Student's name and surname: Grzegorz Koszczał
ID: 175405
Cycle of studies: postgraduate
Mode of study: Full-time studies
Field of study: Informatics
Specialization: Distributed Applications and Internet Services

MASTER'S THESIS

Title of thesis: Research on comparison of various methods for energy measurement for computational systems

Title of thesis (in Polish): Badanie porównawcze różnych metod pomiaru zużycia energii systemów obliczeniowych

Supervisor: dr hab. inż. Paweł Czarnul

GDAŃSK UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS

DECLARATION regarding the diploma thesis titled:
Research on comparison of various methods for energy measurement for computational systems

First name and surname of student: Grzegorz Koszczał
Date and place of birth: 11.02.1999, Olsztyn
ID: 175405

Faculty: Faculty of Electronics,Telecommunications and Informatics
Field of study: informatics

Cycle of studies: postgraduate
Mode of study: Full-time studies

Type of the diploma thesis: master's thesis

Aware of criminal liability for violations of the Act of 4th February 1994 on Copyright and Related Rights (Journal of Laws 2021, item 1062 with later amendments) and disciplinary actions set out in the Act of 20th July 2018 on the Law on Higher Education and Science (Journal of Laws 2022 item 574 with later amendments),[1] as well as civil liability, I hereby declare that the submitted diploma thesis is my own work.

This diploma thesis has never before been the basis of an official procedure associated with the awarding of a professional title.

All the information contained in the above diploma thesis which is derived from written and electronic sources is documented in a list of relevant literature in accordance with art. 34 of the Copyright and Related Rights Act.

15.08.2023, Grzegorz Koszczał

Date and signature of the student or authentication on the university portal Moja PG

**) The document was drawn up in the IT system, on the basis of paragraph 15 clause 3b of the Decree of the Ministry of Science and Higher Education of 12 May 2020, amending the decree concerning university studies (Journal of Laws of 2020, item 853). No signature or stamp required.*

[1] The Act of 20th July 2018 on the Law on Higher Education and Science:

Art. 312, section 3. Should a student be suspected of committing an act referred to in Art. 287 section 2 items 1–5, the rector shall forthwith order an enquiry.

Art. 312, section 4. If the evidence collected during an enquiry confirms that the act referred to in section 5 has been committed, the rector shall suspend the procedure for the awarding of a professional title pending a judgement of the disciplinary committee and submit formal notification on suspicion of committing a criminal offence.

ABSTRACT

The goal of the project is to conduct experiments and compare various methods of power draw measurement of parallel applications. The used measurement tools are both software (Intel RAPL, NVIDIA NVML) and hardware (Yokogawa WT310E). Moreover, the conditions of implementing mentioned solutions are evaluated, as well as their limitations for newest CPUs and GPUs.

Keywords: Informatics, parallel programming, High-Performance Computing Systems, green computing, NAS Parallel Benchmark, Linux Perf, Intel RAPL, NVML, PAPI, Internal Power Sensors, External Power Meters

Field of science and technology in accordance with OECD requirements: [PLACEHOLDER] ask Dr. Czarnul

CONTENTS

LIST OF IMPORTANT SYMBOLS AND ABBREVIATIONS

GPU – Graphics Processing Unit

INTRODUCTION

The Information and Communication Technology sector is responsible for a significant share of global electricity use. In 2020, the data centers, communication networks and user devices accounted for an estimated $4-6\%$ of global electricity use [35]. This value has grown exponentially over the last years, mainly due to technological advances such as cloud computing and the rapid growth of the use of Internet services [4]. Moreover, it is predicted that ICT energy use is likely to increase until 2030 and may reach approximately 13% of global electricity use. According to the increased energy demand, vast efforts has been put in order to improve the energy efficiency across the ICT sector with a success. As a result the overall energy use remains mostly flat according to some estimates.

The goal of energy efficiency increase is reached using various methods. Some of them involve the kernels optimizations and using the energy methods of computing [11]. Such methods focus mainly on load imbalance, mixed precision in floating-point operations. Other methods of increasing computational efficiency are related to power-capping [14]. Such an approach assumes setting a certain power limit level on CPU or GPU in order to achieve a power/performance trade-off. Appropriate limitations of power draw results in slightly longer execution times and significant savings, which render this method a viable option.

In order to make such implementation meaningful, we must make sure that the tools used to validate them are also as precise as possible. For the measurement of the power draw of a CPU, Intel provides its own interface, called Running Average Power Limit (RAPL) [23]. On the GPU side, NVIDIA provides the NVIDIA Management Library (NVML) [28]. Unfortunately, concerns rise on the precision of such softwares, mainly because their providers don't share any information about estimated error of power draw readings, leaving researchers questioning their practical use. In order to verify the precision of software measurement tools, the external measurement tool is used – the Yokogawa WT310E [39] [40]. Its high precision, backed up by certificates, makes it an excellent tool to benchmark the precision of Intel RAPL and NVML.

1. RESEARCH GOAL

1.1. PURPOSE AND RESEARCH QUESTION

The goal of the project is to verify the accuracy and reliability of the CPU and GPU power draw measurement tools during the computational-straining benchmark applications.

1.2. SCOPE AND LIMITATION

The scope of the project is to verify whether the software power measurement tools are precise, based upon the results of the certified external measurement tool and to specify their error range in case of inaccuracy.

In case of benchmark applications, a hypothesis worth considering is whether the change of computational data impacts the power draw measurements between hardware and software tools[NEED QUOTATION]. There are experiments[NEED QUOTATION] that proved that increasing the data used in the benchmarks influences the increasingly different results from the tools. Another hypothesis, however, makes the claim that the application of power capping does not impact the results. In order to make a conclusion, all those configurations should be investigated independently. Another aspect worth investigating is the use of various power supply units, both the server and consumer grade. Those tests would give us insight, whether they differ in total power draw or is it more likely associated with the certificates they have.

