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

March 20, 2023

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

1 Introduction

In recent years there has been rapid growth in Information and Communications Technology (ICT) which has led to an increase in energy consumption that potentially can harm the environment. Furthermore, it is expected that the rapid growth of ICT will continue in the future. [1, 2] As the use of ICT rises the demand for computational power rises as well, therefore energy efficiency has or perhaps should become more of a concern for companies and software developers alike.

In this paper, we investigate the energy consumption of some test cases during execution and compare the results obtained using different measuring instruments. Furthermore, the energy consumption and execution time of sequential and parallel execution of test cases are compared to analyze which method is the most energy efficient and what the tradeoffs are in terms of energy consumption and execution time. These experiments will be run on two different Device Under Tests (DUTs), where one of them will be an Intel Coffee Lake which has the traditional setup of all P-cores and the other is an Intel Raptor Lake CPU, which is comprised of a set of P- and E-cores. This allows us to analyze the impact this type of Asymmetric Multicore Processor (AMP) has on the parallel execution of test cases compared to a traditional Symmetric MultiCore Processor. To facilitate the structure of our procedure the following research questions have been formulated:

- **RQ1**: How does the compiler used to compile the test cases impact the energy consumption?
- **RQ2**:What are the advantages and drawbacks of the different types of measuring instruments in terms of accuracy, ease of use, and cost?
- **RQ3**:What effect does parallelism have on the energy consumption of the test cases?
- **RQ4**:What effect do P- and E-cores have on the parallel execution of a process, compared to a traditional desktop CPU?

To answer these research questions a command line framework will be created to assist with running a series of different experiments each with its own goal of answering one of the research questions.

In Section **2** the related work which lays the foundation for our work will be covered, including our previous work. This is followed by Section **3** which will include the necessary background information about e.g. CPUs and schedulers. Thereafter in Section **4** our experimental setup is presented, which includes the different measuring instruments which are tested in our work and the different test cases. Then in Section **6** the results are presented whereafter they are discussed in Section **7** and finally a conclusion can be made in Section **8** which will present the final answer to our research questions.

2 Related Work

2.1 Previous Work

This paper builds upon the knowledge gathered in our previous work *"A Comparison Study of Measur-*

ing Instruments"[3] where different measuring instruments were compared to explore whether a viable software-based measuring instrument was available for Windows. It was found that Intel Power Gadget (IPG) and Libre Hardware Monitor (LHM) on Windows have similar correlation to hardware-based measuring instruments as Intel's Running Average Power Limit (RAPL) has on Linux. This chapter builds upon the related work chapter of the previous work and as such will not be repeated, however, it will be expanded upon.

2.2 Parallel Software

Amdahl's law describes the maximum potential speedup that can be achieved through the parallelization of an algorithm based on the proportion of the algorithm that can be parallelized and the number of cores used.[4] In [5] Amdahl's law, was extended to also estimate the energy consumption. Then 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 rises 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. However, the comparison was based on an analytical model and not real measurements.[5]

[6] presented a program that solves Laplace equations and compares the observed speedup for computing the 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 speedup and the estimates it was clear that Gustafson's law is more optimistic than Amdahl's law, however, both estimate smaller speedups than the observed speedup on two and four cores. [6]

In [7], three different thread management constructs from Java are explored and analyzed regarding energy consumption. It found that as the number of threads increased, the energy consumption would do the same. This was however only to a certain point, where from this point the energy consumption would start to decrease as the number of threads started to approach the number of cores in the CPU. However, the peak of this energy consumption was application dependent.It also found that in eight out of nine benchmarks, there was a decrease in execution time going from sequential execution on one thread to using multiple threads. However, it should be noted that four of their benchmarks are embarrassingly parallel whereas only one was embarrassingly serial. It should also be noted that decreased execution time does not necessarily mean a decreased energy consumption, because in six out of nine benchmarks the lowest energy consumption was found in the sequential version using one thread. Furthermore, it investigated the energy-performance trade-off using the Energy-Delay-Product (EDP), which was the product between energy consumption and execution time. Using EDP it generally found parallel execution to be more favorable however depending on the benchmark increasing the number of threads may not be aligned with an improvement of EDP.[7]

