Summary

The paper 'Exploring the Energy Consumption of Highly Parallel Software on Windows' investigates several different challenges and aspect of this process will be covered, with a primary focus on doing it on the windows operating system. The research area of energy consumption of software is inherently Linux centric, which avoids some of the measuring challenges that are present on windows. This has caused there to be a large body of knowledge regarding linux measurements, how should they be conducted? and what tools can be utilized. This paper aims to understand and overcome these challenges to make measurements on the most commonly used operating system more accessible. This includes finding and evaluating tools available on windows. The research will be conducted on both Linux and Windows, and two different DUTs, one with homogenous cpu architecture and one with a newer heterogenous, several benchmarks will be used to test and measure them in different situations, to observe the differences.

To achieve this goal four research questions have been formulated to assist in this.

The first research questions covers the evaluation of different C++ compilers, to find the most energy efficient compiler, to run the microbenchmarks on, that is then used throughout the paper. The selected microbenchmark are the Fannkuch-redux and Mandelbrot, both being highly parallel benchmarks. The result here is that the Intel oneApi best C++ compiler, this is because it was found to be the most energy efficient compiler and also the fastest in terms of execution speed.

The second research question looked into which measurements tools worked best interms of accuracy, ease of use, and availability. When evaluating this a groundtruth was introduced, in the form of a current clamp which measured all the energy consumed by the system. The software based tools then used to measure a time series for the benchmarks, these were then compared to the groundtruth using several statical methods, Shapiro-Wilk to see whether or not the data was normally distributed, Mann-Whitney U Test if they are statically different from eachother, and finally Kendall Tau Correlation Coefficient to get the correlation between the measurement instruments.

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

May 19, 2023

Abstract

With the evolution of CPUs over the last few years, increasing the number of cores has become the norm. Through four research questions, this research investigates the performance gains obtained from the additional processing power and the impact of the P- and E-cores on parallel software. The analysis uses benchmarks, considering energy consumption and execution time on a per-core basis and an increasing number of cores. The experiments are conducted primarily on Windows, where Intel's Running Average Power Limit is unavailable, with Linux as a reference point. To compare the performance of measuring instruments made for Windows when running both micro- and macrobenchmarks, we refer to where the body of knowledge is more extensive in this domain, Linux.

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. [1, 2] As the use of ICT rises, the demand for computational power also rises. Therefore energy efficiency has become more of a concern for companies and software developers.

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 Device Under Tests (DUTs): an Intel Coffee Lake CPU with a traditional performance core setup and an Intel Raptor Lake CPU with performance and efficiency cores (P- and E-cores). We analyze the impact of Asymmetric Multicore Processors (AMPs) on parallel execution compared to traditional Symmetric Multi-Core 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 noises 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 larger macrobenchmarks. The macrobenchmarks will be run on an increasing number of cores to explore the benefits of runtime and energy consumption. The following research questions are formulated to assist with the process:

- RQ**1**: How does the C++ compiler used to compile the benchmarks impact the energy consumption?
- RQ**2**: What are the advantages and drawbacks of the different types of measuring instruments in terms of 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 assist with running a series of different experiments.

In Section **2** the related work which lay the foundation for our work is covered, including our previous work. This is followed by Section **3** which includes the necessary background information about e.g. CPUs and schedulers. Thereafter in Section **4** our experimental setup is presented. In Section **5** the results are presented whereafter they are discussed in Section **6** and finally a conclusion is made in Section **7**.

2 Related Work

This section provides an overview of related work in energy consumption, parallel software, compilers, and asymmetric multicore processors. It also builds upon our previous work in comparing measuring instruments.

2.1 Previous Work

In our previous work *"A Comparison Study of Measuring Instruments"*[3], 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 have similar correlation to hardware-based measuring instruments as Intel's Running Average Power Limit (RAPL) has on Linux. The remainder of this chapter builds upon the related work chapter in [3].

2.2 Variations in Energy Measurements

In [3] found that energy consumption measurements can vary between measurements. This topic is also explored in [4], where it was discovered that numerous variables affect energy consumption measurements. This was explored through experiments on 100 nodes to investigate the impact of controllable parameters and achieved 30 times lower variation.

One such parameter is temperature, which has produced conflicting conclusions. [5] observed energy consumption variation on identical processors, without any correlation between temperature and performance. In contrast, [6] found the opposite to be true. In [4] an experiment was performed, where benchmarks were executed on three different configurations, right after each other, with a one minute sleep between benchmarks executions and a restart between benchmark executions. The configurations were not found to have an impact on the energy variation.

[4] also examined the effect of C-states on energy variation. 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 [7] had previously found that disabling Turbo Boost reduced variation from 1% to 16%, [4] could not find any evidence supporting this.

An experiment in [4] explored whether the overhead introduced by Linux and its activities and processes affected energy variation. Disabling non-essential processes, such as Wi-Fi and logging modules, yielded no substantial difference. [8, 9] conducted experiments on CPUs of different generations to examine how energy variation differs between them, finding that older CPUs exhibited lower deviation. Although a similar experiment in [4] 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)[1] deviated less.

2.3 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.[11] In [12] Amdahl's law, was extended to also estimate the energy consumption, after which three different many-core designs were compared with different amounts of cores using the extended Amdahl's law. The comparison showed that a CPU can lose its energy efficiency as the number of cores increases and it was argued that knowing how parallelizable a program is before execution allows for calculating the optimal number of active cores for maximizing performance and energy consumption.[12]

[13] compares 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 the speedup of a parallel program based on the size of the problem and the number of cores. Unlike Amdahl's law which assumes a fixed problem size and a fixed proportion of the program that can be parallelized, Gustafson's law takes into account that larger problems can be solved when more cores are available and that the parallelization of a program can scale with the problem size. Comparing the observed and estimated speedup it was clear that Gustafson's law is more optimistic than Amdahl's law, where both underestimated the speedup on two and four cores.[13]

In [14], three different thread management constructs from Java are explored and analyzed. When allocating additional threads, the energy consumption would first 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, though, 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.[14]

In [15], found that a larger number of cores in the execution pool results in a lower running time and energy consumption, and conclude that parallelism can help reduce energy consumption for genetic algorithms.

In [16], four different language constructs which

[1] The power consumption under the maximum theoretical load[10]

could be used to implement parallelism in C# were tested. The test was conducted on varying amounts of threads and a sample of micro- and macro-benchmarks. They found that workload size has a large influence on run time and energy efficiency and that a certain limit must be reached before improvements can be observed when changing a sequential program into a parallel one. Comparing micro- and macro-benchmarks the findings remain consistent, although the impact becomes lower for the macrobenchmarks due to an overall larger energy consumption.

2.4 Compilers

In [17], several C++ compilers were compared, with the goal of finding a balance between performance and energy efficiency. The study was conducted on different microbenchmarks, where the effect of different coding styles was explored.The C++ compilers used in [17] included MinGW GCC, Cygwin GCC, Borland C++, and Visual C++, and the energy measurements were taken using Windows Performance Analyzer (WPA). The compilers were used with their default settings, and no optimizations options were used. They 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, which involved an election sort algorithm, 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 both decreased energy consumption.

2.5 Asymmetric Multicore Processors

Asymmetric Multicore Processors are CPUs where not all cores are treated equally. One example of this could be the combination of P- and E-cores, as seen in Intel's Alder Lake and Raptor Lake. Intel's Thread Director (ITD) was introduced alongside Intel's Alder Lake, where the purpose of ITD was to assist the operating system (OS) when deciding which cores to run a thread. In [18], support for utilizing ITD in Linux was developed, and some SPEC benchmarks was conducted to analyze the estimated Speedup Factor (SF) from the ITD compared to the observed SF. SF is the relative benefit a thread receives from running on a P-core. The study examined which classes were assigned to different threads in the benchmark and found that 99.9% of class readings were class 0 or 1. Class 0 are for threads performing similarly on P- and E-cores, while Class 1 are for threads where P-cores are preferred.[19] Class 3, which are for threads preferred to be on an E-core, was not used. The experiment indicated that the ITD overestimated the SF of using the P-cores for many threads but also underestimated it for some threads. 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. 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. It found that the PMC-based model provided superior SF predictions compared to ITD.[18] Official support for ITD has since been released.

3 Background

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

3.1 CPU States

CPU-states (C-state) manage a systems energy consumption during different operational conditions. On a CPU, each core has its own state, which dictates how much the core is shut down in order to conserve power. The C0 state represents the normal operation of a core under load.[20, 21] The number of states vary between CPUs and the number of supported states vary between motherboards. The CPU used in [3] 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.[3].

The C-states can have a large impact on the energy consumption of the benchmarks, especially the idle case as was found in [3].

3.2 Performance and Efficiency cores

For the CPU architecture x86, the core layout has historically comprised of identical cores. However, the ARM architecture introduced the big.LITTLE layout in 2011[22]. big.LITTLE is an architecture utilizing two types of cores, a set for maximum energy efficiency and a set for maximum computer performance.[23]. Intel introduced a hybrid architecture in 2021[24] similar to big.LIGGLE, 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 focusing 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[25].

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 the process only executes on the assigned core(s) each time it is scheduled.[26]

When setting the affinity for a process in C#, it is done through a bitmask, where each bit represents a CPU core. An example is found in Listing **1**, where the process is allowed to execute on core #0 and #1.

```
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#

3.4 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 with care, as other threads in the system will not get any processor time while that process is running. 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.[27]

```
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.[27]

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 both the thread priority level and the priority class, where a table showing the scheduling priority given these two parameter can be found in [27]. 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 both 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, with support for multiple OSs and compilers.[28] 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.[28]

