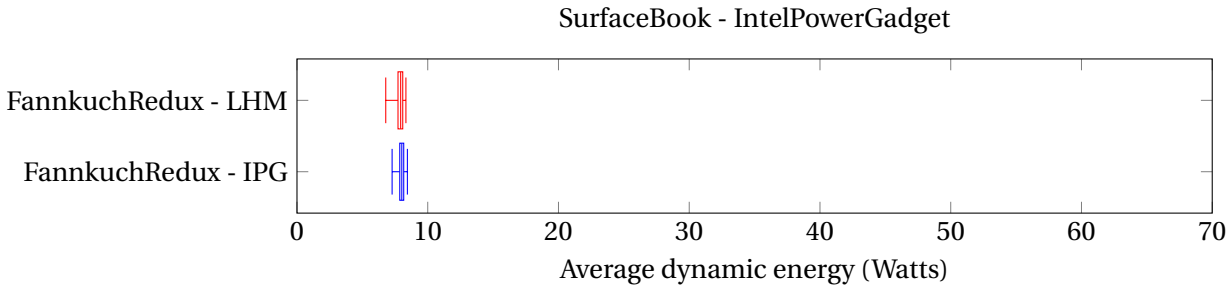


Figure A.11: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT SurfaceBook and test case FannkuchRedux, where the impact of the first profiler can be seen (without outliers)

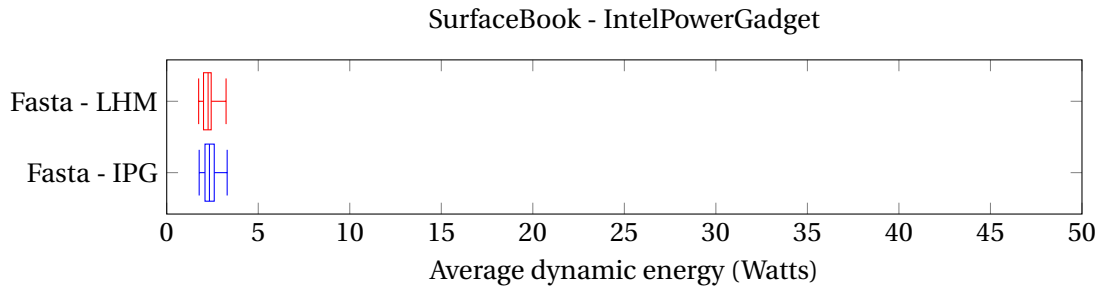


Figure A.12: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT SurfaceBook and test case Fasta, where the impact of the first profiler can be seen (without outliers)

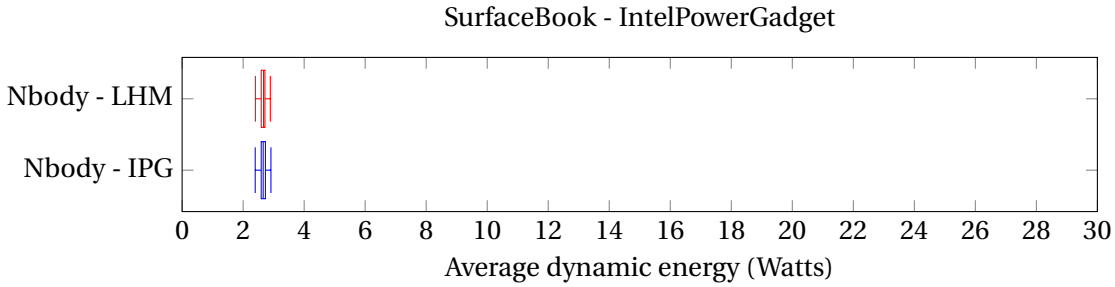


Figure A.13: R3 validation for dynamic energy measurements by IntelPowerGadget for the Cores for DUT SurfaceBook and test case Nbody, where the impact of the first profiler can be seen (without outliers)

A.7 The Battery Levels Impact on Performance

Results from the experiment about battery level in subsection 4.1.5.

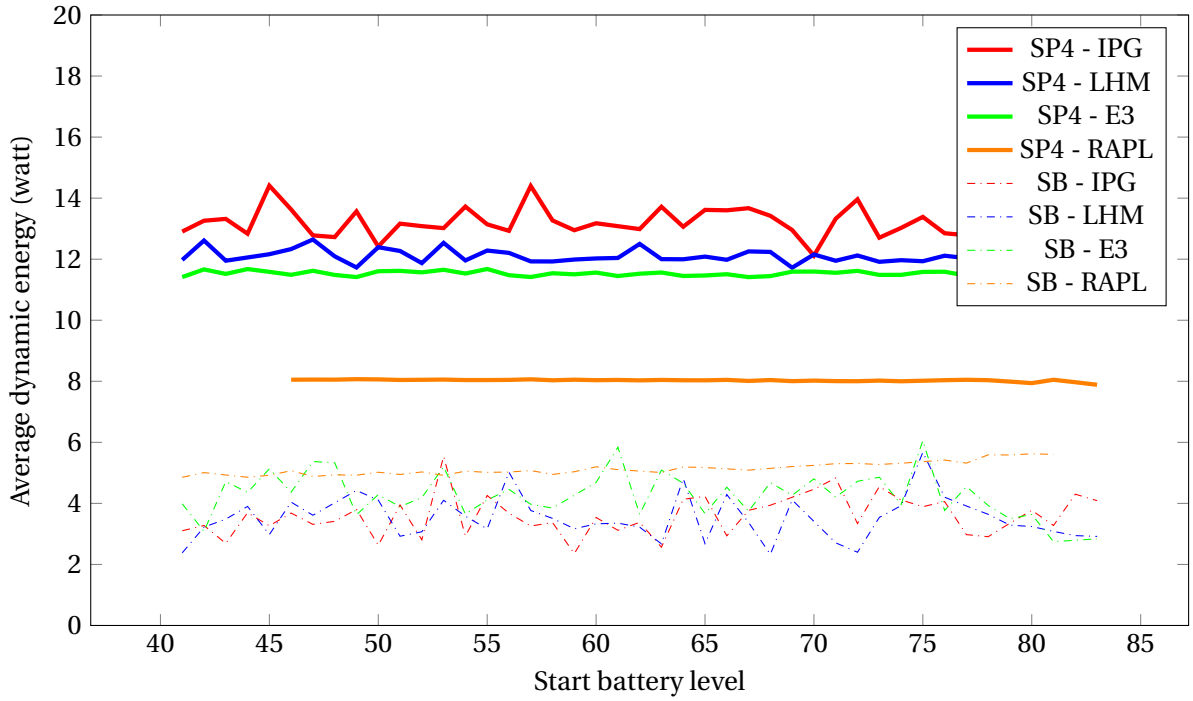


Figure A.14: A graph illustrating the energy consumption of Cores for test case BinaryTrees with regards to the battey level of the DUT (without outliers)

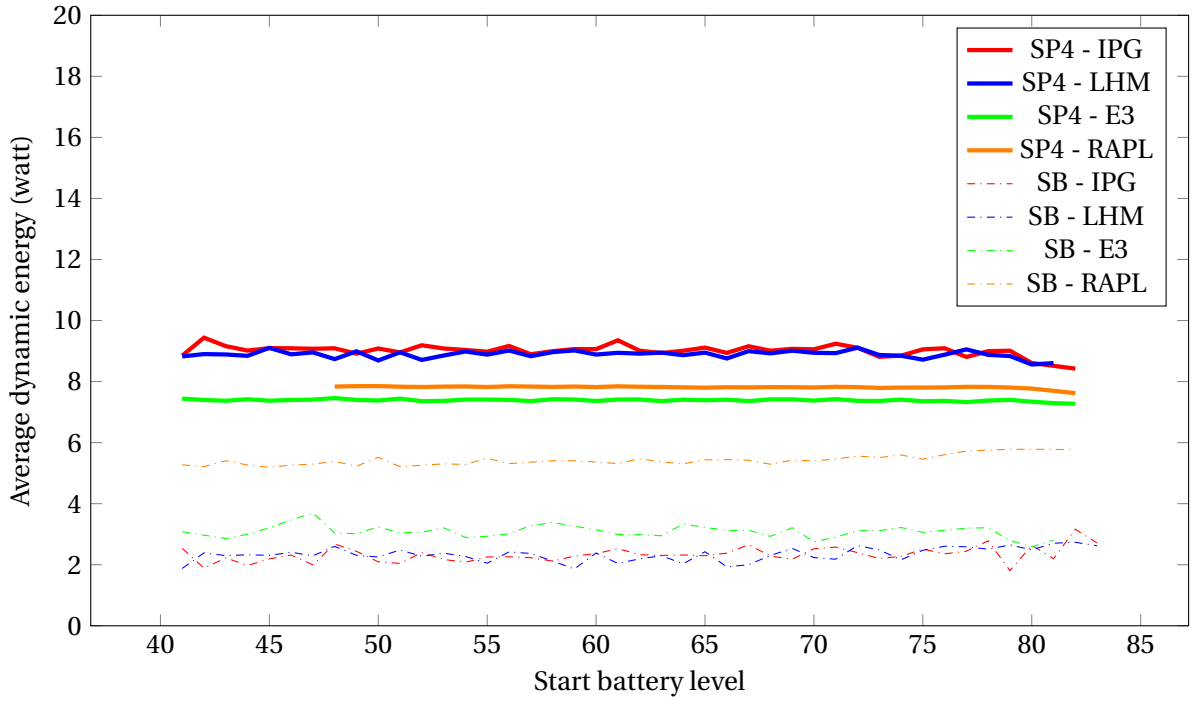


Figure A.15: A graph illustrating the energy consumption of Cores for test case Fasta with regards to the battey level of the DUT (without outliers)