In [8], the energy consumption for sequential and parallel genetic algorithms was explored, where one research question aimed to explore the impact on energy consumption when using different numbers of cores. It 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. Parallelism's ability to reduce energy consumption was argued to be due to the large number of cores working to solve the problem simultaneously, where the combination of more cores, and more parallel operations per time unit will require less energy. When considering parallel software, it also found asynchronous implementations to use less energy, because there are no idle cores waiting for data in asynchronous implementations, while in synchronous implementations cores can be blocked during runtime, while waiting for responses from other cores.

In [9], the behavior of parallel applications and the relationship between execution time and energy consumption are explored. It tests four different language constructs which can be used to implement parallelism in C#. Furthermore, it uses varying amounts of threads and a sample of micro- and macro-benchmarks. It found that workload size has a large influence on running 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. Additionally, it was found that execution time and energy consumption of parallel benchmarks do not always correlate. Comparing micro- and macro-benchmarks the findings remain consistent, although the impact becomes low for the macrobenchmarks due to an overall larger energy consumption. Furthermore, it has included some recommendations, which should be considered:[9]

- Shield cores: Avoid unintended threads running on the cores used in the benchmarking

- PowerUp: Can be used to ensure that benchmark is not optimized away during compilation
- Static clock: Make the clock rate of the CPU as static as possible
- Interrupt request: Avoid interrupt requests being sent to cores used in the benchmarking
- Turn off CPU turbo boost
- Turn off hyperthreading

2.3 Compilers

In [10] the language C++ and different compilers are explored and compared to find the impact of using different coding styles and compilers, where the goal is to find a balance between performance and energy efficiency. The different coding styles introduced explore the impact of splitting CPU and IO operations and interrupting the CPU-intensive instructions with sleep statements. The C++ compilers used in [10] include MinGW GCC, Cygwin GCC, Borland C++, and Visual C++, and the energy measurements are performed using Windows Performance Analyses (WPA). All compilers are used with default settings, and no optimizations were chosen. This decision was made based on works like [11], where it was found how mainstream compilers will apply multiple optimizations to the final code, where these optimizations in the worst case will result in worse performance and increased energy consumption. The issue of optimizations being very machine dependent was also shown in [12], where analysis and optimizations were done on a Texas Instruments C6200 DSP CPU. In [12] it was found that a large portion of the energy is used by fetching instructions which was addressed by introducing a fetch packet mechanism, and also find loop-unrolling to reduce energy consumption. While these optimizations decrease the energy consumption for the Texas Instruments C6200 DSP CPU, they note that for other CPUs varying results are expected. A similar conclusion is also found in [10], where they find that when choosing a compiler and coding style the energy reduction depends on the nature of the target machine and application. Based on the test case used, this being an election sort algorithm, they find the best performance with the Borland compiler, and the lowest energy with the Visual C++ compiler. When considering the coding styles, they find that both separating IO and CPU operations and interrupting the CPU-intensive instructions with sleep statements also decrease the energy consumption.

2.4 Asymmetric Multicore Processors

AMPs are CPUs where all the cores are not treated equally. One example of this is the combination of performance cores and efficiency cores as seen in Intel's Alder Lake and Raptor Lake. Intel's Thread Director (ITD) was presented with Intel's Alder Lake. The purpose of ITD is to assist the operating system decided on which cores to run a thread. In [13] support for utilizing ITD in Linux is created, however, official support for ITD has since been released. Additionally, some SPEC benchmarks were conducted to analyze the estimated Speedup Factor(SF) from the ITD compared to the observed SF. SF is the relative benefit a thread gets from running on a P-core. It looked at which classes were assigned to different threads in the benchmark, where it found that 99.9% of class readings were class 0 or 1. Class 0 is for threads that perform similarly on P- and E-cores. Class 1 is for threads where P-cores are preferred.[14] Furthermore class 3 was not used, which is for threads that are preferred to be on an E-core. 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 it still produced some mistakes, however, the correlation coefficient was higher at (> 0.8). Then it 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 gave superior SF predictions than ITD.[13]