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.[28]

```
#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.[28]

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 that is not utilized by the circuit[29]. The ratio between active and reactive energy is known as the power factor[29].

4 Experimental Setup

In the following section a detailed description of the equipment, benchmarks, and procedures used in the study will be presented.

4.1 Measuring Instruments

The measuring instruments used in this work are based on what was used in [3], with a few additions. Thew new additions will get a more detailed introduction, while the others are briefly introduced but can be found with more detail in [3].

Intel's Running Average Power Limit (RAPL): is a Mac and Linux exclusive software-based measuring instrument most frequently used in the litterature.[3]. RAPL uses model-specific-registers (MSRs) and Hardware performance counters to calculate how much energy the CPU uses. The MSRs used by RAPL are those for the power domains PKG, DRAM, PP0, and PP1, covered in [3].

In [3] RAPL was found to have a correlation of 0.81 with the ground truth.[3]

Intel Power Gadget (IPG): is a Windows and Mac exclusive software tool created by Intel, which can estimate the energy consumption on Intel processors. IPG uses the same hardware counters and MSRs as RAPL[30], and is therefore expected to have similar measurements to RAPL, which was found to be the case in [3]. [3] 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.[3]

Libre Hardware Monitor (LHM): is a fork of Open Hardware Monitor, without a GUI, and is supported on Windows and Linux.[31] Both projects are open source and LHM uses the same hardware counters and MSRs as RAPL and IPG. In [3] 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 in order to measure and log measurements continuously.[3] The intrinsic error is reported to be 2%[32].

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.[33]

Scaphandre (SCAP): is a monitoring agent able to measure energy consumption.[34] SCAP is designed for Linux and uses RAPL and can in addition to this measure the energy consumption of some virtual machines, specifically Qemu and KVM hypervisors. SCAP can also be used on Windows, as a kernel driver exists which allows SCAP to read RAPL measurements from Windows.[35]. The Windows version of SCAP can report the energy consumption of the power domain PKG using the MSR. 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, the performance of SCAP and SCAPs ability to isolate the energy of a process will both be used, where the latter will be referenced as SCAPI.

4.2 Dynamic Energy Consumption

Dynamic Energy Consumption (DEC) represents a way to isolate the energy consumption of a process and was utilized in [3, 36]. DEC was used to enable a comparison between 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 [36] 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.[36]

4.3 Statistical Methods

In this sections the statistical method used to analyze the results are presented. This section was based on [3] and can be referred to for further detail.

Shapiro-Wilk Test: was used to examine if the data followed a normal distribution. The data is not expected to be normally distributed, as this was the finding in [3]. Understanding the distribution of the data was important, as some statistical methods assumes the data is normally distributed.[37]

Mann-Whitney U Test: to evaluate if there was a statistical significant difference between samples, the Mann-Whitney U Test was used. The Mann-Whitney U Test is a non-parametric test which does not assume normality in the data.[38]

Kendall's Tau Correlation Coefficient: is a non-parametric measure of association able to evaluate the strength and direction of relationships between ordinal variables, when the underlying data does not adhere to a normal distribution.[39]. Kendall's Tau Correlation Coefficient was used to asses the correlation between measurements and is evaluated using the Guilford scale in Table **1**.[40, p. 219]

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 [40, p. 219]

Cochran's Formula: is used to determine an appropriate sample size for how many measurements are required. With this formula a required sample size to achieve a desired level of confidence can be calculated.[41]

In summary, the selection of the Shapiro-Wilk test, Mann-Whitney U test, Kendall's Tau correlation coefficient, and Cochran's made it possible to analyze the non-normal distributed data obtained, while determining statistically significant differences, correlations, and an appropriate sample size for the 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 conducted. Both DUTs were chosen to enable comparison between CPUs with and without P- and E-cores. When the two DUTs were set up, they were updated to have the same version of Windows and Linux. In Tables **2** and **3** the specifications of the two workstations can be seen, referred to as DUT 1 and DUT 2.

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 introduces the various C++ compilers used in the first experiment. MSVC and MinGW were included as [17] found those to exhibit the lowest energy consumption. Additionally, the Intel's oneApi and Clang were included as both can be found on lists of the most popular C++ compilers[42–44]. The versions of the compilers is illustrated in Table **4**
Clang: is an open source compiler building on the LLVM optimizer and code generator. It is available for both Windows and Linux[45]

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.[46]

Intel's oneAPI C++ (oneAPI): is a suite of libraries and tools aimed at simplifying development across different hardware. One of these tools is the C++ compiler, which implements SYCL, this being an evolution of C++ for heterogeneous computing. It is available for both Windows and Linux.[47]

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[48].

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

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 are selected, are 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 may not provide an accurate representation of overall performance.[49]

The microbenchmarks used in this work are from the Computer Language Benchmark Game [2]. The selected benchmarks included both single- and multi-threaded microbenchmarks, which were compatible with the compilers and OSs used in this work. Certain libraries, such as `<sched.h>`, were used in many implementations and were not available on Windows, which limited the pool of compatible microbenchmarks. The chosen microbenchmark benchmarks and their abbreviation are presented in Table **5**, where the parameters are 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.[49] Application-level benchmarks are a type of marco benchmarks testing an application, which provides a more realistic benchmark scenario.

The macrobenchmarks used in this work are made by UL Solutions. The first one was 3DMark (3DM) which is a set of benchmarks for scoring both GPU's and CPU's based on gaming performance. From 3DM, the CPU Profiler benchmark was used, as this work focuses on the energy consumption of the CPU and not the GPU. The CPU Profile benchmarks runs a 3D graphic, but the main component of the workloads is from a boids flocking behavior simulation.[50]. The second macrobenchmarks was PCMark 10 (PCM) which is a benchmark meant to test various different task seen at a workplace. PCM has three test groups including e.g. web browsing, video conferencing, working in spreadsheets and photo editing, the full list can be seen in Tables **13** and **14**.[51] 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

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

4.7 Background Processes

To limit background processes on Windows, a few step were taken. 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 are 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 was set to `High`, unless specified otherwise by the particular experiment. In addition to this, suggestions made by [4, 16] were followed, meaning C-states, Turbo Boost and hyper-threading was disabled. On Linux, Wi-Fi was disabled when benchmarks were running, but no background processes were stopped as no effect of doing this was found by [4]. 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 [4] 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 [3] found that to be fitting.

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

5.1 Experiment One

The first experiment investigated RQ **1**. This experiment 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 [3]. This experiment was made based on an hypothesis that the different compilers would produce assembly with a varying energy consumption and execution time, as was also found in [17].

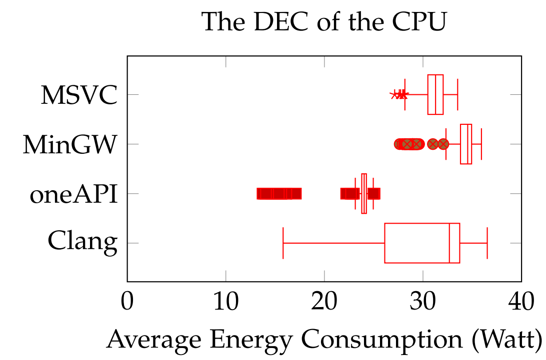


Figure 1: CPU measurements by IPG on DUT 1 for benchmark(s) FR

Compiler Initial Measurements: As was presented in subsection **4.3**, Cochran's formula was used to ensure there was confidence in the measurements made. The initial measurements were taken to gain insight into the number of measurements required before making additional measurements if required. The number 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[52].

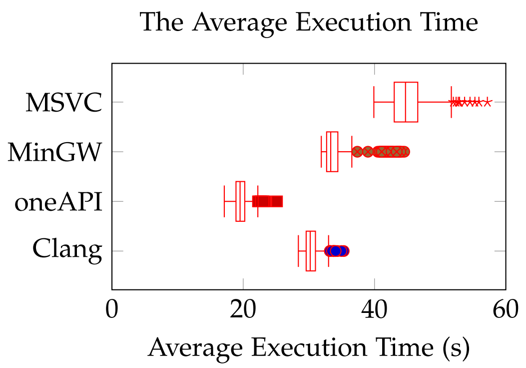


Figure 2: Execution time measurements by IPG on DUT 1 for test case(s) FR

After the initial 30 measurements, Cochran's formula was applied to the measurements, and the required measurements was illustrated in Table **8**, where it was evident that the required samples varied between compilers and benchmarks. When the benchmarks were analyzed it was found that MB deviates less than FR, with MB requiring as little as 3 measurements with MinGW, while FR requires 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. oneAPI also displayed the lowest energy consumption. 550 additional measurements were conducted for the next step.

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 stands out where 61.086 measurements were required. Because this number was so much higher than other compilers, additional measurements were taken using this compiler. After 10.000 measurements, Cochran's formula now 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 **12**, and for MB in Appendix **E**, oneAPI had the lowest DEC and execution time for both benchmarks. Clang deviated the most in Figure **1**.

In the first experiment, it was concluded that the different compilers have a huge impact on the DEC, execution time and on how many measurements were required. In the end, oneAPI had the lowest DEC and execution time and was used in the next experiment.

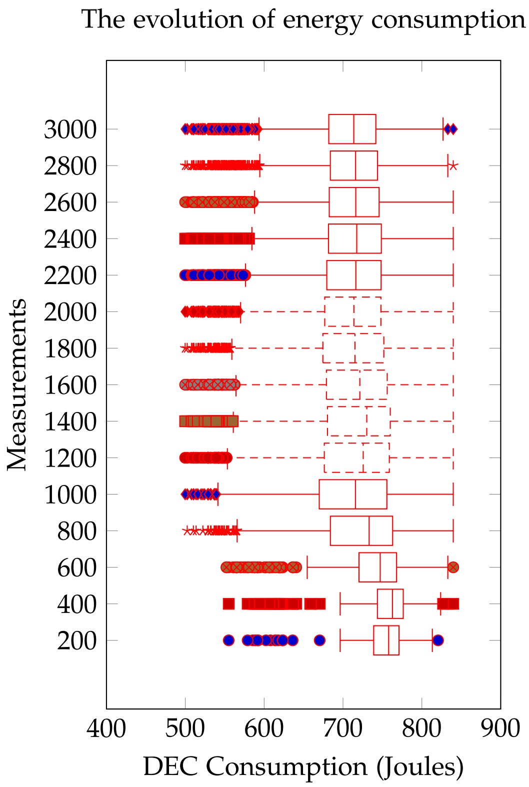


Figure 3: A visual representation of how the energy measurements evolve as more measurements are made by Clamp on DUT 2 for benchmark MB

5.2 Experiment Two

The second experiment investigated RQ **2**, in order to identify the best measuring instrument on Windows. The best measuring instrument was in this study be based on a combination of different 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 benchmark 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 of the benchmark to $\sim$ 14 seconds. This avoided a scenario where the Plug only had a single data point per measurement. For this experiment, FR was executed 550 times, while MB was executed 222 times, based on Table **8**.

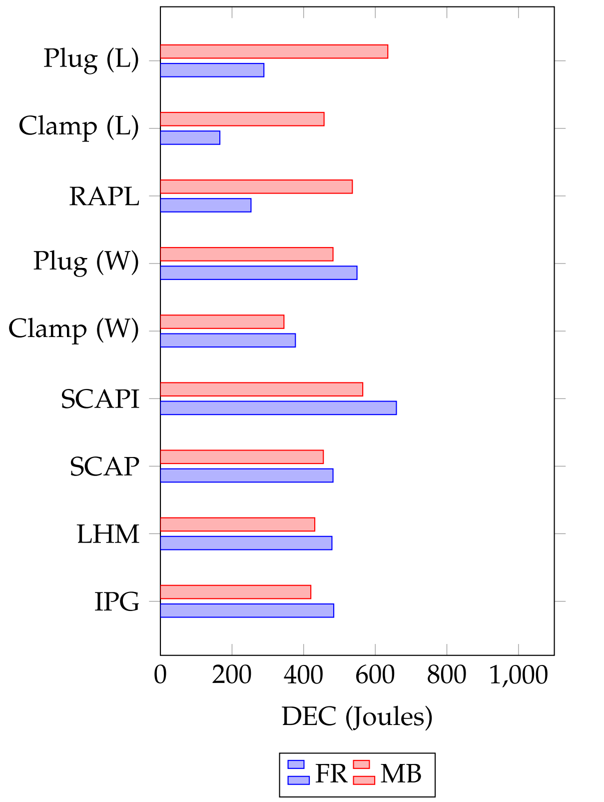


Figure 4: The average DEC for DUT 1, where both benchmarks are compiled on oneAPI

Measuring Instrument Initial Measurements: the required number of measurements for this experiment, found by applying Cochran's formula to the measurements, can be found in Appendix **G**. From Appendix **G** it was found that the Clamp required more measurements compared to other measuring instruments, which is why 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**. In Figure **3** the evolution of the average DEC is illustrated, as more measurements were made, where the median DEC was found to decrease by 5.84% between when 200 measurements were made to when 3.000 measurements were made, and by 0.3% between 2.800 and 3.000 measurements. A pattern was observed, where the median decreased as more measurements were made, until measurements 1.000, after which the average DEC increased until measurement 1.400 by 2%, after which it decreased again. In the last 1.400 measurements the average DEC had converged and only increased by 0.2%. The average DEC at 1.000 measurements was 0.29% from the DEC at 3.000, and due to the time required to run the experiments, the maximum amount of measurement were capped at 1.000 for this experiment. After 3.000 measurements, the number of measurements required ended up being 15.137. This number is higher compared to other measuring instruments, and this will be analyzed further in the discussion. In Appendix **H** a graph is illustrated, showing how many measurements Cochran's formula indicated would be required, as the number of measurements increased.

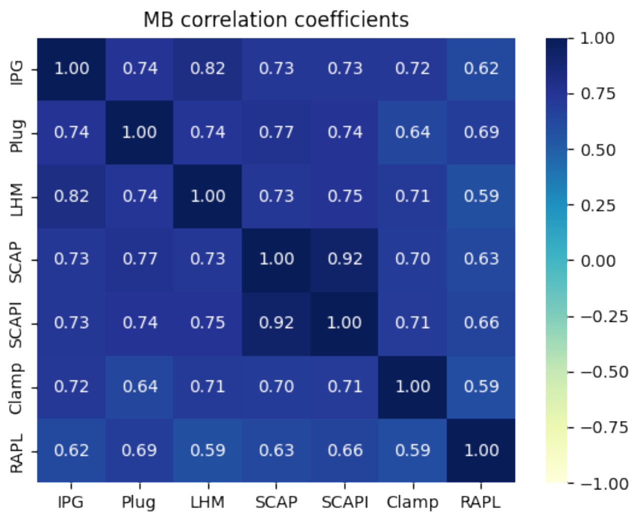


Figure 5: Heatmap showing the correlation coefficient between all of the measurement instruments for MB on dut 1

Measuring Instrument Results: the results for this experiment, is presented in Figure **4**. In Figure **4**, MB was found to consume less energy than FR for Windows, where the opposite was the case for Linux. 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, where RAPL over reports compared to the Clamp in all cases, which was also found on Windows for all measuring instruments.

When the statistical method from subsection **4.3** were applied, the data was was found not to follow a normal distribution, was was also found in [3, 53].

The correlation between the measuring instruments for MB on DUT 1 can be seen 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. The Plug was found to have a moderate correlation of 0.64 to the Clamp. The correlations for FR were found to be more correlated compared to MB, but still within the same categories of either a moderate or high correlation, except for on linux on DUT1. All of the correlation results can be found in Appendix **G**. The similar correlation between the different software based measuring instruments was expected, because of them using the same hardware counters and MSRs. 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.

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

Table 9: The performance difference between cores of the same type

5.3 Experiment Three

The third experiment investigated RQs **3** and **4**, by taking a loot at 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, an analysis on the per-core performance of both CPUs was conducted, 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. After, Cochran's formula was applied to the result to determine if more measurements were required. The results from this can be found in Appendix **J**, where additional measurements were made accordingly.

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 10: The average performance difference between P and E cores on DUT 2, SN

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

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 11: The average performance difference between P and E cores on DUT 2, NB

When comparing how much performance could differ between cores of the same type, on the same CPU, a table illustrating the average energy consumption and how much it deviated was illustrated in Table **9**. Table **9** illustrated DUT 1 deviates more, with the highest deviation for NB, while the lowest was for DUT 2, on P cores.

Macrobenchmark Initial Measurements: following the analysis of the per-core performance, the two macrobenchmarks introduced in subsection **4.6** was executed on an increasing number of cores, starting from the most efficient one. An initial 30 measurements were made, as the per-core experiment illustrated how 250 were too much for DUT 2, illustrated in Appendix **J**. 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 was unable to execute spreadsheet and photo editing for unknown reasons. Because of this, DUT 2 will start at 2 cores to include web browsing. For DUT 1, web browsing was unable to execute, so this scenario was excluded for this DUT. Despite attempts, we were unable to figure out why PCM had were unable to run certain scenarios on few cores, as no error logs were created when the error occurred and the error was presented as an unknown error by PCM. After the initial 30 measurements, Cochran's formula was applied to the data, to ensure enough measurements were made. The amount of required measurements can be found in Appendix **K**.

Macrobenchmark Results: The results for DUT 1 can be seen in Figure **6** and Figure **7** for 3DM and PCM respectively, and for DUT 1 in Appendix **L**.

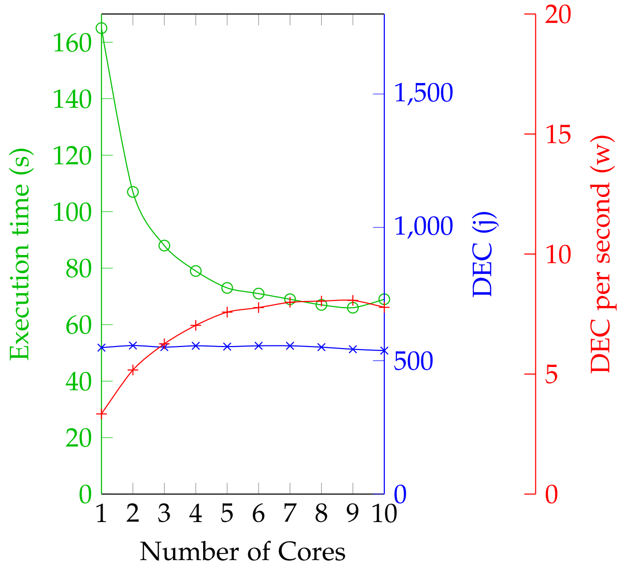


Figure 6: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to 3DM on DUT 2

For both DUTs and macrobenchmarks, a similar observation was made, 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, which was because PCM include scenarios only utilizing a single thread. This meant that only a part 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. This meant the numbers reported in Figure **6** would be even higher if the parallel part were isolated. The diminishing return gained from allocating more resources to PCM is also illustrated, discussed and compared to 3DM in Appendix **N**.

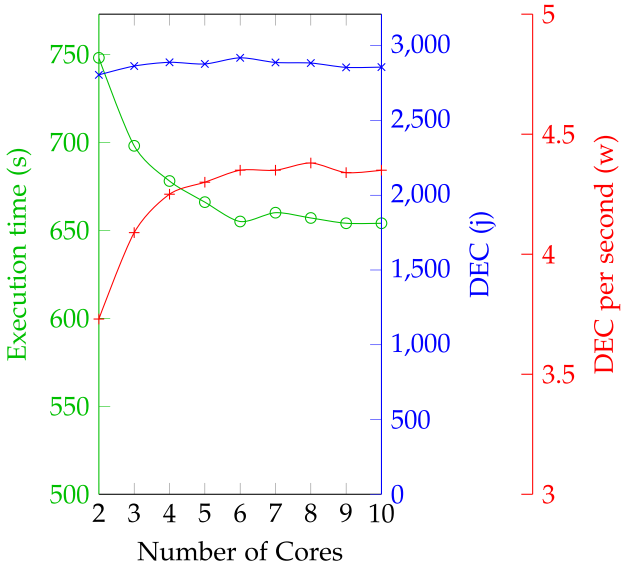


Figure 7: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to PCM on DUT 2. Note that the x- and y- axis does not start at zero.

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 running 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). This was because PCM would not utilize the entire CPU, meaning that the P and E cores could be used when the OS sees it fit. For this experiment, 30 initial measurements were made, and additional were made after Cochran's formula was applied to the results, if required. This can be found in Appendix **L**.

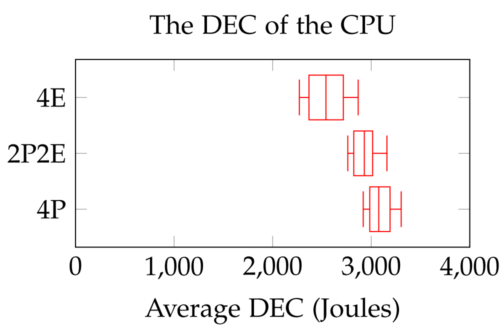


Figure 8: CPU measurements by IPG on DUT 2 for test case(s) PCM

P vs. E Results: The results for the execution time and DEC are illustrated in Figure **9** and Figure **8** respectively, while the DEC per second could be found in Appendix **M**. For the DEC, 4E was found the use the least energy, 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 the use case of E cores, where a lower energy consumption could be achieved, given a higher execution time.