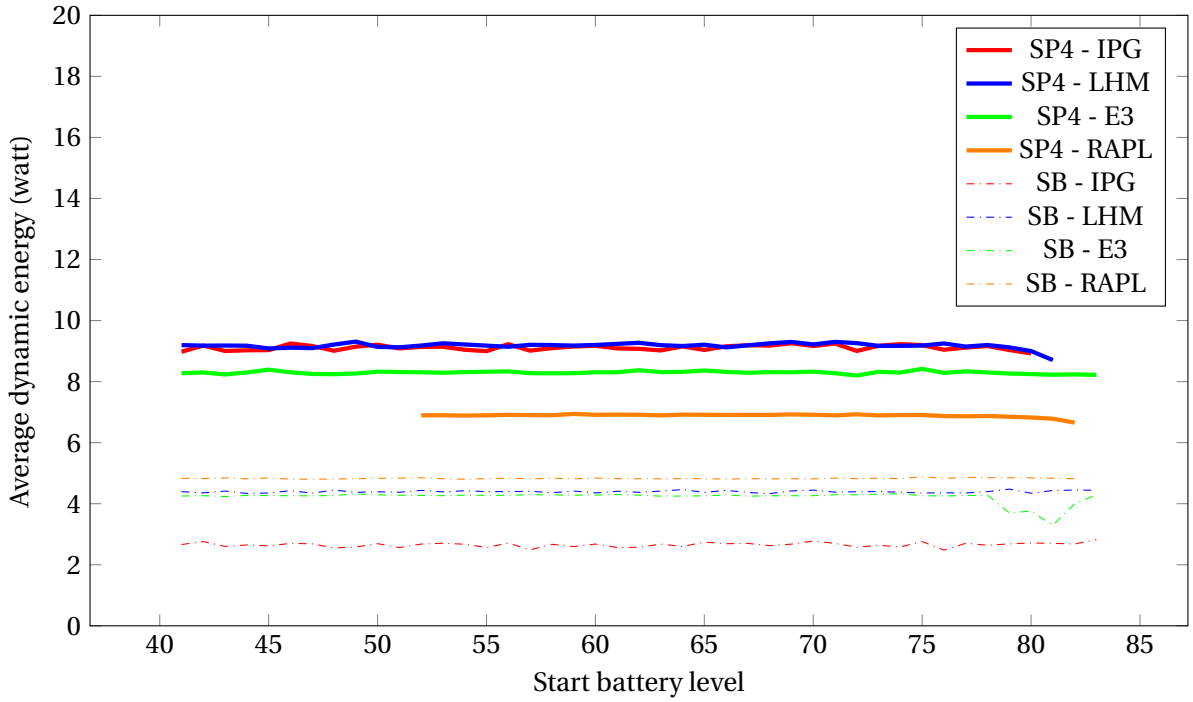


Figure A.16: A graph illustrating the energy consumption of Cores for test case Nbody with regards to the battey level of the DUT (without outliers)

A.8 The Temperatures Levels Impact on Performance

Results from the experiment about temperature in subsection 4.1.5.

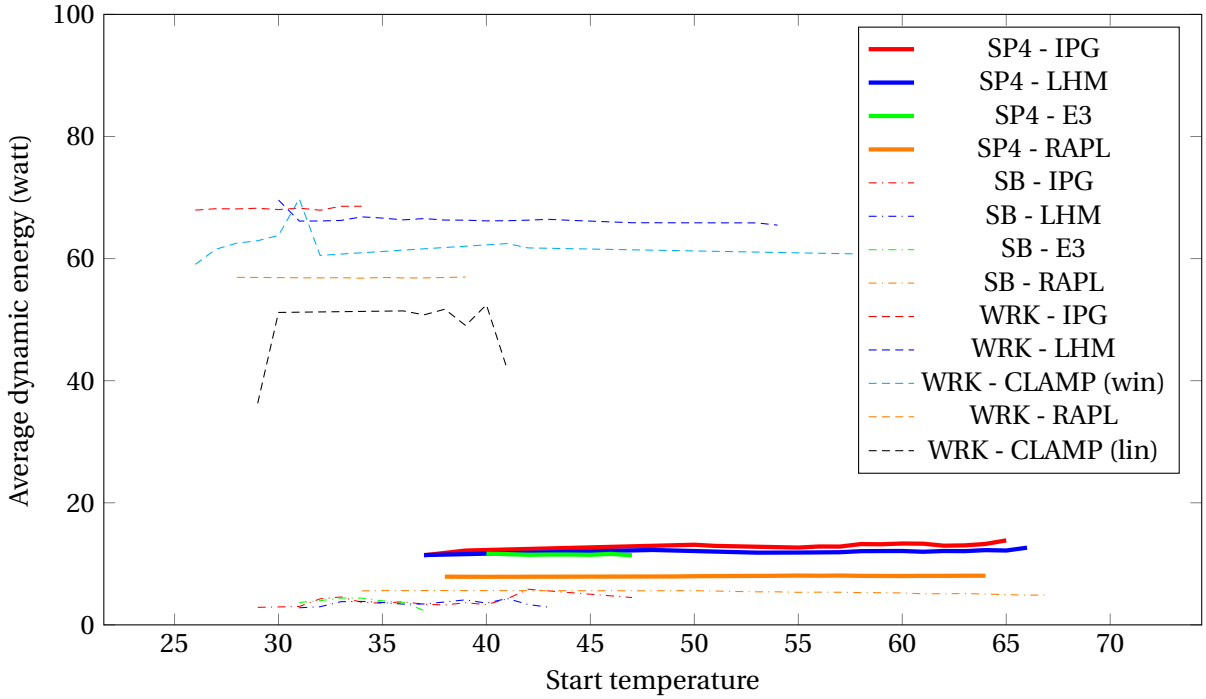


Figure A.17: A graph illustrating the energy consumption of Cores for test case BinaryTrees with regards to the temperature of the DUT (without outliers)

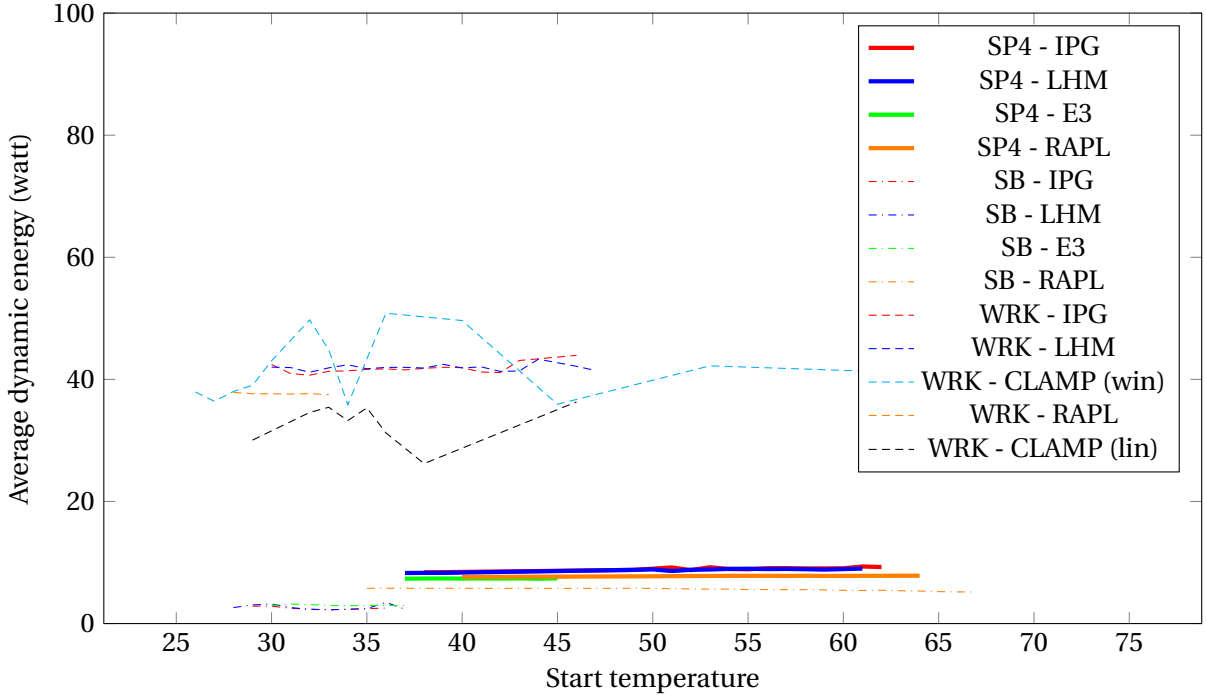


Figure A.18: A graph illustrating the energy consumption of Cores for test case Fasta with regards to the temperature of the DUT (without outliers)