The project is limited to testing CPUs and software released by Intel Corporation and to testing the GPUs and software released by NVIDIA Corporation.

1.3. PROJECT REQUIREMENTS

The project requires a reliable testbed in order to run the computations, verified benchmark applications and certified external measurement tool. Moreover, the computational station must be exclusively reserved for the time of research in order to prevent other user's applications from interfering in the tests results, as well as the tests itself should be repeated several times in order to maintain credibility.

The workstations on which the experiments will be conducted are the two different university computational nodes. One of them consists of two server grade CPUs and eight GPUs, while the second one is equipped with two server grade CPUs and four GPUs. Both nodes are adapted to use a custom power strip that supports the use of external measurement tool.

The tests benchmark applications are considered suitable for the purpose of the project, when they are able to strain the workstation's hardware, using their entire computational resources. The chosen applications for this task are mainly NAS Parallel Benchmarks [5] [26] – a set of benchmarks designed to evaluate the performance of parallel computing systems. The NPB suite contains a variety of benchmarks, including linear algebra, FFT, stencil computations and others. The benchmarks are intended to be representative of some important real-world application problems and can be used to assess the performance of various systems under different conditions.

The external measurement tool is Yokogawa WT310E. It's a precise and reliable equipment that will serve as the ground truth in verification of reading from the software tools.

2. THEORETICAL BACKGROUND

2.1. MEASUREMENT SOFTWARE

2.1.1. INTEL RAPL

Intel RAPL (Running Average Power Limit) [17] is an interface [8], which allows software to set a power limit that hardware ensures and any power control system takes it as an input and tunes behavior to ensure that this operating limit is respected. That way, it is possible to set and monitor power limits both on processor and DRAM, and by controlling the maximum average power, it matches the expected power and cooling budget. RAPL exposes its energy counters through model-specific registers (MSRs) It updates these counters once in every 1 ms. The energy is calculated as a multiple of model specific energy units. For Sandy Bridge, the energy unit is 15.3 µJ, whereas it is 61 µJ for Haswell and Skylake.

Moreover, the Intel RAPL divides the platform into four domains, which consists of:

- PP0 (Core Devices) – Includes the energy consumption by all the CPU cores in the socket(s).
- PP1 (Uncore Devices) – Includes the power consumption of integrated graphics processing unit (unavailable on the server platforms)
- DRAM – The energy consumption of the main memory.
- Package – The energy consumption of the entire socket including core and uncore.

2.1.2. NVIDIA NVML

NVIDIA Management Library (NVML) [28] – A C-based API for monitoring and managing various states of the NVIDIA GPU devices. It provides direct access to the queries and commands exposed via nvidia-smi [32]. The runtime version of NVML ships with the NVIDIA display driver, and the SDK provides the appropriate header, stub libraries and sample applications. Each new version of NVML is backwards compatible and is intended to be a platform for building third party applications.

There are various techniques of computing the energy consumption using the NVIDIA Management Library, which query the onboard sensors and read the power usage of the device. Such techniques are either from the native NVML API, like Sampling Monitoring Approach (SMA) or Multi-Threaded Synchronized Monitoring (MTSM) or from CUDA component and it's called Performance Application Programming Interface (PAPI) [30].

Sampling Monitoring Approach (SMA) – The C-based API provided by NVML that can query the power usage of the device and provide an instantaneous power measurement. Therefore, it can be programmed to keep reading the hardware sensor with a certain frequency. The *nvmlDeviceGetPowerUsage()* function is used to retrieve the power usage reading for the device, in milliwatts. This function is called and executed by the CPU. The highest frequency possible is 66.7 Hz, which means the measurements are done every 15 ms.

Performance Application Programming Interface (PAPI) provides an API to access the hardware performance counters found on modern processors. The various performance metrics can be read through simple programming interfaces from C or Fortran. It could be used as a middleware in different profiling and tracing tools. PAPI can work as a high-level wrapper for different components. Previously it used only Intel RAPL's interface to report the power usage and energy

consumption for Intel CPUs, but recent updates added the NVML component, which supports both measuring and capping power usage on modern NVIDIA GPU architectures. The major advantage of using PAPI is that the measurements are by default synchronized with the kernel execution. The NVML component implemented in PAPI uses the function, *getPowerUsage()* which query *nvmlDeviceGetPowerUsage()* function. According to the documentation, this function is called only once when the command "papi end" is called. Thus, the power returned using this method is an instantaneous power when the kernel finishes execution.

Multi-Threaded Synchronized Monitoring (MTSM) – This method differs from SMA approach in the measurement period, a specific, exact window of the kernel execution is identified which results in recording of only the power reading of the kernel solely. Monitoring part is performed by the host CPU in that way the master thread calls and then monitors the kernel and other threads records the power, therefore it requires the use of parallel programming execution models, such as Pthreads [24] or OpenMP [33]. This approach at first initializes the volatile variable (at master thread) that is used later in recording of power readings. Then, the remaining threads execute the monitoring function in parallel and start measuring the time and power draw as the benchmark kernel starts doing the computation. After its work is done, the timing is ended and the stored measurements are synchronized, giving the precise logs of power consumption during the test period.

2.2. MEASUREMENT HARDWARE

2.2.1. YOKOGAWA WT310E

In order to perform comparison and to check the reliability of software power measurement methods, such as mentioned above Intel RAPL or NVIDIA NVML, it is mandatory to use a certified tool that would serve as the ground truth in such tests. Such a tool is Yokogawa WT310E – an external power meter that will serve this purpose in tests in this paper. It is a digital power analyzer that provides extremely low current measurement capability down to 50 micro-Amps, and a maximum of up to 26 Amps RMS. This device follows standards and certificates such as Energy Star® [34], SPECpower [36] and IEC62301 [20] / EN50564 [37] testing. This model comes from a WT300E's family of appliances that offer a wide range of functions and enhanced specifications, allowing handling of all the measurement applications from low frequency to high frequency inverters using a single power meter. The WT300E series with the fast display update rate of 100ms, offer's engineers a short tact time in their testing procedures. The basic accuracy for all input ranges is 0.1% rdg + 0.05% rng (50/60Hz) and DC 0.1% rdg + 0.2% rng.

