Summary

The paper *'Exploring the Energy Consumption of Highly Parallel Software on Windows* delves into the challenges of measuring the energy consumption of software on the Windows operating system. Whereas most existing work for measuring the energy consumption of software is based on Linux, this work aims to overcome the challenges of measuring energy consumption on Windows and make it more accessible. The paper investigates different aspects of the process, such as finding and evaluating tools available on Windows, testing DUTs with symmetric and asymmetric CPU architectures, and using benchmarks to measure and compare their energy consumption in various situations. The work is based on the following four research questions:

- RQ1: How does the C++ compiler used to compile the benchmarks impact energy consumption?
- RQ2: What are the advantages and drawbacks of the different measuring instruments for Windows regarding accuracy, ease of use, and availability?
- RQ3: What effect does parallelism have on the energy consumption of the benchmarks?
- RQ4: What effect do P- and E-cores have on the parallel execution of a process?

The first experiment focuses on RQ1, where different C++ compilers are tested to find the most energy-efficient one. The results showed that the Intel oneAPI was the most energy-efficient and had the fastest execution time for both benchmarks. Given the good performance of oneAPI, it is used in the second experiment. The second experiment focuses on RQ2 and aims to find the best measuring instrument defined in terms of accuracy, ease of use, and availability, where the experiment was conducted by comparing the correlation of the measuring instruments against a ground truth. In the end, similar performance is found between the tested measuring instruments. However, given the ease of use, Intel Power Gadget was the most usable instrument among those tested. Therefore, the third experiment uses Intel Power Gadget, which focuses on RQ3 and RQ4. In the third experiment, the effect of parallelism on the energy consumption of benchmarks is examined by executing two macrobenchmarks on an increasing number of cores. This experiment finds that more resources results in a lower execution time, while the energy consumption will remain the same. Finally, the effect of P- and E-cores on the parallel execution of a process is analyzed. There are two parts, where one part executes the same microbenchmark on one core at a time, and the other by executing macrobenchmarks on four cores, as either four P-cores, 2 P- and 2 E-cores, or 4 E-cores. This experiment found that P-cores will have a lower execution time, while E-cores, in some cases, have a lower energy consumption.

The paper's discussion section provides further insights into the research. Firstly, Cochran's formula was used to determine the number of measurements needed to gain confidence in the results. Cochran's formula found that results deviated between benchmarks, measuring instruments, DUTs, and even cores on the same DUT. This could be due to the variability in the fabrication process, which can cause changes in the exact characteristics of each core. Another cause of this could be the CPU, as it is found that CPUs with a high TDP can result in a high deviation, which results in many measurements being required. Because of this, an upper limit of 1.000 measurement was introduced, as the additional measurements had a limited effect on the results. Secondly, the paper analyzes the results of comparing different C++ compilers, where it was found that the energy consumption, execution time, and measurements required deviated between compilers. The analysis showed that the code compiled by the oneAPI performs better because it uses AVX functions and Advanced Vector Extensions to perform calculations in parallel. A discussion is also made about some observed energy usage trends during the experiments. The trend observed saw the energy consumption of the idle computer increase during work hours and decrease during non working hours. This was suspected to be caused by reactive energy consumption, which depends on the time of the day. Another topic discussed is time synchronization issues, as devices like the Raspberry Pi and an Analog Discovery 2 each kept their time, which could have caused issues if they were not synchronized. The data acquisition process was changed to ensure the devices were synchronized every second, but small time drifts could still occur over time.

In conclusion, this research addresses the challenges of measuring software energy consumption on Windows and provides valuable insights into the impact of different factors on energy consumption and execution time. The paper demonstrates the importance of choosing an energy-efficient compiler and using the right tools to measure energy consumption accurately. The findings also suggest that using E-cores can limit energy consumption and that the impact of parallelism on energy consumption is mixed. This paper's results demonstrate the need for more research into software energy consumption on Windows and the importance

of considering various factors when developing energy-efficient software. Future research can build on this study’s findings and help reduce the environmental impact of software development.

Exploring the Energy Consumption of Highly Parallel Software on Windows

Mads Hjuler Kusk*, Jeppe Jon Holt*
and Jamie Baldwin Pedersen*
Department of Computer Science, Aalborg University, Denmark
*{mkusk18, jholt18, jjbp18}@student.aau.dk

June, 2023

Abstract

With the evolution of CPUs in recent years, increasing the number of cores has become the norm resulting in additional resources for software to utilize. Through four research questions, we investigate the energy consumption and performance gains obtained from the additional processing power and the impact of P- and E-cores on parallel software. The experiments are conducted on two computers, where the analysis is made based on energy consumption and execution time on a per-core basis and on an increasing number of cores. The experiments are primarily conducted on Windows, where Intel's Running Average Power Limit is unavailable. The energy consumption on Windows is measured on the best performing measuring instrument, found through a set of experiments. Through the experiments, it is found, that more measurements are required to gain confidence in the results than is generally seen in the literature. Furthermore, potential issues when presenting multiple measurements taken over a period of time as a single value were discovered, and that no correlation between energy consumption and execution time when executing benchmarks on more cores could be found.

1 Introduction

In recent years there has been rapid growth in Information and Communications Technology (ICT), leading to an increase in energy consumption. Furthermore, it is expected that the rapid growth of ICT will continue, triggering an increase in computational power needs.[1, 2] Therefore, energy efficiency has become more of a concern for companies and software developers. As ICT has increased, top-of-the-line desktop CPUs from 2023[3] contain not only a higher quantity of cores than CPUs ten years ago[4], but also different types of cores. This modification aims to boost performance and energy efficiency[5].

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 $\approx 6\%$[6]. 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[7]. Because of this, a more in-depth understanding of the energy consumption of software is interesting, as software could help reduce the energy usage[6]. 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.[8] 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[8].

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[9]. 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.

In this paper, we investigate the energy consumption of various benchmarks on Windows 11, comparing the efficiency and tradeoffs between sequential and parallel execution. Our experiments involve two Devices Under Test (DUTs): an Intel Coffee Lake CPU with a traditional performance core setup and an Intel Raptor Lake CPU with a performance and efficiency cores (P- and E-cores) setup. This work analyzes the impact of Asymmetric Multicore Processors (AMPs) on parallel execution compared to traditional Symmetric MultiCore Processors. The first two experiments will focus on C++, where different C++ compilers and measuring instruments for Windows are compared and explored using microbenchmarks. C++ was chosen to avoid noise from, e.g., garbage collectors or just-in-time compilation. The third experiment will use the best-performing measuring instrument to go beyond C++ programs to instead focus on larger macrobenchmarks. The macrobenchmarks will be run on various amounts of cores to explore what impact additional resources have on execution time and energy consumption. The following research questions are formulated:

- RQ**1**: How does the C++ compiler used to compile the benchmarks impact energy consumption?
- RQ**2**: What are the advantages and drawbacks of the different measuring instruments for Windows regarding accuracy, ease of use, and availability?
- RQ**3**: What effect does parallelism have on the energy consumption of the benchmarks?
- RQ**4**: What effect do P- and E-cores have on the parallel execution of a process?

To answer these research questions, a command line framework is created to run a series of different experiments.

Section **2** covers the related work, laying the foundation for this work, Section **3** includes the necessary background information about, e.g., CPUs and schedulers and Section **4** coverers our experimental setup. In Section **5**, Section **6** and Section **7** the experiments are presented, discussed and concluded upon respectively. The source code for the project can be found on GitHub[1]

2 Related Work

This section provided an overview of related work in energy consumption, parallel software, compilers, and AMPs. It was built upon our previous work *"A Comparison Study of Measuring Instruments"*[10], where different measuring instruments were compared to explore whether a viable software-based measuring instrument was available for Windows. It was found that Intel Power Gadget (IPG) and Libre Hardware Monitor (LHM) on Windows and Intel's Running Average Power Limit (RAPL) on Linux had a similar correlation to hardware-based measuring instruments.[10]

2.1 Variations in Energy Measurements

[10] found that energy consumption can vary between measurements, as was also explored in [11], which discovered that numerous variables affect variations in energy measurement. [11] managed to reduce the variation by 30 times by analyzing different controllable parameters conducted on 100 different nodes.

One such parameter is temperature, which has produced conflicting conclusions. [12] observed energy consumption variation on identical processors without any correlation between temperature and performance. In contrast, [13] found the opposite to be true. [11] performed an experiment where benchmarks were executed on three different configurations, either right after each other, with a one-minute sleep between executions or a restart between benchmark executions. The configurations were not found to have an impact on the energy variation.

[11] found that disabling C-states resulted in measurements varying five times less on lower workloads with a 50% higher energy consumption, while no difference was observed on higher workloads. While [14] had previously found that disabling Turbo Boost reduced variation from 1% to 16%, [11] could not find any evidence supporting this.

[11] found that disabling non-essential processes, such as Wi-Fi and logging modules, yielded no substantial difference in energy variation. [15, 16] found that older CPUs exhibited lower deviation than newer CPUs. Although a similar experiment in [11] did not confirm that older CPUs always vary less, they argued that it depends on the generation and observed that CPUs with lower Thermal Design Power (TDP)[2] deviated less.

2.2 Parallel Software

Amdahl's law describes the potential speedup achieved by running an algorithm in parallel based on the proportion of the algorithm that can be parallelized and the number of cores used.[18] In [19], Amdahl's law was extended to estimate energy consumption with different amounts of cores. It was argued that a CPU could lose its energy efficiency as the number of cores increased, and that knowing how parallelizable a program is before execution allowed for calculating the optimal number of active

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

[2]The power consumption under the maximum theoretical load[17]

cores for maximizing performance and energy consumption.[19]

[20] compared the observed speedup of computing Laplace equations with one, two, and four cores, with estimates given by Amdahl's law and Gustafson's law. Gustafson's law evaluates a parallel program's speedup based on the problem's size and the number of cores. Unlike Amdahl's law which assumed a fixed problem size and a fixed proportion of the program that could be parallelized, Gustafson's law takes into account that larger problems could be solved when more cores are available and that the parallelization of a program could scale with the problem size. Comparing the observed and estimated speedup, it was clear that Gustafson's law was more optimistic than Amdahl's law, where both underestimated the speedup on two and four cores.[20]

In [21], three different thread management constructs from Java were explored and analyzed. When allocating additional threads, the energy consumption was found to increase until a certain point where the energy consumption would start to decrease. The exact peak point was, however, found to be application-dependent. The study also found that in eight out of nine benchmarks, there was a decrease in execution time when transitioning from sequential execution on one thread to using multiple threads. It should be noted that four of their benchmarks were embarrassingly parallel, while only one was embarrassingly serial. The results also showed how a lower execution time does not imply a lower energy consumption, which was the case in six out of nine benchmarks.[21]

[22] found that a larger amount of cores in the execution pool resulted in a lower running time and energy consumption and concluded that parallelism could help reduce energy consumption for genetic algorithms.

In [23], four different language constructs used to implement parallelism in C# were tested by executing a set of micro- and macrobenchmarks on a different number of threads. It was found that workload size greatly influenced execution time and energy efficiency. A limit was found for a sequential program, after which changing it to executing in parallel would be beneficial. The findings remained consistent between micro- and macrobenchmarks. However, the impact was less significant for the macrobenchmarks due to a higher total energy consumption.

2.3 Compilers

[24] sought to optimize energy efficiency and performance across various C++ compilers. They conducted comparisons using different coding styles and microbenchmarks on compilers such as MinGW GCC, Cygwin GCC, Borland C++, and Visual C++. Energy measurements were collected via the Windows Performance Analyzer (WPA). The compilers were used with their default settings, and no optimizations options were used. [24] found that when choosing a compiler and coding style, energy reduction depended on the specification of the target machine and the individual application. Based on the benchmark used, the lowest execution time was achieved with the Borland compiler and the lowest energy consumption was observed with the Visual C++ compiler. When considering the coding styles, the study found that separating IO and CPU operations and interrupting CPU-intensive instructions with sleep statements decreased energy consumption.

2.4 Asymmetric Multicore Processors

Asymmetric Multicore Processors are CPUs where not all cores are created to be identical, one example being the combination of P- and E-cores seen in Intel's Alder Lake and Raptor Lake. The use of P- and E-cores when deciding which cores to run a thread on are handled by the OS and assisted by Intel's Thread Director (ITD). Support for ITD on Linux was introduced in [25], where SPEC benchmarks were executed to evaluate the estimated Speedup Factor (SP) against the observed SF. SF represents the relative benefit a thread receives from running on a P-core. [25] found that 99.9% of class readings from the benchmark performed equally on P- and E-cores or preferred P-cores. The experiment indicated that the ITD overestimated the SF of using the P-cores for many threads and underestimated it for some. Overall, it was found that the estimated SF had a low correlation coefficient (< 0.1) with the observed values. Furthermore, a performance monitoring counter (PMC) based prediction model was trained, where the model outperformed ITD but still produced errors. However, the correlation coefficient was higher at (> 0.8). The study implemented support for the IDT in different Linux scheduling algorithms and compared the results from using the IDT and the PMC-based model. [25] found that the PMC-based model provided superior SF predictions compared to ITD. Official support for ITD has since been released.

3 Background

In the following section, different technologies used for the experiment are introduced.

3.1 CPU States

CPU-states (C-state) manage a system's energy consumption during different operational conditions. On a CPU, each core has its own state, which dictates how much it is shut down to conserve power. The `C0` state represents the normal operation of a core under load where the number of states varies between CPUs, and the number of supported states varies between motherboards.[26, 27] The CPU used in [10] had 10 states, where states higher than C0 represents an increasingly shut down core, and the highest states will mean the core is almost inactive.[10]. The C-states can greatly impact the energy consumption of the benchmarks, especially the idle case, as was found in [10].

3.2 Performance and Efficiency cores

For the CPU architecture x86, the core layout has historically been comprised of identical cores. However, the ARM architecture introduced the big.LITTLE layout in 2011[28]. big.LITTLE is an architecture utilizing two types of cores, including a set for maximum energy efficiency and a set for maximum performance.[29]. Intel introduced a hybrid architecture in 2021[30] similar to big.LITTLE, codenamed Alder Lake. Alder Lake has two types of cores: P-cores and E-cores, each optimized for different tasks. P-cores are standard CPU cores that focus on maximizing performance, and E-cores are designed to maximize performance per watt and are intended to handle smaller non-time critical jobs, such as background services[31].

```
void ExecuteWithAffinity(string path)
{
    var process = new Process();
    process.StartInfo.FileName = path
    process.Start();

    // Set affinity for the process
    process.ProcessorAffinity =
        new IntPtr(0b0000_0011)
}
```

Listing 1: An example of how to set affinity for a process in C#

3.3 Processor Affinity

Processor affinity allows applications to bind or unbind a process to a specific set of cores. When a process is pinned to a core, the OS ensures that the process only executes on the assigned core(s) each time it is scheduled.[32] When setting the affinity for a process in C#, it is done through a bitmask, where each bit represents a CPU core. An example was illustrated in Listing **1**, where the process was allowed to execute on core #0 and #1.

3.4 Scheduling Priority

When executing threads on Windows, they are scheduled based on their scheduling priority, which is decided based on the priority class of the process and the priority level of the thread. The priority class can be either `IDLE`, `BELOW NORMAL` `NORMAL`, `ABOVE NORMAL`, `HIGH` or `REALTIME`, where the default is `NORMAL`. It is noted that `HIGH` priority should be used carefully, as other threads in the system will not get any processor time while that process runs. If a process needs `HIGH` priority, it is recommended to raise the priority class temporarily. The `REALTIME` priority class should only be used for applications that ”talk” to hardware directly, as this class will interrupt threads managing mouse input, keyboard inputs, etc.[33]

```
void ExecuteWithPriority(string path)
{
  var process = new Process();
  process.StartInfo.FileName = path
  process.Start();

  // Set priority class for process
  process.PriorityClass =
    ProcessPriorityClass.High;

   // Set priority level for threads
  foreach (var t in process.Threads)
  {
    thread.PriorityLevel =
      ThreadPriorityLevel.Highest;
  }
}
```

Listing 2: An example of how to set priorities for a process in C#

The priority level can be either `IDLE`, `LOWEST`, `BELOW NORMAL`, `NORMAL`, `ABOVE NORMAL`, `HIGHEST` and `TIME CRITICAL`, where the default is `NORMAL`. A typical strategy is to increase the level of the input threads for applications to ensure they are responsive and to decrease the level for background processes, meaning they can be interrupted as needed.[33]

The scheduling priority is assigned to each thread as a value from zero to 31, where this value is called the base priority. The base priority is decided using the thread priority level and the priority class, where a table showing the scheduling priority given these two parameters can be found in [33]. The idea of having different priorities is to treat threads with the same priority equally by assigning time slices to each thread in a round-robin fashion, starting with the highest priority.

When setting scheduling priority, the priority class is supported for Windows and Linux, while the priority level is only supported for Windows. An example of how both priorities are set for a process and its threads can be seen in Listing **2**.

3.5 Open Multi-Processing

Open Multi-Processing (OpenMP) is a parallel programming API consisting of a set of compiler directives and runtime library routines, supporting multiple OSs and compilers.[34] The directives provide a method to specify parallelism among multiple threads of execution within a single program without having to deal with low-level details, while the library provides mechanisms for managing threads and data synchronization.[34]

The directives provide a method to specify parallelism among multiple threads of execution within a single program without having to deal with low-level details, while the library provides mechanisms for managing threads and data synchronization.[34]

When executing using OpenMP, the parallel mode used is the Fork-Join Execution Model. This model begins with executing the program with a single thread called the master thread. This thread is executed serially until parallel regions are encountered, in which case a thread group is created consisting of the master thread and additional worker threads. After splitting up, each thread will execute until an implicit barrier at the end of the parallel region. When all threads have reached this barrier, only the master thread continues.[34]

```
#pragma omp directive-name [
    clause[ [,] clause]...
    ]
```

Listing 3: The basic format of OpenMP directive in C/C++

The basic format of using OpenMP can be seen in Listing 3. By default, the parallel regions are executed using the number of present threads in the system, but this can also be specified using `num_threads(x)`, where `x` represents the number of threads.[34]

3.6 Apparent Energy

In a circuit, two types of energy can be identified: active energy, which performs useful work, and reactive energy, which does not. The combination of these two energies is called apparent energy, which is what is measured by hardware-based measuring instruments. Reactive energy occurs because of inductive or capacitive loads in a circuit, resulting in an energy loss not utilized by the circuit[35]. The ratio between active and reactive energy is known as the power factor[35].

4 Experimental Setup

In this section, we present a detailed description of the equipment, benchmarks, and procedures used in the study.

4.1 Measuring Instruments

The measuring instruments used in this work are based on those used in [10], with a few additions. The new additions will be introduced in more detail, while the others will be briefly mentioned, with more detail available in [10].