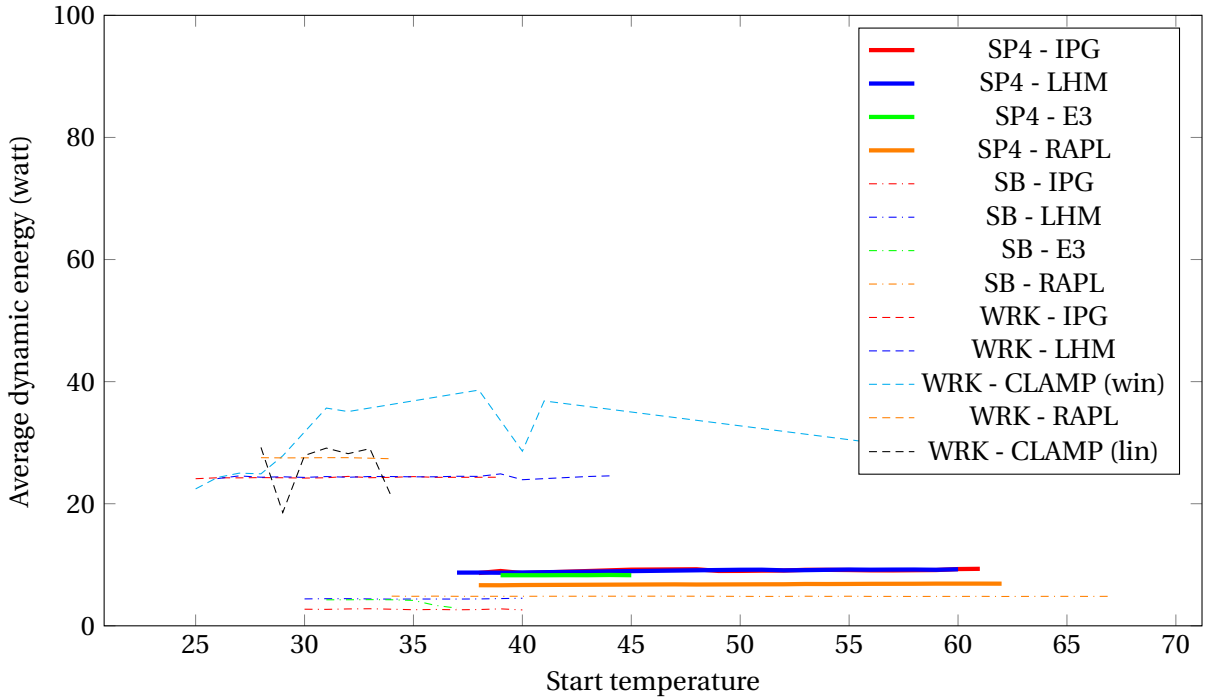


Figure A.19: A graph illustrating the energy consumption of Cores for test case Nbody with regards to the temperature of the DUT (without outliers)

A.9 Comparing the Dynamic Energy Consumption between DUT's

A comparison of the different DUT's, presented in section 5.3.

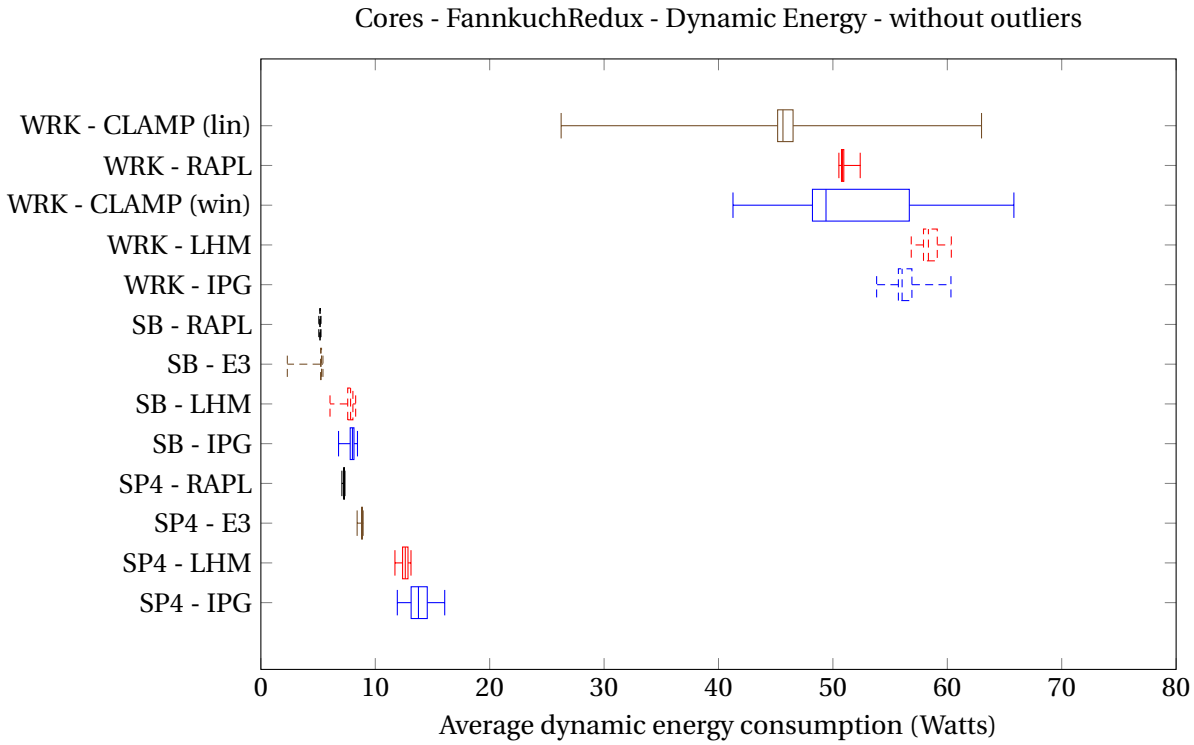


Figure A.20: A comparison of Cores dynamic energy consumption for test case FannkuchRedux for all DUT's and OS's (without outliers)