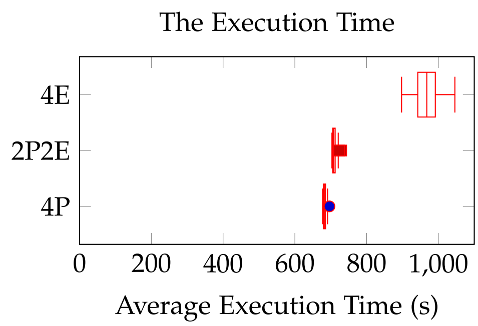


Figure 9: Execution time measurements by IPG on DUT 2 for test case(s) PCM

6 Discussion

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

6.1 Deviating Results

In this work Cochran's formula was used to determine how many measurements were required in order to gain confidence in our results. When analyzing the results from Cochran's formula, it was found that the amount of measurements could deviate a lot between benchmarks, measuring instrument, DUTs and even cores on the same DUT.

Results can deviate from core to core 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.[54] In our work we explored how much the variability in the fabrication process can effect the performance of the cores, and found the energy consumption could have a standard deviation of up to 5.67 among cores of the same type.

When comparing the results from Cochran's formula 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 all software was the same version. Both DUTs executed on a fresh install of windows, and 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 [4]. [4] however found that a lower TDB results in a lower energy variation, and DUT 2 had a TDP of 65 W opposed to DUT 1 with a TDP of 95 W which could explain the difference[55].

6.2 C++ Benchmarks Analysis

In the first experiment presented in subsection **5.1**, different C++ compilers were compared. Through this experiment 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. This could be because the benchmark was removed as dead code by the compiler, which is why an analysis was conducted in Appendix **O**, 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 unique intel functions, Advanced Vector extensions to perform calculations in parallel and loop unrolling. Opposed to oneAPI, MinGW used general purpose registers more in a combination with the C++ Standard Library.

6.3 Energy usage trends

The trend where the DEC decreases 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 we hypothesized that the observed reduction in energy consumption may be caused by changes in the reactive energy[29] consumption.

Based on this, two hypotheses have been constructed to explain why the energy consumption of the DUTs are changing over measurements. Firstly, noise on the electrical network could interfere with the phase synchronization. This may be due to many machines being connected to the same electrical network, and disrupting the harmonics of the network[56]. However, if the power supply generates reactive energy because it is out of phase with the electrical network, a reduction in noise could help synchronize them again. Therefore, the observed changes in energy consumption may be related to the time of the day and week where the measurements are taken, with consumption decreasing when there is less devices connected to the electrical network, during the night and weekends.

Alternatively, the DUTs' PSU may be correcting the phase over time. PSU's can contain a power factor correction circuit that attempts to reduce the amount of reactive power by correcting the phase. There are two main types of power factor correction: passive and active [57]. The behavior seen in the results may be the result of an active power factor correction circuit. We reached out to both of the manufactures of the PSU in used in our DUTs. Corsair confirmed that the PSU in DUT 1 does has Power Factor Correction, however more information could not be supplied due to it being considered internal. Therefore we could not determine which type of Power Factor Correction was used. Cougar did not reply to out email.

To try and confirm or reject these, smaller additional measurements were made to compare the measurements from night and day to see if the trends in the two are different to confirm or reject the first hypothesis the results shows that the seems to be an increasing trend during the working hourssubsection **P**. This also show that the same trend is present in the weekends but to a lesser degree, as the energy consumption is lower throughout the day. The same trend is present on Linux, but generally consumes a bit less energy.(UPDATE WITH LINUX AND DUT1 measurements later)

For the second hypothesis, a mail was writing to one of the producers of the power Supplies specifically Corsair in the DUTs for further details. They confirmed that the power supplies did contain some form of PFC, but would not go into more details due to trade secrets. We are unable to determine the exact cause of the changes in energy consumption.

Previous research in this field, which also utilizes hardware measurements, has not addressed this phenomenon. For example, [58], [53], and [59] did not report similar findings, although their studies might have had a similar environmental setting to ours. While these studies are not directly comparable, we would have anticipated some resemblance, indicating that previous research utilizing hardware measurements might not have been extensive enough, as this trend has not been revealed previously to our knowledge.

6.4 Time synchronization

When measuring the ground truth, four different devices are used. These devices include the DUTs, a Raspberry Pi, and an Analog Discovery 2. Each of these devices kept its own time, which could cause issues if they were not synchronized. This was particularly problematic for external measurement instruments, as even small differences in time 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[60] and would therefore become increasingly inaccurate over time. Additionally, the execution time of IO events for the clamp and plug could result in a slight time difference, although this is expected to have minimal impact on the results, since resynchronization happens every second but this is a subject for a future work.

6.5 Windows

Compared to the litterature, this work stood out by its use of Windows over Linux[58, 59, 61]. Windows was interesting as it was a very popular OS, and because the only study who looked into measuring instruments and energy consumption on Windows, to our knowledge, was [3].

When comparing results between Linux and Windows in Appendix **F**, Windows was found to have a lower DEC, similar to what was found in [3]. 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 they compatibility for Windows. This was a problem in the first experiment, where the benchmarks had to be compatible for all four compilers on Windows. The original idea was also to find macrobenchmarks written in C++, compiled on the most energy efficient compiler, which we were not able to find. Instead PCM and 3DM was chosen, where each had their own issues. For PCM, each DUT had some scenarios it was unable to run, 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. 3DM also caused bluescreen with stop code `VIDEO_TDR_FAILURE` on DUT 2 in rare cases, which was found to be 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

In this work, Cochran's formular was used to ensure enough measurements were taken. In the 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. This means 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 or error of 0.03, or 95% if the margin or 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.

6.7 Estimated Speedup Factor

BRRRRRR GO FAST!

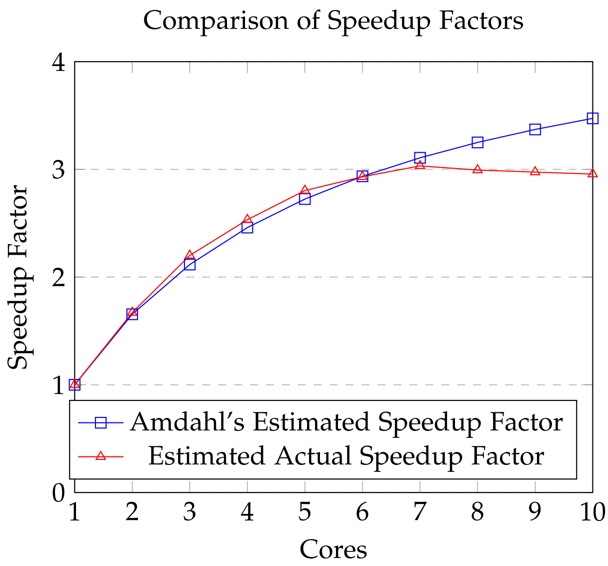


Figure 10: A comparison on the estimated speedup factor from Amdahl's law, Gustafson's law and the actual Speedup factor for 3DM on DUT 2

7 Conclusion

This work explores parallelism, P- and E-cores and how this effects energy consumption and execution time, with a primary focus on Windows, with Linux as a reference point. This study is based on four research questions about areas not explored in the literature. The first research question revolves around the impact the compiler has when compiling benchmarks, both in terms of energy consumption but also execution time. The second research question looks into different software based measuring instruments for Windows, while the third research question looks into the effect parallelism have on energy consumption, and the forth 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 them. 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. This work also introduces an upper limit of 1.000 measurements, as the gain from additional measurements is found to be limited. While Windows provides valuable depth to the analysis of energy consumption, Linux is overall found to be the more convenient OS choice due to its minimalist nature with less pre-installed software and background processes. We however also find that 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 energy consumption of software.

Since RAPL is not available on Windows, we compare alternatives by measuring energy consumption on C++ microbenchmarks, compiled with the most energy efficient compiler of the ones we test. The most energy efficient C++ compiler is found to be Intel's oneAPI through the first experiment, where a significant difference in performance between compilers is observed. Through an analysis, oneAPI achieves the best performance due to its utilization of AVX for parallelism, and other optimizations, such a loop unrolling.

We test different measuring instruments in the second experiment and decide which to use on Windows by comparing microbenchmarks compiled with oneAPI. A moderate to high correlation between 0.59 - 0.80 is found between the ground truth for the different software-based measuring instruments for Windows, and we expect similar performance between them to be a result of a utilization of the same registers when reporting the energy consumption. We choose Intel Power Gadget as our preferred software-based measuring instrument, because of its usability compared to other measuring instruments. In addition to different software measuring instruments, a cheaper alternative to the ground truth is also included, this being the plug. When comparing the correlation between the plug and clamp, similar correlations to the software based measuring instruments are found. There are also some aspects not included in the comparison between measuring instruments, which could be interesting to include in a future work. This could be by extending the analysis to include factors such as the overhead of the measuring instruments for Windows, to see how they compare to RAPL.

In the third experiment, we analyze the performance of P- and E-cores, which shows a 17.40% higher energy consumption for P cores, while E cores has a 29.52% higher execution time when executing on four cores. This shows that E cores can be used to limit the energy consumption, when a higher execution time can be afforded.

In the third experiment, we explore parallelism and its effect on energy consumption 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 numbers of cores to examine the effects of additional resources. For both macrobenchmarks, similar observations are found, where as more cores are allocated, the execution time decreases, DEC per second increases but the DEC remains the same. This shows that there is no correlation between execution time and energy consumption.

Acknowledgements

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. Holt, J., Kusk, M. H. & Pedersen, J. B. *A Comparison Study of Measuring Instruments* (Aalborg University Department of Computer Science, 2023).

4. 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.

5. 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.

6. 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).

7. 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.

8. 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.

9. 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.

10. *Thermal Design Power (TDP) in Intel Processors* https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html. 23/02/2023.

11. 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.

12. 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).

13. 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).

14. 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).

15. 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).

16. 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).

17. 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).

18. 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.

19. 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.

20. 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.

21. 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.
22. 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.
23. ARM. *Processing Architecture for Power Efficiency and Performance* https://www.arm.com/technologies/big-little. 09/03/2023.
24. 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.
25. Rotem, E. *et al.* Intel alder lake CPU architectures. *IEEE Micro* **42,** 13–19 (2022).
26. *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.
27. Karl-Bridge-Microsoft. *Scheduling priorities - win32 apps* https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities. 03/03/2023.
28. Michael Womack, A. W. *Parallel Programming and Performance Optimization With OpenMP* https://passlab.github.io/OpenMPProgrammingBook/cover.html. 03/03/2023.
29. *MORE POWER WITH LESS COPPER* https://fortop.co.uk/knowledge/white-papers/reactive-power-reducing-compensating/.
30. Mozilla. *tools/power/rapl* https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html. 24/02/2023.
31. source, O. *Libre Hardware Monitor* https://github.com/LibreHardwareMonitor/LibreHardwareMonitor. 03/03/2023.
32. Arnoux, C. *AC current clamps user's manual* 2013.
33. CloudFree. *CloudFree EU Smart Plug* https://cloudfree.shop/product/cloudfree-eu-smart-plug/. 10/03/2023.
34. Hubblo. *Scaphandre* https://github.com/hubblo-org/scaphandre. 23/02/2023.
35. Hubblo. *windows-rapl-driver* https://github.com/hubblo-org/windows-rapl-driver. 23/02/2023.
36. Fahad, M., Shahid, A., Manumachu, R. R. & Lastovetsky, A. A comparative study of methods for measurement of energy of computing. *Energies* **12,** 2204 (2019).
37. 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).
38. 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).
39. Han, A. K. Non-parametric analysis of a generalized regression model: the maximum rank correlation estimator. *Journal of Econometrics* **35,** 303–316 (1987).
40. Guilford, J. P. Fundamental statistics in psychology and education (1950).
41. Cochran, W. G. *Sampling Techniques, 3rd edition* ISBN: 0-471-16240-X (John Wiley & sons, 1977).
42. 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.
43. Pedamkar, P. *Best C++ Compiler* 2023. https://www.educba.com/best-c-plus-plus-compiler/. 20/03/2023.
44. *Top 22 Online C++ Compiler Tools* 2023. https://www.softwaretestinghelp.com/best-cpp-compiler-ide/. 20/03/2023.
45. *Clang Compiler Users Manual* https://clang.llvm.org/docs/UsersManual.html. 20/03/2023.
46. *MinGW FAQ* https://home.cs.colorado.edu/~main/cs1300/doc/mingwfaq.html. 20/03/2023.
47. *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.
48. *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.
49. 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.
50. UL. *3DMark* https://www.3dmark.com/. 14/04/2023.
51. UL. *PCMark* https://benchmarks.ul.com/pcmark10. 14/04/2023.

52. *Central Limit Theorem* https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_probability/BS704_Probability12.html. 20/03/2023.

53. 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.

54. Mauzy, R. Parallelizing Neural Networks to Break Physical Unclonable Functions (2020).

55. *Compare 2 Intel Products* https://www.intel.com/content/www/us/en/products/compare.html?productIds=186605,230495.

56. Kullarkar, V. T. & Chandrakar, V. K. Power quality analysis in power system with non linear load. *Int. J. Electr. Eng* **10,** 33–45 (2017).

57. McDonald, B. & Lough, B. *Power Factor Correction (PFC) Circuit Basics* in *Texas Instruments Power Supply Design Seminar* (2020).

58. Georgiou, S. & Spinellis, D. Energy-delay investigation of remote inter-process communication technologies. *Journal of Systems and Software* **162,** 110506 (2020).

59. 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).

60. *Syncing Time from the Network on the Raspberry Pi* = https://pimylifeup.com/raspberry-pi-time-sync/. 2022.

61. Pereira, R. *et al. Energy efficiency across programming languages: how do energy, time, and memory relate?* in (Oct. 2017), 256–267.

62. Sestoft, P. Microbenchmarks in Java and C#. *Lecture Notes, September* (2013).

63. *Intel Default Libraries* https://www.cita.utoronto.ca/~merz/intel_c10b/main_cls/mergedProjects/bldaps_cls/cppug_ccl/bldaps_def_libs_cl.htm.

64. *x86 and amd64 instruction reference* https://www.felixcloutier.com/x86/.

65. *Intel extensions* https://www.intel.com/content/www/us/en/developer/tools/isa-extensions/overview.html.

66. 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

On 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 **2.5**
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	subsection **2.1**
Libre Hardware Monitor	LHM	subsection **2.1**
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	subsection **2.1**
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 **2.5**
Model-specific-registers	MSRs	subsection **4.1**
Open Multi-Processing	OpenMP	subsection **3.5**
Performance Monitoring Counter	PMC	subsection **2.5**
Speedup Factor	SF	subsection **2.5**

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

B The Framework

The framework used in this work was an extension to [3]. The framework was called Biks Diagnostics Energy (BDE) and was a command line tool which 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. Information like the `compiler`, `language`, etc could also be specified the configuration. 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 was be used. When multiple affinities are specified, each benchmark would be run on both. Limits for the temperature the benchmarks should be executed within could also be specified, and lastly, `AdditionalMetadata` could be used to specify relevant aspects about the experiment, which 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. This included the ability to enable and disable the Wi-Fi, stop background processes, ect.