Intel's Running Average Power Limit (RAPL): is a Mac and Linux exclusive software-based measuring instrument frequently used in the literature.[10] RAPL uses model-specific-registers (MSRs) and Hardware performance counters to calculate the energy consumption of the CPU. The MSRs used by RAPL are those for the power domains PKG, DRAM, PP0, and PP1, covered in [10].

In [10] RAPL was found to have a correlation of 0.81 with the ground truth.[10]

Intel Power Gadget (IPG): is a Windows and Mac exclusive software tool created by Intel, which can estimate the energy consumption of Intel processors. IPG uses the same hardware counters and MSRs as RAPL[36], and is therefore expected to have similar measurements to RAPL, which was found to be the case in [10]. [10] found that IPG had a correlation of 0.78 with the ground truth on Windows and a correlation of 0.83 with RAPL on Linux.[10]

Libre Hardware Monitor (LHM): is a fork of Open Hardware Monitor supported on Windows and Linux.[37] LHM is open source and uses the same hardware counters and MSRs as RAPL and IPG. In [10], LHM on Windows was found to have a correlation of 0.76 with the ground truth on Windows and a correlation of 0.85 with IPG.

MN60 AC Current Clamp (Clamp): is a current clamp connected to the phase of the wire going into the power supply unit (PSU), which serves as the ground truth. The clamp is connected to an Analog Discovery 2, where the Analog Discovery 2 is connected to a Raspberry Pi 4 to measure and log measurements continuously.[10] The intrinsic error is reported to be 2%[38].

CloudFree EU smart Plug (Plug): is a smart plug with energy measuring capabilities, used as an alternative lower-priced, easier-to-use hardware-based measuring instrument. The accuracy and sampling rate of the plug is unknown.[39]

Scaphandre (SCAP): is a monitoring agent able to measure energy consumption.[40] SCAP is designed for Linux and uses RAPL, and can also measure the energy consumption of some virtual machines. SCAP can also be used on Windows, as a kernel driver exists which allows SCAP to read RAPL measurements on Windows.[41]. The Windows version of SCAP can report the energy consumption of the power domain PKG using the MSRs. SCAP can also estimate the energy consumption for individual processes by storing CPU usage statistics alongside the energy counter values and then calculating the ratio of CPU time for each Process ID (PID). Using the calculated ratio, SCAP estimates the subset of energy consumption belonging to a specific PID. In this work, both the performance of SCAP and SCAPs ability to isolate the energy of a process will be used, where the latter will be referenced as SCAPI.

4.2 Dynamic Energy Consumption

Dynamic Energy Consumption (DEC) represented a way to isolate the energy consumption of a process and was utilized in [10, 42]. DEC was used to compare software- and hardware-based measuring instruments, where the former measures energy consumption of the CPU only and the latter the entire DUT. A brief explanation of DEC based on [42] is given:

$$E_D = E_T - (P_S * T_E) \tag{1}$$

In Equation (1) E_D is the DEC, E_T is the total energy consumption of the system, P_S is the energy consumption when the system is idle, and T_E is the execution time of the program execution. E_D thus represents the energy consumption of the running process only, as the idle energy consumption is subtracted.[42]

4.3 Statistical Methods

This section presents the statistical method used to analyze the results. The selection of the various statistical methods is partly inspired by Koedijk et al.[43] and Fahad et al.[42].

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.[45] Because of this, the Shapiro-Wilk test will be used to check for normal distributions. The formula for the Shapiro-Wilk test is:

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

The null hypnosis H_0 for the test is[45]:

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[46]. If the p_{Value} is less than α, it can reject H_0. Using the p_{value} is commonplace in statistical testing[47], 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:

$$t_{Value} = \frac{|\bar{x_1} - \bar{x_2}|}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{3}$$

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.[48]

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:

$$U_1 = n_1 * n_2 + \frac{n_1 * (n_1 + 1)}{2} - T_1 \tag{4}$$

$$U_2 = n_1 * n_2 + \frac{n_2 * (n_2 + 1)}{2} - T_2 \tag{5}$$

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.

$$\varphi U = \frac{n_1 * n_2}{2} \tag{6}$$

$$\sigma U = \sqrt{\frac{n_1 * n_2 * (n_1 + n_2 + 1)}{12}} \tag{7}$$

Then calculate the value z.

$$z = \frac{U - \varphi U}{\sigma U} \tag{8}$$

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: It comes in two different variations, the one-sample and two-sample Kolmogorov-Smirnov test[49]. 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[49].

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

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 $D_{n,m} > c(\alpha)\sqrt{en}$[49]

Correlation

Correlation is a statistical measurement informing about the relationship between two samples[50]. 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[51]. 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:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \tag{9}$$

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[52]. 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 [53]. 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:

$$\frac{C-D}{C+D} \tag{10}$$

C is concordant pairs and D is discordant. A concordant pair is ordered, while discordant is disordered pairs[54]. To evaluate the correlations, the scale presented by Guildford in [55, p. 219] can be used.

Values	Label
$< .20$	Almost negligible correlation
$.20 - .40$	Low correlation
$.40 - .70$	Moderate correlation
$.70 - .90$	High Correlation
$.90 - 1$	Very high correlation

Table 1: The values for the scale presented by Guildford in [55, p. 219]

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

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[8, 43, 56]. Another method is to base the number of measurements on how many times the experiment can be run within a time-frame[57]. In this paper, 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.[58–60], 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 Equation (11)[3] is for categorical data when the population is large[60]:

$$n_0 = \frac{Z^2 * p * q}{e^2} \tag{11}$$

Where

[3]The terminology used by Cochran is different to the one utilized in this paper

- 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.[59, 60] The equation is shown in Equation (12).

$$n = \frac{n_0}{1+\frac{(n_0-1)}{N}} \tag{12}$$

Where n is the new sample size and N population size.

Furthermore, there is a simplified formula called Yamane's formula, shown in Equation (13)[60].

$$n = \frac{N}{1-N(e)^2} \tag{13}$$

Lastly, there is another version of Cochran's formula shown in Equation (14), Which is for continuous data.

$$n_0 = \frac{Z^2 * \sigma^2}{e^2} \tag{14}$$

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.

It should still be considered if categorical variables will play an important role in the data analysis, then Equation (11) should be used[59]. For our data Equation (14) seems to most accurately fit the description and is therefore chosen. The correction formula Equation (12) 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.[59] 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[59]. 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[43, 59], 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[61]. 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.

Workstation 1 (DUT 1)	
Processor:	Intel i9-9900K
Memory:	DDR4 16GB
Disk:	Samsung MZVLB512HAJQ
Motherboard:	ROG STRIX Z390 -F GAMING
PSU:	Corsair TX850M 80+ Gold
Ubuntu:	22.04.2 LTS
Linux kernel:	5.19.0-35-generic
Windows 11:	10.0.22621 Build 2262

Table 2: The specifications for DUT 1

4.4 Device Under Tests

Two workstations were used in the experiments. The DUTs were chosen to compare CPUs with and without P- and E-cores. Upon setup of the DUTs, they were updated to have the same version of Windows and Linux, and in Tables **2** and **3**, the specifications of the two workstations, referred to as DUT 1 and DUT 2, can be seen.

Workstation 2 (DUT 2)	
Processor:	Intel i5-13400
Memory:	DDR4 32GB
Disk:	Kingston SNV2S2000G
Motherboard:	ASRock H610M-HVS
PSU:	Cougar GEX 80+ Gold
Ubuntu:	22.04.2 LTS
Linux kernel:	5.19.0-35-generic
Windows 11:	10.0.22621 Build 22621

Table 3: The specifications for DUT 2

4.5 Compilers

This section introduced the various C++ compilers used in the first experiment. MSVC and MinGW were included as [24] found those to exhibit the lowest energy consumption, and Intel's oneApi and Clang were included as both could be found on lists of the most popular C++ compilers[62–64]. The versions of the compilers were illustrated in Table **4**

C++ Compilers	
Name	Version
Clang	15.0.0
MinGW	12.2.0
Intel OneAPI C++	2023.0.0.20221201
MSVC	19.34.31942

Table 4: C++ Compilers

Clang: is an open-source compiler building on the LLVM optimizer and code generator and is available for both Windows and Linux[65]

Minimalist GNU for Windows (MinGW): is an open-source project which provides tools for compiling code using the GCC toolchain on Windows. It includes a port of GCC. Additionally, MinGW can be cross-hosted on Linux.[66]

Intel's oneAPI C++ (oneAPI): is a suite of libraries and tools to simplify development across different hardware. One of these tools is the C++ compiler, which implements SYCL, an evolution of C++ for heterogeneous computing. It is available for both Windows and Linux.[67]

Microsoft Visual C++ (MSVC): comprises a set of libraries and tools designed to assist developers in building high-performance code. One of the included tools is a C++ compiler, which is only available for Windows[68].

4.6 Benchmarks

This work employed microbenchmarks and macrobenchmarks to asses the measuring instruments and analyze the energy consumption. The introduction and the rationale behind why the specific benchmarks were selected can be found in this section.

Microbenchmarks		
Name	Parameter	Focus
NBody (NB)	$50 * 10^6$	single core
Spectra-Norm (SN)	5.500	single core
Mandelbrot (MB)	16.000	multi core
Fannkuch-Redux (FR)	12	multi core

Table 5: Microbenchmarks

Microbenchmarks: are small, focused benchmarks testing only a specific operation, algorithm, or piece of code. They are useful for measuring the performance of some particular code precisely while minimizing the impact of other factors, but they may not accurately represent overall performance.[69]

The microbenchmarks used in this work were from the Computer Language Benchmark Game [4]. The selected benchmarks included single- and multi-threaded microbenchmarks compatible with the compilers and OSs used in this work. Certain libraries, such as `<sched.h>`, were unavailable on Windows, limiting the pool of compatible microbenchmarks. The chosen microbenchmark benchmarks and their abbreviation were presented in Table **5**, where the parameters were those specified by the Computer Language Benchmark Game. During compilation, the only parameter given is `-openmp` for the multi-core benchmarks, ensuring optimization for all cores of the DUT.

Macrobenchmarks	
Name	Version
3D Mark (3DM)	2.26.8092
PC Mark 10 (PCM)	5.61.1173.0

Table 6: Macrobenchmarks

Macrobenchmarks: are large-scale benchmarks testing the performance of an entire application or system. Macrobenchmarks provide a more comprehensive overview of how the system performs in real-world scenarios and are more suitable for understanding the overall performance of an application or system.[69] Application-level benchmarks are a type of marcobenchmarks testing an application, which provides a more realistic benchmark scenario.

The two macrobenchmarks used in this work were made by UL Solutions. One was 3DMark (3DM), a set of benchmarks for scoring GPUs and CPUs based on gaming performance. From 3DM, the CPU Profiler benchmark was used, as this work focuses on the energy consumption of the CPU. The CPU Profile benchmarks executes a 3D graphic, but the main component of the workloads is from a boids flocking behavior simulation.[70]. The other macrobenchmarks was PCMark 10 (PCM), a benchmark meant to test various tasks seen at a workplace. PCM has three test groups, including web browsing, video conferencing, working in spreadsheets, and photo editing. The full list can be seen in Tables **13** and **14**.[71] The versions of both macrobenchmarks can be seen in Table **6**.

Background Processes
Name
searchapp
runtimebroker
phoneexperiencehost
TextInputHost
SystemSettings
SkypeBackgroundHost
SkypeApp
Microsoft.Photos
GitHubDesktop
OneDrive
msedge
AsusDownLoadLicense
AsusUpdateCheck

Table 7: Background Processes

[4]https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

4.7 Background Processes

Steps were taken to limit background processes on Windows. When the DUTs were set up, all startup processes in the Task Manager and non-Microsoft and Intel-related services found in the System Configuration were disabled

During runtime, different background processes were also stopped. These processes were found by looking at the running processes using command `Get-Process`. A list of processes was found which are killed using the `Stop-Process` command before running the experiments. The list can be found in Table **7**.

5 Experiments

In the following section, the conducted experiments were analyzed. All experiments were carried out on the framework presented in Appendix **B**, with the results stored in the database introduced in Appendix **C**. During the experiments, the `ProcessPriorityClass` for the measuring instrument, framework, and benchmarks were set to `High` unless specified otherwise. In addition, suggestions made by [11, 23] were followed, meaning C-states, Turbo Boost, and hyperthreading was disabled. On Linux, Wi-Fi was disabled when benchmarks were running, but no background processes were stopped as [11] found it had no noticeable effect. No analysis on the effect of background processes on Windows was found, which is why the background processes presented in Section **4.7** was disabled in addition to the Wi-Fi. The benchmarks were executed right after each other in all experiments, as [11] did not find any effect of either restarts or sleep between executions. When using Cochran's formula, a confidence level of 95% and a margin of error of 0.03% was used, as [10] found that to be fitting.

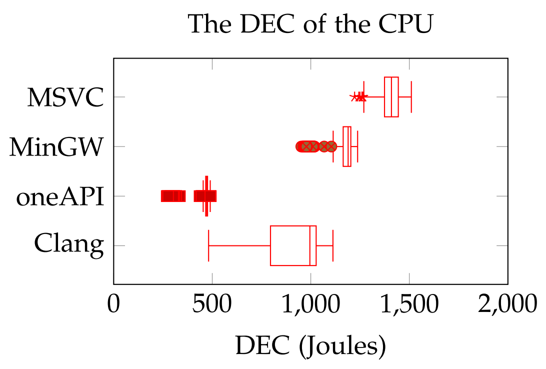


Figure 1: CPU measurements by IPG on DUT 1 for test case(s) FR

5.1 Experiment One

The first experiment investigated RQ **1** and employed both multi-core microbenchmarks presented in subsection **4.6**, and the measurements were performed using IPG on DUT 1. IPG was chosen based on its performance in [10]. This experiment was conducted based on a hypothesis that the different compilers would produce assembly code with varying energy consumption and execution time, as was found in [24].

Compiler Initial Measurements: The initial measurements were taken to gain insight into the number of measurements required by computing Cochran's formula on the results from the initial measurements. After that, additional measurements were made if required. The number of measurements chosen for the initial measurements was 30, as the central limit theorem suggests that a sample size of at least 30 is usually sufficient to ensure that the sampling distribution of the sample mean approximates normality, regardless of the underlying distribution of the population[72]. After the initial 30 measurements, Cochran's formula was applied to the measurements, and the required measurements were illustrated in Table **8**, where it was evident that the required samples varied between compilers and benchmarks.

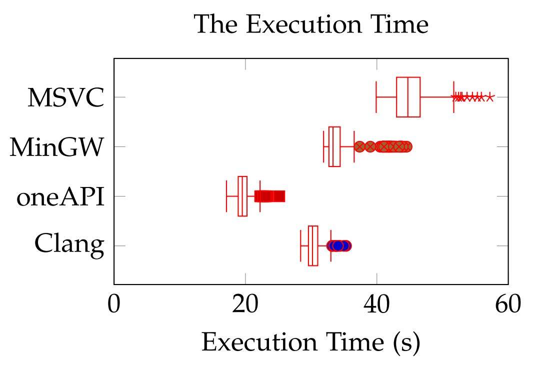


Figure 2: Execution time measurements by IPG on DUT 1 for test case(s) FR

When the benchmarks were analyzed, it was found that MB deviated less than FR, with MB requiring as little as 3 measurements with MinGW, while FR required up to 61.086 samples with Clang. When the compilers were analyzed, oneAPI had the lowest required samples for FR but the highest for MB. 550 additional measurements were conducted for the next step.

Initial Measurements		
Name	FR	MB
Clang	61.086	40
MinGW	1.644	3
oneAPI	550	222
MSVC	2.994	10

Table 8: The required samples to gain confidence in the measurements made by IPG on Windows

Compiler Results: After 550 measurements were obtained, the reported measurements required by Cochran's formula still indicated that MSVC, MinGW, and Clang needed more measurements compared to oneAPI. Between the different compilers, Clang stood out where 61.086 measurements were required. Because this number was much higher than other compilers, additional measurements were taken for this compiler. After 10.000 measurements, Cochran's formula indicated that 1.289 measurements were required, which is more in line with other compilers.

When looking at the results for FR in Figures **1** and **2**, and for MB in Appendix **E**, oneAPI had the lowest DEC and execution time for both benchmarks and Clang deviated the most in Figure **1**.

The first experiment found that the different compilers had a considerable impact on the DEC, execution time, and on how many measurements were required. Ultimately, oneAPI had the lowest DEC and execution time and was therefore used in the following experiment.

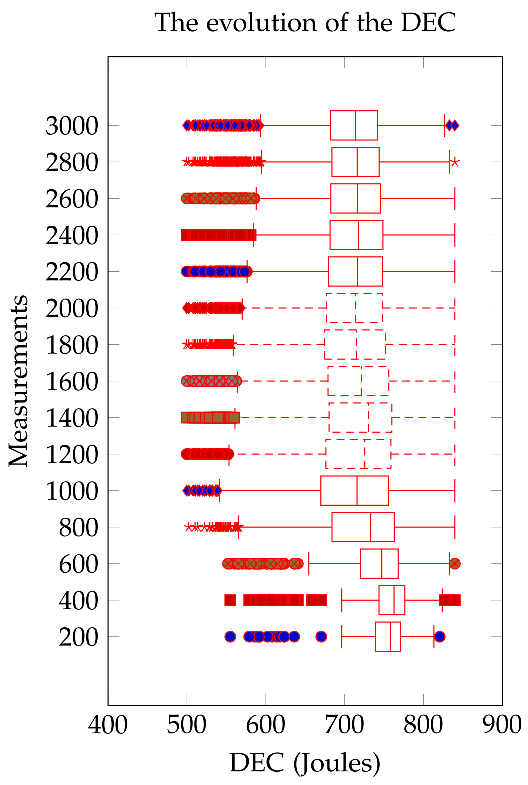


Figure 3: A visual representation of how the DEC evolved as more measurements were made by Clamp on DUT 2 for benchmark MB

5.2 Experiment Two

The second experiment investigated RQ **2** to identify the best measuring instrument on Windows for this study. This decision was based on a combination of factors, including correlation to the ground truth, ease of use, and availability.

A couple of changes were made in the experimental setup for experiment two. Firstly, due to some issues with SCAP and SCAPI, where the sampling rate significantly decreased when the DUT was under full load, the process priority class of the benchmarks was set to `Normal`. Secondly, due to an execution time of less than a second for MB when compiled with oneAPI, MB's parameter was changed from 16.000 to 64.000, which increased the execution time to 14 seconds. This avoided a scenario where the Plug only had one data point per measurement. FR was executed 550 times for this experiment, while MB was executed 222 times, based on Table **8**.