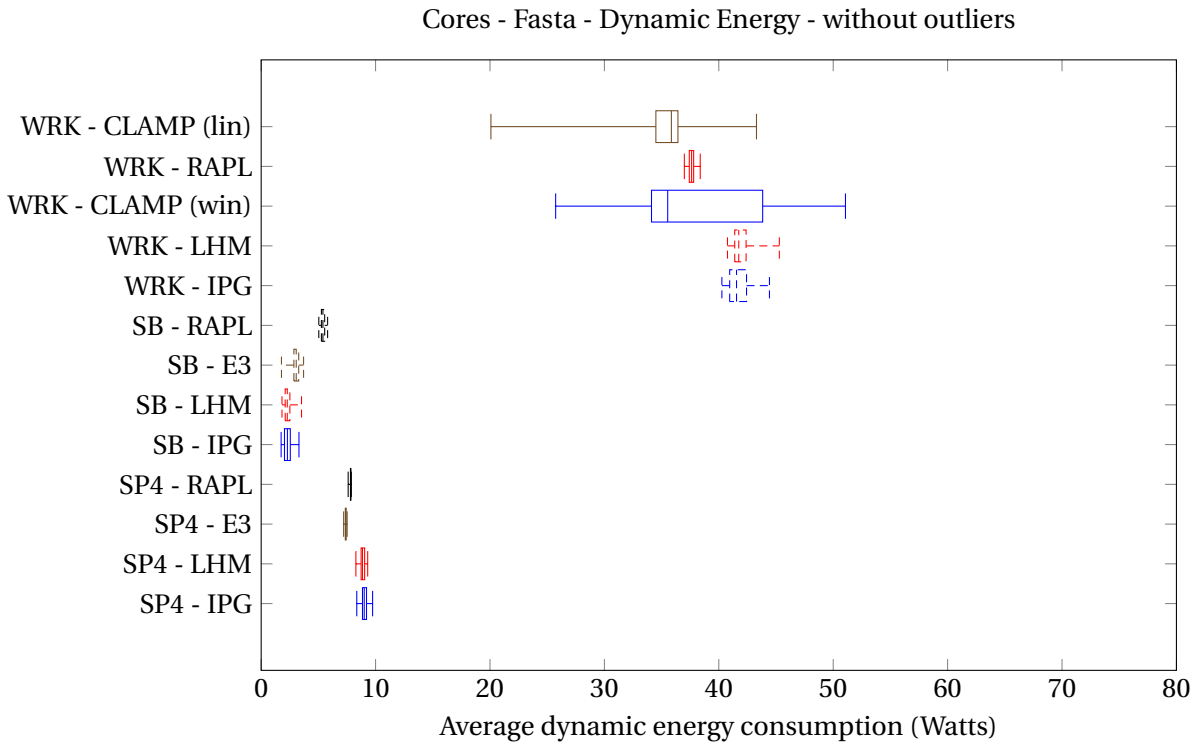


Figure A.21: A comparison of Cores dynamic energy consumption for test case Fasta for all DUT's and OS's (without outliers)

A.10 Comparing the Dynamic Energy Consumption Between the Workstation

A comparison of the different the workstation, presented in section 5.3.

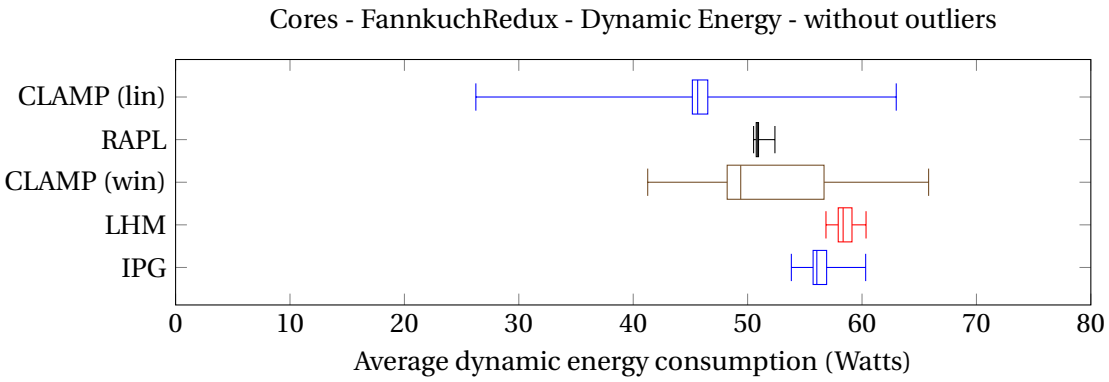


Figure A.22: A comparison of Cores dynamic energy consumption for test case FannkuchRedux for the workstation, (without outliers)

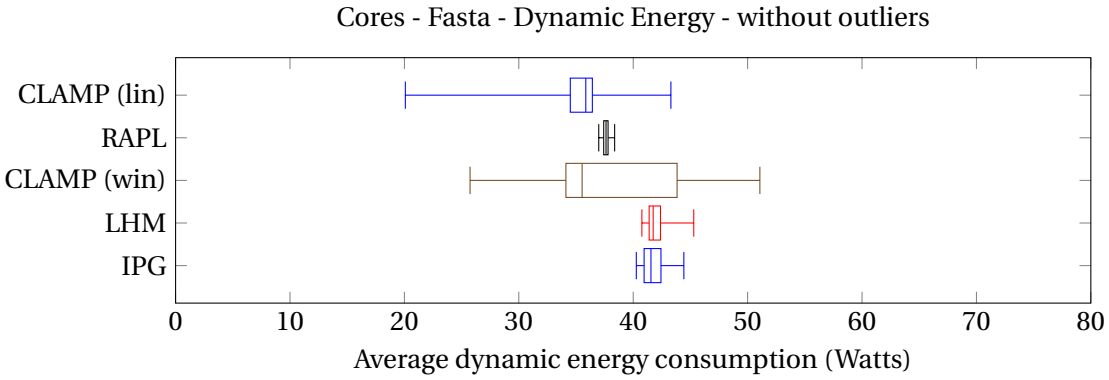


Figure A.23: A comparison of Cores dynamic energy consumption for test case Fasta for the workstation, (without outliers)

A.11 Comparing the Dynamic Energy Consumption Between the Surface Pro 4

A comparison of the different the Surface Pro 4, presented in section 5.3.

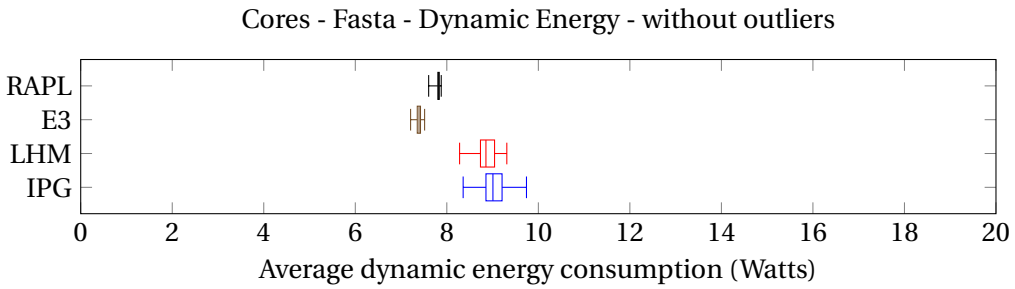


Figure A.24: A comparison of Cores dynamic energy consumption for test case Fasta for the Surface4Pro, (without outliers)

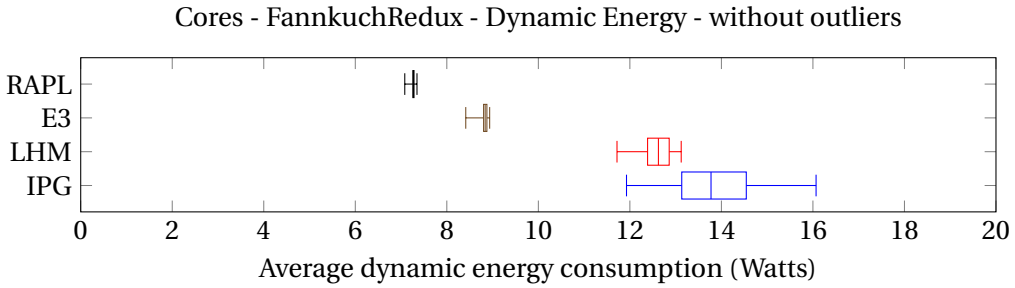


Figure A.25: A comparison of Cores dynamic energy consumption for test case FannkuchRedux for the Surface4Pro, (without outliers)

A.12 Comparing the Dynamic Energy Consumption Between the Surface Book

A comparison of the different Surface Book, presented in section 5.3.

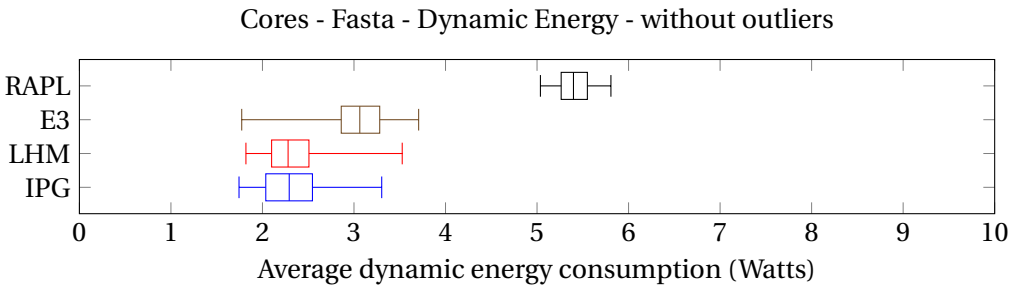


Figure A.26: A comparison of Cores dynamic energy consumption for test case Fasta for the SurfaceBook, (without outliers)

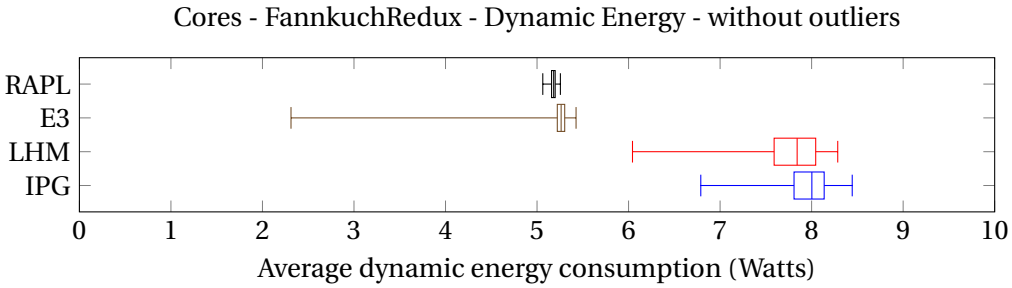


Figure A.27: A comparison of Cores dynamic energy consumption for test case FannkuchRedux for the SurfaceBook, (without outliers)

A.13 Comparing the Dynamic Energy Consumption Over Time

Results from the experiment about the evolution of energy consumption over time, as presented in section 5.2.