```
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, which was why they could be overwritten. 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.

```
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

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.

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 [3], a MySQL database was used to store the measurements made by the different measuring instruments. In this work, a similar database was also used, but with some modifications to accommodate the different focus compared to [3]. The design of the database was illustrated in Figure **11**, 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.

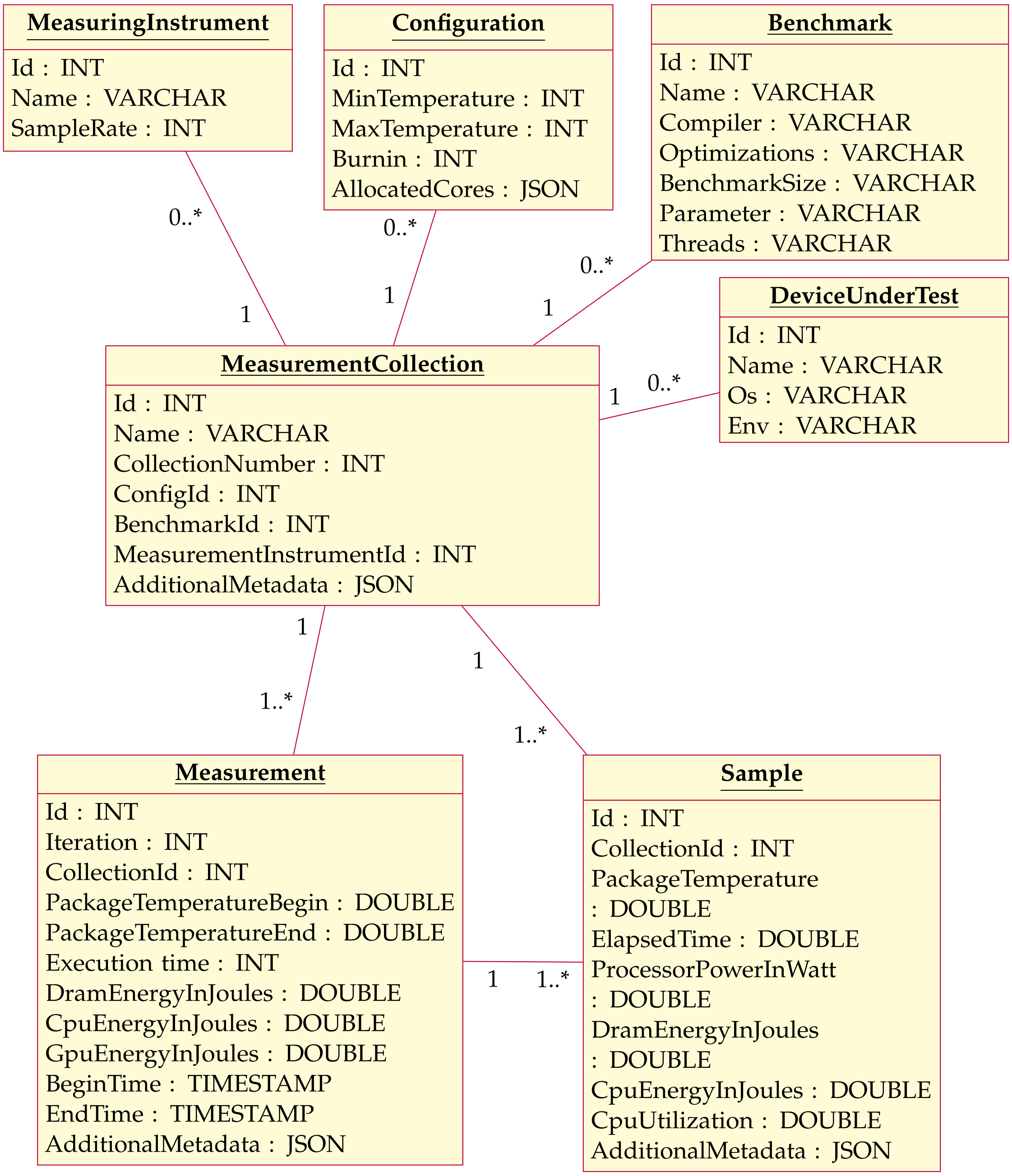


Figure 11: An UML diagram representing the tables in the SQL database

In the `MeasurementCollection`, the columns `CollectionNumber` and `Name` represented which experiment the measurement was from, and the name of the experiment respectively.

The `Measurement` contained values for the energy consumption during the entire execution time 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

Not all of the benchmarks are executed on our DUTs, because some of them would crash. The different workloads and whether they are included or not are shown on Tables **13** and **14**. Further detail about the workloads can be found in [51].

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

Measurements made on benchmark MB for the first experiment, found in subsection **5.1**.

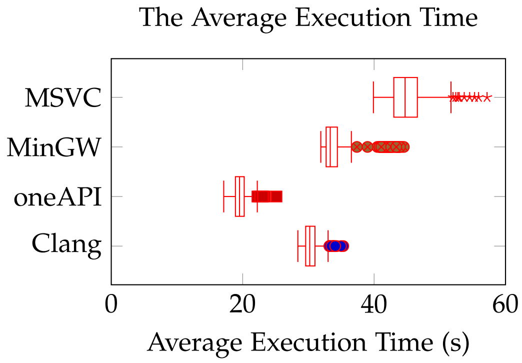


Figure 12: Execution time measurements by IPG on DUT 1 for test case(s) FR

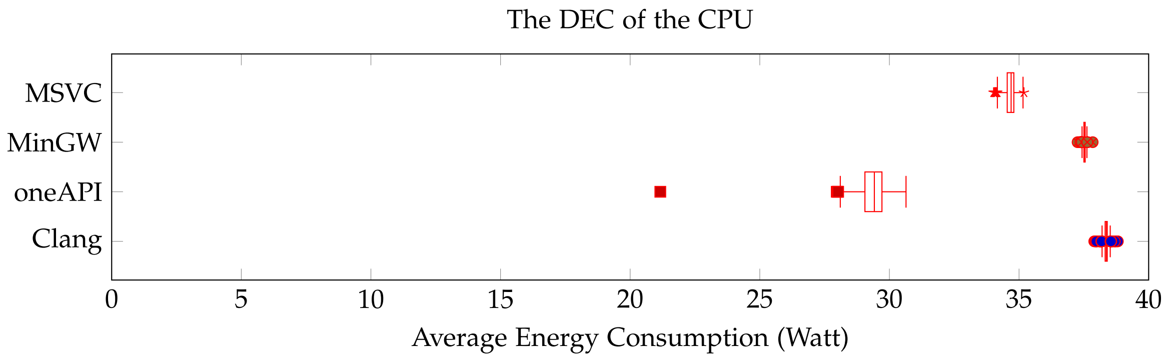


Figure 13: CPU measurements by IPG on DUT 1 for benchmark(s) MB

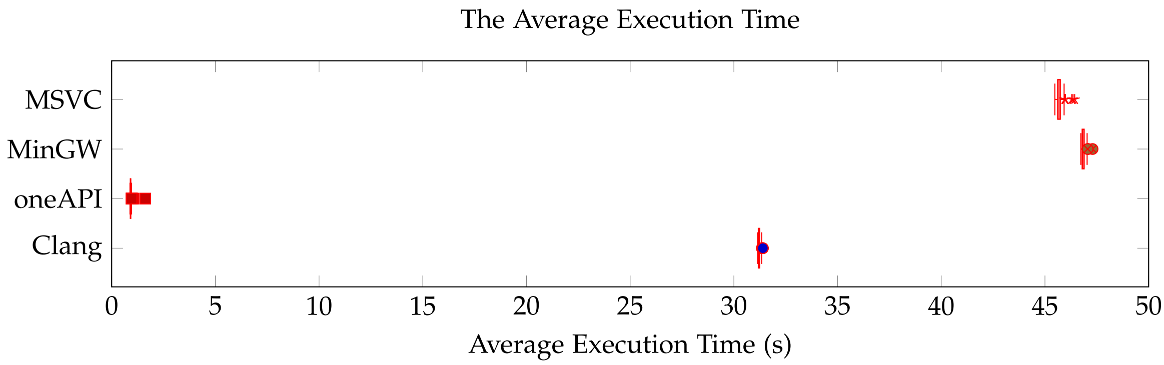


Figure 14: Execution time measurements by IPG on DUT 1 for test case(s) MB

F Experiment Two

Measurements made on for the second experiment, aiming to find the best measuring instrument, found in subsection **5.2**.

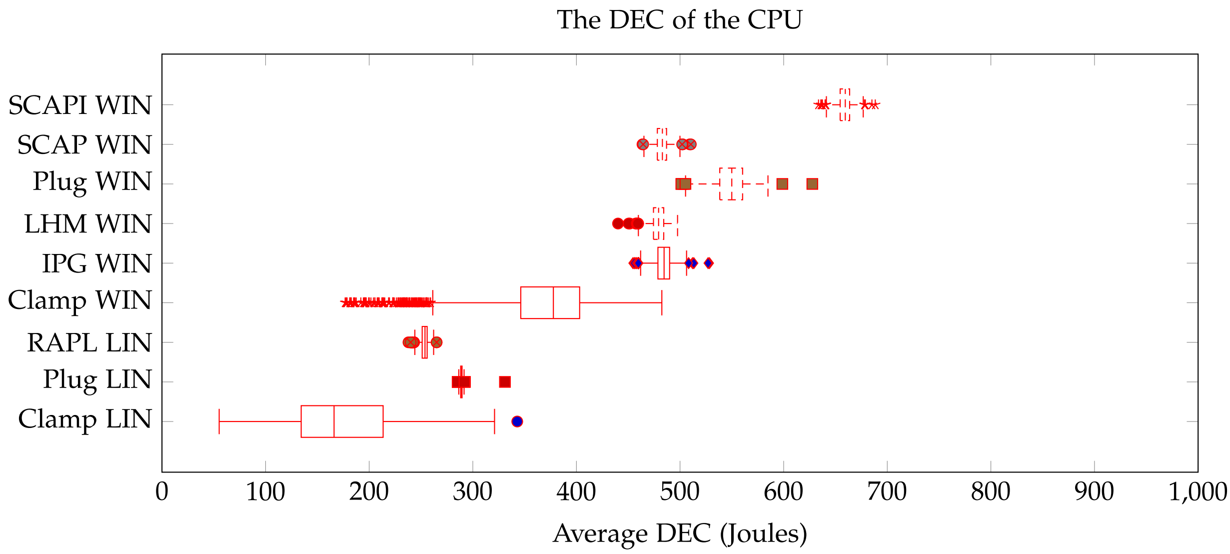


Figure 15: CPU measurements on DUT 1 for test case(s) FR compiled on oneAPI

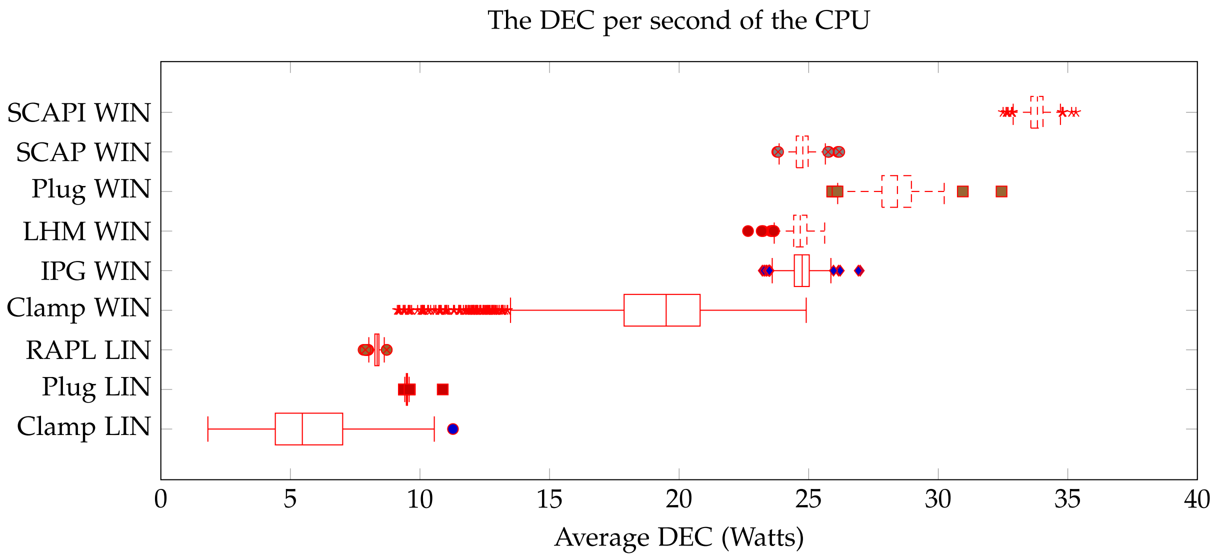


Figure 16: CPU measurements on DUT 1 for test case(s) FR compiled on oneAPI

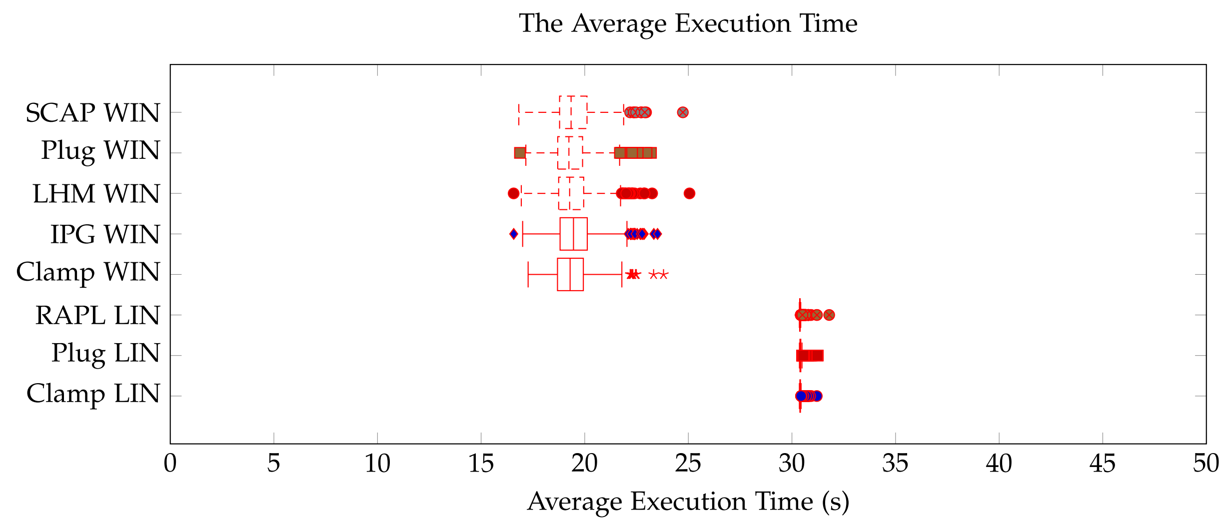


Figure 17: Execution time measurements on DUT 1 for test case(s) FR compiled on oneAPI

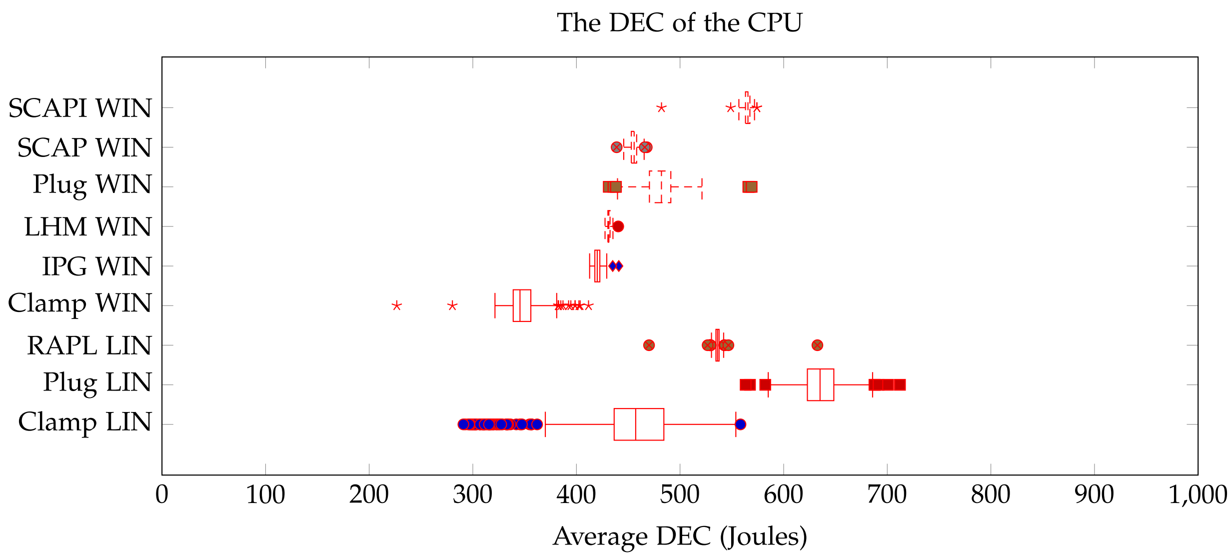


Figure 18: CPU measurements on DUT 1 for test case(s) MB compiled on oneAPI

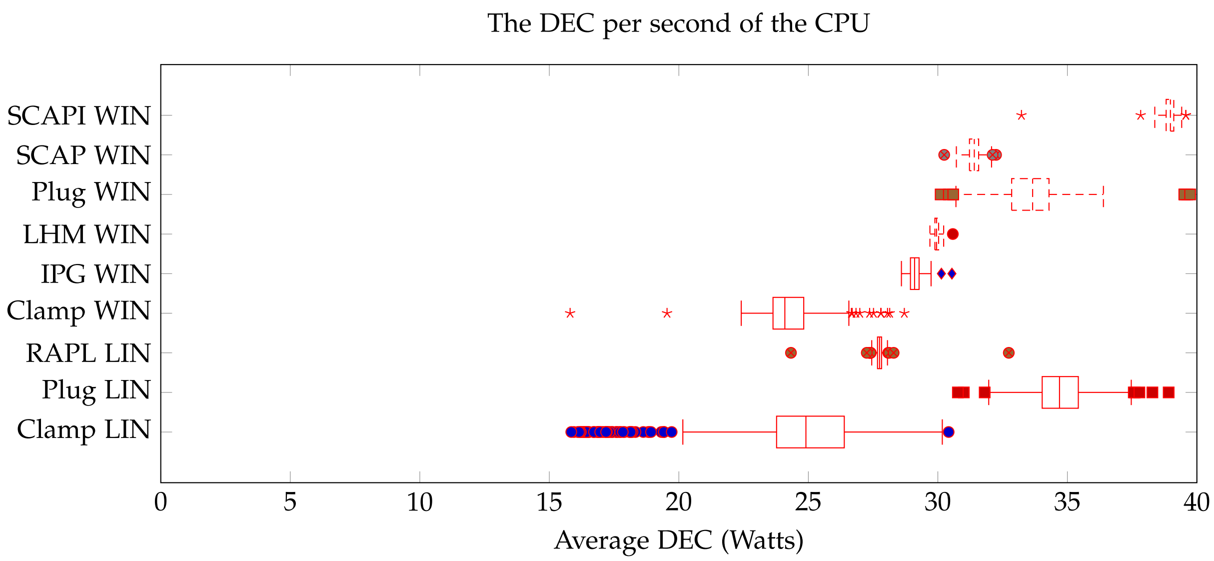


Figure 19: CPU measurements on DUT 1 for test case(s) MB compiled on oneAPI

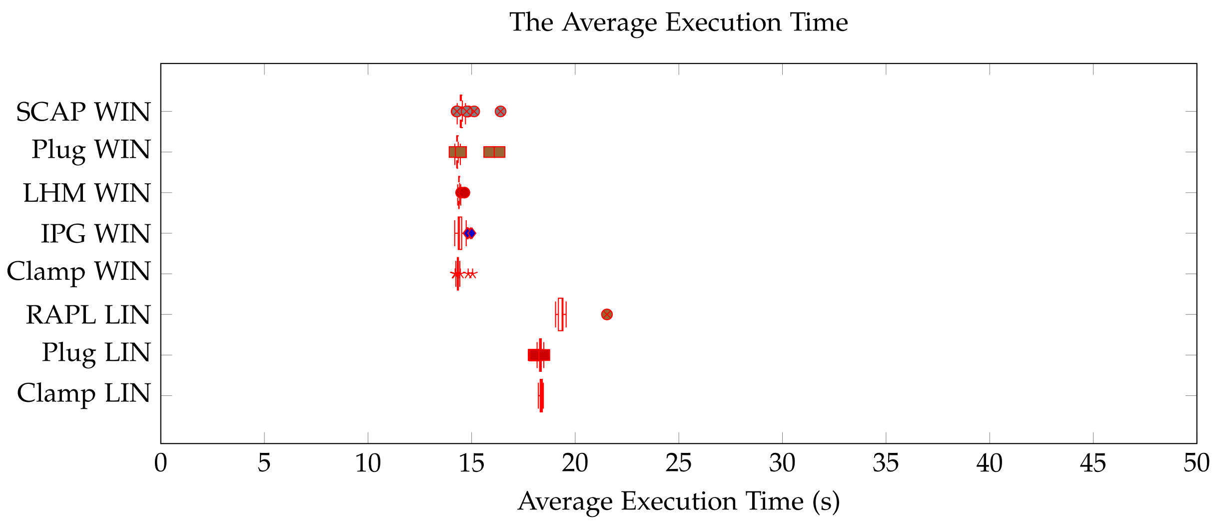


Figure 20: Execution time measurements on DUT 1 for test case(s) MB compiled on oneAPI

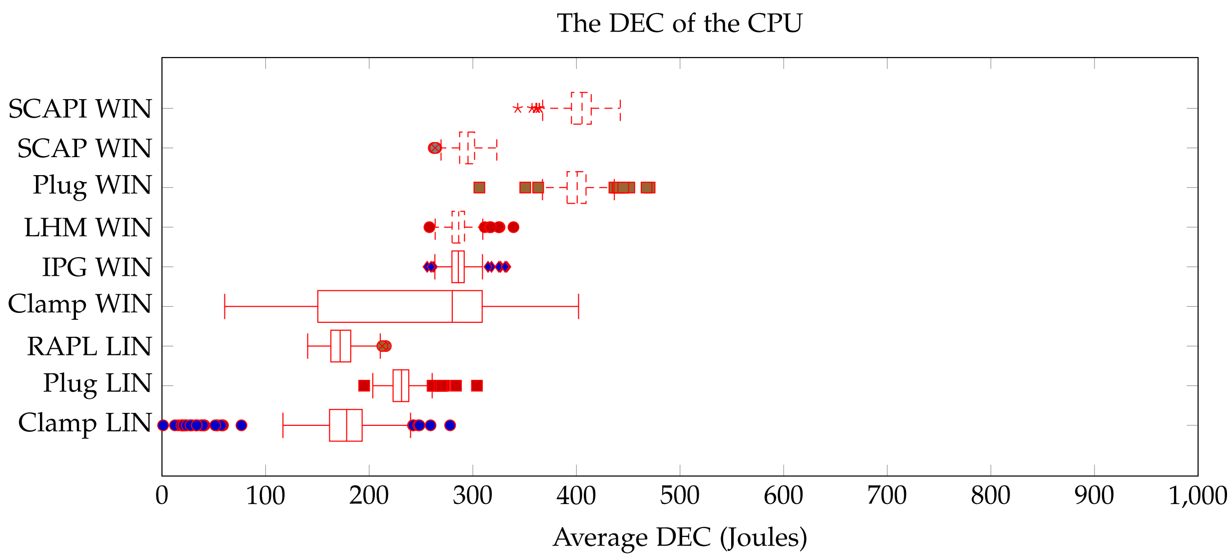


Figure 21: CPU measurements on DUT 2 for test case(s) FR compiled on oneAPI

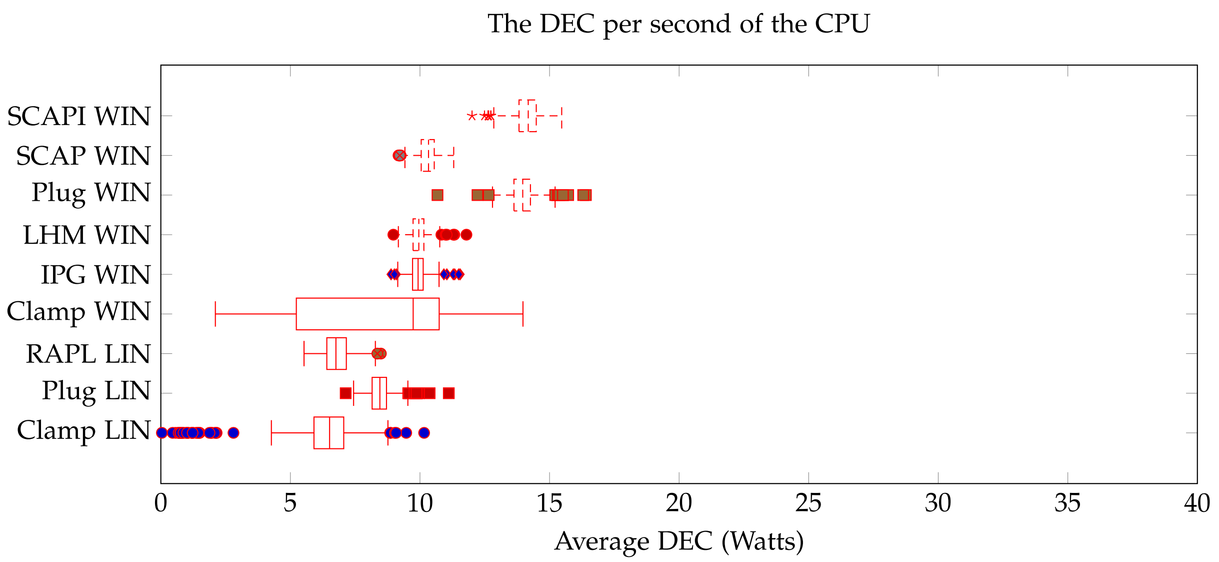


Figure 22: CPU measurements on DUT 2 for test case(s) FR compiled on oneAPI

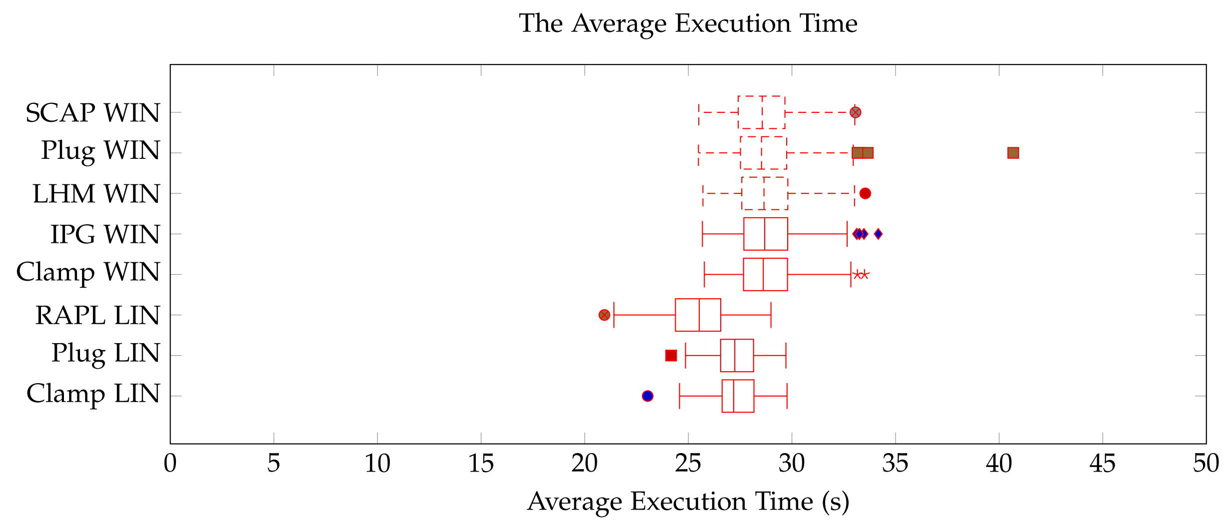


Figure 23: Execution time measurements on DUT 2 for test case(s) FR compiled on oneAPI

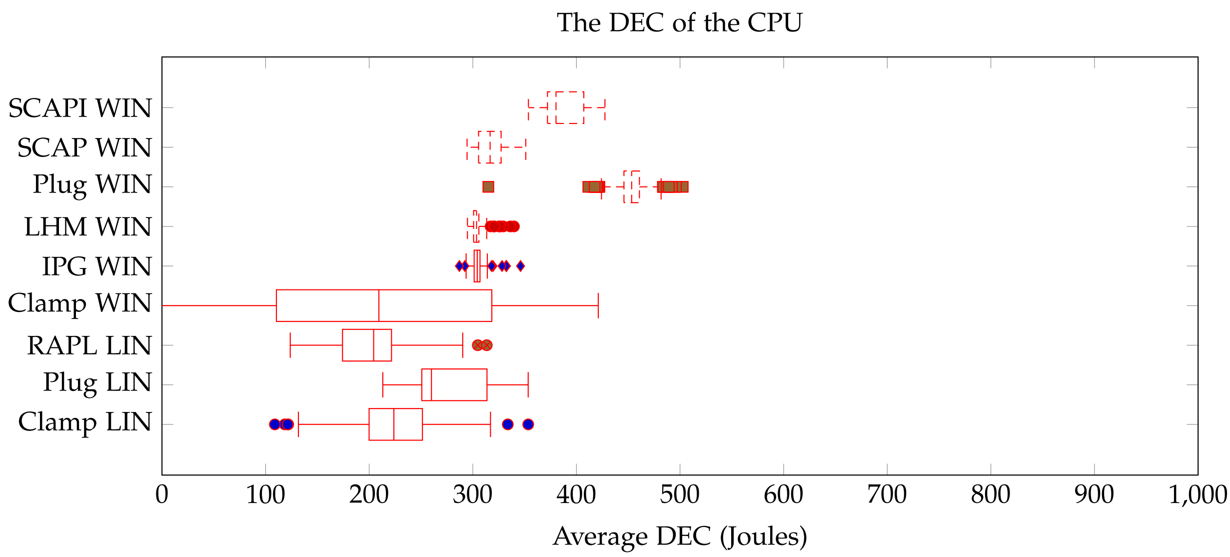


Figure 24: CPU measurements on DUT 2 for test case(s) MB compiled on oneAPI

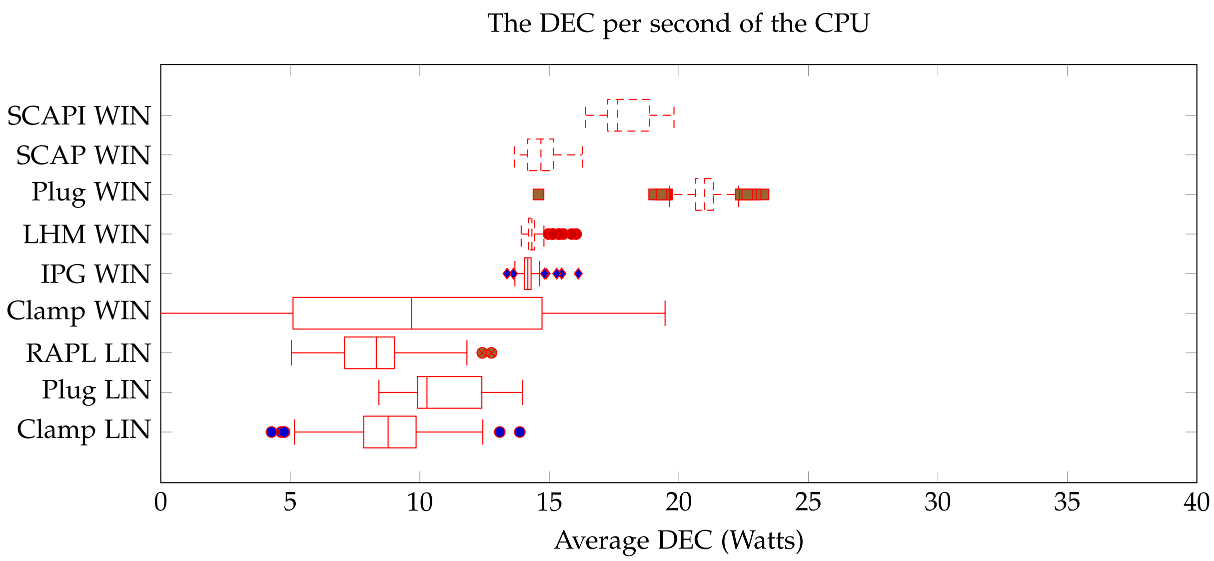


Figure 25: CPU measurements on DUT 2 for test case(s) MB compiled on oneAPI

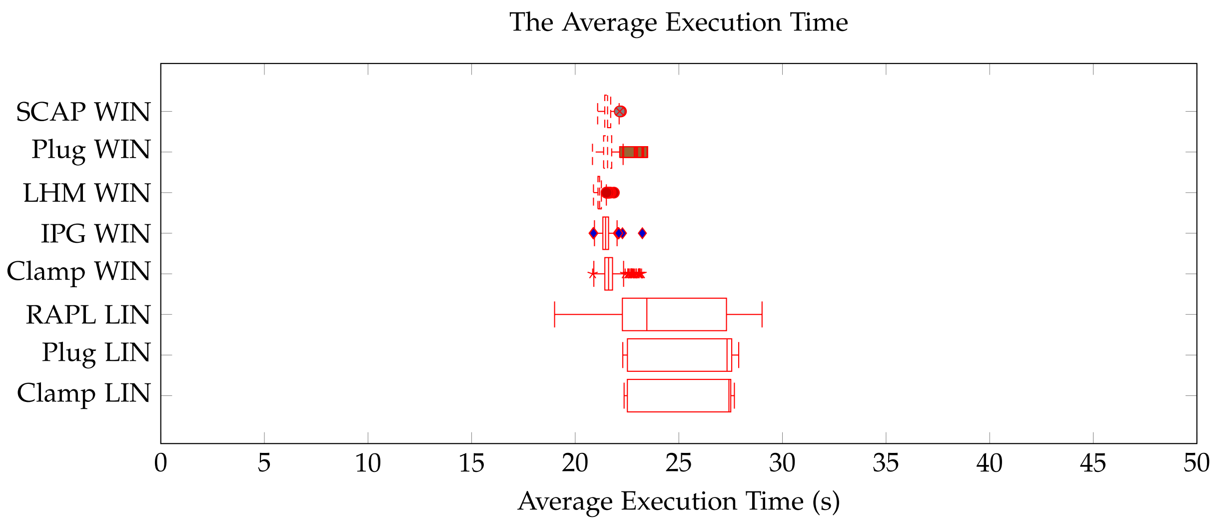


Figure 26: Execution time measurements on DUT 2 for test case(s) MB compiled on oneAPI