Measuring Instrument Initial Measurements: The required number of measurements for this experiment, found by applying Cochran's formula to the measurements, was be found in Appendix **H**. From Appendix **H**, it was found that the Clamp required more measurements than other measuring instruments, so a more in-depth analysis was conducted. This analysis was made by performing 3.000 MB measurements by the Clamp on DUT 2, where the result from this experiment was illustrated in Figure 3.

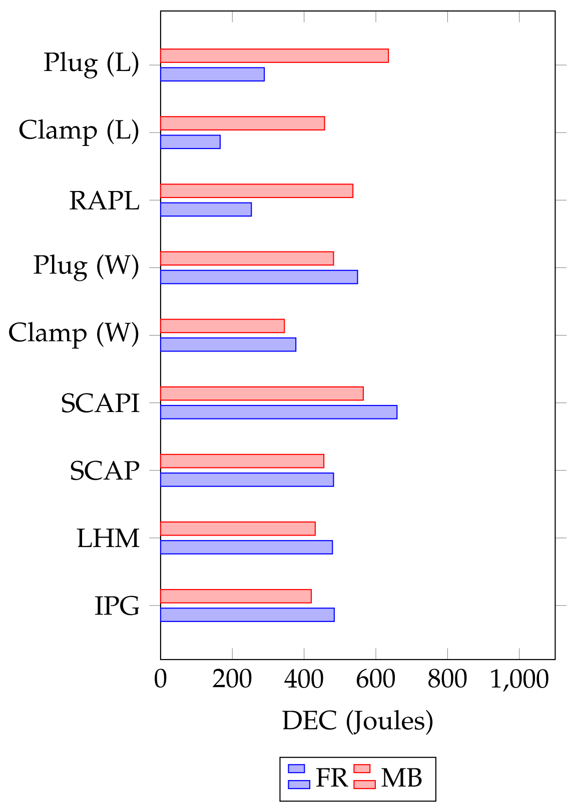
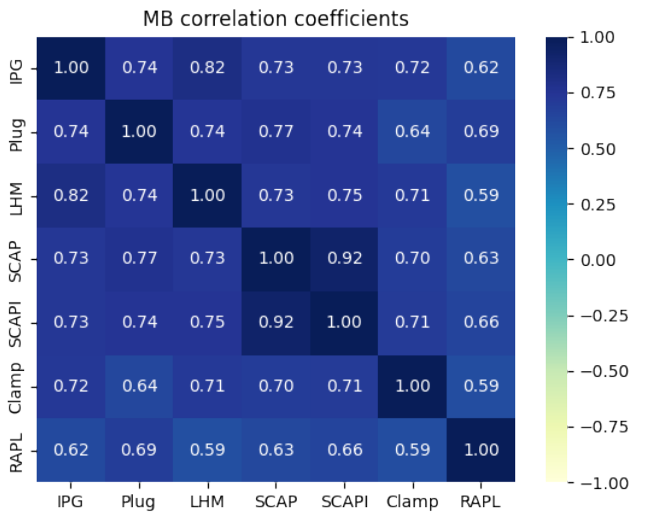


Figure 4: The DEC for DUT 1, where both benchmarks were compiled on oneAPI

In Figure 3, the evolution of the DEC was illustrated, where the DEC was found to decrease by 5.84% between 200 and 3.000 measurements and by 0.3% between 2.800 and 3.000 measurements. A pattern was observed, where the DEC decreased between 200 and 1.000 measurements, after which the DEC increased until measurement 1.400 by 2%, and then decreased and converged. The DEC at 1.000 measurements was 0.29% from the DEC at 3.000, and due to the time required to run the additional 2.000 measurements, the maximum amount of measurement was capped at 1.000 for this experiment. After 3.000 measurements, the number of measurements required was 15.137. This number is higher than other measuring instruments, which will be covered further in the discussion. In Appendix I, a graph was illustrated, showing how many measurements Cochran's formula indicated would be required as the number of measurements increased.

Measuring Instrument Results: This experiment's results were presented in Figure 4 for DUT 1, and for DUT 2, the results were presented in Appendix F. In Figure 4, MB consumed less energy than FR for Windows, whereas the opposite was the case for Linux and for DUT 2 in most cases. When comparing the different software measuring instruments for Windows, SCAP, IPG, and LHM were in all cases within 25 joules of each other, where IPG reported the lowest DEC and SCAPI reported the highest DEC. When the hardware measuring instruments were compared, the Plug reported a higher DEC than the Clamp in all cases. Between OSs, Linux reported a lower DEC for FR but a higher DEC for MB for both the Clamp and Plug on DUT 2 but not DUT 1. When comparing RAPL to the Clamp, it overreports in all cases, which was also found on Windows for all measuring instruments, except for FR on DUT 2.



	IPG	Plug	LHM	SCAP	SCAPI	Clamp	RAPL
IPG	1.00	0.74	0.82	0.73	0.73	0.72	0.62
Plug	0.74	1.00	0.74	0.77	0.74	0.64	0.69
LHM	0.82	0.74	1.00	0.73	0.75	0.71	0.59
SCAP	0.73	0.77	0.73	1.00	0.92	0.70	0.63
SCAPI	0.73	0.74	0.75	0.92	1.00	0.71	0.66
Clamp	0.72	0.64	0.71	0.70	0.71	1.00	0.59
RAPL	0.62	0.69	0.59	0.63	0.66	0.59	1.00

Figure 5: Heatmap showing the correlation coefficient between all of the measurement instruments for MB on dut 1

Based on the results presented in this section, it was difficult to find conclusions that were true in all cases across all measuring instruments, DUTs, OSs, and benchmarks. This is similar to what was found in [11], where conclusions from other work could not be proven on their setup.

When the statistical methods from subsection 4.3 were applied to the results, it showed that they did not follow a normal distribution and did not come from the same distribution, which was also found in [10, 43].

The correlation between the measuring instruments for MB on DUT 1 was illustrated in Figure 5, where it was found that all software-based measuring instruments had a moderate to high correlation between 0.59 - 0.72 to the Clamp when assessed with the Guildford Scale, where the Plug also had a moderate correlation of 0.64. The correlations were higher for FR than MB, but still within the same categories of moderate or high correlation, as shown in Appendix G,. Given the similar correlation between the different software-based measuring instruments, this was as expected, as they all used the same hardware counters and MSRs to monitor the energy consumption, as presented in subsection **4.1**. When choosing the best measuring instrument, SCAP and SCAPI were excluded despite a high correlation given a low sample rate and a tedious setup process. Between IPG and LHM, the performance was equal, where IPG was more correlated on MB and LHM on FR. The choice ended up being on IPG, given a better user experience.

SN measurements on DUT 2			
Metric	E-core	P-core	Difference
Execution time	58.96 s	13.96 s	−76.32%
DEC	31.87 j	26.49 j	−16.88%
DEC per second	0.53 w	1.88 w	+254.71%

Table 9: The average performance difference between P- and E-cores on DUT 2, SN

5.3 Experiment Three

The third experiment investigated RQs **3** and **4**, by analyzing what benefit macrobenchmarks gained from additional allocated cores by executing PCM and 3DM on an increasing number of cores, measured by IPG only. Before this was done, the per-core performance of both CPUs was analyzed, where the single-core benchmarks introduced in subsection **4.6** were used. This allowed a comparison between the energy consumption of the P- and E-cores on DUT 2 and the P-cores on DUT 1. When the measurements were performed, the limit of 1.000 measurements set in subsection **5.2** was still used.

Per-Core Initial Measurements: An initial 250 measurements were made for each benchmark on each core, on both DUTs. Afterward, Cochran's formula was applied to the result to determine if more measurements were required, as was presented in Appendix **K**.

NB measurements on DUT 2			
Metric	E-core	P-core	Difference
Execution time	29.59 s	11.54 s	−60.96%
DEC	19.04 j	26.00 j	+36.55%
DEC per second	0.66 w	2.23 w	+237.87%

Table 10: The average performance difference between P- and E-cores on DUT 2, NB

Per-Core Results: For the per-core results, the analysis was based primarily on DUT 2, with results from DUT 1 presented in Appendix **J**. When comparing the difference in performance between P- and E-cores, the results were illustrated in Table **9** for SN, and in Table **10** for NB. for both SN and NB, E-cores were observed to have a higher execution time and lower DEC per second compared to P-cores, where the DEC was higher for P-cores on NB, while it was lower for SN.

Performance Between Cores				
DUT	Core	Benchmark	Average	STD
1		SN	79.16 j	1.75 j
1		NB	47.88 j	5.67 j
2	P	SN	26.00 j	0.27 j
2	E	SN	19.02 j	0.78 j
2	P	NB	26.49 j	0.32 j
2	E	NB	31.87 j	1.30 j

Table 11: The performance difference between cores of the same type

When comparing how much performance could differ between cores of the same type on the same CPU, Table **11** illustrated the average energy consumption and how much it deviated. Table **11** illustrated DUT 1 deviates more, with the highest deviation for NB, while the lowest was for DUT 2, on P-cores for SN.

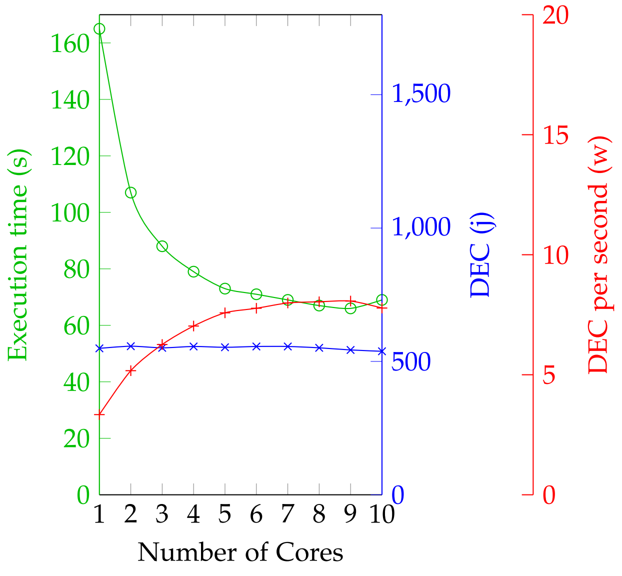


Figure 6: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to 3DM on DUT 2

Macrobenchmark Initial Measurements: Following the analysis of the per-core performance, the two macrobenchmarks introduced in subsection **4.6** were executed on an increasing amount of cores, starting from the most efficient one. An initial 30 measurements were made, as the per-core experiment showed how 250 were too many measurements for DUT 2, as illustrated in Appendix **K**. The initial idea was to start at one core, which was done for 3DM on both DUTs and PCM on DUT 1. On DUT 2, PCM could not execute web browsing on a single core and could not execute spreadsheet and photo editing on any amount of cores for unknown reasons. Because of this, DUT 2 started at two cores to include web browsing. For DUT 1, web browsing could not execute, so this scenario was excluded for this DUT. No solution was found to these issues. After the initial 30 measurements, Cochran's formula was applied to ensure enough measurements were made. The number of required measurements was presented in Appendix **L**.

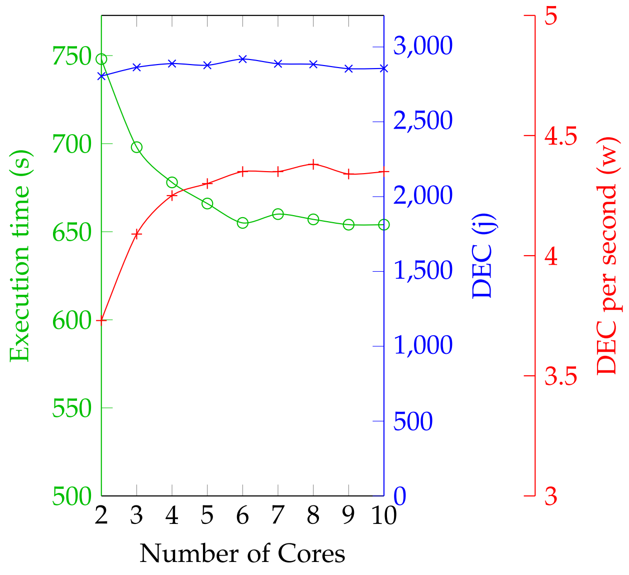


Figure 7: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to PCM on DUT 2. Note that the x- and y- axis does not start at zero.

Macrobenchmark Results: The results for DUT 2 were illustrated in Figure **6** and Figure **7** for 3DM and PCM, respectively, and for DUT 1 in Appendix **M**.

A similar observation was made for both DUTs and macrobenchmarks, where the execution time decreased and the DEC per second increased as more cores were allocated, while the DEC remained the same. A difference between 3DM and PCM was how the execution time decreased more for 3DM than for PCM. This is partly due to PCM having more scenarios only utilizing a single thread, but also because several of the scenarios in PCM had constant execution times, e.g., a video took the same amount of time regardless of the processing power, which means that only parts of the benchmarks could benefit from the additional allocated cores. For 3DM, the benchmark itself was embarrassingly parallel, but the measurements included a startup and shutdown period, which was not parallel. An additional analysis of the energy consumption over time by PCM and 3DM was analyzed more in-depth in Appendix **O**.

P vs. E Initial Measurements: when executing macrobenchmarks on an increasing number of cores, it was difficult to illustrate how P cores perform against E cores. This experiment therefore explores how P and E cores compares, when executing macrobenchmarks. This was achieved by running PCM on four cores, either with four P cores (4P), four E cores (4E) or two of each (2P2E). PCM was chosen as it represented a usecase with low CPU usage, where some jobs were of lower priority, which could see the DEC reduced, when E cores were available. For this experiment, 30 initial measurements were made, and additional were made after Cochran's formula was applied to the results, if required. The required measurements were presented in Appendix **M**.

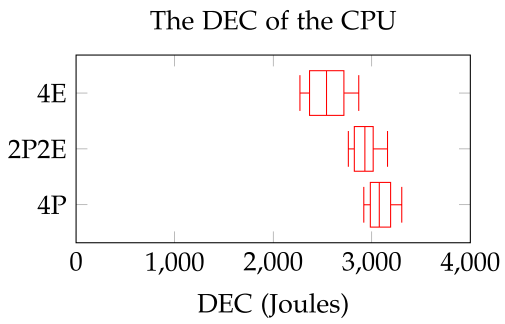


Figure 8: CPU measurements by IPG on DUT 2 for test case(s) PCM

P vs. E Results: The results for the execution time and DEC are illustrated in Figures **8** and **9**, while the DEC per second was presented in Appendix **N**. 4E used the least energy for the DEC, while 4P and 2P2E used 17.40% and 13.28% more energy, respectively. For execution time, the order was the opposite, where 4P had the lowest execution time, where 2P2E and 4E executed 3.74% and 29.52% slower, respectively. This illustrated a use case where E-cores had a lower energy consumption but a higher execution time.

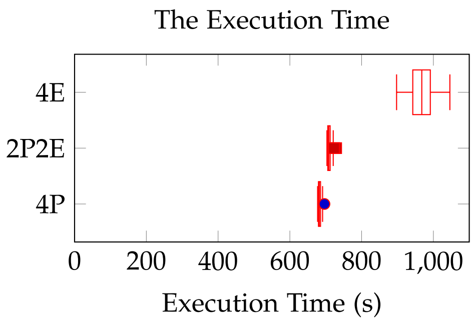


Figure 9: Execution time measurements by IPG on DUT 2 for test case(s) PCM

Estimated Speedup and Actual Speedup: An additional analysis was conducted by comparing the actual speedup when executing on more cores against an estimate provided by Amdahl's law. This analysis was conducted in Appendix **R**, where it was found that the estimation followed the actual speedup closely until the E-cores were used. The actual speedup showed that the E-cores do not contribute to a speedup of the 3DM benchmark.

6 Discussion

In the following section, results from Section **5** are discussed.

6.1 Deviating Results

Cochran's formula was used to determine how many measurements were required in order to gain confidence in the results. Using Cochran's formula, the amount of required measurements deviated between benchmarks, measuring instrument, DUTs and even cores on the same CPU.

The required number of measurements deviated between cores on the same CPU as a result of the variability in the fabrication process, where the exact characteristics of each core can change, despite being assembled in the same way.[73] When comparing cores of the same type, it was found that a DEC of 47 j could have a standard deviation of up to 5.67 j among cores of the same type.

When comparing the required amount of measurements between DUTs, DUT 2 required less measurements than DUT 1. The cause of this could be either software or hardware based. When setting up both DUTs, effort was put into ensuring the DUTs had software with the same versions. Both DUTs executed on a fresh install of Windows, had the same software downloaded and the same background processes were disabled, so it seemed unlikely this was the cause. When comparing hardware, the two DUTs were from different generations of intel CPUs, released five years apart. DUT 1 was the older of the two, but no evidence of newer CPUs deviating more was found by [11]. It was however found in [11] that a lower TDP could result in a lower energy variation, and DUT 2 had a lower TDP of 65 W opposed to the TDP of 95W found on DUT 1, which could explain the difference[74].

6.2 C++ Benchmarks Analysis

In the first experiment presented in subsection **5.1**, different C++ compilers were compared and it was found that the energy consumption, execution time and measurements required deviated between compilers. For the oneAPI, a low runtime was observed for the MB benchmark, compared to the other compilers. This could be because the benchmark was removed as dead code by the compiler, which is why an analysis was conducted in Appendix **P**, where the instructions from the decompiled executables were compared between MinGW and oneAPI. The analysis showed that the benchmark was not removed as dead code, but rather that oneAPI achieved a better performance as it used AVX instructions, Advanced Vector extensions to perform calculations in parallel, where MinGW used general purpose registers more, in a combination with the C++ Standard Library.

6.3 Energy usage trends

The trend where the DEC decreased as more measurements were made, illustrated in subsection **5.2**, was found for both the Clamp and Plug, which indicated that it was not caused by faulty measurements. The trend was however not observed on any software-based measuring instruments, which is why the observed reduction in energy consumption may be caused by changes in the reactive energy consumption.

In order to test if reactive energy consumption caused the trends of a lower DEC as more measurements were made, an analysis was conducted in subsection **Q**. This analysis was conducted based on energy measurements from both DUTs and OSs, when the DUT was on idle, where the energy consumption during working hours and non-working hours were compared. In the end, the analysis found that there was a difference between the energy consumption during working hours and non-working hours for both OSs, where during the working hours the energy consumption increased, and decreased again during the night. This was likely due to more machines being turned on during working hours.

When comparing the energy trends during the day between OSs, both Linux and Windows had spikes, where more energy was consumed, with higher spikes on Windows. The spikes and the trends indicated that not all background processes were disabled on Windows, which impacted how many measurements were required according to Cochran's formula. If all background processes were identified and disabled on Windows, it would mean that less measurements were required, but given the different trends during working- and non-working hours, measurements would still deviate.