3 Background

3.1 CPU States

This section provides an overview of CPU-states. The concept of CPU-states is concerned with how a system manages its energy consumption during different operational conditions. The C-states are a crucial aspect of CPU-states, as they dictate the extent to which a system shuts down various components of the CPU to conserve energy. The C0 state represents the normal operation of a computer under load.[15, 16] As the system moves from C0 to C10 [3], progressively more components of the CPU are shut down until, in C10, the CPU is almost inactive. It is important to note that the number of C-states supported may vary depending on the CPU and motherboard in use, in [3] the workstation used supported from C0 to C10 states.

In our work the C-states can have a large impact on the energy consumption of the test cases, especially the idle case as was found in [3].

3.2 Performance and Efficiency cores

The CPU architecture x86 has had a core layout comprised of identical cores. However, the ARM architecture introduced the big.LITTLE layout in 2011[17]. It is an architecture that utilizes two types of cores, a set for maximum energy efficiency and a set for maximum computer performance.[18]. Intel introduced a hybrid architecture in 2021[19] codenamed Alder lake, which is similar to ARM's big.LITTLE architecture. Alder lake also has two types of cores: performance cores (P-cores) and efficiency cores (E-cores). These types of cores are optimized for different tasks. P-cores are standard CPU cores, which focus on maximizing performance. In contrast, the E-cores are designed to maximize performance per watt and are intended to handle smaller non-time critical jobs, such as background services[20].

3.3 CPU Affinity

Affinity is a feature in operating systems(OSs) that enables processes to be bound to specific cores in a multi-core processor. In OSs, jobs and threads are constantly rescheduled for optimal system performance, which means that the same process can be assigned to different cores of the CPU. Processor affinity allows applications to bind or unbind a process to a specific set of cores or range of cores/CPUs. When a process is pinned to a core, the OS ensures it only executes on the assigned core(s) or CPU(s) each time it is scheduled.[21]

```
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#

Processor affinity is particularly useful for scaling performance on multi-core processor architectures that share the same global memory and have local caches referred to as the Uniform memory access architecture. Processor affinity is also useful for out study, as this allows the framework to assign a single or a set of cores and threads to a process.[21]

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

3.4 Scheduling Priority

Scheduling threads on Windows, is done based on each thread's scheduling priority level and the priority class of the process. For the priority the value 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.[22]

For the priority level, the levels 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.[22]

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 [22]. When assigning a base priority where both the priority class and thread priority are the default values, e.i.`NORMAL`, the base priority is 8.[22]

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. In the case of none of the highest priority threads being ready to run, the lower priority threads will be assigned time slices. The lower-priority threads will then execute until a higher-priority thread is available, in which case the system will assign a full time slice to the thread, and stop executing the lower-priority threads, without time to finish using its time slice.[22]

```
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#

Note when setting priority class and priority level for a process through C#, the priority class is supported for both Windows and Linux, while the priority level is only supported for Windows. An example of how both the priority class and priority level can be set for a process and its threads can be seen in Listing **2**.