To use the Yokogawa WT310E power meter, a special software has been written for easy use – the Yokotool [18]. Yokotool is a command-line tool for controlling Yokogawa power meters in Linux. The tool is written in Python and works with both Python 2.7 and Python 3. The tool comes with the 'yokolibs.PowerMeter' module which can be used from Python scripts.

(Work-In-Progress – Here I can cover a bit more of the use of Yokotool in this project)

(Work-In-Progress: add honorable mentions, such as WattsUp and Kill-A-Watt [12] and their use in previous works)

2.3. BENCHMARK APPLICATIONS

(Work-In-Progress)

[TO DO:

1. Cite 2 works from NPB-CPP and NPB-CUDA github repos TEST
2. Write more about my own multi-gpu benchmark, maybe cite myself]

For the purpose of the experiments, benchmark applications should fully utilize the resources of the tested CPUs and GPUs, as well as be able to run on various configurations parameters. Such parameters are: being able to run in parallel on various numbers of logical processors, being able to run on one or more GPUs and being able to execute on various input parameters class sizes.

There are four benchmark sets, that satisfies mentioned goals:

- NAS Parallel Benchmarks (C++ with OMP)
- NAS Parallel Benchmarks (Fortran with MPI)
- NAS Parallel Benchmarks (CUDA)
- Custom deep learning model, based on XCeptionnet with MPI communication

In a general sense, the NAS Parallel Benchmarks are a set of programs designed and created in order to help evaluate the performance of parallel supercomputers. They are based on computational fluid dynamics applications and originally consisted of five kernels and three pseudo-applications. Later on, the benchmark suite has been extended with more kernels, such as adaptive meshes, parallel I/O, multi-zone applications, and computational grids. Every application comes with predefined and indicated problem size, labeled as class size. Moreover, the benchmark kernels are available in widely-used programming models like MPI and OpenMP, which allows for easy configuration of use with various number of CPU threads (NPB-OMP and NPB-MPI) [15], as well as execution on two or more computational nodes (NPB-MPI only). For the computations on GPUs, different set has been created, which excels in tests on a single devices (NPB-CUDA) [2] [3]. In order To run GPUs benchmarks in multi-GPUs and multi-node environments, a custom deep learning model has been created solely for the purpose of this task.

Below is listed the original set of eight NPB benchmarks, which are tested later in the experiments conducted for the purpose of this work.

Five kernels:

- **IS** – Integer Sort (random memory access)
- **EP** – Embarassingly Parallel
- **CG** – Conjugate Gradient (irregular memory access and communication)
- **MG** – Multi-Grid on a sequence of meshes
- **FT** – Discrete 3D fast Fourier Transform (all-to-all communication)

Three pseudo applications:

- **BT** – Block Tri-diagonal solver
- **SP** – Scalar Penta-diagonal solver
- **LU** – Lower-Upper Gauss-Seidel solver

The three pseudo applications mentioned earlier also comes with the multi-zone versions, designed to exploit multiple levels of parallelism. Moreover, NPB suite also offers benchmarks for unstructured computation, parallel I/O and data movement. For the purpose of this work only the original single-zone kernels and applications were used, therefore these benchmarks suites are mentioned only.

In addition to solving different computational problems, each kernel or application operates on various sizes of input data, determined during compilation, known as classes. Those classes helps choosing the right benchmark in term of execution time, which helps in measurements. Too short benchmarks may cause measurement error, due to low sampling rate of measurement instruments and too long benchmarks are unnecessary, because they prolong the overall experiments.

Benchmark classes:

- **Class S** – Very small, used for quick test purpose. Nowadays obsolete.
- **Class W** – The so-called '90's workstation' size, nowadays consisted small.
- **Classes A, B, C** – Standard test problems (∼4 times size increase from each of the previous classes)
- **Classes D, E, F** – Large test problems (∼16 times size increase from each of the previous classes)

2.3.1. NPB FOR CPU, C++ WITH OMP

NAS Parallel Benchmarks for CPU has been ported from Fortran to C++ – a programming language that has been developed for a long time as an Object-Oriented successor of a very popular and successful C language. It is a strongly typed, highly portable language with very high performance, compatibility and excellent yet difficult, manual memory management, which makes it a grea choice for writing code, that is meant to use in High Performance Computing, where fast execution times and low memory usages are among the most important factors. C++ offers also many well-optimised libraries to choose from, based on our goals, as well as extensive amount of documentations and books, explaining the principles of the language.

OpenMP (Open Multi-Processing) is an API (Application Programming Interface) and set of directives for parallel programming in shared-memory multiprocessing environments, primarily used for multi-threading. It's designed to simplify the development of parallel applications by allowing developers to add parallelism to their code through compiler directives and library functions. OpenMP is particularly popular in scientific and high-performance computing applications where performance optimization is crucial.

Since NPB-OMP comes with up to eight benchmarks and each benchmark can be compiled with various class sizes, it creates a very diverse and flexible test cases to choose from, depending of our needs. More informations about choosing the correct configurations for our purposes are explained in Chapter 4: 'Proposal of the Solution'

Another major advantage of these benchmarks are the fact, that after the compilation, one can be executed and pinned to specific logical processor using Linux's *taskset* command [25], which is mainly used to set or retrieve the CPU affinity of a running process. In practice, it allows to set exact number of created benchmark processes and allows to manually pin them to the desired threads, which gives us the absolute control over choosing the benchmark configuration and gives us confidence, that they are correctly executed.