The analysis in subsection **Q** also showed how DEC in Equation (1), presented in subsection **4.2**, had some assumptions which were unrealistic. When calculating the DEC, it was based on the total and idle energy consumption. But given how the idle case and benchmark would be measured at different times, which could be at the bottom and top of a spike, the DEC could be too high/low.

6.4 Time synchronization

When measuring the ground truth, four different devices were used. These devices include the DUTs, a Raspberry Pi and an Analog Discovery 2. Each of these devices kept its own time, which were not necessarily synchronized. This was particularly problematic for external measurement instruments, as even small differences like 100ms could result in inaccurate data.

To address this issue, the data acquisition process was changed to ensure that the devices were synchronized every second. However, some problems may still exist, as small time drifts can occur over time. For example, the Raspberry Pi did not have a real-time clock[75] and would therefore become increasingly inaccurate over time. Additionally, the execution time of IO events for the Clamp and Plug could result in a time difference. This is however expected to have minimal impact on the results, as resynchronization happens every second.

6.5 Windows

Compared to the literature, this work stood out by its use of Windows over Linux[8, 76, 77]. Windows was interesting as it is a very popular OS, and because the only study analyzing measuring instruments and energy consumption on Windows, to our knowledge, is [10].

When comparing results between Linux and Windows in Appendix **F**, Windows was found to have a lower DEC in some cases, similar to what was found in [10]. One issue on Windows was finding compatible benchmarks. Because most studies were conducted on Linux, most micro- and macrobenchmarks were made for Linux, which does not guarantee compatibility for Windows. This was a problem in the first experiment, where the C++ benchmarks not only had to be compatible on Windows, but also the four compilers. The original idea was to find macrobenchmarks written in C++, compiled on the most energy efficient compiler, but we did not succeed. Instead PCM and 3DM were chosen, where each had their own issues. For PCM, each DUT had some scenarios it was unable to execute, making it difficult to compare the performance of the two DUTs. For 3DM, when starting multiple times after each other, loading times became increasingly large, until 3DM was restarted. These loading times did not effect the energy measurements, but meant the experiments took additional time to run. Both PCM and 3DM also caused bluescreen with stop code `VIDEO_TDR_FAILURE` on DUT 2 in rare cases, which was found to be a GPU related issues on the `igdkmdn64.sys` process. Neither of the mentioned issues related to PCM or 3DM was resolved, but is something to explore in a future work.

6.6 Cochran's Formula

Cochran's formula was used to ensure enough measurements were taken. In subsection **5.2**, an upper limit was however introduced of 1.000 measurements, as additional measurements were found to have a limited effect on the results. This means that the confidence level of 95% was not met for all results shown in this work. E.g. a case where 1.300 measurements were required, the confidence level was 92% when the margin of error was 0.03 or 95% when the margin of error was 0.034. When 3.000 measurements were required, the confidence level is 75% with a margin of error of 0.03, or 95% if the margin of error is 0.05, and when 5.000 measurements are required, the confidence level was 63.2% with a margin of error of 0.03, or 0.95% with a margin of error of 0.067.

The evolution of the confidence levels and margin of errors presented, represents what impact it has when not enough measurements are made. This shows that in order to gain more confidence in values presented in this paper, some measuring instruments and benchmarks could benefit from additional measurements, but that is a subject for a future work.

7 Conclusion

This work explores parallelism, P- and E-cores, and how this affects energy consumption and execution time, focusing primarily on Windows, with Linux as a reference point. This study is based on four research questions about areas that, until now, have yet to be explored on Windows in the literature. The first research question revolves around compilers' impact on energy consumption and execution time. The second research question looks into different software-based measuring instruments for Windows. The third research question looks into the effect parallelism has on energy consumption. Finally, the fourth research question analyzes and compares P- and E-cores.

For each experiment, initial measurements are made before analyzing the results. The initial measurements are made to ensure confidence in the results by applying Cochran's formula to the results. Cochran's formula is used in this work to ensure enough measurements are made, given a desired confidence level and margin of error. We find that the sample size determined by Cochran's formula is, in many cases, larger than what is currently seen in the literature. The sample size found is, in some cases, so large that an upper limit of 1.000 measurements is introduced, as the gain from additional measurements is found to be limited.

This work has a primary focus on Windows. Through the analysis and comparisons with Linux, Windows is found to provide valuable depth to the analysis of energy consumption, whereas Linux is overall found to be the more convenient OS choice due to its minimalist nature with less pre-installed software and background processes. Reaching definitive conclusions is challenging as the results are very hardware and compiler dependent, and similar observations are not guaranteed between OSs. Given this, we conclude that Windows will be a valuable addition to any research about the energy consumption of software.

When presenting the results, dynamic energy consumption is used to isolate the energy consumption of the benchmark. However, an analysis of the idle energy consumption found that the energy consumption varies between working and non-working hours. Therefore, it is worth exploring the advantages and disadvantages of representing multiple measurements taken over an extended period as a single value in future work.

Since RAPL is not available on Windows, we compare alternative measuring instruments by measuring energy consumption on C++ microbenchmarks compiled with the most energy efficient compiler. In the first experiment, exploring RQ **1**, Intel's oneAPI was found to be the most energy efficient, where a significant difference in performance between compilers is observed. We found that oneAPI achieves the best performance due to its utilization of AVX for parallelism and other optimizations.

The second experiment tests different measuring instruments to decide which to use on Windows, as was formulated in RQ **2**. The experiment compares measurements made by different measuring instruments against a ground truth, where a moderate to high correlation between 0.59 - 0.80 is found. Between the different software-based measuring instruments, similar measurements are made, which is expected to result from using the same registers when reporting energy consumption. In the end, Intel Power Gadget is chosen as our preferred software-based measuring instrument because of its usability compared to other measuring instruments. In addition to different software-based measuring instruments, a cheaper alternative to the ground truth is also included: a smart plug. Similar correlations to the software-based measuring instruments are found when comparing the correlation between the plug and clamp.

In order to answer RQ **4**, the third experiment analyze the performance of P- and E-cores, which in one case shows a 17.40% higher energy consumption for P-cores, while E-cores have a 29.52% higher execution time when executing on four cores, showing that E-cores can be used to limit energy consumption when a higher execution time can be afforded. However, there are cases where the E-cores have a higher energy consumption than P-cores.

Lastly, RQ **3** is also answered in the third experiment, where parallelism and its effect on energy consumption are explored using two macrobenchmarks, PCMark 10 and 3DMark. One represents a realistic use case, including tasks such as video conferencing, web browsing, and video editing, while the other simulates a more demanding workload. Both macrobenchmarks are executed on an increasing amount of cores to examine the effects of additional resources. For both macrobenchmarks, similar observations are found. When more cores are allocated, the execution time decreases, and DEC per second increases, but the DEC remains the same. This shows that there is no correlation between execution time and energy consumption.

8 Future Work

Based on the work presented, certain aspects could be interesting to take a further look into in a future work. This could be an extension to the first experiment, to include more compilers of different versions, more benchmarks and to explore how the different compiler flags impacts the energy consumption. It could also be interesting to extend the second experiment to include a similar experiment to the one conducted in [76] on RAPL, an analyze the overhead of Windows measuring instruments. Future work can further investigate the impact of reactive energy consumption and how it affects the DEC calculation. It may be useful to explore more accurate ways of representing and calculating the DEC that take into account changes in reactive energy consumption and differences in energy consumption during working and non-working hours. Furthermore, additional research could examine the impact of background processes on energy consumption during benchmark execution and explore methods for identifying and disabling these processes to improve measurement accuracy. By addressing these limitations and exploring alternative approaches, future work could improve the accuracy and reliability of energy consumption analysis in software. Finally, it would be relevant to extend this study to other OSs and other hardware, as the two computers used in this work would sometimes contradict each other. The inclusion of more hardware could help back the conclusions found in this work.

Acknowledgements

We would like to express our gratitude to our two supervisors, Lone Leth Thomsen and Bent Thomsen, for their guidance and support throughout the research and writing process of this master thesis. Their expertise and encouragement have been instrumental in shaping this work. The usage of ChatGPT is covered in Appendix **S**

References

1. Jones, N. *et al.* How to stop data centres from gobbling up the world's electricity. *Nature* **561,** 163–166 (2018).
2. Andrae, A. S. & Edler, T. On global electricity usage of communication technology: trends to 2030. *Challenges* **6,** 117–157 (2015).
3. *Intel Core i9-13900K Processor* https://www.intel.com/content/www/us/en/products/sku/230496/intel-core-i913900k-processor-36m-cache-up-to-5-80-ghz/specifications.html.
4. *Intel Core i7-4770K Processor* https://www.intel.com/content/www/us/en/products/sku/75123/intel-core-i74770k-processor-8m-cache-up-to-3-90-ghz/specifications.html.
5. Intel. *Intel performance hybrid architecture & software optimizations Development Part Two: Developing for Intel performance hybrid architecture* https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwj1j9no9c79AhX9S_EDHXGiDMgQFnoECA8QAQ&url=https%3A%2F%2Fcdrdv2-public.intel.com%2F685865%2F211112_Hybrid_WP_2_Developing_v1.2.pdf&usg=AOvVaw2dfqExqLBgFMeS5To1sjKM. 09/03/2023.
6. Somavat, P., Namboodiri, V., *et al.* Energy consumption of personal computing including portable communication devices. *Journal of Green Engineering* **1,** 447–475 (2011).
7. Procaccianti, G., Vetro, A., Ardito, L. & Morisio, M. *Profiling power consumption on desktop computer systems* in *International conference on information and communication on technology* (2011), 110–123.
8. Pereira, R. *et al. Energy efficiency across programming languages: how do energy, time, and memory relate?* in (Oct. 2017), 256–267.
9. *Operating System Market Share Worldwide* https://gs.statcounter.com/os-market-share. 2022.
10. Holt, J., Kusk, M. H. & Pedersen, J. B. *A Comparison Study of Measuring Instruments* (Aalborg University Department of Computer Science, 2023).
11. Ournani, Z. *et al. Taming Energy Consumption Variations In Systems Benchmarking* in *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Association for Computing Machinery, Edmonton AB, Canada, 2020), 36–47. ISBN: 9781450369916. https://doi.org/10.1145/3358960.3379142.
12. Von Kistowski, J. *et al. Variations in CPU Power Consumption* in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering* (Association for Computing Machinery, Delft, The Netherlands, 2016), 147–158. ISBN: 9781450340809. https://doi.org/10.1145/2851553.2851567.
13. Wang, Y., Nörtershäuser, D., Le Masson, S. & Menaud, J.-M. Potential Effects on Server Power Metering and Modeling. *Wirel. Netw.* **29,** 1077–1084. ISSN: 1022-0038. https://doi.org/10.1007/s11276-018-1882-1 (Nov. 2018).
14. Acun, B., Miller, P. & Kale, L. V. *Variation Among Processors Under Turbo Boost in HPC Systems* in *Proceedings of the 2016 International Conference on Supercomputing* (Association for Computing Machinery, Istanbul, Turkey, 2016). ISBN: 9781450343619. https://doi.org/10.1145/2925426.2926289.
15. Marathe, A. *et al. An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors* in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing* (Association for Computing Machinery, Denver, CO, USA, 2017). ISBN: 9781450351324. https://doi.org/10.1145/3149412.3149421.
16. Wang, Y., Nortershauser, D., Masson, S. & Menaud, J.-M. *Experimental Characterization of Variation in Power Consumption for Processors of Different Generations* in (July 2019), 702–710.
17. *Thermal Design Power (TDP) in Intel Processors* https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html. 23/02/2023.
18. Amdahl, G. M. *Validity of the single processor approach to achieving large scale computing capabilities* in *Proceedings of the April 18-20, 1967, spring joint computer conference* (1967), 483–485.
19. Woo, D. H. & Lee, H.-H. S. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer* **41,** 24–31 (2008).
20. Prinslow, G. Overview of performance measurement and analytical modeling techniques for multi-core processors. *UR L: http://www. cs. wustl. edu/˜ jain/cse567-11/ftp/multcore* (2011).
21. Pinto, G., Castor, F. & Liu, Y. D. Understanding Energy Behaviors of Thread Management Constructs. *SIGPLAN Not.* **49,** 345–360. ISSN: 0362-1340. https://doi.org/10.1145/2714064.2660235 (2014).

22. Abdelhafez, A., Alba, E. & Luque, G. A component-based study of energy consumption for sequential and parallel genetic algorithms. *The Journal of Supercomputing* **75,** 1–26 (Oct. 2019).

23. Lindholt, R. S., Jepsen, K. & Nielsen, A. Ø. *Analyzing C# Energy Efficiency of Concurrency and Language Construct Combinations* (Aalborg University Department of Computer Science, 2022).

24. Hassan, H., Moussa, A. & Farag, I. Performance vs. Power and Energy Consumption: Impact of Coding Style and Compiler. *International Journal of Advanced Computer Science and Applications* **8** (Dec. 2017).

25. Saez, J. C. & Prieto-Matias, M. *Evaluation of the Intel thread director technology on an Alder Lake processor* in *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems* (2022), 61–67.

26. Intel. *Energy-Efficient Platforms* https://www.intel.com/content/dam/develop/external/us/en/documents/green-hill-sw-20-185393.pdf. 2011. 07/03/2023.

27. hardwaresecrets. *Everything You Need to Know About the CPU Power Management* https://hardwaresecrets.com/everything-you-need-to-know-about-the-cpu-c-states-power-saving-modes/. 2023. 07/03/2023.

28. ARM. *MEDIA ALERT: ARM big.LITTLE Technology Wins Linley Analysts' Choice Award* https://www.arm.com/company/news/2012/01/media-alert-arm-biglittle-technology-wins-linley-analysts-choice-award. 2012. 09/03/2023.

29. ARM. *Processing Architecture for Power Efficiency and Performance* https://www.arm.com/technologies/big-little. 09/03/2023.

30. Intel. *Intel Unveils 12th Gen Intel Core, Launches World's Best Gaming Processor, i9-12900K* https://www.intel.com/content/www/us/en/newsroom/news/12th-gen-core-processors.html. 2021. 09/03/2023.

31. Rotem, E. *et al.* Intel alder lake CPU architectures. *IEEE Micro* **42,** 13–19 (2022).

32. *1.3.1. processor affinity or CPU pinning* https://www.intel.com/content/www/us/en/docs/programmable/683013/current/processor-affinity-or-cpu-pinning.html. 03/03/2023.

33. Karl-Bridge-Microsoft. *Scheduling priorities - win32 apps* https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities. 03/03/2023.

34. Michael Womack, A. W. *Parallel Programming and Performance Optimization With OpenMP* https://passlab.github.io/OpenMPProgrammingBook/cover.html. 03/03/2023.

35. *MORE POWER WITH LESS COPPER* https://fortop.co.uk/knowledge/white-papers/reactive-power-reducing-compensating/.

36. Mozilla. *tools/power/rapl* https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html. 24/02/2023.

37. source, O. *Libre Hardware Monitor* https://github.com/LibreHardwareMonitor/LibreHardwareMonitor. 03/03/2023.

38. Arnoux, C. *AC current clamps user's manual* 2013.

39. CloudFree. *CloudFree EU Smart Plug* https://cloudfree.shop/product/cloudfree-eu-smart-plug/. 10/03/2023.

40. Hubblo. *Scaphandre* https://github.com/hubblo-org/scaphandre. 23/02/2023.

41. Hubblo. *windows-rapl-driver* https://github.com/hubblo-org/windows-rapl-driver. 23/02/2023.

42. Fahad, M., Shahid, A., Manumachu, R. R. & Lastovetsky, A. A comparative study of methods for measurement of energy of computing. *Energies* **12,** 2204 (2019).

43. Koedijk, L. & Oprescu, A. *Finding Significant Differences in the Energy Consumption when Comparing Programming Languages and Programs* in *2022 International Conference on ICT for Sustainability (ICT4S)* (2022), 1–12.

44. Kaur, A. & Kumar, R. Comparative analysis of parametric and non-parametric tests. *Journal of computer and mathematical sciences* **6,** 336–342 (2015).

45. Razali, N. M., Wah, Y. B., *et al.* Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics* **2,** 21–33 (2011).

46. Wasserstein, R. L., Schirm, A. L. & Lazar, N. A. *Moving to a world beyond "p¡ 0.05"* 2019.

47. Greenland, S. *et al.* Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations. *European journal of epidemiology* **31,** 337–350 (2016).

48. Mann, H. B. & Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics,* 50–60 (1947).

49. Massey, J. & J, F. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* **46,** 68–78 (1951).

50. Godfrey, K. R. Correlation methods. *Automatica* **16,** 527–534 (1980).

51. Armstrong, R. A. Should Pearson's correlation coefficient be avoided? *Ophthalmic and Physiological Optics* **39,** 316–327 (2019).