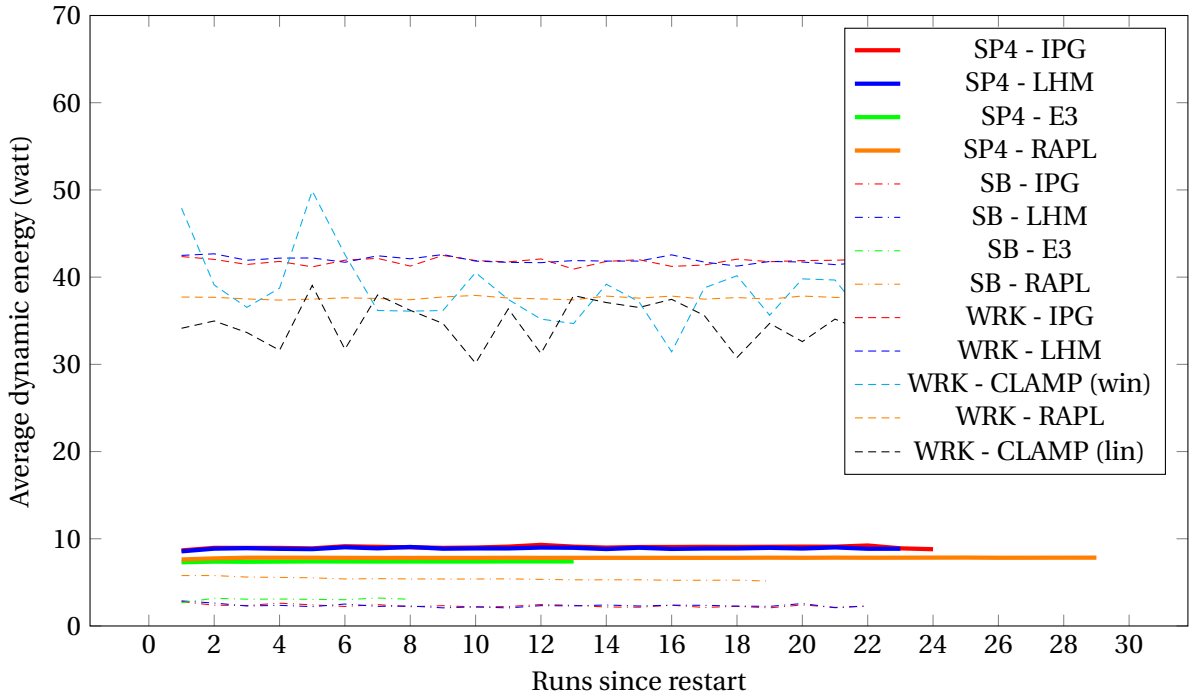


Figure A.28: A graph illustrating the energy consumption of Cores for test case Fasta with regards to how long ago the DUT was restarted (without outliers)

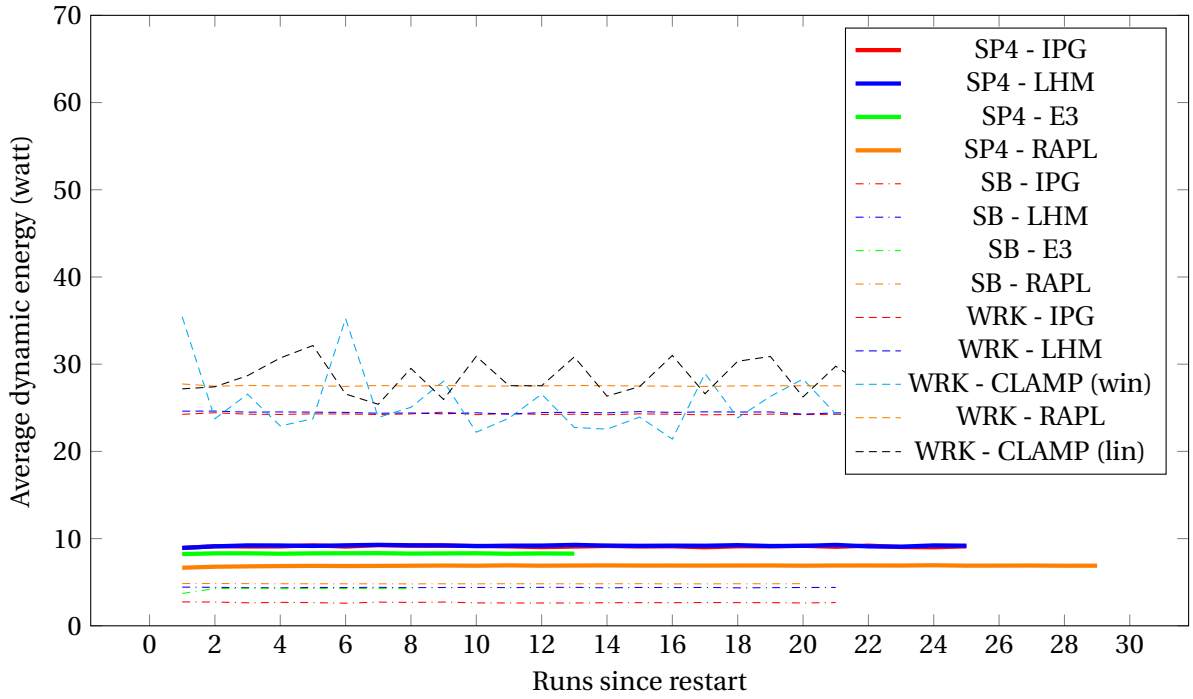


Figure A.29: A graph illustrating the energy consumption of Cores for test case Nbody with regards to how long ago the DUT was restarted (without outliers)

A.14 Results from Experiment #2

The results from the second experiment, presented in section 5.6.

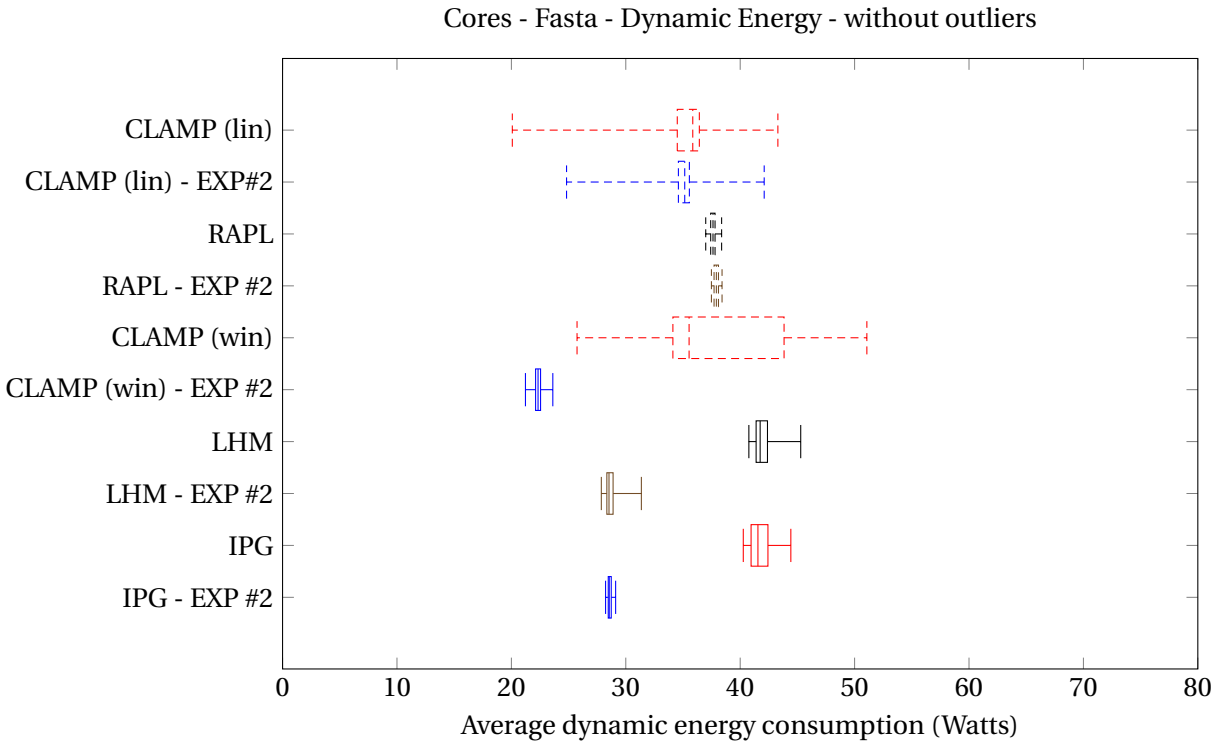


Figure A.30: A comparison of Cores dynamic energy consumption for test case Fasta for the workstation, experiment #2 (without outliers)