3.5 OpenMP

OpenMP (Open Multi-Processing) is a parallel programming API consisting of a set of compiler directives and runtime library routines, with support for multiple platforms like Linux, macOS, and Windows as well as multiple compilers like GCC, LLVM/Clan, and Intel's OpenApi. OpenMP allows programmers to write parallel code for multi-core CPUs and GPUs.[23]

The directives provide a way to specify parallelism among multiple threads of execution within a single program, while the library provides mechanisms for managing threads and data synchronization. When using OpenMP programmers can write parallel codes and take advantage of multiple processors without having to deal with low-level details.[23]

When executing using OpenMP, the parallel mode used is called the Fork-Join Execution Model. This model works by first 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. This process is called a fork. 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.[23]

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

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

When using OpenMP, the parallel regions are identified using a series of directives and clauses, where the basic format 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.[23]

4 Experimental Setup

4.1 Measuring Instruments

This section present the different measuring instruments utilized in our work. The measuring instruments utilized in our previous work will only be briefly introduced, however more detail can be found in [3]. In this paper, four software-based measuring instruments and one hardware-based measuring instrument is used, where the hardware-based measuring instrument represents the ground truth.

Running Average Power Limit Intel's RAPL is a commonly used software-based measuring instrument seen in the literature.[3] It uses model-specific-registers (MSRs) and Hardware performance counters to calculate how much energy the processor uses. The MSRs RAPL uses include *MSR_PKG_ENERGY_STATUS*, *MSR_DRAM_ENERGY_STATUS*, *MSR_PP0_ENERGY_STATUS* and *MSR_PP1_ENERGY_STATUS*. Which corresponds to the power domains, PKG, DRAM, PP0, and PP1 which are explained in [3]. RAPL has previously only been directly accessible on Linux and Mac. In [3] we found that RAPL had a high correlation of 0.81 with our ground truth on Linux.[3]

Intel Power Gadget IPG is a software tool created by Intel, which can estimate the power of Intel processors. It contains a command line version called Powerlog which allows accessing the energy consumption using callable APIs. It uses the same hardware counters and MSRs as RAPL[24], therefore it is expected to observe similar measurements to that of RAPL. Which is also shown in [3] where we found that IPG had a high correlation of 0.78 with our ground truth on Windows. We also found that IPG had a high correlation of 0.83 with RAPL, although the measurements is on different operating systems.[3]

Libre Hardware Monitor LHM[25] is a fork of Open Hardware Monitor, where the difference is that LHM does not have a UI. Both projects are open source. LHM can use the same hardware counters and MSRs as RAPL and IPG and as such can measure the power domains PKG, DRAM, PP0, and PP1. Since it uses the methods to read energy consumption, a similar measurement is expected between LHM and IPG. We found that LHM correlated 0.76 with our ground truth on Windows. LHM was also found to have a high correlation of 0.85 with IPG.[3]

AC Current Clamp Serving as our ground truth measurement is our hardware-based measuring setup which is comprised of an MN60 AC clamp that is connected to the phase of the wire that goes into the

PSU. It is also connected to an Analog Discovery 2 which is used as an oscilloscope which in turn is then connected to a Raspberry Pi 4. This setup allows us to continuously measure and log our data. The accuracy is reported to be 2% For more detail see [3].

CloudFree EU smart Plugs Two CloudFree EU Smart Plugs[26] are used, as a lower-priced hardware-based measuring instrument, which also has greater ease of use than the AC Current Clamp setup. We have not found any information about their accuracy or sampling rate.

Scaphandre One measuring instrument not used in our previous work is Scaphandre[27]. Scaphandre is described as a monitoring agent which can measure energy consumption and is made for Linux where it can use Powercap RAPL which is a Linux kernel subsystem where data can be read from RAPL. It also has the functionality of measuring the energy consumption of some virtual machines, specifically Qemu and KVM hypervisors. A driver also exists which allows for installing RAPL on Windows.[28] Doing so allows using Scaphandre on a Windows computer where the sensor is RAPL which is utilizing the MSRs to update its counters. The Windows version of Scaphandre has some limitations but is able to report the energy consumption of the power domain PKG, using the MSR *MSR_PKG_ENERGY_STATUS*. Furthermore, it can also give an estimation of the energy consumption for individual processes. It does so by storing CPU usage statistics alongside the values of the energy counters. Then it is able to calculate the ratio of the CPU time for each Process ID (PID). With the calculated ratio a new calculation is made to get the subset of the energy consumption which is estimated to belong to a specific PID. A Linux exclusive feature is that the monitoring system Prometheus can be used with Scaphandre to get the energy consumption of an application which consist of several PIDs.