G Initial Measurements in Experiment Two

This section illustrates how many measurements are required from the different measuring instruments in order to gain confidence in them, and are used 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

Table 15: The required samples to gain confidence in the measurements made by the different measuring instruments, on both OSs for 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

Table 16: The required samples to gain confidence in the measurements made by the different measuring instruments, on both OSs for DUT 2

Correlation from Experiment Two

This section shows the correlation heatmap from subsection **5.2**.

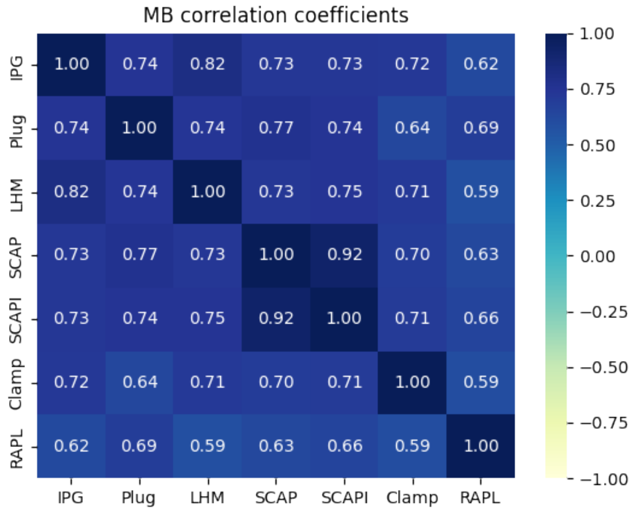


Figure 27: Heatmap showing the correlation coefficient between all of the measurement instruments on windows for the MB benchmark for dut 1.

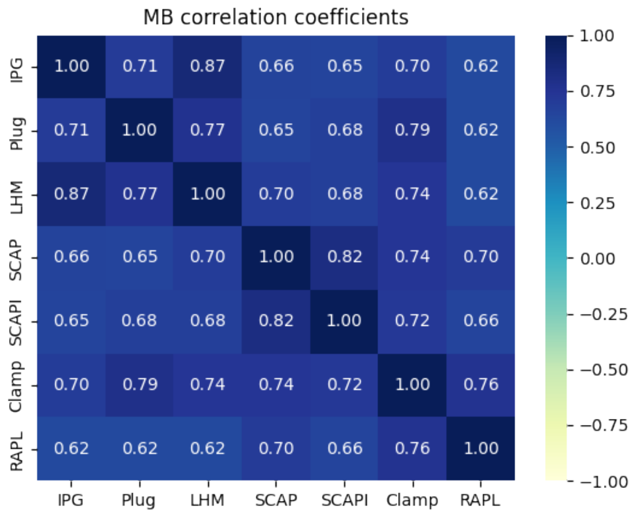


Figure 28: Heatmap showing the correlation coefficient between all of the measurement instruments on windows for the MB benchmark for dut 2.

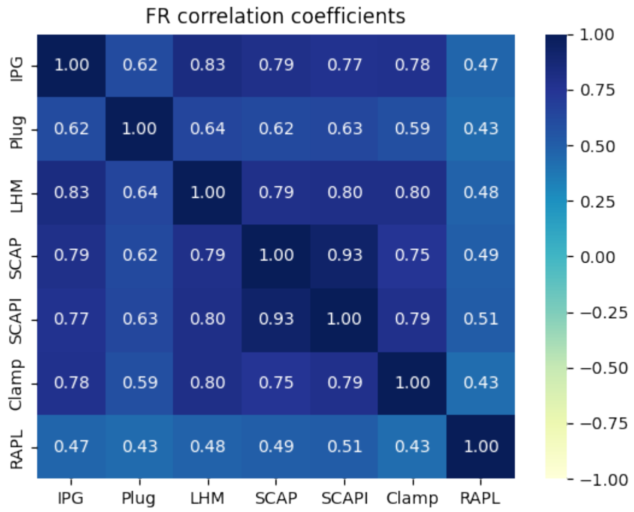


Figure 29: Heatmap showing the correlation coefficient between all of the measurement instruments on windows for the FR benchmark for dut 1.

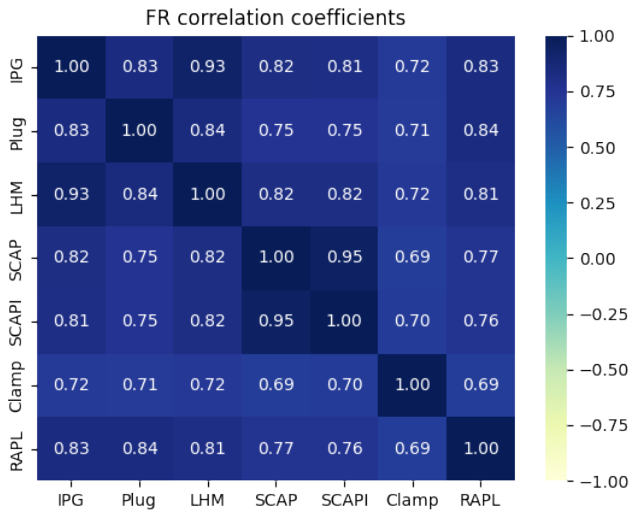
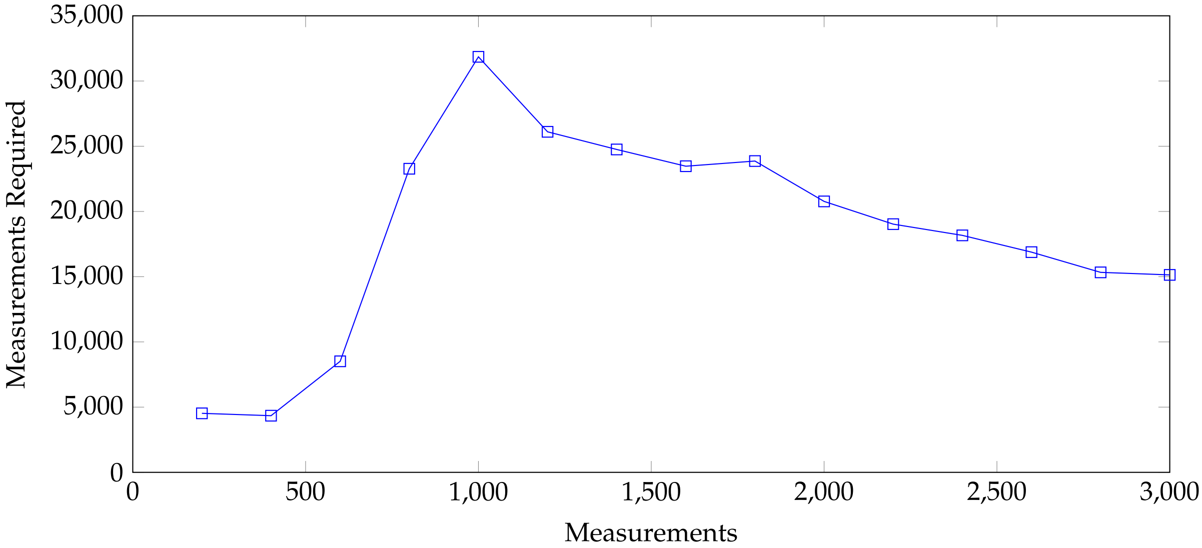


Figure 30: Heatmap showing the correlation coefficient between all of the measurement instruments on windows for the FR benchmark for dut 2.

H Cochran's Formula Evolution for Experiment Two

This section shows the evolution of how many measurements are required, calculated using Cochran's formula on different amount of measurements, as used in subsection **5.2**.



I Experiment Three

This section includes the results from the third experiment, as can be found in subsection **5.3**

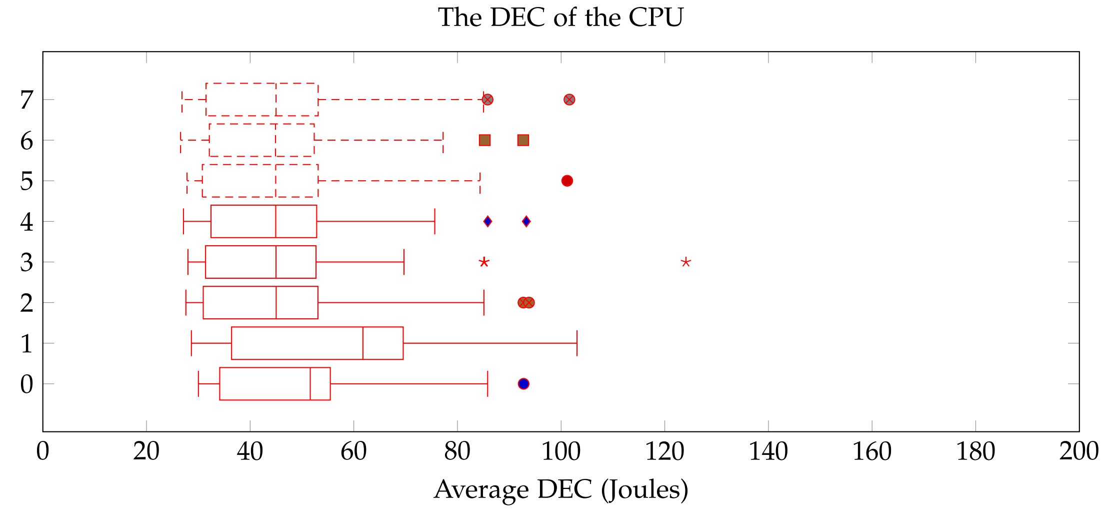


Figure 31: CPU measurements by IPG on DUT 1 for test case(s) NB compiled on oneAPI

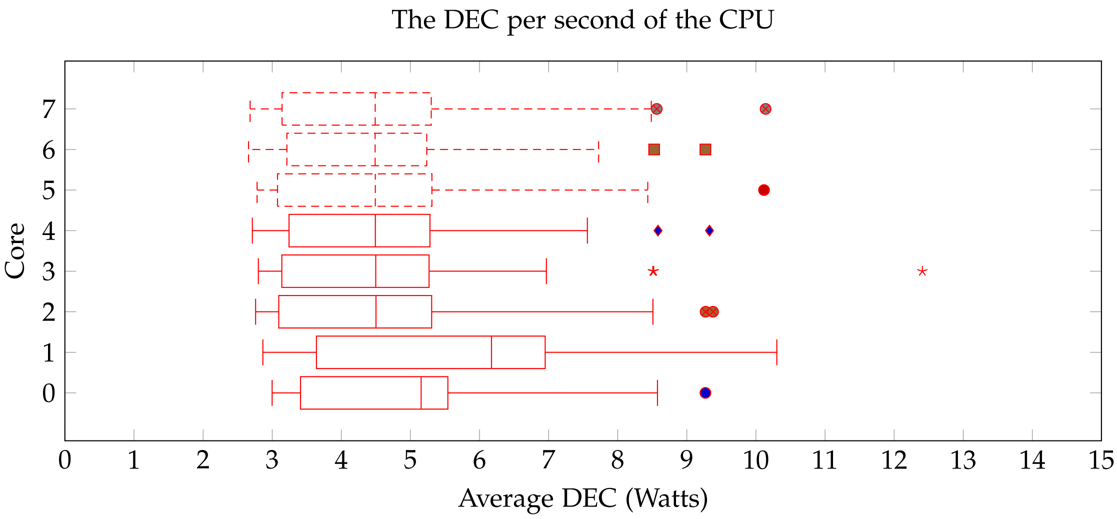


Figure 32: CPU measurements by IPG on DUT 1 for test case(s) NB compiled on oneAPI

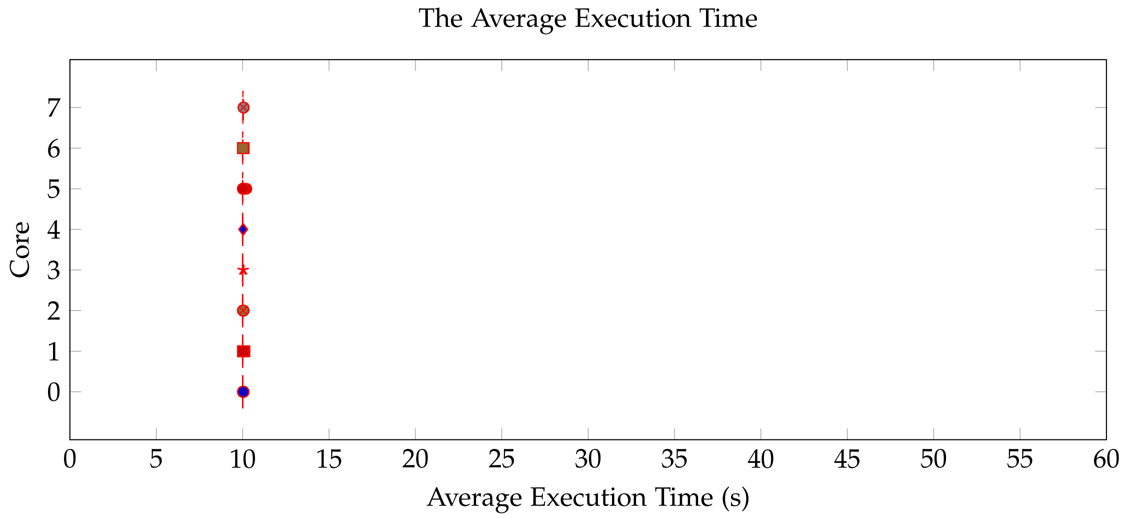


Figure 33: Execution time measurements by IPG on DUT 1 for test case(s) NB compiled on oneAPI

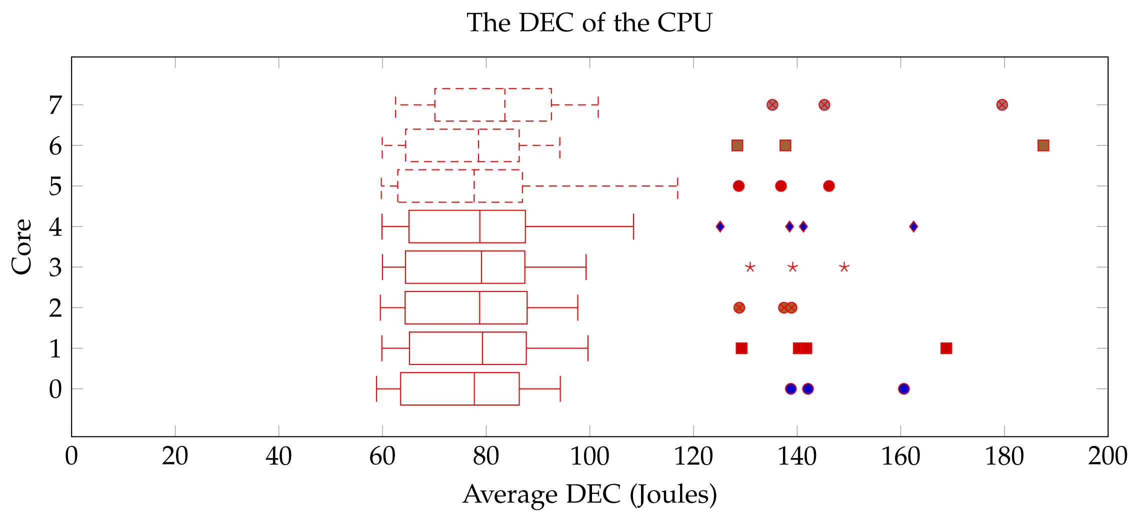


Figure 34: CPU measurements by IPG on DUT 1 for test case(s) SN compiled on oneAPI

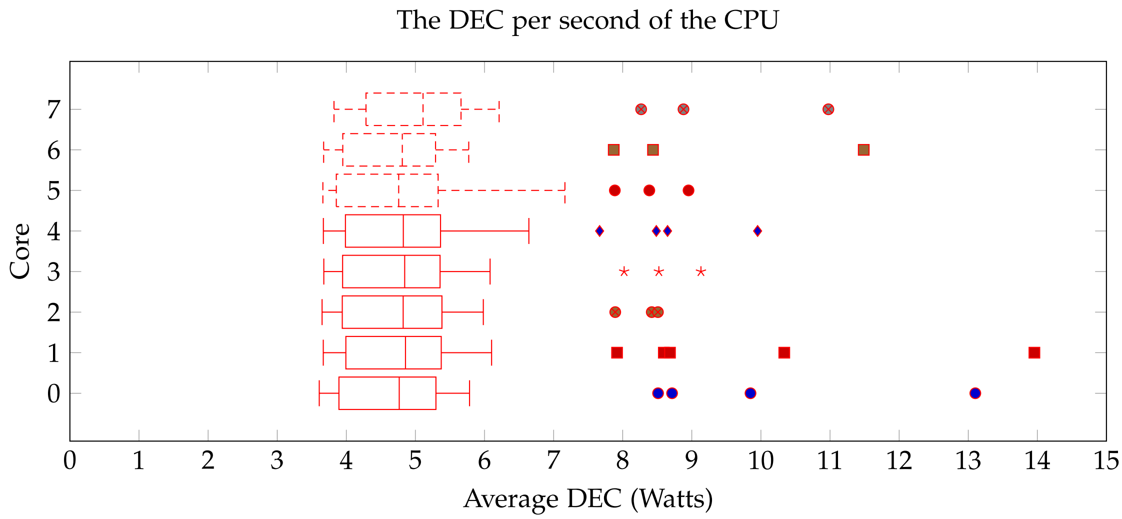


Figure 35: CPU measurements by IPG on DUT 1 for test case(s) SN compiled on oneAPI

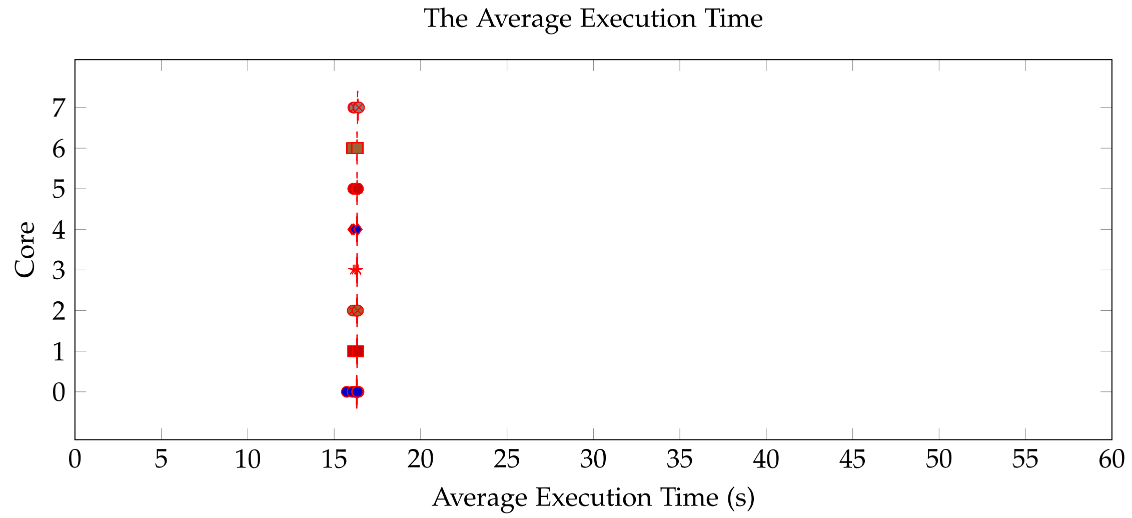


Figure 36: Execution time measurements by IPG on DUT 1 for test case(s) SN compiled on oneAPI

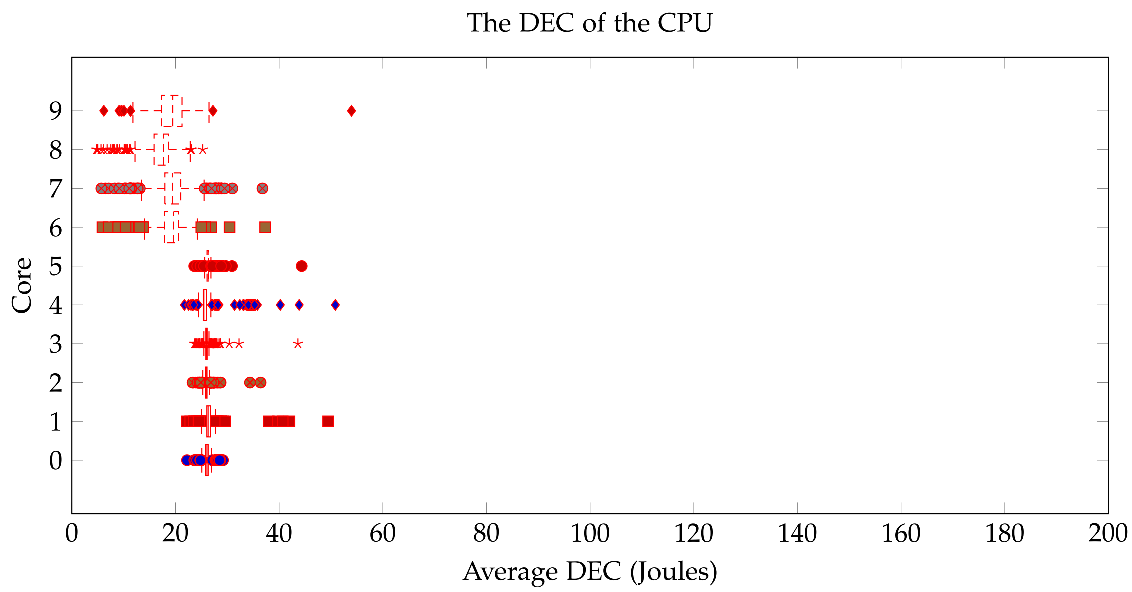


Figure 37: CPU measurements by IPG on DUT 2 for test case(s) NB compiled on oneAPI

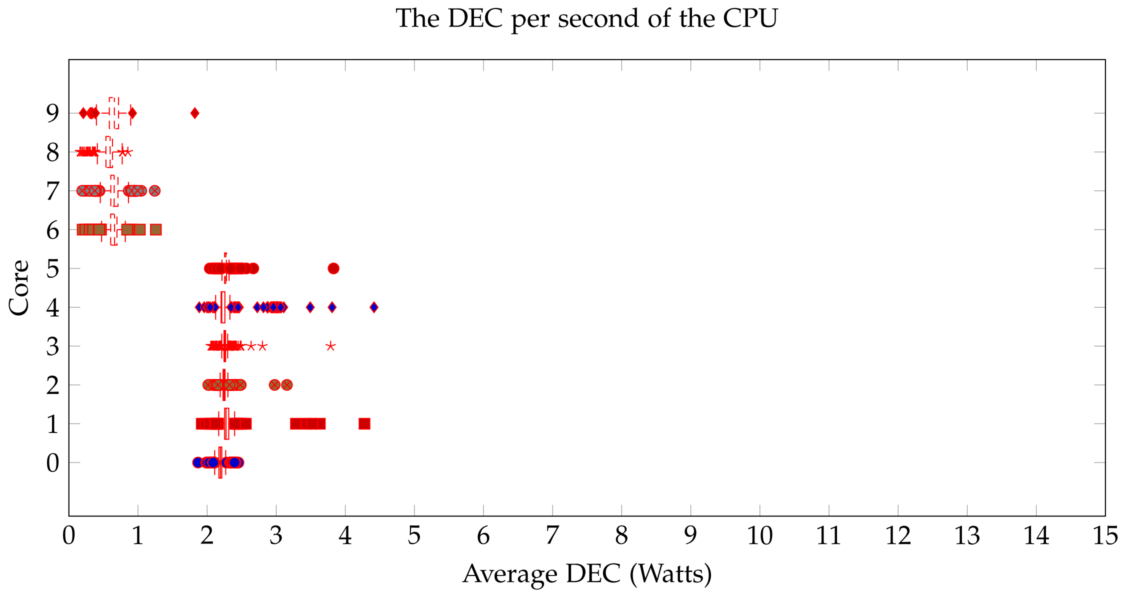


Figure 38: CPU measurements by IPG on DUT 2 for test case(s) NB compiled on oneAPI

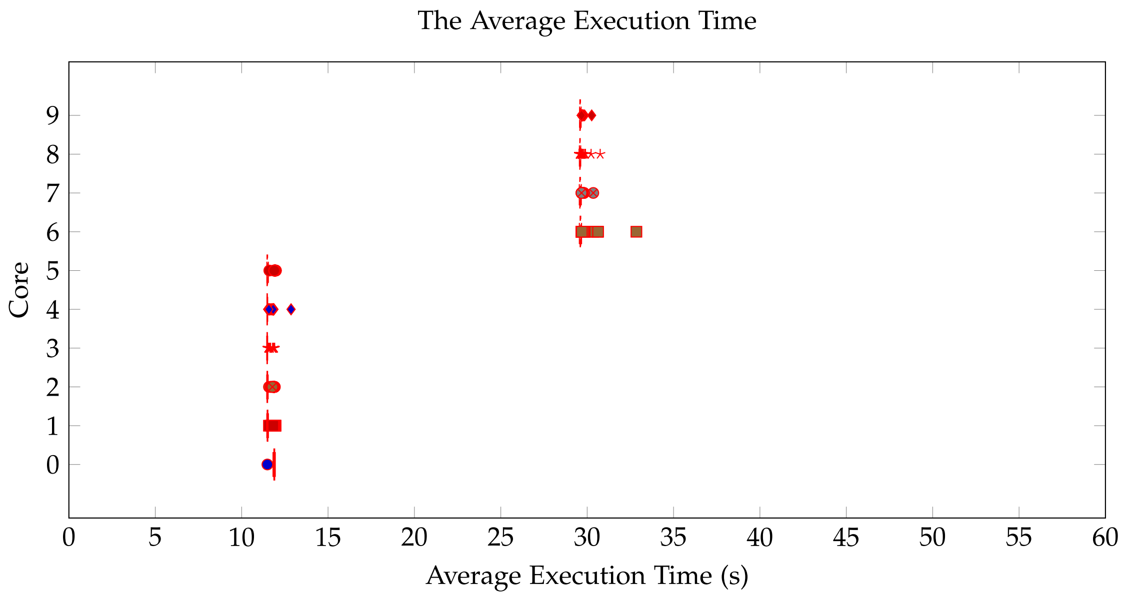


Figure 39: Execution time measurements by IPG on DUT 2 for test case(s) NB compiled on oneAPI

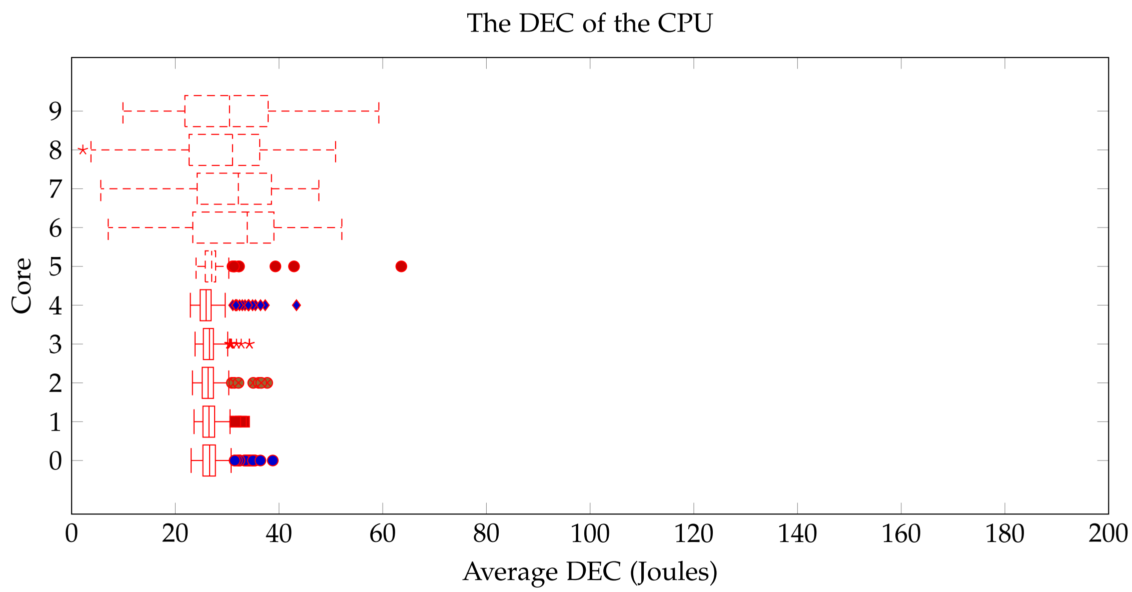


Figure 40: CPU measurements by IPG on DUT 2 for test case(s) SN compiled on oneAPI

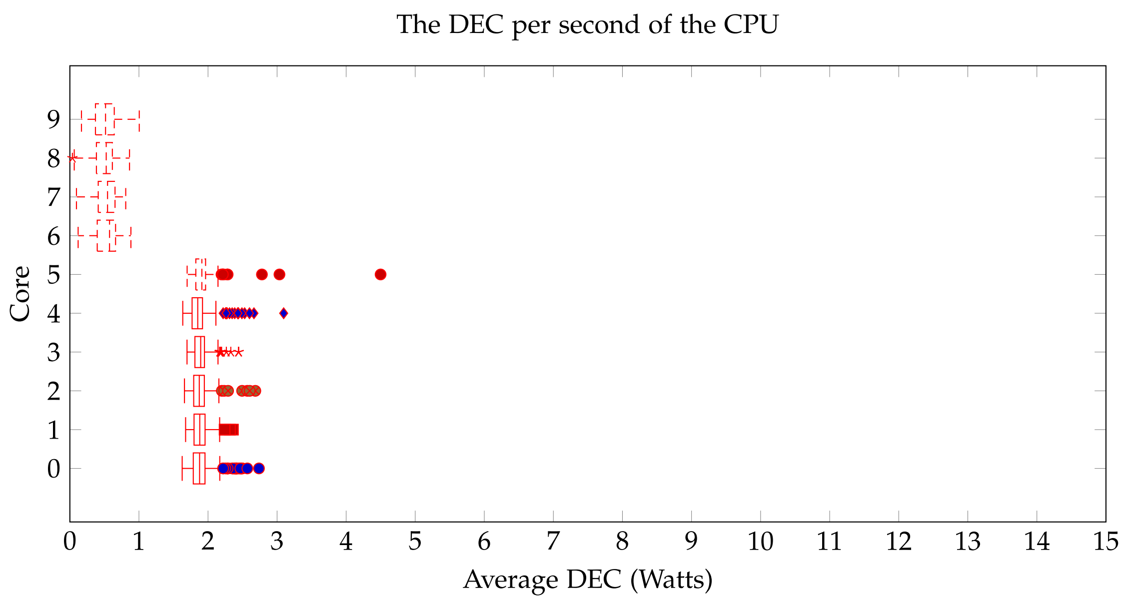


Figure 41: CPU measurements by IPG on DUT 2 for test case(s) SN compiled on oneAPI

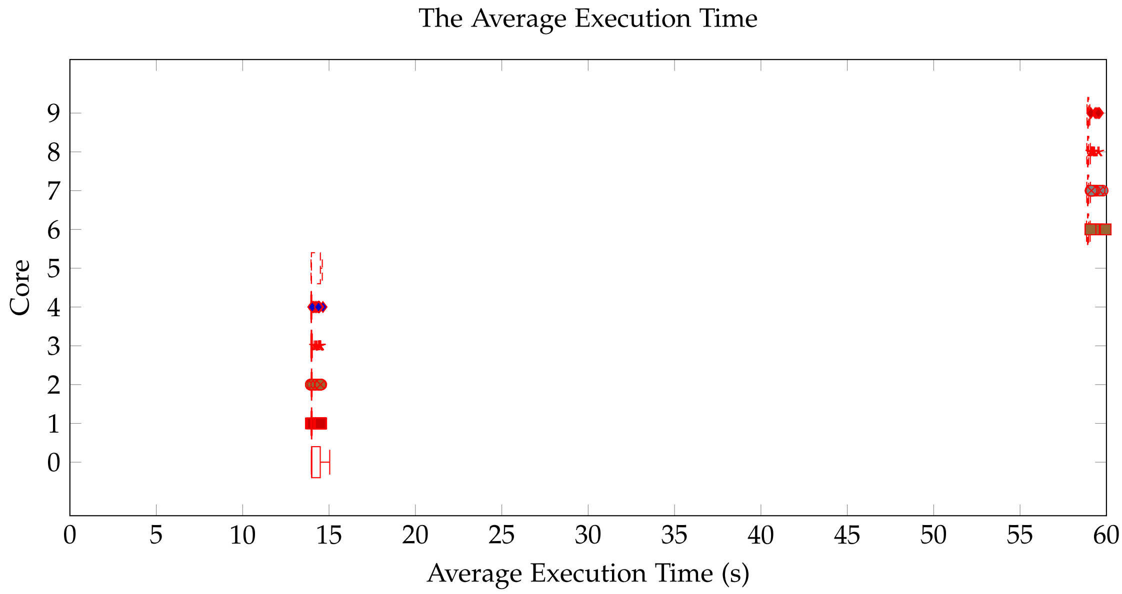


Figure 42: Execution time measurements by IPG on DUT 2 for test case(s) SN compiled on oneAPI

J Measurements Required in Experiment Three Cores