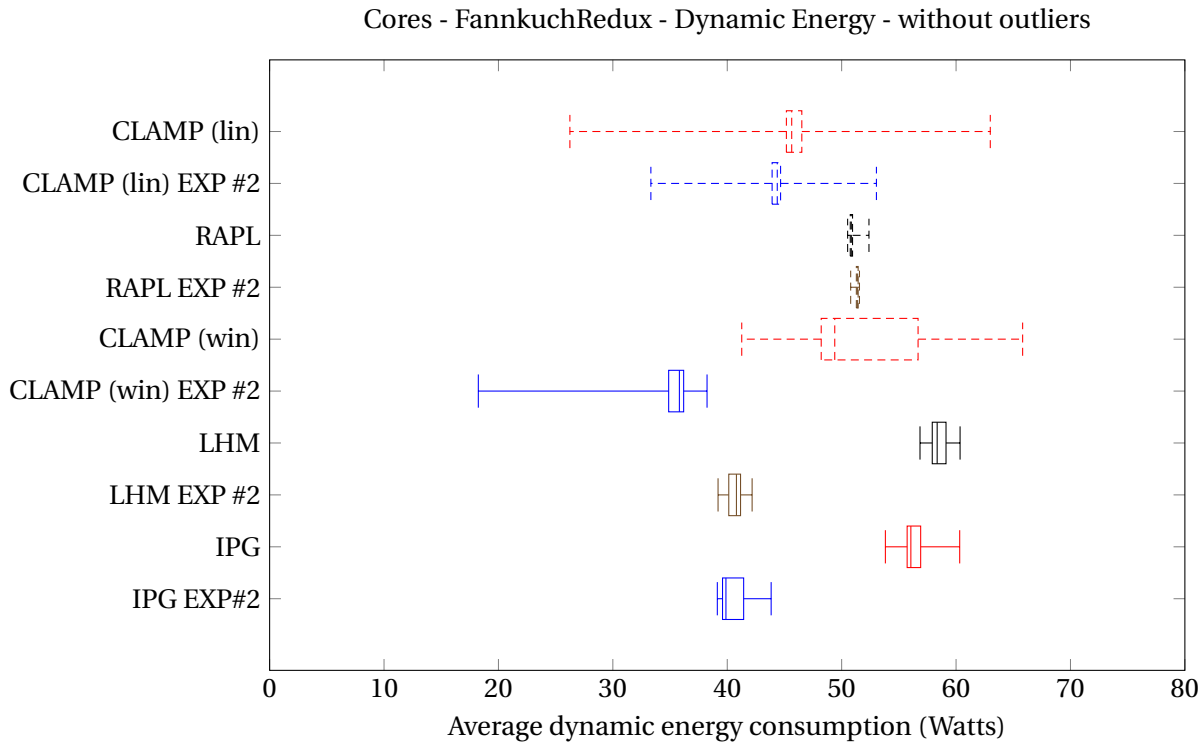


Figure A.31: A comparison of Cores dynamic energy consumption for test case FannkuchRedux for the workstation, experiment #2 (without outliers)

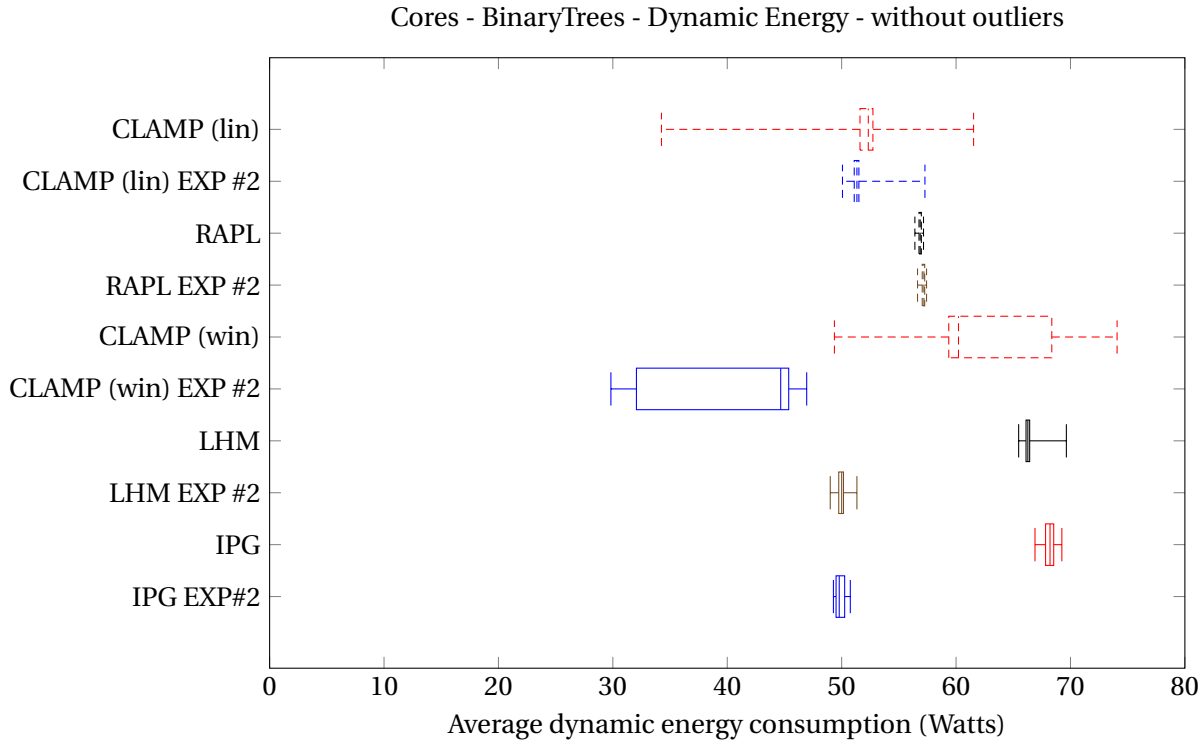


Figure A.32: A comparison of Cores dynamic energy consumption for test case BinaryTrees for the workstation, experiment #2 (without outliers)

	TestCaseIdle	BinaryTrees	FannkuchRedux	Nbody	Fasta
IntelPowerGadget	0.0004	**0.3685**	0.0007	0.0	**0.0809**
HardwareMonitor	0.0	0.0033	**0.088**	0.0	0.0002
E3	**0.9307**	**0.2229**	0.0	**0.0966**	0.0002
RAPL	0.0152	0.0311	0.0	0.0	0.0007

	TestCaseIdle	BinaryTrees	FannkuchRedux	Nbody	Fasta
IntelPowerGadget	0.0004	0.3685	0.0007	0.0	0.0809
HardwareMonitor	0.0	0.0033	0.088	0.0	0.0002
E3	0.9307	0.2229	0.0	0.0966	0.0002
RAPL	0.0152	0.0311	0.0	0.0	0.0007

A.15 Statistical results from Experiment #1

The results from the statistical analysis for the first expriment, presented in section 5.1.