4.2 Energy Consumption Analysis

Dynamic Energy Consumption In [3, 29] dynamic energy consumption (DEC) was utilized to enable comparison between the software-based measuring instruments and the hardware-based measuring instruments. DEC is also used in our work. A brief explanation of DEC based on [29] is given:

$$E_D = E_T - (P_S * T_E) \quad (1)$$

In Equation (1) E_D is the dynamic energy consumption, 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 duration of the program execution. With this equation the energy consumption of the test case is isolated. Using DEC requires also measuring the energy consumption on an idle case. [29]

4.3 Device Under Tests

Two workstations are used as DUTs in the experiments. This was chosen to enable comparison between a CPU with and without P- and E-cores. In Tables **1** and **2** the specifications for the two workstations can be seen. They will be referred to as DUT 1 and DUT 2.

Workstation 1	
Processor:	Intel i9-9900K
Memory:	DDR? ??GB
Disk:	??
Motherboard:	??
PSU:	??
Ubuntu version:	??
Linux kernel:	??
Windows 11 version:	Build: ??

Table 1: The specifications for DUT 1

Dummy station	
Processor:	Intel ??
Memory:	DDR? ??GB
Disk:	??
Motherboard:	??
PSU:	??
Ubuntu version:	??
Linux kernel:	??
Windows 11 version:	Build: ??

Table 2: The specifications for DUT 2

4.4 Test cases

In this work, micro benchmarks and application benchmarks will be used to evaluate measuring instruments. This section will present the test cases used and why they were chosen.

Microbenchmarks : The microbenchmarks in this study will be used for initial experiments, and will be from the Computer Language Benchmark Game (CLBG)[1]. When chosing benchmarks, both single and multi-threaded versions were chosen, as experiments focusing on both single core and multi core performance will be run. One implication when chosing, was that each benchmark should be compiled on all compilers used in this study, and on both Windows

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

and Linux, and since a library like `<sched.h>`, which was used in many implementations, was only available on Windows, the pool of compatible benchmarks was limited. When executing the benchmakrs, the used parameters were the highest parameter used by CLBG. The microbenchmarks can be seen in Table **3**.

Microbenchmarks		
Name	Parameter	Focus
NBody	$50 * 10^6$	single core
Spectra-Norm	5.500	single core
Mandelbrot	16.000	multi core
Fannkuch-Redux	12	multi core

Table 3: Microbenchmarks

Application benchmarks

4.5 Compilers

In this section the different C++ compilers used will be introduced. Some of the compilers are based on [10], where it was found that applications compiled by Microsoft Visual C++ and MinGW had the lowest energy consumption. In addition to this Intel OneApi C++ compiler and Clang was also included as both can be found on lists of the most popular C++ compilers[30–32].

Microbenchmarks	
Name	Version
Clang	15.0.0
Min-Gw	12.2.0
Intel OneAPI C++	2023.0.0.20221201
MSVC	19.34.31942

Table 4: C++ Compilers

Clang : The Clang compiler is open source and builds on the LLVM optimizer and code generator. The compiler was released in 2007, and is available on both Linux and Windows.[33]

MinGW : The MinGW (Minimalist GNU for Windows) is an open source compiler based on the GNU GCC project, which compiles code to be run on Windows. MinGW can can also be cross-hosted on Linux.[34]

Intel OneApi C++ : The Intel OneApi is a set of libraries and other tools aiming to simplify development across different hardware. One of these tools is the C++ compiler, which implements SYCL, this being an evolution of C++ for heterogeneous computing. The compiler is available for both Windows and Linux.[35]

MSVC : Microsoft Visual C++ (MSVC) is a set of libraries and tools aiming to help developers build high performance code. One of the tools included is a C++ compiler. This compiler is only available on Windows.[36]

5 Experiments

In the following section the experiments conducted will be introduced. All experiments conducted in this section will be done on the framework presented in Appendix **A**, where the results will be saved in the database presented in Appendix **B**.