52. Han, A. K. Non-parametric analysis of a generalized regression model: the maximum rank correlation estimator. *Journal of Econometrics* **35,** 303–316 (1987).
53. Gilpin, A. R. Table for conversion of Kendall's Tau to Spearman's Rho within the context of measures of magnitude of effect for meta-analysis. *Educational and psychological measurement* **53,** 87–92 (1993).
54. Kendall, M. G. A new measure of rank correlation. *Biometrika* **30,** 81–93 (1938).
55. Guilford, J. P. Fundamental statistics in psychology and education (1950).
56. Georgiou, S. & Spinellis, D. Energy-Delay investigation of Remote Inter-Process communication technologies. *Journal of Systems and Software* **162,** 110506. ISSN: 0164-1212. https://www.sciencedirect.com/science/article/pii/S0164121219302808 (2020).
57. Sestoft, P. Microbenchmarks in Java and C#. *Lecture Notes, September* (2013).
58. Cochran, W. G. *Sampling Techniques, 3rd edition* ISBN: 0-471-16240-X (John Wiley & sons, 1977).
59. Kotrlik, J. & Higgins, C. Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research. *Information technology, learning, and performance journal* **19,** 43 (2001).
60. Israel, G. D. Determining sample size (1992).
61. Ross, S. M. *INTRODUCTORY STATISTICS, 3rd edition* ISBN: 978-0-12-374388-6 (Elsevier Inc, 2010).
62. Saqib, M. *What are the Best C++ Compilers to use in 2023* 2023. https://www.mycplus.com/tutorials/cplusplus-programming-tutorials/what-are-the-best-c-compilers-to-use-in-2023/. 20/03/2023.
63. Pedamkar, P. *Best C++ Compiler* 2023. https://www.educba.com/best-c-plus-plus-compiler/. 20/03/2023.
64. *Top 22 Online C++ Compiler Tools* 2023. https://www.softwaretestinghelp.com/best-cpp-compiler-ide/. 20/03/2023.
65. *Clang Compiler Users Manual* https://clang.llvm.org/docs/UsersManual.html. 20/03/2023.
66. *MinGW FAQ* https://home.cs.colorado.edu/~main/cs1300/doc/mingwfaq.html. 20/03/2023.
67. *Intel oneAPI Base Toolkit* https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit.html#:~:text=The%20Intel%C2%AE%20oneAPI%20Base,of%20C%2B%2B%20for%20heterogeneous%20computing.. 20/03/2023.
68. *C and C++ in Visual Studio* 2022. https://learn.microsoft.com/en-us/cpp/overview/visual-cpp-in-visual-studio?view=msvc-170. 20/03/2023.
69. appfolio. *Microbenchmarks vs Macrobenchmarks (i.e. What's a Microbenchmark?)* https://engineering.appfolio.com/appfolio-engineering/2019/1/7/microbenchmarks-vs-macrobenchmarks-ie-whats-a-microbenchmark. 24/03/2023.
70. UL. *3DMark* https://www.3dmark.com/. 14/04/2023.
71. UL. *PCMark* https://benchmarks.ul.com/pcmark10. 14/04/2023.
72. *Central Limit Theorem* https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_probability/BS704_Probability12.html. 20/03/2023.
73. Mauzy, R. Parallelizing Neural Networks to Break Physical Unclonable Functions (2020).
74. *Compare 2 Intel Products* https://www.intel.com/content/www/us/en/products/compare.html?productIds=186605,230495.
75. *Syncing Time from the Network on the Raspberry Pi* = https://pimylifeup.com/raspberry-pi-time-sync/. 2022.
76. Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K. & Ou, Z. RAPL in Action: Experiences in Using RAPL for Power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* **3,** 1–26 (2018).
77. Georgiou, S. & Spinellis, D. Energy-delay investigation of remote inter-process communication technologies. *Journal of Systems and Software* **162,** 110506 (2020).
78. *Intel Default Libraries* https://www.cita.utoronto.ca/~merz/intel_c10b/main_cls/mergedProjects/bldaps_cls/cppug_ccl/bldaps_def_libs_cl.htm.
79. *x86 and amd64 instruction reference* https://www.felixcloutier.com/x86/.
80. *Intel extensions* https://www.intel.com/content/www/us/en/developer/tools/isa-extensions/overview.html.
81. Kullarkar, V. T. & Chandrakar, V. K. Power quality analysis in power system with non linear load. *Int. J. Electr. Eng* **10,** 33–45 (2017).
82. McDonald, B. & Lough, B. *Power Factor Correction (PFC) Circuit Basics* in *Texas Instruments Power Supply Design Seminar* (2020).
83. Igor Wallossek, A. M. *Picking The Right Power Supply: What You Should Know* https://www.tomshardware.com/reviews/psu-buying-guide,2916-3.html. 2016.

A Abbreviations

In Table **12**, a list of all the terms which are abbreviated in this work can be found. They are alphabetically sorted within their categories. Their first occurrence can also be seen.

Abbreviations used in this work		
General Technology and Hardware Terms	**Abbreviation**	**First Occurrence**
Device Under Test	DUT	Section **1**
Efficiency core	E-core	Section **1**
Information and Communications Technology	ICT	Section **1**
Operating System	OS	subsection **2.4**
Performance core	P-core	Section **1**
Power supply unit	PSU	subsection **4.1**
Measuring Instruments	**Abbreviation**	**First Occurrence**
Clamp Linux	Clamp (L)	subsection **5.2**
Clamp Windows	Clamp (W)	subsection **5.2**
CloudFree EU smart Plug	Plug	subsection **4.1**
Intel Power Gadget	IPG	Section **2**
Libre Hardware Monitor	LHM	Section **2**
MN60 AC Current Clamp	Clamp	subsection **4.1**
Plug Linux	Plug (L)	subsection **5.2**
Plug Windows	Plug (W)	subsection **5.2**
Running Average Power Limit	RAPL	Section **2**
Scaphandre	Scap	subsection **4.1**
Scaphandre isolated	SCAPI	subsection **4.1**
Benchmarks	**Abbreviation**	**First Occurrence**
3DMark	3DM	subsection **4.6**
Fannkuch-Redux	FR	subsection **4.6**
Mandelbrot	MB	subsection **4.6**
Nbody	NB	subsection **4.6**
PCMark 10	PCM	subsection **4.6**
Spectra-Norm	SN	subsection **4.6**
Compilers	**Abbreviation**	**First Occurrence**
Intel's oneAPI C++	oneAPI	subsection **4.5**
Microsoft Visual C++	MSVC	subsection **4.5**
Minimalist GNU for Windows	MinGW	subsection **4.5**
Energy Consumption Terms	**Abbreviation**	**First Occurrence**
Dynamic Energy Consumption	DEC	subsection **4.2**
Other terms	**Abbreviation**	**First Occurrence**
Biks Diagnostics Energy	BDE	Appendix **B**
Intel's Thread Director	ITD	subsection **2.4**
Model-specific-registers	MSRs	subsection **4.1**
Open Multi-Processing	OpenMP	subsection **3.5**
Performance Monitoring Counter	PMC	subsection **2.4**
Speedup Factor	SF	subsection **2.4**

Table 12: Abbreviations used in this work and their first occurrences. In alphabetical order.

B The Framework

The framework introduced for the experiments in this work, was called Biks Diagnostic Energy (BDE) and was a command line tool. It was an extension of the work presented in [10]. BDE could be executed in two ways, as illustrated in Listing **4**, where one was with a configuration, and one was with a path to an executable file.

```
.\BDEnergyFramework --config path/to/config.json

.\BDEnergyFramework --path path/to/file.exe --parameter parameter
```

Listing 4: An example of how BDE can be started

When using `--config`, the user specified a path to a valid JSON file formatted like Listing **5**. Through Listing **5**, it was possible to specify paths to executable files and assign each executable file with a parameter in `BenchmarkPaths` and `BenchmarkParameter` respectively. It was also possible to specify the affinity of the benchmark through `AllocatedCores`, where an empty list represented the use of all cores and the list `1,2` stated core 1 and 2 were used. When multiple affinities were specified, each benchmark was executed on both. Lastly, `AdditionalMetadata` could specify relevant aspects of the experiment that could not already be specified through the configuration.

```
[
  {
    "MeasurementInstruments": [ 2 ],
    "RequiredMeasurements": 30,
    "BenchmarkPaths": [
        "path/to/one.exe", "path/to/two.exe"
    ],
    "AllocatedCores": [
        [], [1,2]
    ],
    "BenchmarkParameters": [
        "one_parameter", "two_parameter",
    ],
    "UploadToDatabase": true,
    "BurnInPeriod": 0,
    "MinimumTemperature": 0,
    "MaximumTemperature": 100,
    "DisableWifi": false,
    "ExperimentNumber": 0,
    "ExperimentName": "testing-phase",
    "ConcurrencyLimit": "multi-thread",
    "BenchmarkType": "microbenchmarks",
    "Compiler": "clang",
    "Optimizations": "openmp",
    "Language": "c++",
    "StopBackgroundProcesses" : false,
    "AdditionalMetadata": {}
  }
]
```

Listing 5: An example of a valid configuration for BDE

When using the parameters `--path`, the `--parameter` was an optional way to provide the executable with parameters. When using BDE this way, a default configuration was set up, containing all fields in the configuration, except `BenchmarkPath` and `BenchmarkParameter`.

```
public interface IDutService
{
    public void DisableWifi();
    public void EnableWifi();
    public List<EMeasuringInstrument> GetMeasuringInstruments();
    public string GetOperatingSystem();
    public double GetTemperature();
    public bool IsAdmin();
    public void StopBackgroundProcesses();
}
```

Listing 6: The DUT interface which allows BDE to work on multiple OSs

Both Windows and Linux were supported on BDE. This was supported through the `IDutService` seen in Listing **6**, where all OS dependent operations were located.

```
1   public class MeasuringInstrument
2   {
3
4       public (TimeSeries, Measurement) GetMeasurement()
5       {
6           var path = GetPath(_measuringInstrument, fileCreatingTime);
7           return ParseData(path);
8       }
9
10      public void Start(DateTime fileCreatingTime)
11      {
12          var path = GetPath(_measuringInstrument, fileCreatingTime);
13
14          StartMeasuringInstruments(path);
15
16          StartTimer();
17      }
18
19      public void Stop(DateTime date)
20      {
21          StopTimer();
22          StopMeasuringInstrument();
23      }
24
25      internal virtual int GetMilisecondsBetweenSamples()
26      {
27          return 100;
28      }
29
30      internal virtual (TimeSeries, Measurement) ParseData(string path) { }
31
32      internal virtual void StopMeasuringInstrument() { }
33
34      internal virtual void StartMeasuringInstruments(string path) { }
35
36      internal virtual void PerformMeasuring() { }
37  }
```

Listing 7: The implementation of the different measuring instruments on BDE

BDE also supported multiple measuring instruments, through a parent class `MeasuringInstrument` in Listing **7** the measuring instruments could inherit from. `MeasuringInstrument` implemented a start (line 10) and stop (line 19) method, and a method which retrieved the data measured between the start and stop. The virtual methods were measuring instrument specific, an could be overwritten by the specific measuring instrument. This included a start (line 34) and stop (line 32) method, a method to parse the measurement data in line 30 and a method in line 36 which performed a measurement by default every 100ms. The method in line 36 was made for measuring instruments like RAPL, where an action is required to read the energy consumption.

Listing **8** showed how BDE performed measurements given the configuration. In the configuration, the burn-in period could be set to any positive integer, where if this value is one, the boolean `burninApplied` would be set to `true`, and the measurements would be initialized in line 7. This initialization would, if the results should be uploaded to the database, mean BDE would fetch existing results from the database, where the configuration was the same, and continue where it was left off. Otherwise, an empty list would be returned. If `burninApplied` was set to `false`, the amount of burn-in specified in the configuration would be performed before initializing the measurements.

```
1      public void PerformMeasurement(MeasurementConfiguration config)
2      {
3          var measurements = new List<MeasurementContext>();
4          var burninApplied = SetIsBurninApplies(config);
5
6          if (burninApplied)
7              measurements = InitializeMeasurements(config, _machineName);
8
9          do
10         {
11             if (CpuTooHotOrCold(config))
12                 Cooldown(config);
13
14             if (config.DisableWifi)
15                 _dutService.DisableWifi();
16
17             PerformMeasurementsForAllConfigs(config, measurements);
18
19             if (burninApplied && config.UploadToDatabase)
20                 UploadMeasurementsToDatabase(config, measurements);
21
22             if (!burninApplied && IsBurnInCountAchieved(measurements, config))
23             {
24                 measurements = InitializeMeasurements(config, _machineName);
25                 burninApplied = true;
26             }
27
28         } while (!EnoughMeasurements(measurements));
29     }
```

Listing 8: An example of how BDE performs measurements

Next, a do-while loop was entered in line 9, which would execute until the condition `EnoughMeasurements` from line 28 was met. Inside the do-while loop, a cooldown would occur in line 12, until the DUT was below and above the temperature limits specified in the configuration. Once this is achieved, the Wi-Fi/Ethernet is disabled, and `PerformMeasurementsForAllConfigs` would then iterate over all measuring instruments and benchmarks specified, and perform one measurement for all permutations. Afterward, a few checks were made. If the burn-in period was over, and the configuration stated that the results should be uploaded to the database, `UploadMeasurementsToDatabase` was called. If the burn-in period was not over yet, but `IsBurnInCountAchieved` is `true`, the measurements was initialized similarly to line 7, and the boolean `burninIsApplied` was set to `true`, indicating that the burn-in period was over, and the measurements were about to be taken.

C The Database

In [10], a MySQL database was used to store the measurements made by the different measuring instruments. In this work, a similar database was used with some modifications to accommodate the different focus compared to [10]. The design of the database was illustrated in Figure **10**, where the `MeasurementCollection` table defines under which circumstances the measurements were made. This includes which measuring instrument was used, which benchmark was running, which DUT the measurements were made on, whether or not there was a burn-in period, etc.

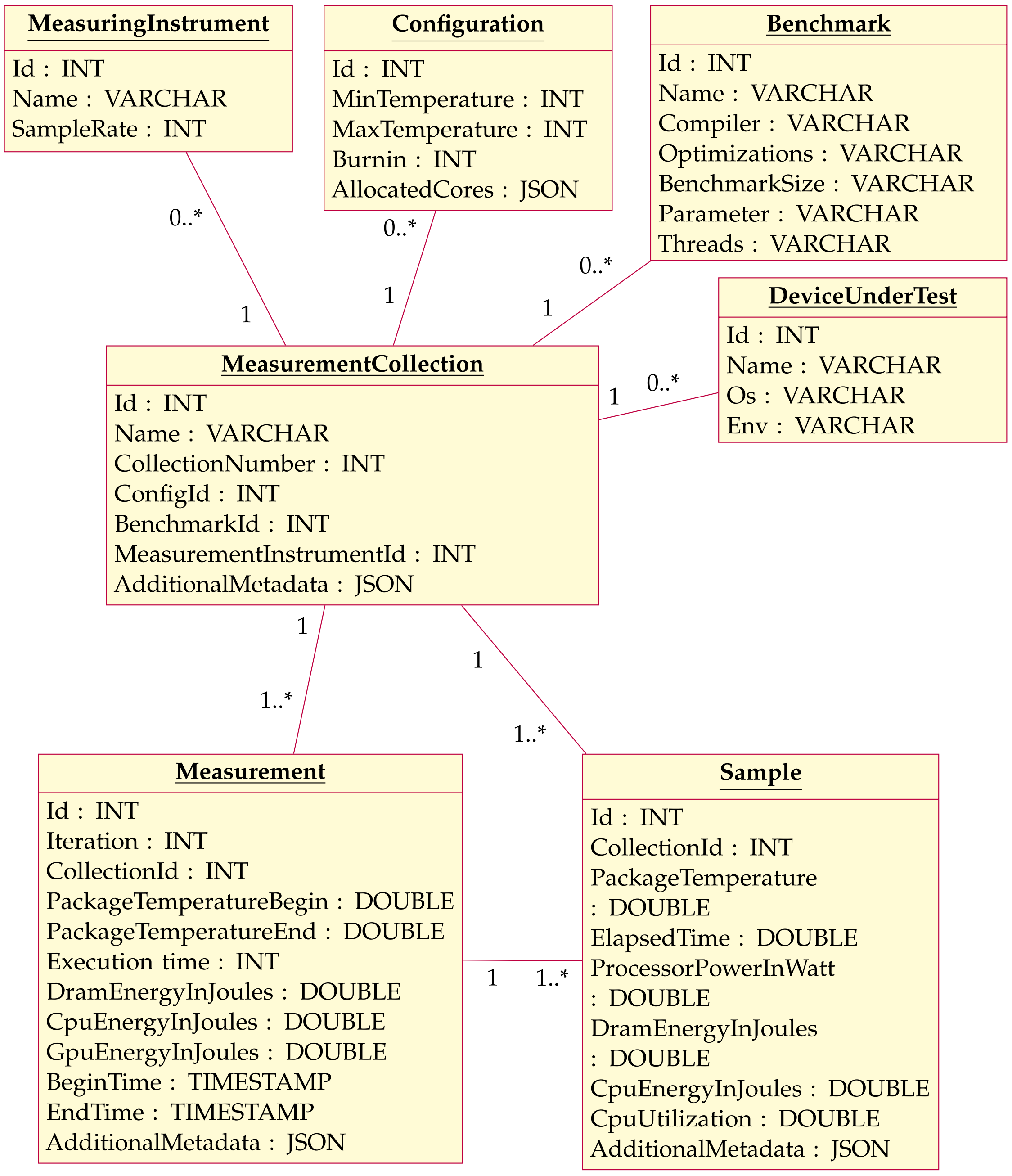


Figure 10: An UML diagram representing the tables in the SQL database

In the `MeasurementCollection`, the columns `CollectionNumber` and `Name` represented which experiment the measurement was from, and the name of the experiment. The `Measurement` contained values for the entire energy consumption during the entire execution of one benchmark, while the `Sample` represented samples taken during the execution of the benchmark. This meant for one row in the `MeasurementCollection` table, there could exist one to many rows in `Measurement`. Each row in `Measurement` was associated with multiple rows in the `Sample` table, where the samples would be a time-series illustrating the energy consumption over time.

D PCMark 10

Given issues with some scenarios on PCM, some scenarios were excluded for the DUTs, where the excluded scenarios were not the same across both DUTs. The different scenarios and whether they are included are shown on Tables **13** and **14**. Further detail about the workloads can be found in [71]. PCM was presented in subsection **4.6**, while the issues were presented in Section **6.5**.

Essentials		**Productivity**		**Digital Content Creation**	
App Start-up		**Writing**		**Photo Editing**	
Chromium	×	Writing simulation	×	Editing one photo	✓
Firefox	×			Editing a batch of photos	✓
LibreOffice Writer	×				
GIMP	×				
Web Browsing		**Spreadsheets**		**Video Editing**	
Social media	×	Common use	✓	Downscaling	✓
Online shopping	×	Power use (More complex)	✓	Sharpening	✓
Map	×			Deshaking filtering	✓
Video 1080p	×				
Video 2160p	×				
Video Conferencing				**Rendering and Visualization**	
Private call	✓			Visualization of a 3D model	✓
Group call	✓			Calculating a simulation	✓

Table 13: List of PCM benchmarks used on DUT1.

Essentials		**Productivity**		**Digital Content Creation**	
App Start-up		**Writing**		**Photo Editing**	
Chromium	×	Writing simulation	×	Editing one photo	×
Firefox	×			Editing a batch of photos	×
LibreOffice Writer	×				
GIMP	×				
Web Browsing		**Spreadsheets**		**Video Editing**	
Social media	✓	Common use	×	Downscaling	✓
Online shopping	✓	Power use (More complex)	×	Sharpening	✓
Map	✓			Deshaking filtering	✓
Video 1080p	✓				
Video 2160p	✓				
Video Conferencing				**Rendering and Visualization**	
Private call	✓			Visualization of a 3D model	✓
Group call	✓			Calculating a simulation	✓

Table 14: List of PCM benchmarks used on DUT2.

E Experiment One

This section illustrated the measurements made for the first experiment, where four different C++ compilers were compared. The first experiment was presented in subsection **5.1**.