2.3.2. NPB FOR CPU, FORTRAN WITH MPI

Another suite of benchmarks created for execution on CPUs – an older set that was written in the world's first high level programming language, the Fortran. Fortran has been widely used in scientific, engineering and numerical computing applications for several decades. Its main

advantages such as high performance, standarized libraries, support for complex numbers and parallel computing makes it a valuable and reliable choice in numerical and scientific computing to this day.

MPI (Message Passing Interface), is a standardized and portable message-passing system designed for parallel and distributed computing. It provides a set of routines and libraries for high-performance communication and coordination among processes or tasks in a parallel or distributed computing environment. MPI is particularly popular in high-performance computing (HPC) and scientific computing, where large-scale parallelism is essential for solving complex problems. In contrast to OMP, this implementation provides inter-node communication for execution of code on several nodes in parallel manner.

Similar to NPB-OMP, this benchmarks suite also comes with many kernels and class sizes to choose from, but it should be noted, that it does not offer complete flexibility in therms of choosing the number of processes – some kernels require a certain number of processes to run which are specified in the code documentation, and are caused by the way, the implementation was created.

2.3.3. NPB FOR GPU, CUDA

CUDA (Compute Unified Device Architecture), is a parallel computing platform and application programming interface (API) developed by NVIDIA Corporation. CUDA is specifically designed for accelerating general-purpose computational tasks on NVIDIA GPUs (Graphics Processing Units). It allows developers to harness the massive parallel processing power of GPUs to accelerate a wide range of applications, including scientific simulations, machine learning, image processing, and more.

Key features and concepts of CUDA include, but are not limited, to:

- Parallel Computing Model: CUDA enables developers to write parallel code that can be executed on GPUs.
- GPU Acceleration: CUDA provides a means to offload computationally intensive tasks from the CPU to the GPU.
- CUDA C/C++ Language Extensions: Developers can write CUDA code using C or C++ with CUDA-specific extensions. These extensions allow developers to define GPU kernels, which are functions that run in parallel on the GPU.
- CUDA Toolkit: The CUDA Toolkit includes the CUDA compiler, runtime libraries, and development tools. It enables developers to write, compile, and optimize CUDA applications.

As mentioned in the previously shown, CPU implementations, this benchmarks suite also comes with various kernels and class sizes to choose from, according to our needs. One important thing should be noted, however – due to the nature of the way it has been implemented, the index number of the GPU, that will be used during tests must be specified in the configurations file during compilation. Therefore this benchmark sets are single-GPU only and require creation of many executable files with various configs, if the test server has many GPUs. This problem is explained in more detail and ultimately resolved in Chapter 4.

2.3.4. CUSTOM DEEP NEURAL NETWORKS MODEL

Last benchmark application used in this work fills the remaining gap for test suite – an multi-GPUs kernel that utilizes MPI and has potential for multi-node training. This application was created solely for the purpose of this paper and utilizes high-level deep learning frameworks,

such as TensorFlow and Keras, widely known and used, general-purpose programming language such as Python, as well as Horovod [21] framework, that utilizes MPI and NCCL communication libraries. The mentioned components used in this benchmark are more precisely described below:

Python is a versatile and widely-used high-level programming language known for its simplicity, readability, and extensive standard libraries. The fact that it is a general-purpose language means, that it can be used in various fields, such as web development, data analysis, scientific computing and artificial intelligence. Its high-level abstraction provides a clean and readable syntax and the fact that its an interpreted language means that there is no need to compile the code before running it, shortening the time between each tests of new features. Moreover, Python comes with a large and comprehensive standard library for basic programming tasks such as file I/O, regular expressions, networking and data manipulation, and in case if someone needs more specific libraries, it features an easy solution for installing additional packages using PIP (Python Index Package).

TensorFlow is a comprehensive deep learning framework that provides low-level and high-level APIs. While one can use TensorFlow for a wide range of machine learning and deep learning tasks, it also includes the Keras API as a high-level component. TensorFlow provides more low-level control, allowing developers to customize their models and training loops extensively. This is useful for researchers and engineers who need fine-grained control over model architecture and training procedures.

Keras is an open-source high-level neural networks API written in Python. It provides a user-friendly, high-level interface for building and training neural networks. Keras was designed with simplicity and ease of use in mind.

Both TensorFlow and Keras offers extensive documentations that allows user to quickly learn the basics of theirs modules and libraries. Moreover, organizations such as Google [0] or Kaggle [0] offers a practical approach in learning such concepts, especially Kaggle comes with a wide array of many datasets and exemplary solutions, created by the community members.

Horovod [0] is an open-source distributed deep learning framework designed to make it easier to train machine learning models on distributed computing environments, such as clusters or cloud-based setups. It was developed by Uber Engineering and is designed to provide efficient and scalable distributed training for deep learning models. Horovod is particularly well-suited for distributed training with popular deep learning libraries like TensorFlow, PyTorch, and MXNet. Main advantages of Horovod framework are: large scale distributed training, that allows multi-GPU and multi-node support, efficient data parallelism, compatibility with deep learning frameworks, such as TensorFlow and Keras, resource awarness and dynamic scaling.

The custom deep learning benchmark, created for the purpose of this Master's Thesis, tackles the problem of image classification. It involves training a neural network to categorize or classify images into predefined classes or categories. The goal is to teach the model to recognize patterns and features in images that distinguish one category from another. This problem is commonly addressed using supervised learning, where the neural network learns from a labeled dataset containing examples of images and their corresponding class labels. The dataset used has been taken from the Kaggle website [0] classes of the dataset represents various species of birds, that are labeled. The birds in nearly all images occupies over 50% of the image, the photos themselves are of good quality $-$ no images are blurry, done in bad lighting or misslabeled. All images are $224 \times 224 \times 3$ color images in .jpg format. Considering all the informations about the dataset mentioned above, one can expect a good results when training deep neural networks models on it. During the tests and fine-tuning phase of the creation of the model, a very high accuracy of over 90% has been achieved in relatively small number of iteration. What is more im-

portant in the terms of this work, the utilization of the GPUs and their power draw has been very high, thus meeting the requirement for a good benchmark application, to use possibly as much of the devices resources during measurements as possible. Main features in this benchmark, that allowed to achieve such great results are: usage of Horovod innate function to automatically allocate memory growth based on chosen number of devices, Horovod handling the deep learning distributed optimizer among the devices via callbacks, TensorFlow's *.cache()* and *.prefetch()* utilizing maximum of GPUs VRAM and finally, usage of XCeption Model – a deep convolutional neural network architecture that involves Depthwise Separable Convolutions, that has been developed mainly for the purpose of solving the images classification problems.