5.1 Experiment One

In the first experiment, **RQ1** will be explored. This experiment will be using both multi core test cases presented in Section **4.4** and the measurements will be performed using IPG. IPG was chosen as this was deemed the best measuring instrument in [3], where it was also observed how IPG and LHM made similar measurements. Given the point of this experiment is to find the best compiler, the expectations would be that a similar conclusion would be made if multiple measuring instruments were used.

6 Results

7 Discussion

8 Conclusion

Acknowledgements

9 Future Works

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. 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.
5. 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).
6. 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).
7. 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 (Oct. 2014).
8. 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).
9. 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).
10. 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).
11. Lima, E., Xavier, T., Faustino, A. & Ruiz, L. *Compiling for performance and power efficiency* in (Sept. 2013), 142–149.
12. Cooper, K. & Waterman, T. Understanding Energy Consumption on the C62x (Jan. 2004).
13. 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.
14. 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.
15. Intel. *https://www.intel.com/content/dam/develop/external/us/en/doc hill-sw-20-185393.pdf* https://www.intel.com/content/dam/develop/external/us/en/documents/green-hill-sw-20-185393.pdf. 2011. 07/03/2023.
16. 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.
17. 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.
18. ARM. *Processing Architecture for Power Efficiency and Performance* https://www.arm.com/technologies/big-little. 09/03/2023.
19. 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.
20. Rotem, E. *et al.* Intel alder lake CPU architectures. *IEEE Micro* **42,** 13–19 (2022).
21. *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.
22. Karl-Bridge-Microsoft. *Scheduling priorities - win32 apps* https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities. 03/03/2023.
23. Michael Womack, A. W. *Parallel Programming and Performance Optimization With OpenMP* https://passlab.github.io/OpenMPProgrammingBook/cover.html. 03/03/2023.
24. Mozilla. *tools/power/rapl* https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html. 24/02/2023.

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

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

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

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

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

30. 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.

31. Pedamkar, P. *Best C++ Compiler* 2023. https://www.educba.com/best-c-plus-plus-compiler/. 20/03/2023.

32. *Top 22 Online C++ Compiler Tools* 2023. https://www.softwaretestinghelp.com/best-cpp-compiler-ide/. 20/03/2023.

33. *Clang Compiler Users Manual* https://clang.llvm.org/docs/UsersManual.html. 20/03/2023.

34. *MinGW FAQ* https://home.cs.colorado.edu/~main/cs1300/doc/mingwfaq.html. 20/03/2023.

35. *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.

36. *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.

A The Framework

B 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 will be used, with some modifications made because the focus of this work is different to [3]. The design of the database can be seen in Figure **1**, where the `MeasurementCollection` table defines under what circumstances the measurements were made under. This includes what measuring instrument was used, what test case was running, what DUT the measurements were made on and whether or not there was a burn-in period ect. Compared to [3], a few extra columns has been added to `TestCase`, as more focus is on the test cases used in this work, compared to [3]. This includes metadata like compiler, optimizations, and parameters used.

In the `MeasurementCollection`, the columns `CollectionNumber` and `Name` represents what experiment the measurement is from, and the name of the experiment respectively. A column found in both `MeasurementCollection`, `Measurement` and `Sample` is `AdditionalMetadata`. This column can be used to set values unique for specific rows, where an example could be how some metrics are only measured by one measuring instrument.

The `Measurement` contains values for the energy consumption during the entire execution time of one test case, while the `Sample` represents samples taken during the execution of the test case. This means for one row in the `MeasurementCollection` table, there can exists one to many rows in `Measurement`. For each row in `Measurement`, there will be many rows in `Sample`, where the samples will be a fine grained representation of the measurement.