	TestCaseIdle	BinaryTrees	FannkuchRedux	Nbody	Fasta
IntelPowerGadget	0.0	0.0	0.0	0.0	0.0001
HardwareMonitor	0.0	0.0	0.0	0.0	0.0
Clamp Win	0.0	0.002	0.0	0.0	0.0
RAPL	0.0	0.0128	0.0	0.0	0.0968
Clamp Lin	0.0	0.0	0.0	0.0	0.0

Table A.2: Here the various values for p using the Shapiro Wilk test for the testcases on the workstation

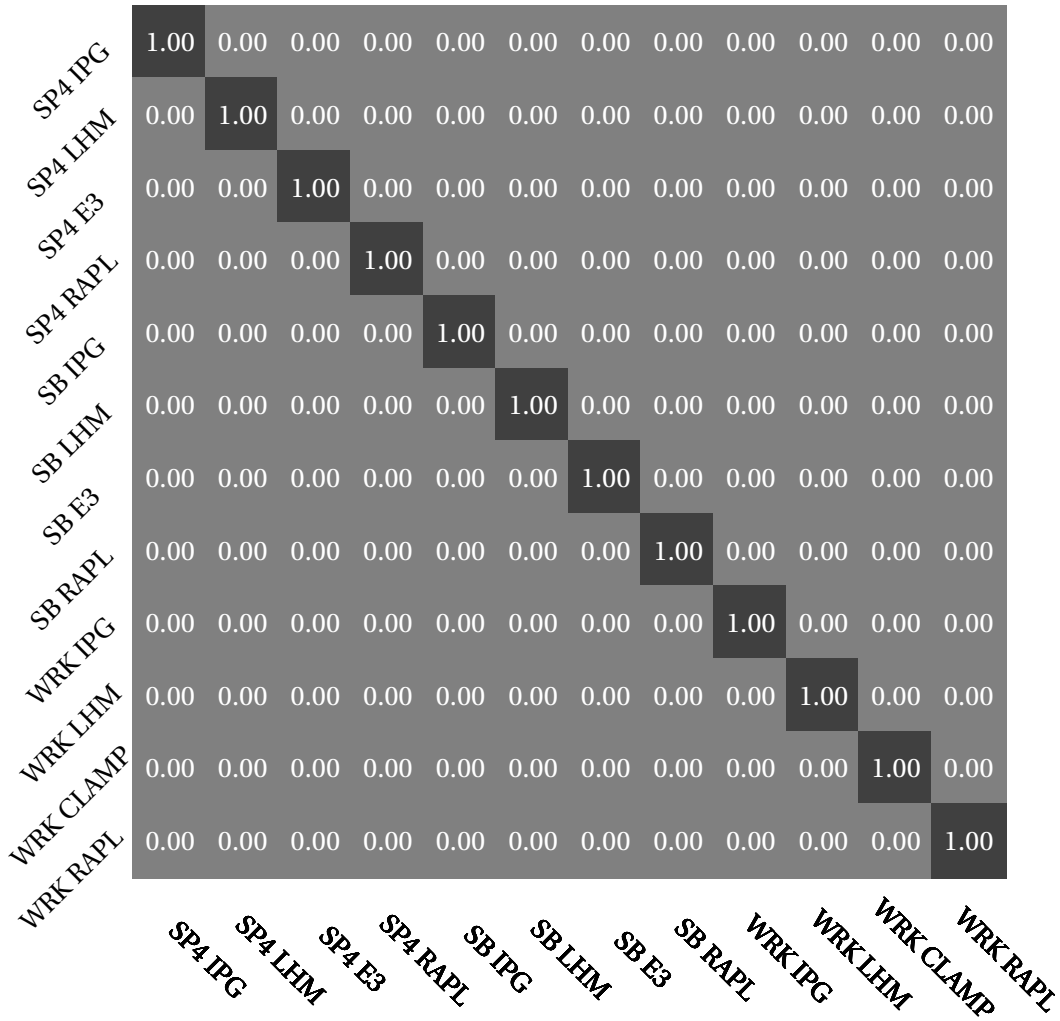


Figure A.33: The results for the BinaryTrees on the Mann Whitney U Test can be seen here. The Range is 0 – 1

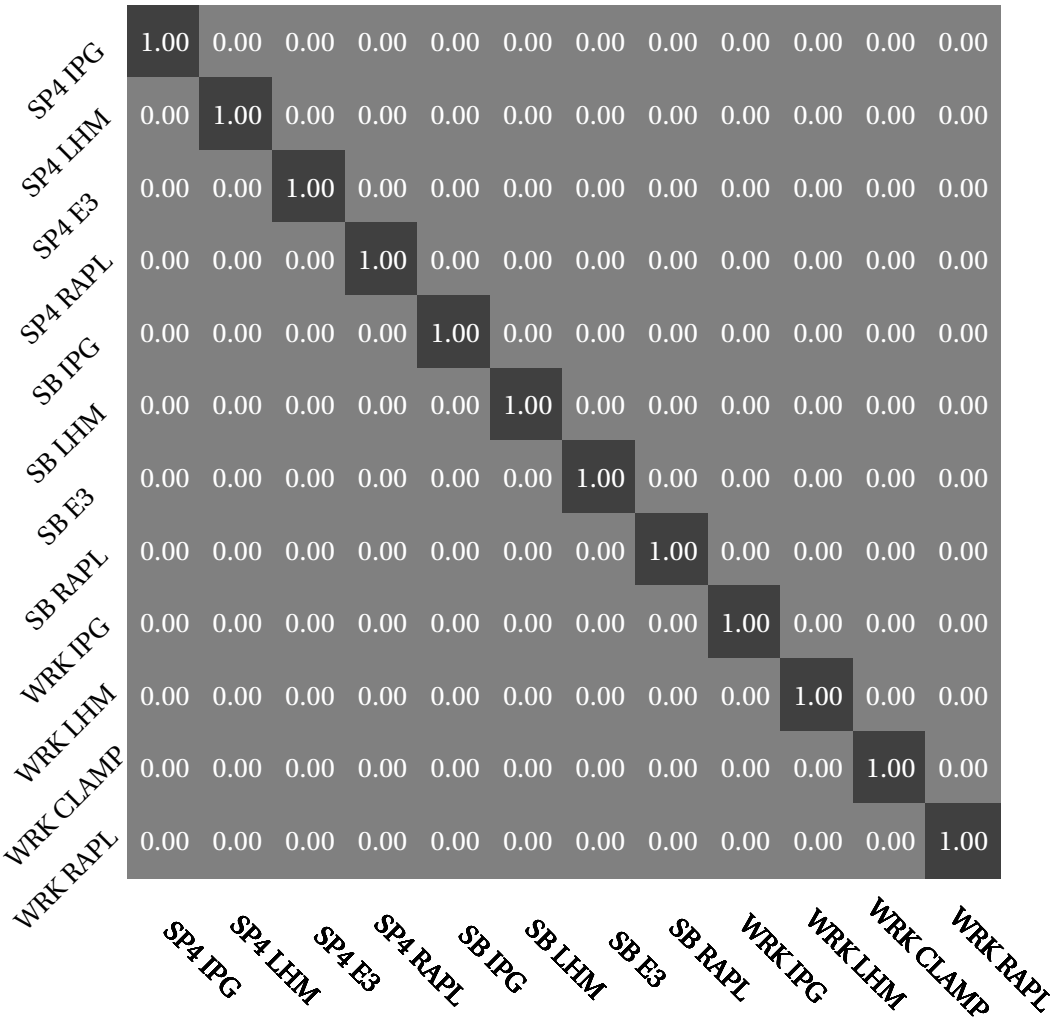


Figure A.34: The results for the Fasta on the Mann Whitney U Test can be seen here. The Range is $0-1$