This section illustrates how many measurements are required from IPG when measuring the energy consumption on different cores in order to gain confidence in them, and are used 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

Table 17: The required samples to gain confidence in the measurements made by IPG in different cores for 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

Table 18: The required samples to gain confidence in the measurements made by IPG in different cores for DUT 2

K Measurements Required in Experiment Three Macrobenchmarks

This section illustrates how many measurements are required from IPG when measuring the energy consumption of the macrobenchmarks in order to gain confidence in them, and are used 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

Table 19: The required samples to gain confidence in the measurements made by IPG for the macrobenchmarks for 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

Table 20: The required samples to gain confidence in the measurements made by IPG on for different macrobenchmarks for DUT 2

L Results for Macrobenchmarks in the Third Experiment

In this section the energy consumption for an increasing number of cores can be found, referenced in subsection **5.3**.

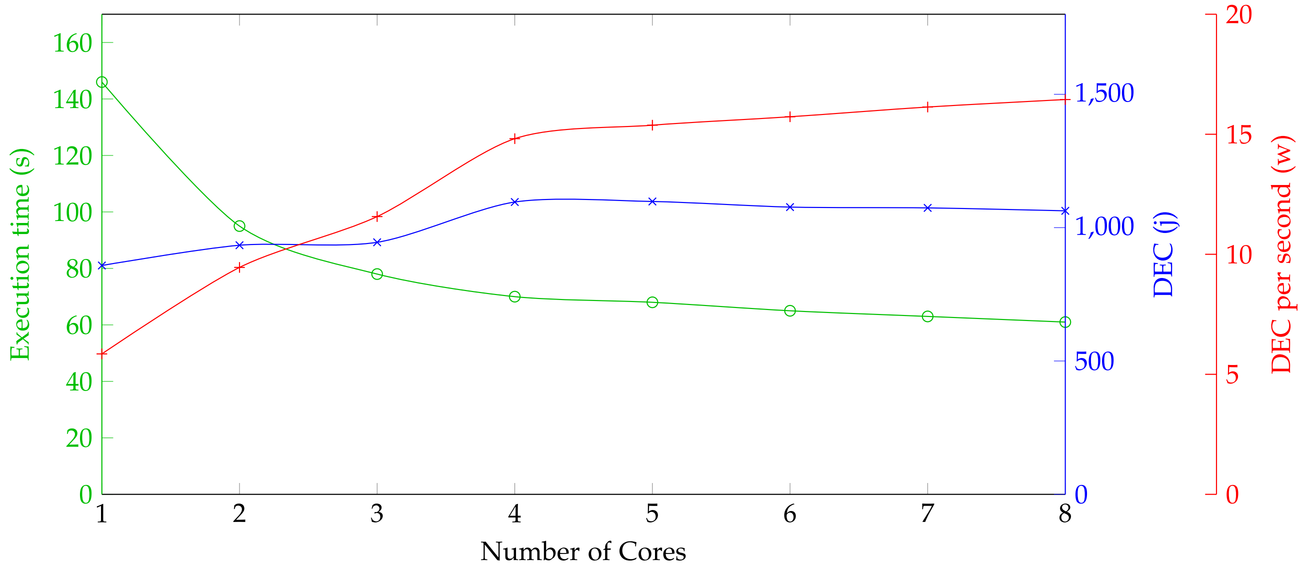


Figure 43: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to 3DM on DUT 1

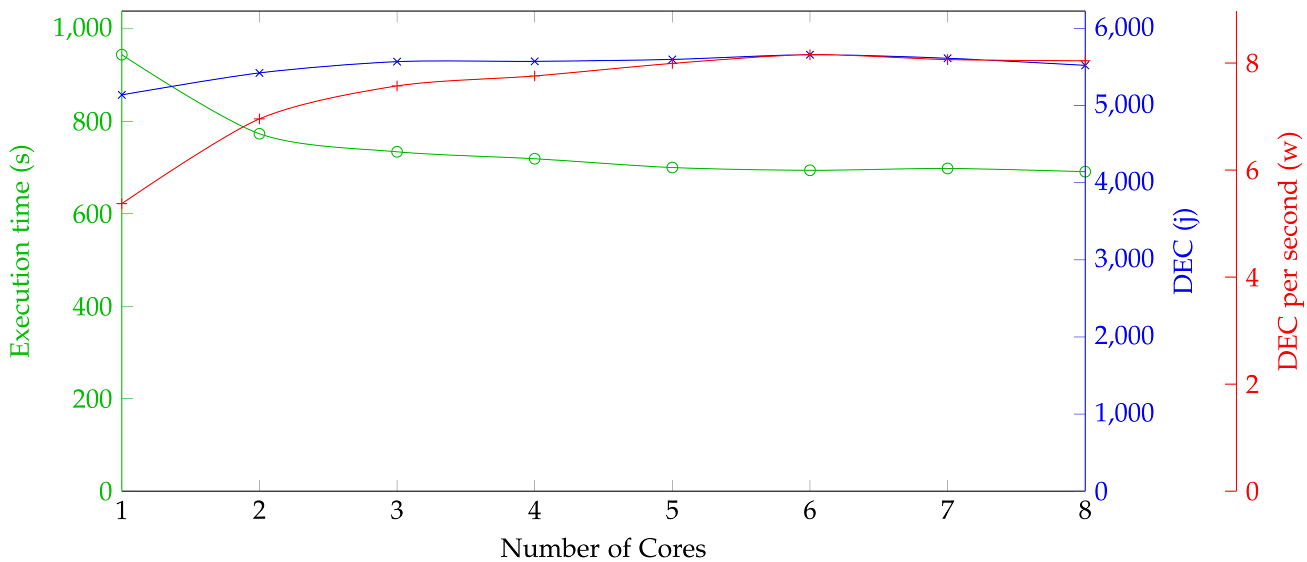


Figure 44: The evolution of the DEC (blue), DEC per second (red) and execution time (green) as more cores are allocated to PCM on DUT 1

Measurements Requied in Experiment Three P vs. E Cores

This section illustrates how many measurements are required from IP when measuring the energy consumption of PCM on four P cores, four E cores, or two of each. These results are referenced in subsection **5.3**

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

Table 21: The required samples to gain confidence in the measurements made by IPG when comparing P and E cores for DUT 2

M Results for P vs. E Cores in Third Experiment

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

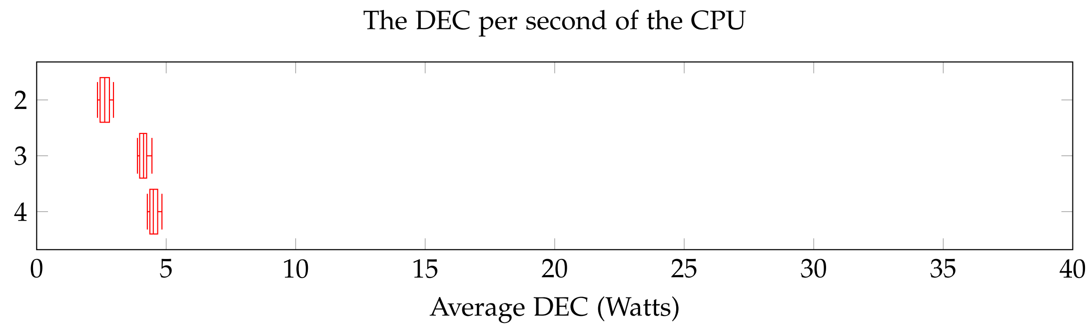


Figure 45: CPU measurements by IPG on DUT 2 for test case(s) PCM

N Energy Consumption Over Time

This section shows how both macrobenchmarks energy consumption evolves over time for DUT 2, used in subsection **5.3**. In this section 3DM and PCM are plotted with two cores and all cores to illustrate the difference the additional resources make.

3DM is illustrated in Figure **46** and Figure **47**, where different phases of 3DM can be observed. For both two and ten cores there is a startup period until around 14 seconds, after which the benchmark starts. On ten cores, load is on 25 watts for 18 seconds, while for two cores the energy consumption is on 12 watts for 60 seconds.

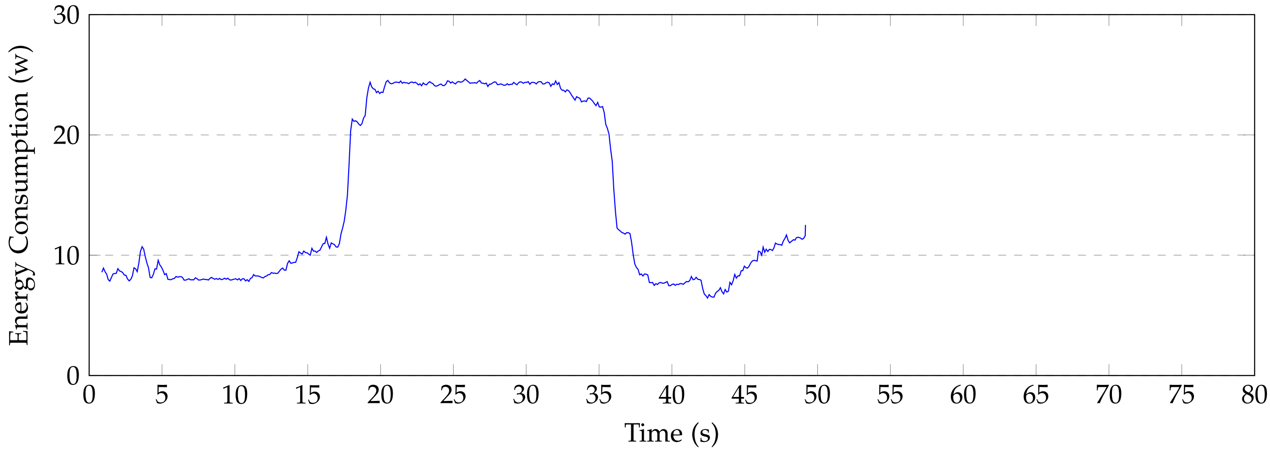


Figure 46: A timeseries of the energy consumption over time for DUT 2 when running 3DM for all cores

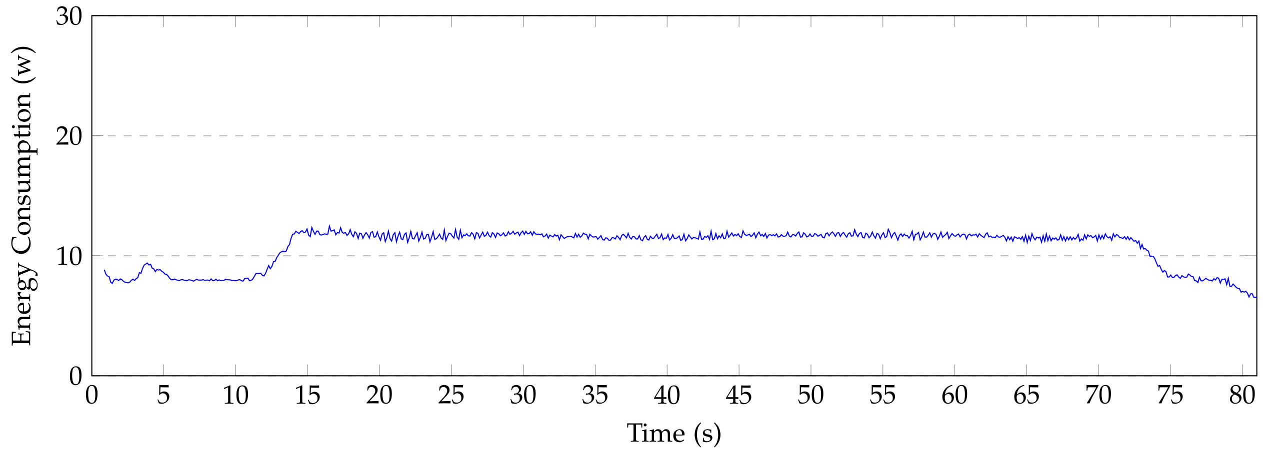


Figure 47: A timeseries of the energy consumption over time for DUT 2 when running 3DM on two cores