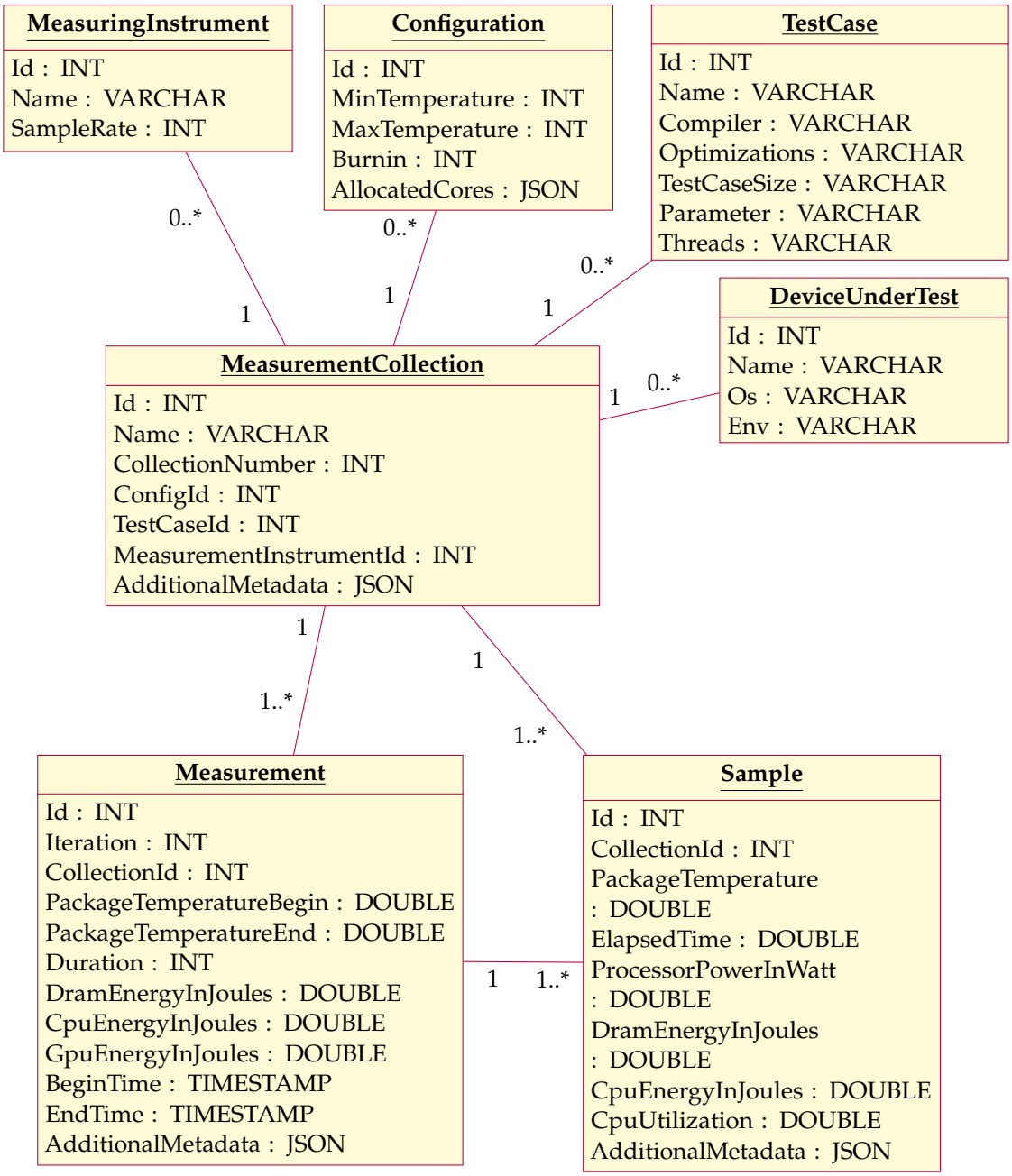

Figure 1: An UML diagram representing the tables in the SQL database