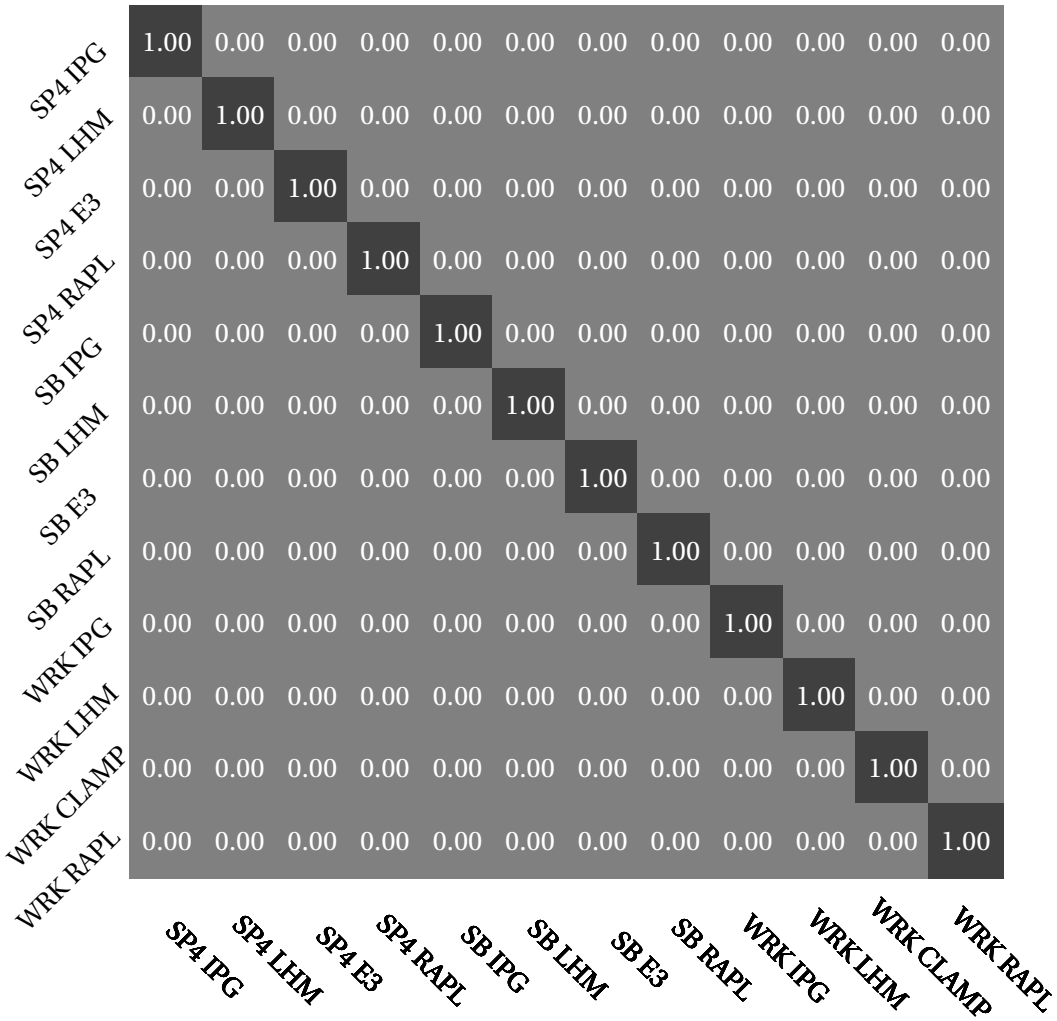


Figure A.35: The results for the Nbody on the Mann Whitney U Test can be seen here. The Range is 0 – 1

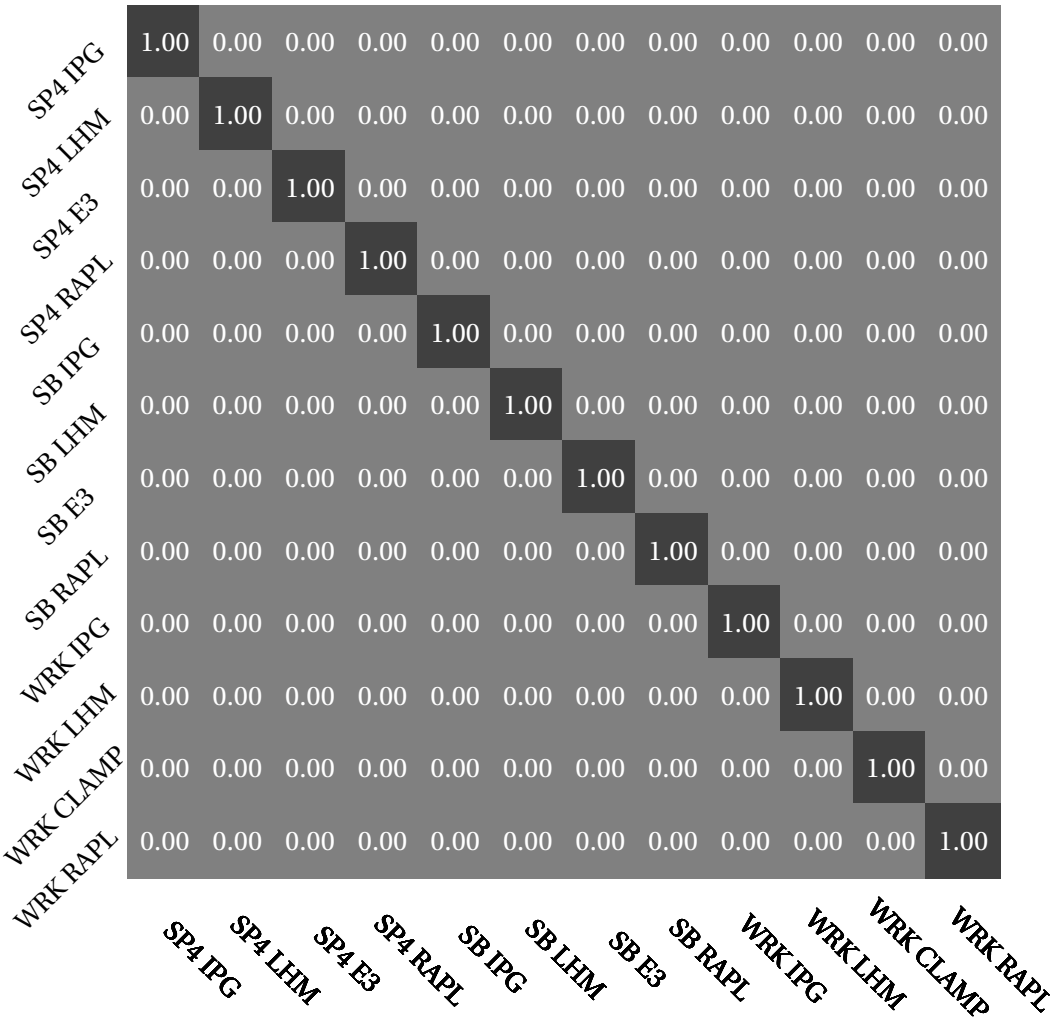


Figure A.36: The results for the Idle case on the Mann Whitney U Test can be seen here. The Range is 0 – 1

A.16 Statistical results from Experiment #2

This results from the statistical analysis of the second experiment, presented in section 5.5.

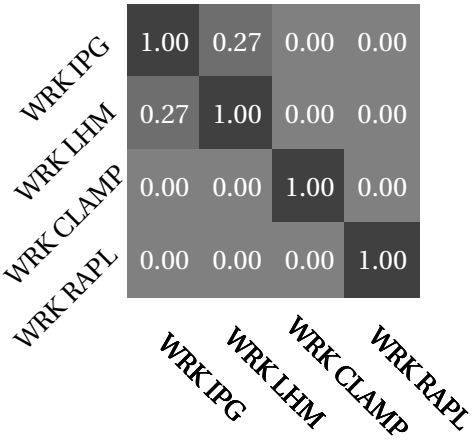


Figure A.37: Here the results for the BinaryTrees on the Mann Whitney U Test can be seen. The Range in 0 – 1

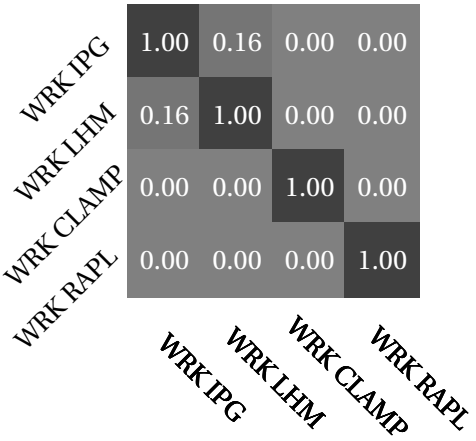


Figure A.38: Here the results for the Fasta on the Mann Whitney U Test can be seen. The Range in 0 – 1

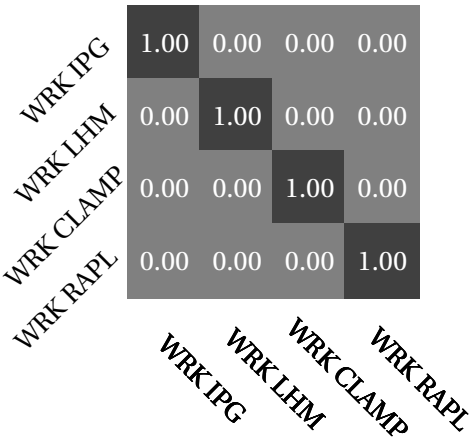


Figure A.39: Here the results for the Idle case on the Mann Whitney U Test can be seen. The Range in 0 – 1

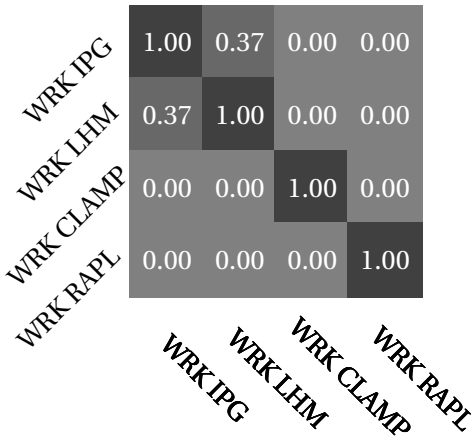


Figure A.40: Here the results for the Nbody on the Mann Whitney U Test can be seen. The Range in $0-1$