For PCM, the graphs illustrating the energy consumption over time, can be seen in Figure **48** and Figure **49**, where a smaller difference could be found between two and ten cores compared to 3DM. One reason was the load of PCM, which did not require as many resources, meaning the additional resources gave a diminishing return. When looking at Figure **49** it was observed that the peak wattage usage of 12 for 3DM on two cores was exceeded during runtime. This happened between $225s - 235s$, $370s - 380s$ and $570s - 620s$, which amounts to 9% of the total runtime. This indicated that the affinity was not set for all processes related to PCM, resulting in too many resources being allocated to some processes. An effort was put into fixing this, but without success. This meant that the performance gained when allocating more cores when executing PCM represents a lower limit. This is because, if the affinity was set correctly, the execution time would be higher on few cores, resulting in larger gains when allocating additional cores.

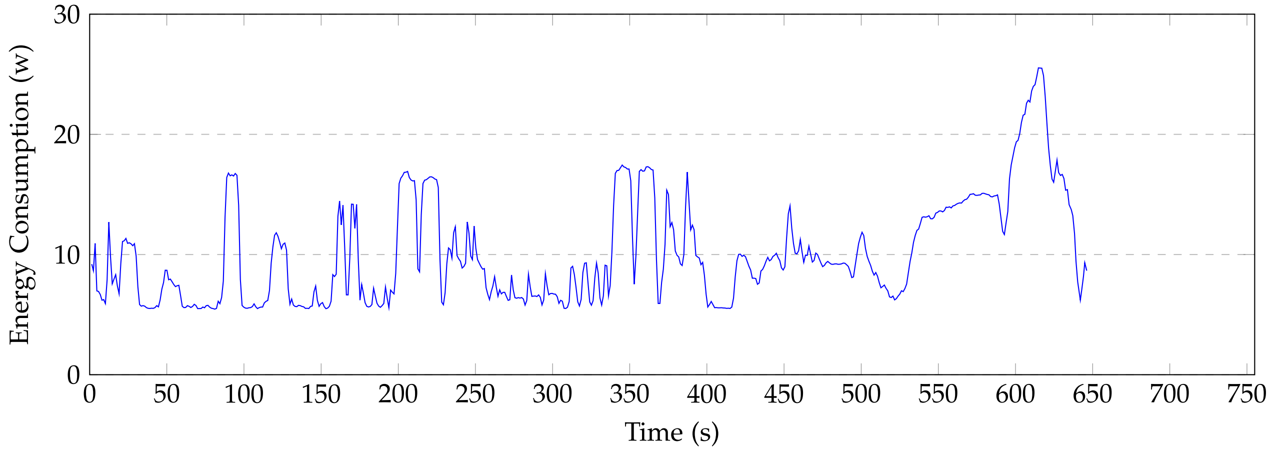


Figure 48: A timeseries of the energy consumption over time for DUT 2 when running PCM for all cores

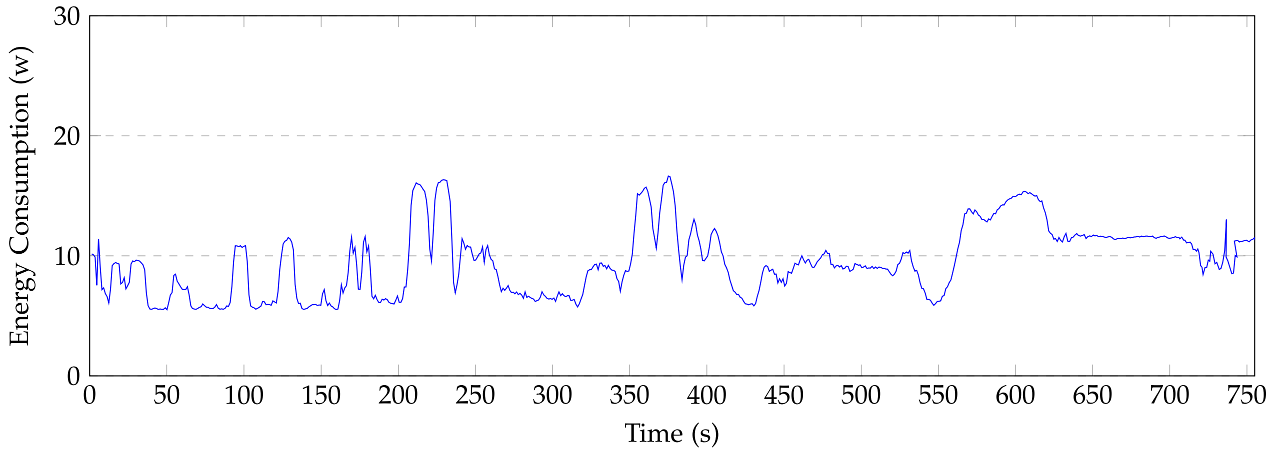


Figure 49: A timeseries of the energy consumption over time for DUT 2 when running PCM on two cores

O C++ Compiler Analysis

In the first experiment in subsection **5.1**, different compilers were compared. This experiment found that both the energy consumption, and measurements required deviated between compilers. This was especially clear when comparing the oneAPI to the other compilers which could be a result of a common optimizations technic, dead code elimination that can remove the benchmark that is being tested[62]. The executable from oneAPI (668 mb) was three times as large as for MinGW (220 mb), so this indicates that the code has not been removed, but further analysis is required to confirm this. The analysis was conducted by comparing the oneAPI against MinGW, by decompiling the executables and comparing the instructions.

When comparing the Assembly code structure between the main functions from the compilers, the oneAPI used several function calls unique for intel process such as `___intel_new_feature_proc_init` and `__intel_fast_memset`, where both functions are part of Intel's default C++ libraries[63]. The MinGW used general purpose registers more and utilizes C++ Standard Library, in the assembly more than oneAPI.

When moving on to the Mandelbrot function, where the differences with the larges effect on runtime are expected, MinGW only use standard X86 instruction[64] to perform the calculations on the floating points and `xmm` registers to store the values. Opposed to this, oneAPI's implementation also utilized Advance Vector extensions (AVX)[65], to perform the calculations, and frequently utilizing `ymm` registers instead of `xmm`. The usage of AVX results in a increase in calculation speed as it allows for multiple calculations to be performed in parallel. The AVX technology is a set of instructions introduced by Intel to enhance the performance of floating-point-intensive applications[65]. Compared to MinGW, oneAPI is better at optimizing the code for the AVX architecture and take full advantage of the processing power of the CPU. This usage of AVX can be seen in Figure **51** and Figure **52**, which show the differences between Mingw and oneApis way of handling the setup for the Mandelbrot function, both are loading values stored in memory into local variables. They are however doing this using different instructions and registers, here it can be seen that oneApi uses vectorized instructions such as `vmovsd`, from AVX, while Mingw uses simple instructions such as `movsd` to move double precision floating points. The Vectorized instructions does so that the operation can be performed on multiple values simultaneously in parallel.