3. RELATED WORK

3.1. 'A COMPARATIVE STUDY OF METHODS FOR MEASUREMENT OF ENERGY OF COMPUTING'

Authors of this work [10] investigated the accuracy of measurement of energy consumption during an application execution in order to ensure the application-level energy minimization techniques. They mentioned three most popular methods of measurement: (a) System-level physical measurements using external power meters; (b) Measurements using on-chip power sensors and (c) Energy predictive models. Later, they presented a comparison of the state-of-the-art on-chip power sensors as well as energy predictive models against system-level physical measurements using external power meters, which played the role of a credible measurement appliance.

The methodology is as follows: The ground truth tests were performed at first, using WattsUp Pro Meter [0]. The group of components that were running the benchmark kernel were defined as abstract processor – a whole that consists of multicore CPU processor, consisting of a certain number of physical cores and DRAM. In order to perform meaningful measurements, only a certain configuration of application was run, that executed solely on an abstract processor, without need of using any other system resources, such as solid state drives or network interface cards and so on. The result of such an approach is that HCLWattsUp reflects solely the power drawn from CPU and DRAM.

The researchers took other important precautions during the experiment. At first, all the resources they used have been reserved and fully dedicated to the experiment, making sure that no unwanted, third-party applications are being run in the background. During the tests, they actively monitored the disk consumption and network to ensure that there is no interference in the measurements. Another important factor of the testbed that draws power and generates heat are the fans. In default configuration, the fans speed is dependent on the increasing temperature of the CPUs, which rises as the time of the training goes on. That generates a dynamic energy consumption that impacts negatively on the outcomes of the experiment. To rule this phenomenon out, all the fans in the entire testbed are set at full speed, therefore they draw the same amount of power regardless of the actual strain put on CPUs and their temperature. All the procedures mentioned above ensure that the dynamic energy consumption obtained using HCLWattsUp, reflects the contribution solely by the abstract processor executing the given application kernel.

After determining the "ground truth" measurements using the configuration with HCLWattsUp, the researchers conducted a series of tests that determined the dynamic energy consumption by the given application using RAPL. At first the Intel PCM/PAPI was used to obtain the base power of CPUs, both core and uncore as well as DRAM, with no straining workload applied. Then, in the next phase, using HCLWattsUp API the execution time of the given application has been obtained. After that, the Intel PCM/PAPI has been used in order to obtain the total consumption of the CPUs and DRAM, all within the execution time of a given benchmark. Lastly, the researchers responsible for the experiment calculated the dynamic energy consumption of the abstract processor by subtracting the base energy from the total energy drawn during the kernel execution. To determine the dynamic energy consumption using HCLWattsUp, all the steps mentioned before has been repeated, but using the HCLWattsUp software instead of the Intel RAPL.

The execution time of the benchmark kernels were the same for both of the power draw measurement tools, so any difference between the energy readings of the tools comes solely from their power readings. Finally, tests were conducted on three different sets of experiments in order

to receive three different types of patterns.

In the first set of experiments, the FFTW and MKL-FFT energy consumption has been explored, by using a given workload size. For many tests on various problem sizes, the Intel RAPL reports showed less dynamic energy consumption for all application configurations than HCLWattsUp, but it follows the same pattern as HCLWattsUp for most of the data points. Therefore it is possible to calibrate the RAPL readings, which resulted in significant decrease of average error for the dynamic energy profile.

Second set of tests was conducted using OpenBLAS DGEMM. Executions were, again, performed using various configurations of data sizes, but results were less satisfactory than in the first tests. Like the first set of experiments, RAPL profiles lag behind the HCLWattsUp profiles. Unlike the first set of experiments where the error between both the profiles could be reduced significantly by calibration, the reduction of the average error for most of the application configurations was only as half as effective contrary to the first set of tests. This calibration, however, is again not the same for all the application configurations.

In the third and last set of experiments the team studied the dynamic energy behavior of FFTW as a function of problem size N x N. The tests were performed in different problem ranges. Researchers claim that for many data points, RAPL reports an increase in dynamic energy consumption with respect to the previous data point in the profile whereas HCLWattsUp reports a decrease and vice versa. Therefore it is impossible to use calibration to reduce the average error between the profiles because of their interlacing behavior.

As a conclusion, readings from Intel RAPL and HCLWattsUp differ strongly based on executed benchmark and data size. In the first set of experiments, the FFTW and MKL-FFT energy consumption test, the Intel RAPL readings followed the pattern of HCLWattsUp readings, being even more accurate after calibrations. The second test however, showed that RAPL does not follow most of the energy consumption pattern of the power meter. This could be tuned to some extent by calibration, but not as good as in the first test case. In the last experiment, however, the RAPL does not follow the energy consumption pattern of the power meter and can not be calibrated, leaving the readings quite troublesome.