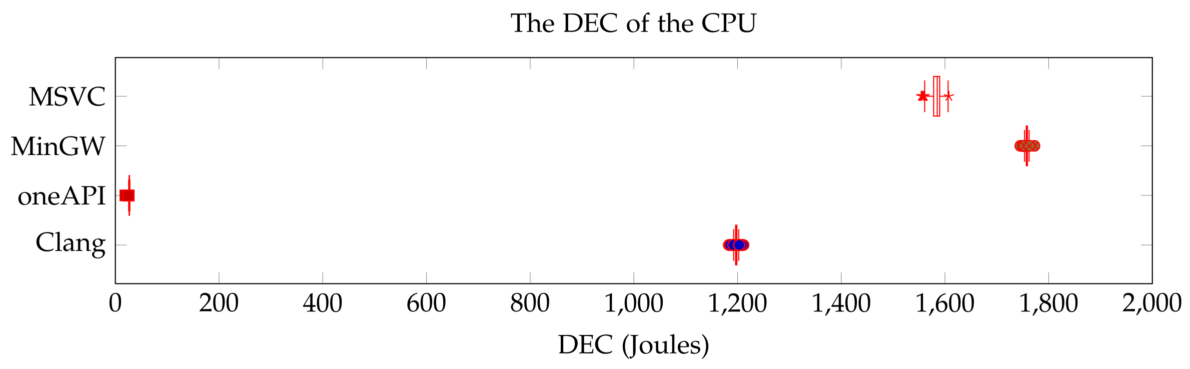


Figure 11: CPU measurements by IPG on DUT 1 for test case(s) MB

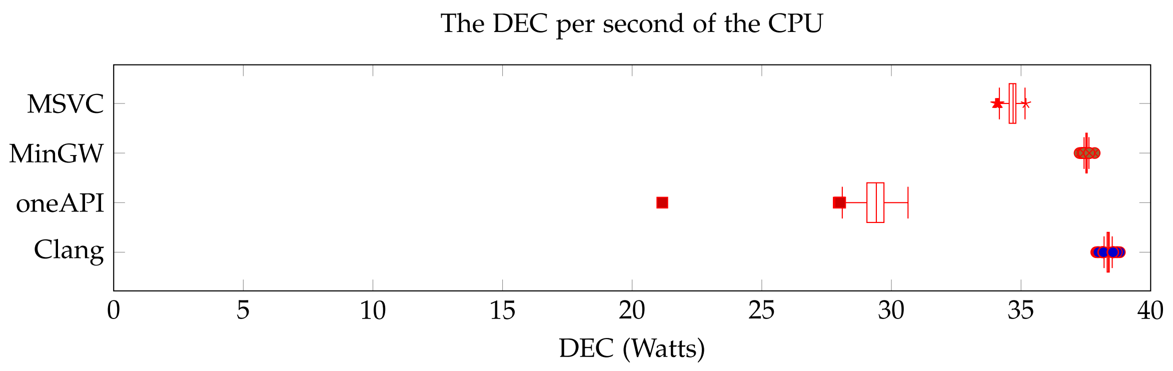


Figure 12: CPU measurements by IPG on DUT 1 for test case(s) MB

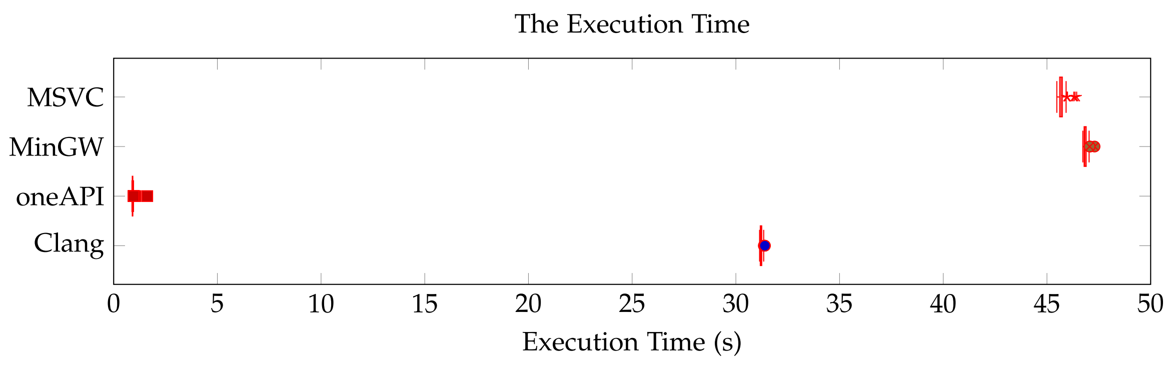
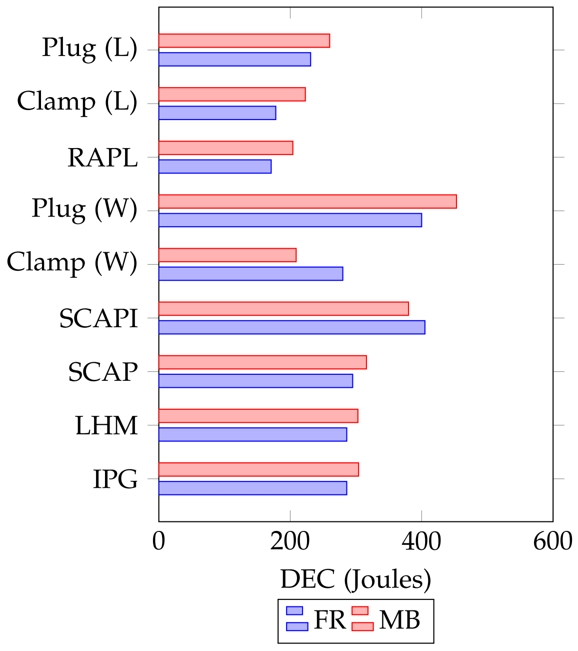


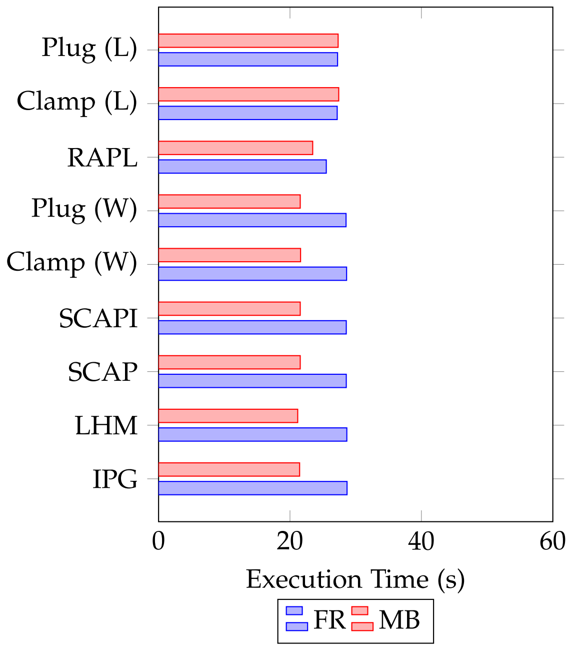
Figure 13: Execution time measurements by IPG on DUT 1 for test case(s) MB

F Experiment Two

This section illustrated the measurements made for the second experiment, which aimed to identify the positive and negative aspect of the different measuring instruments. The second experiment was presented in subsection **5.2**.



(a) DEC



(b) Execution Time

Figure 14: DEC and execution time for DUT 2, where FR and MB were measured with different measuring instruments

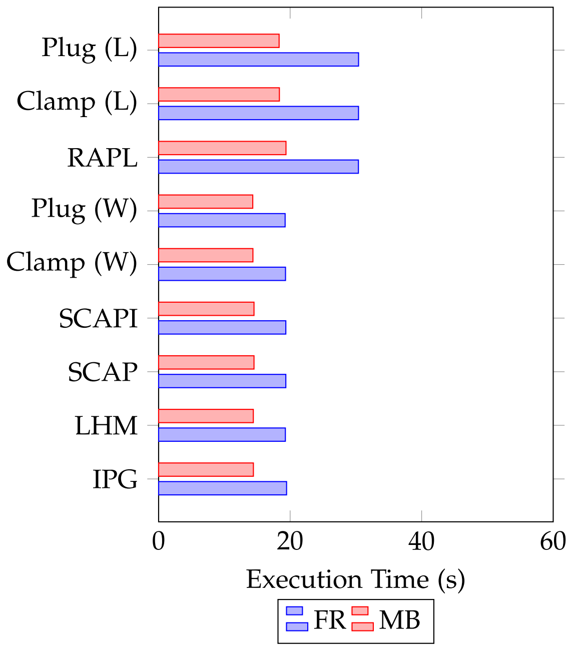
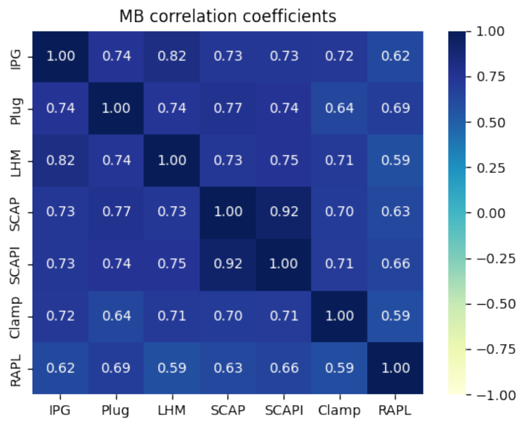


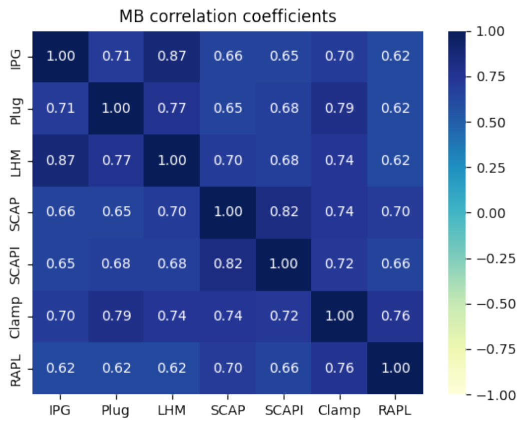
Figure 15: The execution time for DUT 1, where both benchmarks are compiled on oneAPI

G Correlation from Experiment Two

This section illustrated the correlation heatmap from the second experiment in subsection **5.2**.

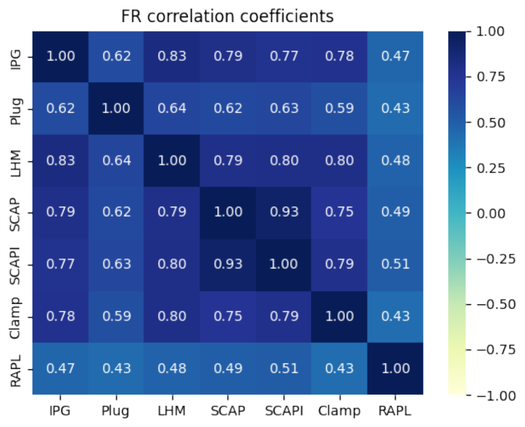


(a) DUT 1

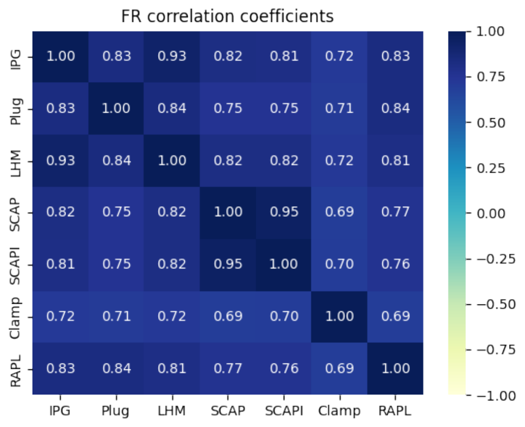


(b) DUT 2

Figure 16: Heatmap showing the correlation coefficient between all of the measurement instruments on Windows for the MB benchmark



(a) DUT 1



(b) DUT 2

Figure 17: Heatmap showing the correlation coefficient between all of the measurement instruments on Windows for the FR benchmark

H Initial Measurements in Experiment Two

This section illustrated the required measurements for the different measuring instruments from the second experiment, which were estimated using Cochran's formula for a desired confidence level of 95% and a margin of error of 0.03%. The second experiment was presented in subsection **5.2**.

Initial Measurements		
Name	FR	MB
Plug (W)	2.474	1.790
Plug (L)	5	4.818
Clamp (W)	16.908	2.855
Clamp (L)	12.837	11.518
RAPL	52	53
SCAP	459	74
SCAPI	453	153
IPG	714	216
LHM	604	45

(a) DUT 1

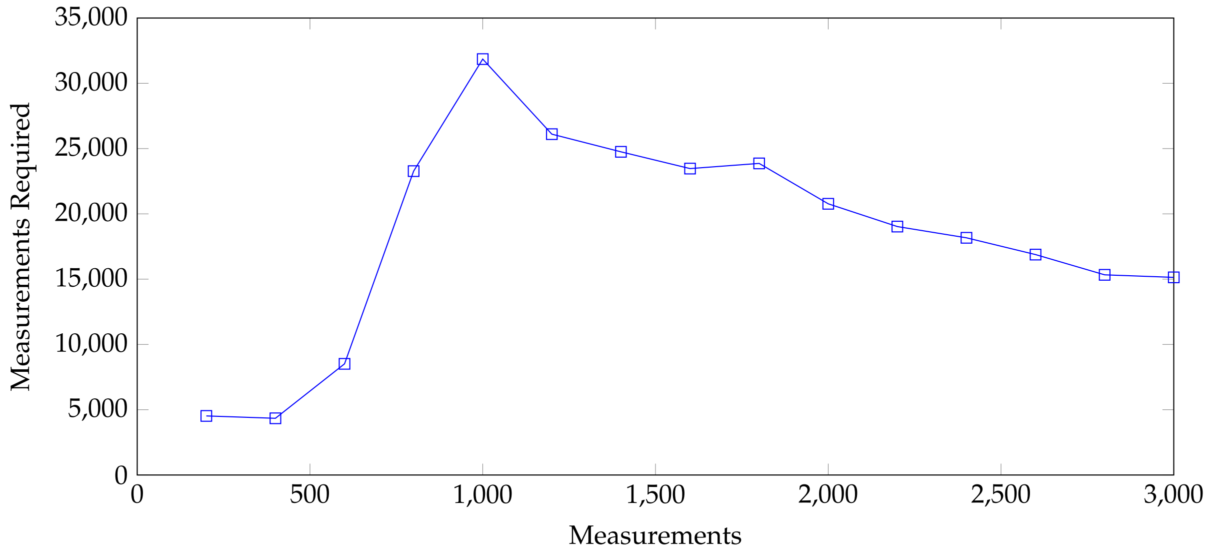
Initial Measurements		
Name	FR	MB
Plug (W)	916	1.088
Plug (L)	738	1056
Clamp (W)	36.558	44.106
Clamp (L)	2.869	7.021
RAPL	1.298	4.340
SCAP	416	1.478
SCAPI	840	3.095
IPG	379	88
LHM	379	31

(b) DUT 2

Figure 18: The required samples to gain confidence in the measurements made by the different measuring instruments on both OSs

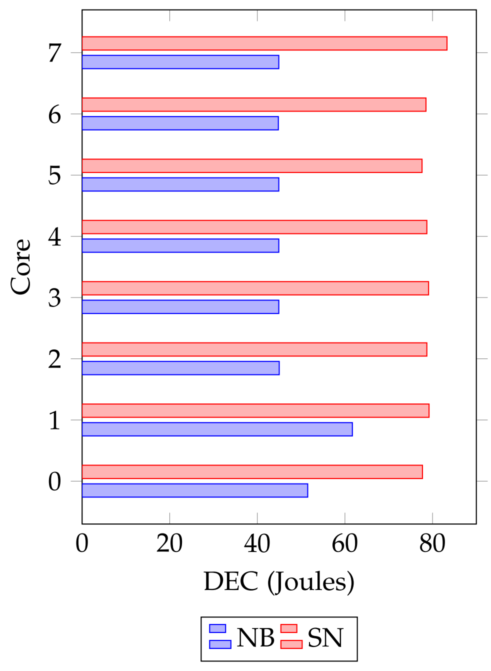
I Cochran's Formula Evolution for Experiment Two

This section illustrated the evolution of how many measurements were required, as estimated using Cochran's formula, for an increasing number of measurements. The graph was used in the second experiment, presented in subsection **5.2**.

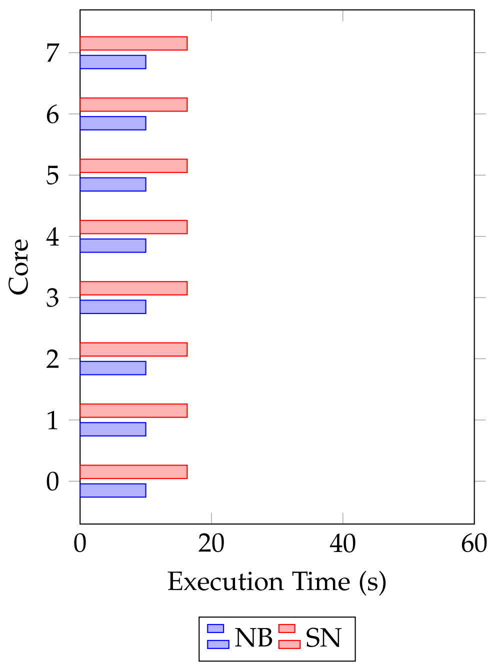


J Experiment Three

This section illustrated the results from the third experiment, where microbenchmarks were executed on one core at a time. The third experiment was presented in subsection 5.3

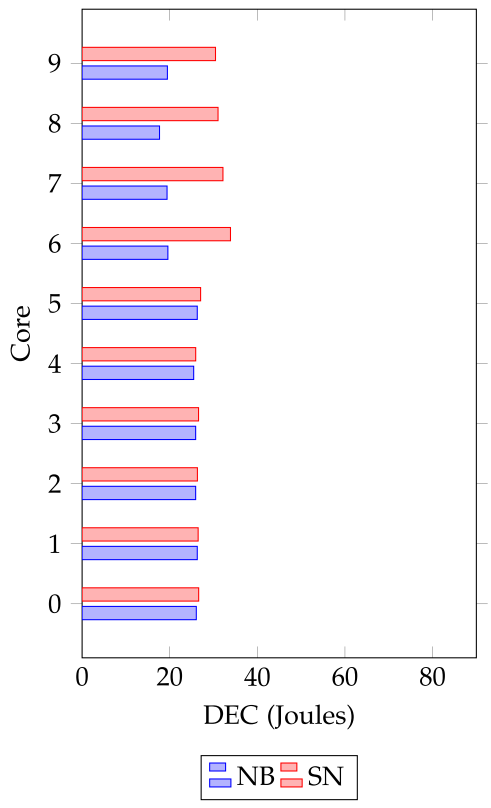


(a) DEC

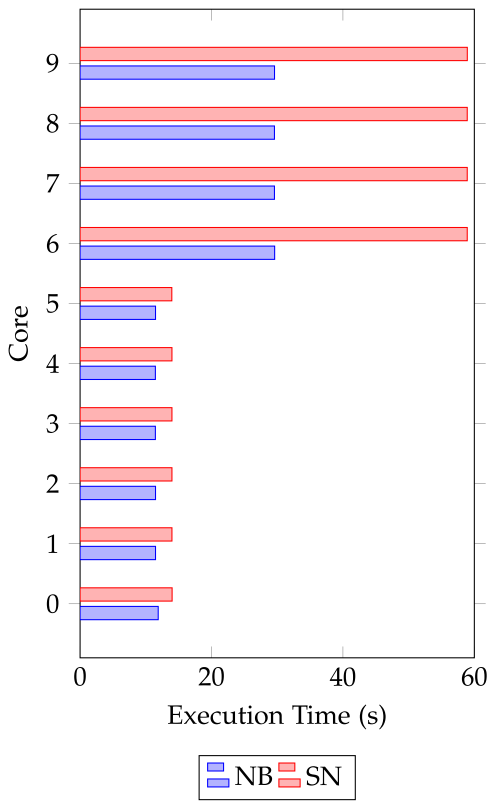


(b) Execution time

Figure 19: DEC and execution time for DUT 1, where NB and SN were executed on one core at a time



(a) DEC



(b) Execution time

Figure 20: DEC and execution time for DUT 2, where NB and SN were executed on one core at a time

K Measurements Required in Experiment Three Cores

This section illustrated the required measurements for IPG when measuring the energy consumption on different cores, estimated using Cochran's formula for a desired confidence level of 95% and a margin of error of 0.03%. The values were used for the third experiment, presented in subsection **5.3**.