The use of AVX to perform large floating point calculations in parallel resulted in a similar energy consumption but a lower execution time, this is an observation also found by other studies about energy consumption and parallelism[16]. The inclusion of extension libraries could be the reasons why the executable made by oneAPI is roughly 3 times larger but with a lower runtime than MinGW and the the other compilers.

P Energy usage trends analysis

To explore our hypothesis regarding electrical network noise interfering with phase synchronization, we categorized the data into `working hours` (7:00 to 16:00) and `non-working hours` (16:00 to 7:00). We found no significant variation in power consumption peaks between the two categories. However, periods of low energy usage were higher during `working hours`, suggesting that they did not effect, benchmark measurements but did impact idle case measurements. This observation aligns with the results presented in Figure **3**, which represent the DEC values. To better understand this, consider that the 3,000 DEC measurements, each consist of the total energy consumption during benchmark execution and a corresponding idle case measurement as explained in subsection **4.2**.

Power supplies are typically less efficient at lower loads[66], causing reactive energy to contribute more to overall usage. To test this hypothesis we conducted another experiment where we measured the computer during weekdays, when the computer was during nothing to see the effects. The results can be seen in Figure **50**, where there are clear difference between the energy consumption from night to day, the cross over is not at the same time each day but the trend is consistent. Another thing to note is the usage pattern that there are spikes and valleys approximately once every 2 hours. The exact cause of this is not known, but we hypothesis that they are scheduled jobs on windows, because we do not see a similar pattern on Linux, some of these can be found using the Task Scheduler, but it is hard to say exactly what would execute at a given peak. This could potentially be determined with further analysis, but the exact jobs are not interesting for this study, but could be a subject for Future work, so this will not be looked into further.

Based on these findings it is clear that there is relationship between the energy consumption and the time of day, and that it seems to be effected by the amount of people on the electrical network. These observations have been made on multiple different DUTs and across different operating systems over multiple days, all showing this same pattern. We can confidently say that the DUTs consumes more energy during the day than it does at night, and during working days than in weekends. Whether the cause is reactive energy or some other unaccounted for factor is a subject for future work.

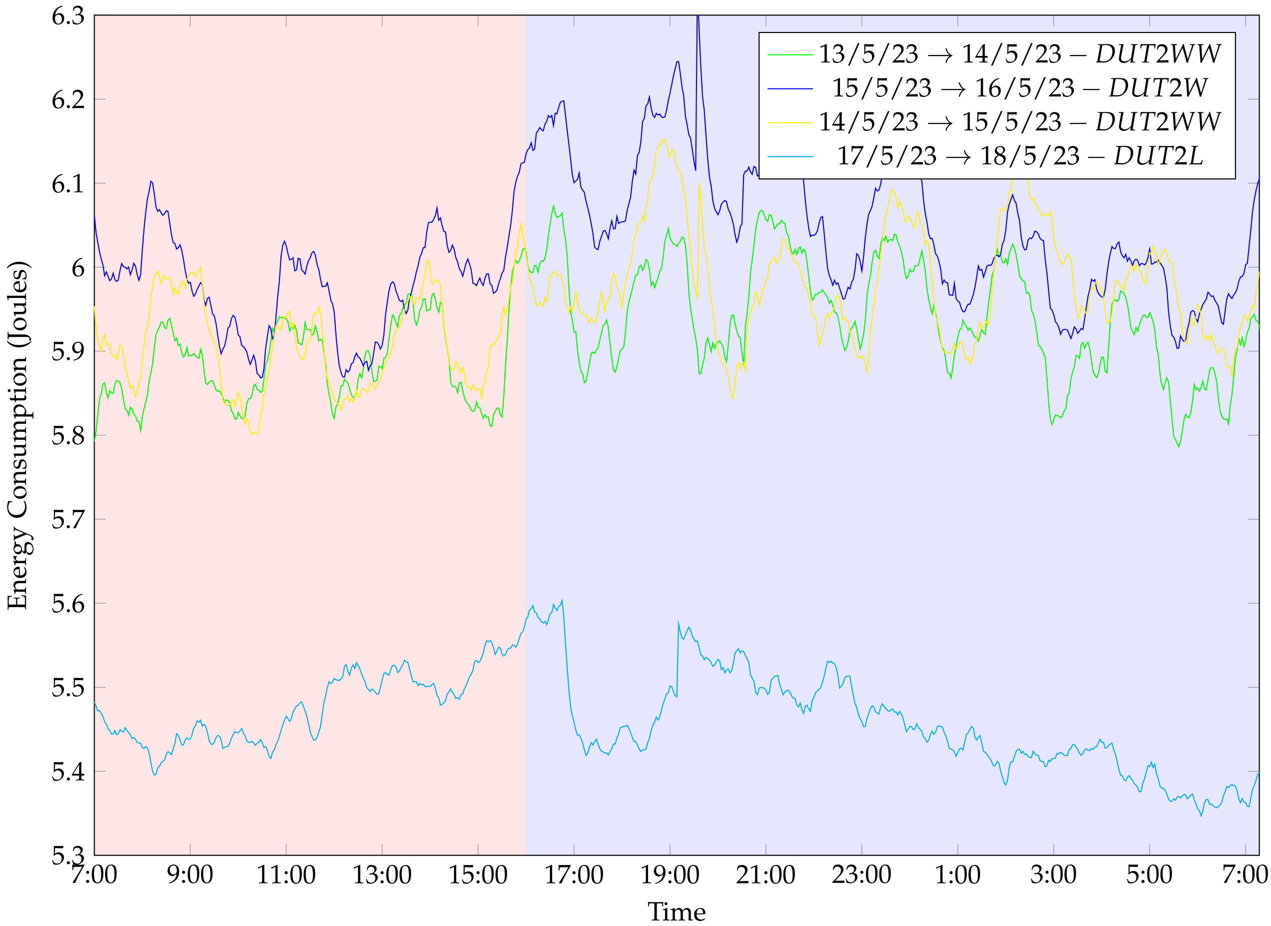


Figure 50: This shows the difference in energy consumbtion, between working hours and non-working hours, when the DUT perform no work. The red represents the working hours and the blue represents the non-working hours

```
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 51: 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, 0xffffffe0
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 52: oneAPI assembly showing the setup for the mandelbrot function