Next experiment conducted in this paper was the comparison of measurements by GPU and Xeon Phi Sensors with HCLWattsUp. The methodology of work is similar to the one explained before – the entire testbed is reserved solely for the purpose of the experiment, the fans are set to the maximum speed and only the abstract processor is measured. To strain the hardware, two applications were used:

The first one was the matrix multiplication (DGEMM), the second one was 2D-FFT. Tests were performed on two NVIDIA GPUs: K40c [13] and P100, and one Intel coprocessor, the Intel Xeon Phi 3120P [16]. To obtain the power values from on-chip sensors on NVIDIA GPUs, the dedicated libraries were used, called NVIDIA NVML [29] and to obtain power values from Intel Xeon Phi, the Intel System Management Controller chip (SMC) [22] was used. Values from the Intel Xeon Phi can be programmatically obtained using Intel manycore platform software stack (MPSS) [17]. The methodology taken to compare the measurements using GPU and Xeon Phi sensors and HCLWattsUp is similar to this for RAPL. Briefly, HCLWattsUp API provides the dynamic energy consumption of an application using both CPU and an accelerator (GPU or Xeon Phi) instead of the components involved in its execution. Execution of an application using GPU/Xeon Phi involves the CPU host-core, DRAM and PCIe to copy the data between CPU host-core and GPU/Intel Xeon Phi. On-chip power sensors (NVML and MPSS) only provide the power consumption of GPU or Xeon Phi only. Therefore, to obtain the dynamic energy profiles of applications, the Intel RAPL was used to determine the energy contribution of CPU and DRAM. Energy contributions

from data transfers using PCIe were considered as not significant.

At first, the DGEMM was used as the test benchmark with various workload sizes. The energy readings from the GPU NVIDIA K40c sensors exhibit a linear profile whereas HCLWattsUp does not. Moreover, the sensor does not follow the application behavior exhibited by HCLWattsUp for approximately two-thirds of the data points. In the case of the Intel Xeon Phi coprocessor, the results seemed to be better – sensors follow the trend exhibited by HCLWattsUp for third-fourth of the data points. However, sensors report higher dynamic energy than HCLWattsUp, but that can be reduced significantly using calibration.

In the case of the second benchmark, the 2D-FFT, the measurements by NVML follow the same trend for the majority of the data points, compared to the results from NCLWattsUp. The sensor of the Intel Xeon Phi followed the trend of HCLWattsUp for over 90% of all data points, which is a good result. Overall, Intel RAPL and NVML both exhibit the same trend for FFT. Therefore, the difference with HCLWattsUp comes from both sensors collectively.

The results of this test allows to draw several conclusions. First, the average error between measurements using sensors and HCLWattsUp can be reduced using calibration, which is, nevertheless, specific for an application configuration. Another important finding is that CPU host-core and DRAM consume equal or more dynamic energy than the accelerator for FFT applications (FFTW 2D and MKL FFT 2D),which means that data transfers (between CPU host-core and an accelerator) consume same amount of energy as that for computations on the accelerator for older generations of NVIDIA Tesla GPUs such as K40c and Intel Xeon Phi such as 3120P. However, for newer generations of Nvidia Tesla GPUs such as P100, the data transfers consume more dynamic energy than computations. It suggests that optimizing the data transfers for dynamic energy consumption is important.

3.2. 'VERIFIED INSTRUCTION-LEVEL ENERGY CONSUMPTION MEASUREMENT FOR NVIDIA GPUS'

Authors of this research paper [1] investigated the actual cost of the power/energy overhead of the internal microarchitecture of various NVIDIA GPUs from four different generations. In order to do so, they compared over 40 different PTX instructions and showed the effect of the CUDA compiler optimizations on the energy consumption of each instruction. To measure the power consumption, they used three different software techniques to read the GPU on-chip power sensors and determined their precision by comparing them to custom-designed hardware power measurement. The motivation of their work comes from the fact that in order to increase the performance of the GPUs, their power consumption must be correctly and reliably measured, because it serves as a primary metric of performance evaluation. This issue is proven even more challenging, since the GPU vendors never publish the data on the actual energy cost of their GPUs' microarchitecture, therefore the independent research should be conducted in order to verify the power measurement software they provide.

The authors of the research paper prepared a set of special micro-benchmarks to stress the GPU, in order to capture the power usage of each PTX instruction [31], so the instructions were written in PTX as well. PTX is a virtual-assembly language used in NVIDIA's CUDA programming environment whose purpose is to control the exact sequence of instructions executing without any overhead. The researchers prepared two kernels for the purpose of this work – first one is tasked with adding integers and second one responsible for dividing variables with unsigned values. Since it is impossible to capture power draw of an execution of a single instruction, a different approach was proposed: the same instruction has been repeated millions of times and the power

drawn during the entire test case has been measured. Then the amount of power reported by the measuring system was divided by the total number of instructions, giving the power consumed by a single PTX instruction. It is worth noting that GPUs drain power as static power and dynamic power. The static power is a constant power that the GPU consumes to maintain its operation. To eliminate the static power and any overhead dynamic power other than the instruction power consumption, the kernel's was computed twice and the energy consumption was measured both times. First, the kernel was run in a configuration to measure the total energy drawn for the operation. In the second run the back-to-back instructions were omitted and the energy measured was defined as overhead energy. Energy used on instruction was defined as subtraction of total energy and overhead energy, divided by the total number of instructions.

In this experiment, the ground truth of energy drawn by the GPUs was set by the external power meter. It was pointed out that the GPUs have two power sources: one is direct DC power, provided by a PSU, another one is the PCI-E power source, provided through the motherboard. In order to capture the total power, the measurement of current and voltage has been done for each power source simultaneously. A clamp meter and a shunt series resistor were used for the current measurement. For voltage measurement, a direct probe on the voltage line using an oscilloscope has been used. In case of measurements of current and voltage on the direct DC power supply source, everything was measured using an oscilloscope, therefore the power draw calculations were performed using a certain formula. The measurement of the PCI-E power source was more difficult. Since there wasn't any possibility to directly receive current or voltage, the authors of this paper decided to set up a special PCI-E riser board that measures the power supplied through the motherboard. Two in-series shunt resistors are used as a power sensing technique. Using the series property, the current that flows through the riser is the same current that goes to the graphics card, same with the voltages.