Initial Measurements		
Name	NB	SN
Core 0	5.162	1.991
Core 1	11.771	1.999
Core 2	5.119	2.047
Core 3	4.678	2.039
Core 4	4.597	1.979
Core 5	5.005	2.082
Core 6	4.622	1.852
Core 7	4.996	1.945

(a) DUT 1

Initial Measurements		
Name	NB	SN
Core 0	4	36
Core 1	7	33
Core 2	1	35
Core 3	1	28
Core 4	3	33
Core 5	2	30
Core 6	17	115
Core 7	22	99
Core 8	18	121
Core 9	39	92

(b) DUT 2

Figure 21: The required samples to gain confidence in the measurements made by IPG in different cores

L Measurements Required in Experiment Three Macrobenchmarks

This section illustrated the required measurements for IPG when measuring the energy consumption of the macrobenchmarks, estimated using Cochran's formula for a desired confidence level of 95% and a margin of error of 0.03%. These numbers were used for the third experiment, presented in subsection **5.3**.

Initial Measurements		
Name	3DM	PCM
1 Core	1207	630
2 Cores	1470	579
3 Cores	1531	770
4 Cores	1524	913
5 Cores	2054	820
6 Cores	2359	883
7 Cores	1810	997
8 Cores	1391	811

(a) DUT 1

Initial Measurements		
Name	3DM	PCM
1 Core	22	
2 Cores	183	84
3 Cores	135	80
4 Cores	56	71
5 Cores	79	54
6 Cores	53	78
7 Cores	25	76
8 Cores	20	78
9 Cores	42	77
10 Cores	44	100

(b) DUT 2

Figure 22: The required samples to gain confidence in the measurements made by IPG for the macrobenchmarks

M Results for Macrobenchmarks in the Third Experiment

This section illustrated the energy consumption for an increasing amount of cores, used in the third experiment, presented in subsection **5.3**.

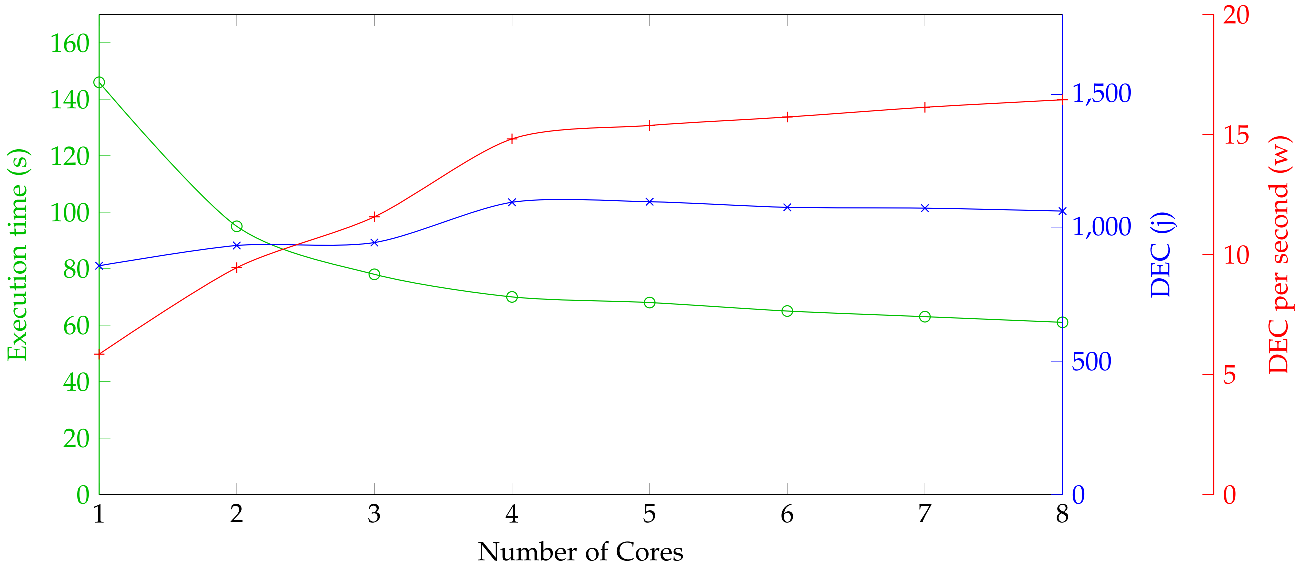


Figure 23: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to 3DM on DUT 1

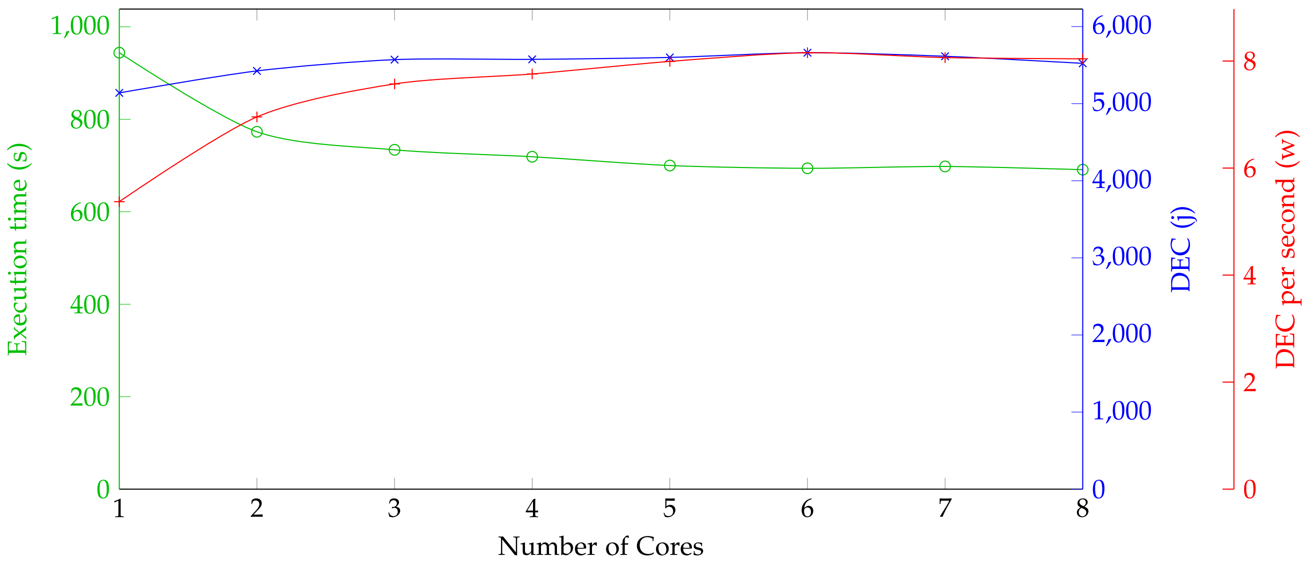


Figure 24: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to PCM on DUT 1

Measurements Requied in Experiment Three P- vs. E-Cores

This section illustrated how many measurements were required from IPG when the energy consumption of PCM on four P-cores, four E-cores, or two of each was measured. These results were used in the the third experiment, presented in subsection **5.3**

Initial Measurements	
Name	PCM
4P	131
4E	125
2P2E	44

Table 15: The required samples to gain confidence in the measurements made by IPG when comparing P- and E-cores for DUT 2

N Results for P- vs. E-Cores in Third Experiment

This section illustrated the results obtained when the performance of four P-cores, four E-cores, or two of each when running PCM on DUT 2. The results were referenced in subsection **5.3**

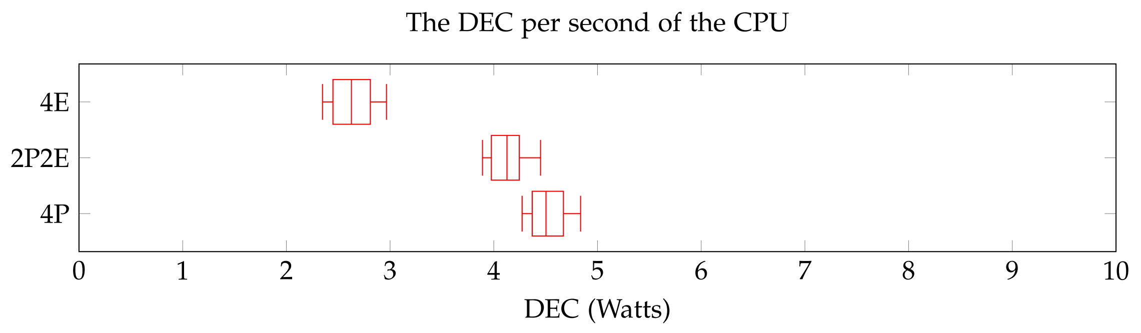


Figure 25: CPU measurements by IPG on DUT 2 for test case(s) PCM

O Energy Consumption Over Time

This section illustrated how the energy consumption of both macrobenchmarks evolved over time for DUT 2. The analysis was used in the third experiment in subsection **5.3**, and compares the energy consumption for two and ten cores, to show the difference.

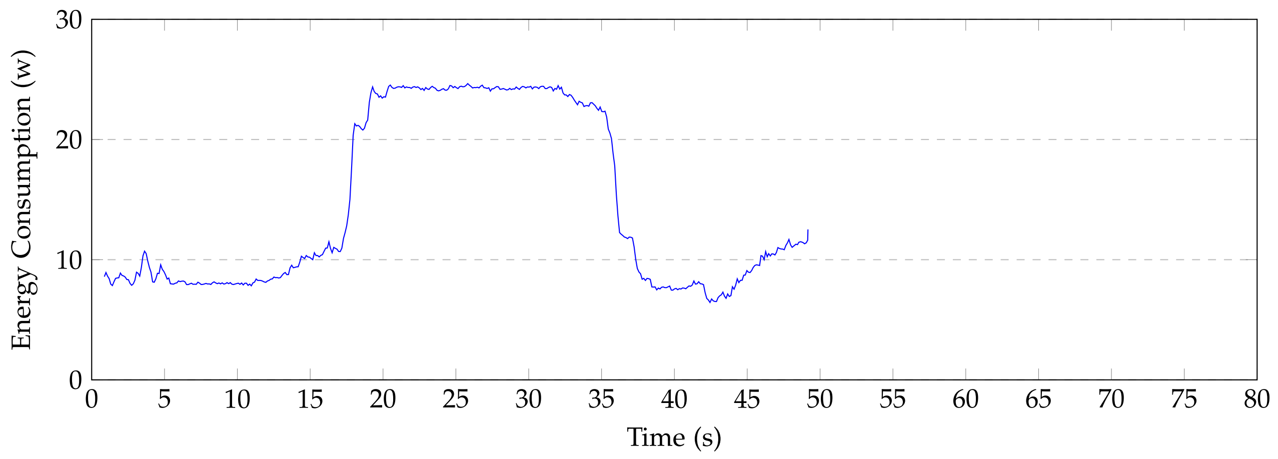


Figure 26: A timeseries of the energy consumption over time for DUT 2 when running 3DM for all cores

Measurements for 3DM were illustrated in Figure **26** and Figure **27** for both two and ten cores, both have a similar startup period until 14 seconds, after which the benchmark started. On ten cores, the load was on 25 watts for 18 seconds, while for two cores the energy consumption was on 12 watts for 60 seconds.

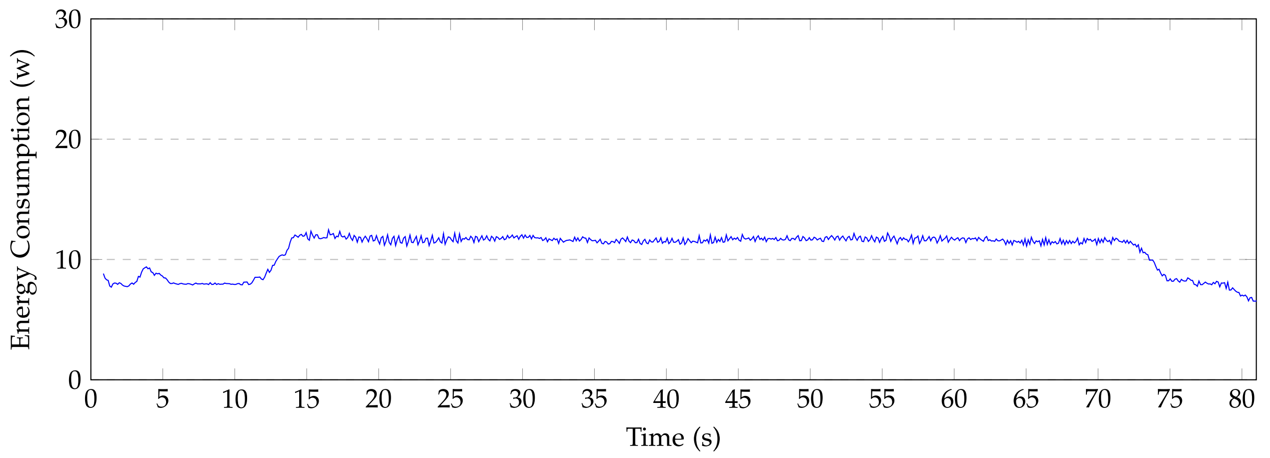


Figure 27: A timeseries of the energy consumption over time for DUT 2 when running 3DM on two cores

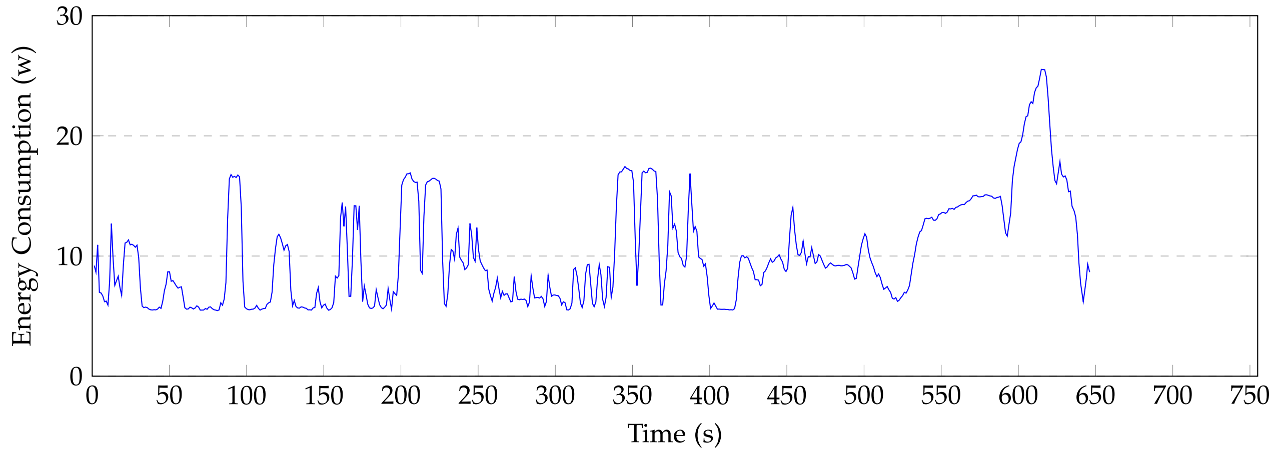


Figure 28: A timeseries of the energy consumption over time for DUT 2 when running PCM for all cores

Measurements for PCM were illustrated in Figure **28** and Figure **29**. Compared to 3DM, there were smaller differences between two and ten cores. One cause was that PCM required fewer resources, meaning additional resources had diminishing returns. When looking at Figure **29**, it was observed that the peak wattage usage of 12 for 3DM on two cores was exceeded during runtime. This occurred between $225s - 235s$, $370s - 380s$, and $570s - 620s$, which amounts to 9% of the total runtime. This indicated that the affinity was only set for some processes related to PCM, which meant that too many resources were allocated to some processes. We could not solve this, meaning the performance gained when allocating more cores to PCM represents a lower limit. Because if the affinity were set correctly, the execution time would be higher on a few cores, resulting in more significant decreases in execution time when allocating additional cores.