The experiment has been conducted for four NVIDIA GPUs from four different generations/architectures: GTX TITAN X from Maxwell architecture, GTX 1080 Ti from Pascal architecture, TITAN V from Volta architecture and TITAN RTX from Turing architecture. To compile and run the previously prepared benchmarks, the CUDA NVCC compiler [27] has been used. The results of the tests show that NVIDIA TITAN V has the lowest energy consumption per instruction among all the tested GPUs. Additionally the tests were performed on both CUDA optimized and non-optimized versions of code, and overall the optimized versions of instruction proved to be less energy hungry than the non-optimized ones. In terms of differences between various software power measuring techniques, namely PAPI versus MTSM, The dominant tendency of the results is that PAPI readings are always more than the MTSM. Although the difference is not significant for small kernels, it can be up to 1 μJ for bigger kernels like Floating Single and Double Precision div instructions. Different software techniques (MTSM and PAPI) have been compared against the hardware setup on Volta TITAN V GPU. Compared to the ground truth hardware measurements, for all the instructions, the average Mean Absolute Percentage Error (MAPE) of MTSM Energy is 6.39 and the mean Root Mean Square Error (RMSE) is 3.97. In contrast, PAPI average MAPE is 10.24 and the average RMSE is 5.04. The results prove that MTSM is more accurate than PAPI as it is closer to what has been measured using the hardware.

3.3. 'MEASURING GPU POWER WITH THE K20 BUILT-IN SENSOR'

Authors of this research paper [7] investigated accurate profiling of the power consumption of GPU code when using the on-board power sensor of NVIDIA K20 GPUs. Moreover, two major anomalies that happened during the tests were more thoroughly analyzed – the first one being related to the doubling a benchmark kernel's runtime resulted with more than double energy usage, the second indicated that running two kernels in close temporal proximity inflates the energy consumption of the later kernel. Based on previous work in a similar field and set of preliminary tests, a new, reliable methodology [19] has been proposed as the conclusion of this experiment.

GPUs used in this project are NVIDIA Tesla K20, equipped with on-board sensors for querying the power consumption at runtime. As noted by the authors of the work, measurement of the power draw of the GPU using its built-in sensor is more complex than it would seem at first glance. The straightforward approach of sampling the power, computing the average, and multiplying by the runtime of the GPU code is likely to yield large errors and nonsensical results, hence the anomalies related to more energy used than expected due to increase of kernel's runtime or kernel's energy consumption increase after consecutive runs. Therefore another approach must be adopted. Methodology of the experiment is as follows: a number of unexpected behaviors when measuring a GPU's power consumption have been noted for further investigation, various observations has been noted during the tests runs conducted on the NVIDIA K20 GPUs and based on those observations and other related work, a correct way of measuring the power and energy consumption using sensor has been created. Later on it was validated for reliability by performing it multiple ways on many GPUs based on Kepler architecture, equipped with power sensor, such as the NVIDIA K20c, K20m, and K20x. The custom tool, created by the authors of the work, has been published for future use by other scientists, as an open source code.

Benchmark applications used in this paper solved two different n-body problem implementations. The algorithm models the simulation of gravity-induced motion of stars in a star cluster. The first kernel, called NB (N-Body), performs precise pairwise force calculations, which means that the same operations are performed for all n bodies, leading to a very regular implementation that maps well to GPUs. Moreover, the force calculations are independent, resulting in large amounts of parallelism. The second code, called BH, uses the Barnes-Hut algorithm to approximately compute the forces [9] [38]. It hierarchically partitions the volume around the n bodies into successively smaller cubes, called cells, until there is just one body per innermost cell. The resulting spatial hierarchy is recorded in an unbalanced octree. Each cell summarizes information about the bodies it contains. The NB code is relatively straightforward, has a high computational density, and only accesses main memory infrequently due to excellent caching in shared memory. In contrast, the BH code is quite complex, has a low computational density, performs mostly irregular [6] pointer-chasing memory accesses, and consists of multiple different kernels. Nevertheless, because of its lower time complexity, it is about 33 times faster than the NB code when simulating one million stars.

In order to conduct the energy measurement from the GPU power sensor, the authors of the work wrote their own tool to query the sensor via the NVIDIA Management Library (NVML) interface, which returns the power readings in milliwatts. The sampling intervals of the measurement are lowest possible – 15 ms between measurements. At first, during the tests, there was a noticeable power lag and measurement distortion – power profiling tends to lag behind the kernel activity and shape of the profile does not match the kernel activity in both shape (minor difference) and time (major difference). The key insight in creating a model of correct measurement is the fact

that the power sensor gradually approaches the true power level rather than doing so instantly. Since the 'curved' power readings between time when kernel start running and time when the power curve stabilizes reminded the authors of this work of capacitors charging and discharging, they tested whether the power profiles can be described by the same formulas. This turned out to work very well in the end. It is assumed that this is the case because the power sensor hardware uses a capacitor of some sort. After this revelation, the authors determined the 'capacitance' of the power sensor by using a single capacitor function to approximate the curve between the kernel start time and kernel stop time. After that, they determined the value of the capacitance that minimizes the sum of the differences between the measured values and the function values. As the capacitance is constant, it only needs to be established once for a given GPU, which is C = 833.333 on all tested K20 GPUs. Computing the true power draw value then become a single function of the slope of power profile derived in time domain and is shown in a function below:

$$P_{true}(t_i) = P_{meas}(t_i) + C \times (\frac{P_{meas}(t_{i+1}) - P_{meas}(t_{i-1})}{t_{i+1} - t_{i-1}})[W] \tag{3.1}$$

Moving back onto the recommended steps for this experiment, following assumption should be considered: highest possible sample rate for NVML (which is 66.7 Hz / 15 ms between intervals) as well as including the time stamps, removal of consecutive samples of the same value that are no more than 4 ms apart of each other, computation of true power with the equation mentioned above and finally, computation of the true energy consumption by integrating the true power, using the time stamps, over all intervals where the power level is above the 'active idle' threshold of 52.5 W.