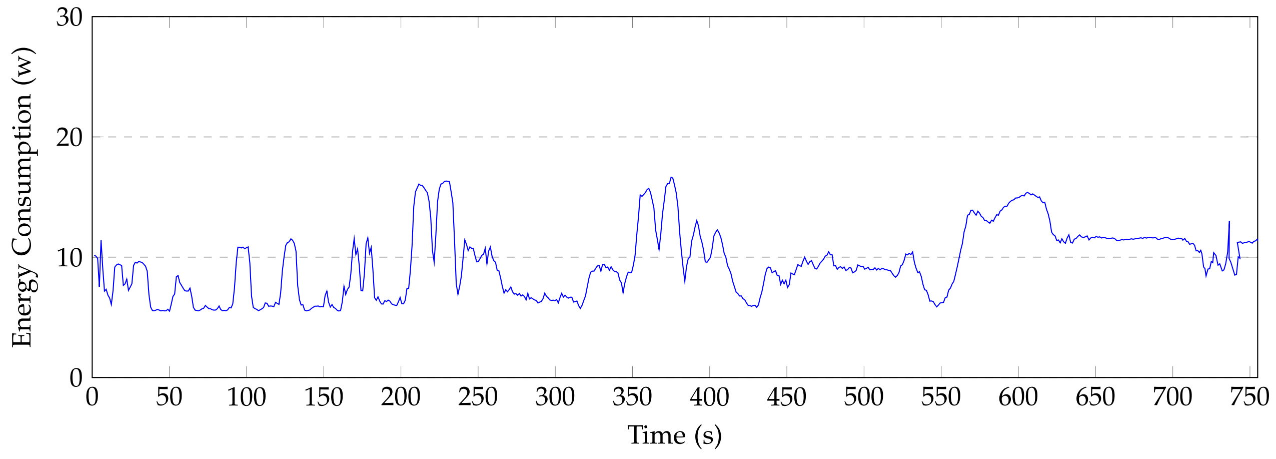


Figure 29: A timeseries of the energy consumption over time for DUT 2 when running PCM on two cores

P C++ Compiler Analysis

Different compilers were compared in the first experiment in subsection **5.1**. Through the analysis it found that both the energy consumption and measurements required deviated between compilers. The largest difference was found when comparing oneAPI against the other compilers, where one potential cause was if oneAPI had removed the entire benchmark as dead code.[57]. An analysis was conducted on MB, where the executables compiled by oneAPI and MinGW were decompiled to compare the instructions used.

The assembly code showed that oneAPI used several function calls unique for Intel processors such as `___intel_new_feature_proc_init` and `__intel_fast_memset`, where both function calls were part of Intel's default C++ libraries.[78] MinGW used more general-purpose registers and utilized the C++ Standard Library more than oneAPI.

```
push     r15 {__saved_r15}
push     r14 {__saved_r14}
push     r13 {__saved_r13}
push     r12 {__saved_r12}
push     rbp {__saved_rbp}
push     rdi {__saved_rdi}
push     rsi {__saved_rsi}
push     rbx {__saved_rbx}
sub      rsp, 0x228
movaps   xmmword [rsp+0x200 {__saved_zmm6}],
    xmm6
movaps   xmmword [rsp+0x210 {__saved_zmm7}],
    xmm7
mov      r13, rcx
mov      rcx, qword [rcx+0x20]
mov      rax, qword [rcx+0x8]
movsd    xmm7, qword [r13+0x10]
subsd    xmm7, qword [r13]
test     rax, rax
js       0x140003a75
```

Figure 30: MinGW assembly showing the setup for the mandelbrot function

```
push     ebp {__saved_ebp}
mov      ebp, esp {__saved_ebp}
push     ebx {__saved_ebx}
push     edi {__saved_edi}
push     esi {__saved_esi}
and      esp, 0xfffffffe0
sub      esp, 0x320
mov      eax, dword [ebp+0x8 {_RawVals
  }]
vmovsd   xmm0, qword [eax+0x10]
vmovsd   xmm1, qword [eax+0x18]
vsubsd   xmm2, xmm0, qword [eax]
vsubsd   xmm0, xmm1, qword [eax+0x8]
vmovsd   qword [esp+0x1f8 {var_148}],
  xmm0
mov      eax, dword [eax+0x20]
mov      ecx, dword [eax+0x4]
vmovss   xmm3, dword [eax+0x4]
vmovss   xmm0, dword [eax+0x8]
xor      edi, edi  {0x0}
cmp      ecx, 0x8
xmmword [esp+0xa0 {var_2a0}], xmm0
jae      0x408fcb
```

Figure 31: oneAPI assembly showing the setup for the mandelbrot function

For Mandelbrot, MinGW only used standard X86 instruction[79] to calculate the floating points and `xmm` registers to store the values. Whereas, oneAPI's implementation utilized Advance Vector extensions (AVX)[80] to perform the calculations and frequently utilized `ymm` registers instead of `xmm`. The usage of AVX increased calculation speed as it allowed for multiple calculations in parallel. The AVX technology is a set of instructions introduced by Intel to enhance the performance of floating-point-intensive applications[80]. Compared to MinGW, oneAPI was better at optimizing the code for the AVX architecture and taking full advantage of the processing power of the CPU. This usage of AVX was illustrated in Figure **30** and Figure **31**, where the differences between MinGW and oneAPIs way of handling the setup for the Mandelbrot function were illustrated. The listings show the two approaches to load values into local variables used by oneAPI and MinGW. When comparing the two approaches, oneAPI used vectorized instructions, such as `vmovsd`, from AVX, while MinGW used simple instructions, such as `movsd`. The Vectorized instructions were better for execution time, as this operation could be performed simultaneously on multiple values.

oneAPIs use of parallel operations and more minor optimizations was likely the reason it achieved a lower execution time.

Q Energy usage trends analysis

Two hypotheses were defined to explore if reactive energy consumption caused DEC consumption to decrease. (**H1:**) *noise on the electrical network could interfere with the phase synchronization.* This could be due to many machines being connected to the same electrical network and disrupting the harmonics of the network[81], which can be caused by non-linear loads. A non-linear load is essentially a device with changing capacitance such as most everyday devices except light bulbs and such. It is mainly caused by electronic devices like computers, which convert AC power to DC power. The interference caused by non-linear loads can lead to increased energy consumption, equipment damage, and instability of the power system. The PSU could have generated reactive energy because it was out of phase with the electrical network, where a reduction in noise could help synchronize again. Therefore, the observed changes in energy consumption may be related to the time of the day and week where the measurements were taken, with consumption decreasing when fewer devices were connected to the electrical network during the night and weekends.

H2: *The DUTs' PSUs may be correcting the phase over time with a Power Factor Correction circuit.* According to [82], there were two main types of Power Factor Correction: passive and active. The behavior observed in the results could result from an active Power Factor Correction circuit. In order to explore **H2**, an email was sent to the manufacturers of both PSUs, to which Cougar did not respond, while Corsair confirmed the PSU had Power Factor Correction but would not give any more details. Therefore we could not determine which type of Power Factor Correction was used.

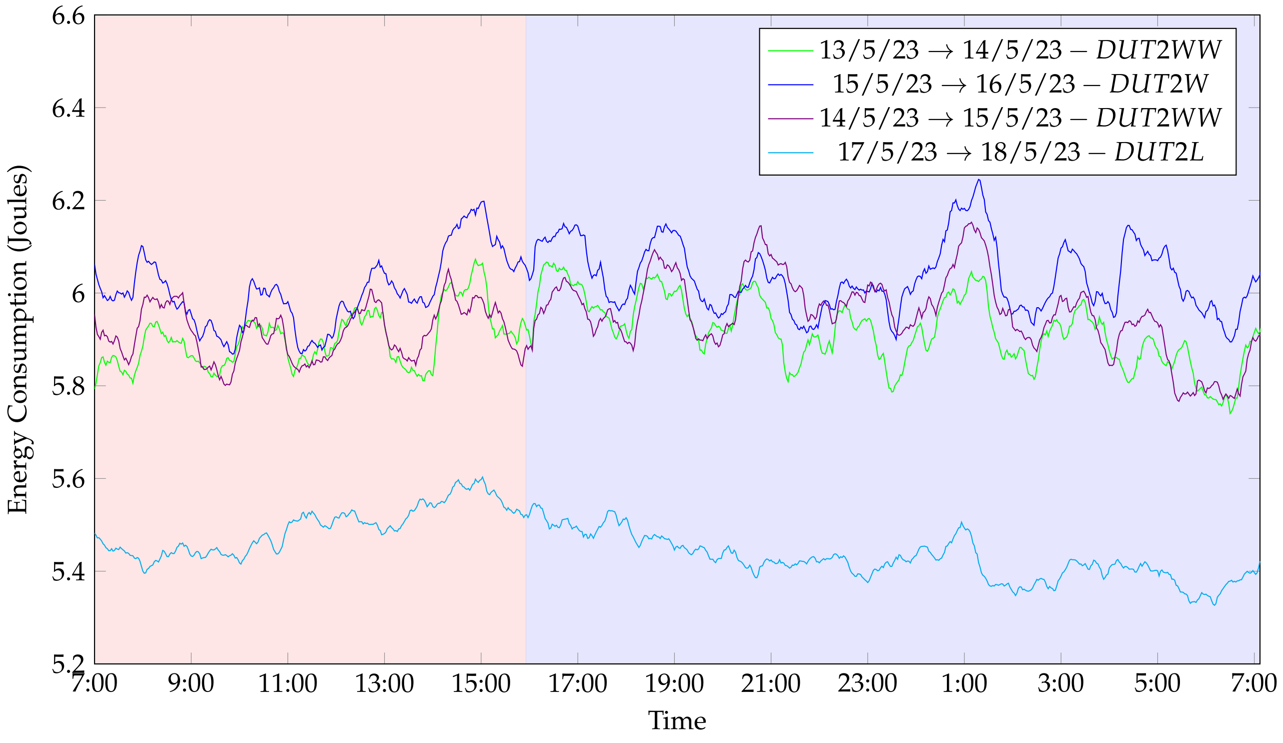


Figure 32: This shows the difference in energy consumbtion, between working hours and non-working hours, for DUT2. The red represents the working hours and the blue represents the non-working hours. The first letter is the OS and the second denoted if it is a weekend. W = Windows, L = "Linux"

To explore **H1** regarding electrical network noise interfering with phase synchronization, measurements were analyzed for both DUTs and OSs over 24 hour periods, where the DUTs executed the same benchmark. The data was then categorized into working hours (7 : 00 to 16 : 00) and non-working hours (16 : 00 to 7 : 00), where no significant variation in power consumption peaks between the two categories was found. However, periods of low energy usage were higher during `working hours`, which suggested that they did not affect benchmark measurements but did impact idle case measurements. This observation aligned with the results presented in Figure **3**, which represent the DEC values. One reason this trend might be easier to see on the idle case is that PSUs are less efficient at low loads, which causes reactive energy to contribute more to overall usage.[83]

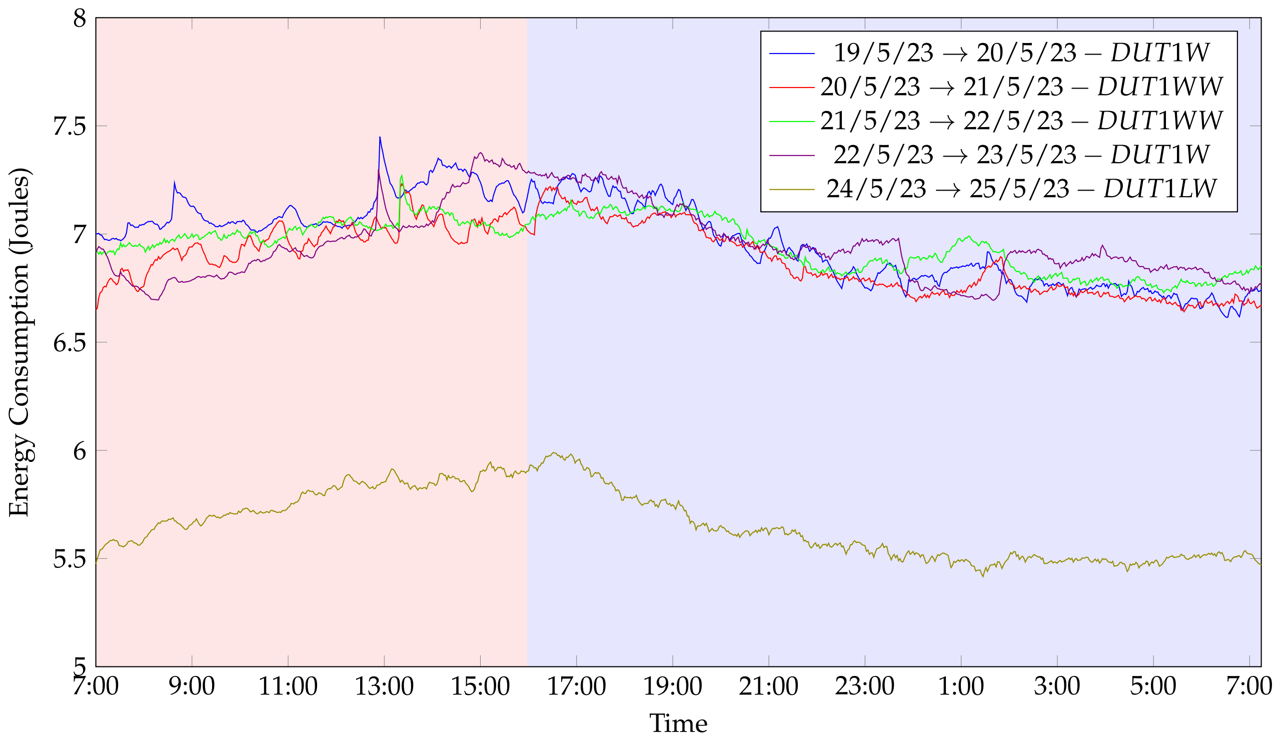


Figure 33: This shows the difference in energy consumbtion, between working hours and non-working hours, for DUT1. The red represents the working hours and the blue represents the non-working hours

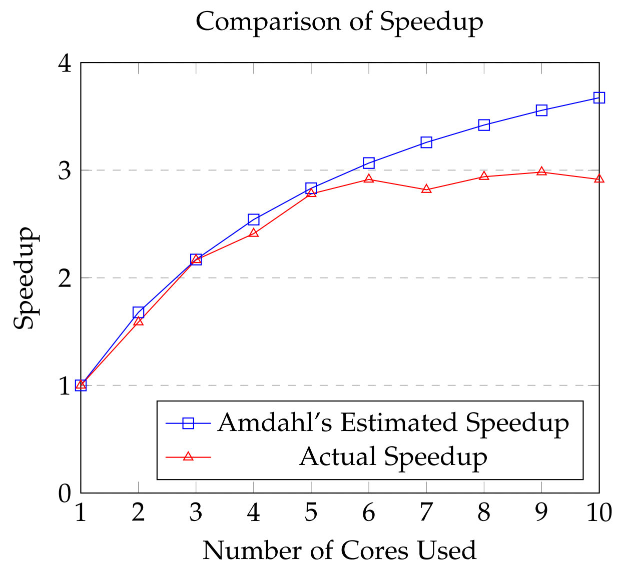
The impact of reactive power on low loads was tested by making idle measurements on both DUTs during weekdays and weekends, with the results presented in Figures **32** and **33**. The results showed differences between the energy consumption between working and non-working hours, although minimal. It could also be observed from the results that the same pattern is present during the weekends, but to a lesser degree. It should be noted that the recorded measurements were taken close to the end of a semester, which could influence the results. For DUT2 in Figure **32**, when comparing OSs, Windows had a more considerable difference between peaks and valleys, occurring approximately once every two hours. The reason why these spikes occur is likely scheduling jobs.Why these peaks only occurred on DUT 2, even with the same versions of Windows, is a subject for future work.

A similar observation of varying energy consumption between working and non-working hours has, to our knowledge, yet to be addressed in existing works[43, 76, 77]. While these studies were not directly comparable, we anticipated some resemblance, indicating that previous research utilizing hardware measurements might have needed to be more extensive, to reveal this trend.

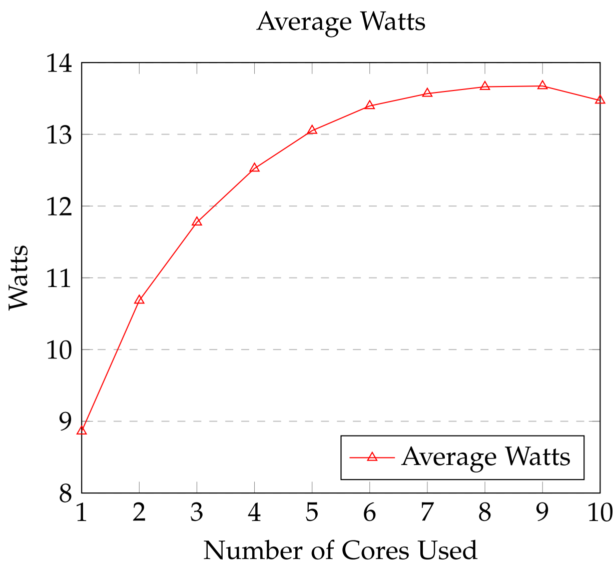
Through this analysis, it was found that there is a relationship between energy consumption and the time of day. It seemed to be affected by the number of people on the electrical network, as suggested by **H1**. These observations were made on two DUTs and two OSs over multiple days, showing the same pattern, ideally more measurements would have been made, but due to time constraints it is limited. When comparing day and night, the energy consumption was higher during working hours compared to weekends and nights, but whether the cause is reactive energy or some other unaccounted-for factor is a subject for future work.

R Estimated Speedup Using Amdahl's Law

We estimated Amdahl's Law on the 3DM benchmark on DUT 2, which required us to know the parallelizable part of the benchmark. The parallelizable part had to be estimated because we did not have the source code for the benchmark. Each measurement starts with a period where the system is idle, then 3DM executes, followed by another period of the DUT being idle. We assumed that the period where 3DM is executed is the parallelizable part of the whole measurement, while the periods before and after are serial. Based on this assumption, we gathered estimates of the parallelizable and serial parts of the measurement. Then we computed the estimated speedup and compared it with the actual speedup.



(a) A comparison on the estimated speedup from Amdahl's law and the actual Speedup for 3DM on DUT 2



(b) Average watts for 3DM on DUT 2

Figure 34: Speedup up and Average Watts for DUT2 on 3DM

On Figure **34a**, the estimated and actual speedup was illustrated at different amounts of cores. From using 1-6 cores, the benchmark was executed using P-cores, while from 7-10, progressively more E-cores were utilized. This can be observed as the estimated and actual speedup follows closely until the DUT uses the E-cores. It should be noted that Amdahl's law is not meant for asymmetric CPUs and, as such, does not account for the difference in computational power of the E-cores. The effect of this was that for 7-10 cores, the estimated speedup did not follow the actual speedup. The actual speedup results showed that the E-cores could not contribute to speeding up the benchmark. On Figure **34b**, it was illustrated that after six cores, the average watts increased slower than from 1 to 6 cores, while the E-cores did not provide a speedup on the benchmark, they also did not increase the energy consumption as much as the P-cores.

S ChatGPT

In this work, ChatGPT, the language model made by OpenAI, has been used to a limited extent to replace Google search. This for example included assistance to setup Scaphandre given the limited documentation, and assistance with C++ given our limited experience with this language. ChatGPT also proved useful when learning a new domain by giving the keywords that could be used for further research on Google. ChatGPT has not given us any information we could not have found on Google, and has not been used to analyze any results or write any of this project.