After incorporating the steps mentioned above in the tests, the authors of the research paper validated their results. To do so, they checked if the computed power profile follows the GPU kernel activity and also they revisited anomalies that they encountered before in order to check if their new approach eliminates them. In the end, the profiling almost instantly shoots up when the kernel starts, stays at a (more or less) constant level during execution, and almost instantly drops to the aforementioned 52.5 W after the kernel stops. Importantly, the power level during execution coincides with the asymptotic power between kernel start and kernel stop, which verifies the above hypothesis. This observation gives insight that power should be integrated from the time point of kernel start to the point of kernel end. Any energy consumption by the GPU before or after the kernel execution is due to idling (at different power levels) and is a function of time but independent of the kernel. In case of anomalies, the first one was regarding the kernel runtime changes and unintuitive increase of measured power draw. In the early tests that were based solely on readings from NVML, after increasing the kernel's runtime by two times, the power draw readings were not increased proportionally as well, they were higher by approximately 8% than expected. The corrected power profile, however, indicates that, in fact, the energy consumption increase follows the total runtime one-to-one, thus resolving this particular anomaly. The second anomaly was that running the same kernel twice in close temporal proximity inflates the energy consumption of the second invocation. Once again, the power profiling proposed by the author clearly resolves this anomaly as the two corrected profiles are now at the same level (and the power level between the kernel runs is at the active-idle level). In other words, the computed power profile of a kernel is unaffected by prior kernel runs, which is an important advantage of this approach. This means that there is no need to have to delay kernel runs until the GPU reaches its idle power level before one can measure the energy consumption of the next kernel. Those tests were also validated on other GPUs and the results showed that they behave in a similar manner to the first one. The only notable difference is that all of the measurements are a few watts higher

on the second GPU. The difference is, however, within the 5 W absolute accuracy of the sensor. The profiled power obtained from tests on other GPUs are all very similar to each other, which further validates the used methodology.

As a conclusion of this work, many results and insights were obtained, such as: Power profile is distorted with respect to the kernel activity; the measured power lags behind the kernel activity; running multiple kernels one after another inflates the power draw of the later kernels; after a short-running kernel, the power draw can even increase for a while; integrating the power to a discernable time after a kernel stops does not correctly compensate for the power lag; the sampling interval lengths vary greatly; the GPU sensor only performs power measurements once in a while; the true sampling rate may be too low to accurately measure short-running kernels and the PCI-bus activity is not included in the sensor's measurements. This paper proposes and evaluates a power- and energy-measurement methodology that is accurate even in the presence of the above problems. It computes the true instant power consumption from the measured power samples, accounts for variations in the sampling frequency, and correctly identifies when kernels are running and when the GPU is in active-idle mode waiting for the next kernel launch.

4. PREPARATIONS TO EXPERIMENTS

4.1. CHOOSING THE CORRECT CONFIGURATIONS

In order to receive a wide set of data from the measurements, tests should be conducted using broad range of benchmark kernels with various classes sizes. To break down the hierarchy of the test cases, it is a sound strategy to start from the biggest piece – the server. Computational nodes used for the experiments are server-grade machines, codenamed 'sanna.kask' and 'vinnana.kask' respectively. Next step are the devices that the individual benchmarks use during runs. Each node utilize three setting: 'CPUs', 'GPUs' and the 'Hybrid' benchmarks, which are combination of previous two. After choosing the device, the correct implementation is specified, based both on the currently used server and device. This way, it has been decided, that in order to diversify the tests, implementation on the servers should vary. Experiments on 'sanna.kask' are utilizing the implementation based on OpenMP, where the 'vinnana.kask' server uses the solutions with MPI, TensorFlow and Horovod. Based on the device utilized, implementation are either OMP-C++ or MPI-Fortran for using on the CPUs and OMP-CUDA or Horovod-Python on GPUs. Another parameter of the tests are the benchmarks, which are the short applications or kernels, that solves various problems based on computational fluid dynamics. The main goal of the benchmark is to evaluate the performance of workstations by utilizing its targeted resources to the maximum. Properly conducted tests should include various benchmarks in order to avoid bias on the collected measurements data. The sets of used benchmarks are more complex that the previously mentioned parts of the configuration and will be explained graphically on the charts in the next section, but it should be mentioned, that all implementation except Horovod-Python uses three different benchmarks with various class sized. The Horovod-Python implementation use only one application, the custom deep neural networks model. Finally, based on both implementation and benchmark, the configuration is defined. By configuration we should expect the resources used – for the CPUs benchmark, such resources are the CPUs themselves and their logical processors, and for the GPUs benchmarks – the GPUs themselves and reserved for them CPUs threads. Again, the amount of configurations set for the purpose of the experiments are significant, therefore this part will also be represented in an easier to understand, graphical way.

All the mentioned benchmarks fulfill the most important requirement, that is, they utilize the targeted devices resources to the maximum or near-maximum. Therefore, the next factor should be the execution time. It should be noted, that this parameter is important, because of two reasons – first, too short benchmark kernel exposes the measurements to inaccuracy. To put this in a practical way, we should at first find the measurements tool with lowest sampling rate, that is possible due to their implementation. In our case it is the Yokogawa WT310E Power Meter – its software, the Yokotool, enables the measurements with the shortest interval of 0.1 second. Other methods, such as Linux Perf and NVML should use this sampling frequency as well, in order to maintain cohesion. In one of the research papers [0] that covered the topic of energy awarness in parallel applications, authors discussed the sampling period of *nvidia-smi*, which is 1 second, and the execution times of various benchmarks. Those values varied between 20 and 200 seconds, which gives the maximal error of GPU power draw reading less than 5% and the minimal error for the longest test runs is less than 0.5%.

This insight gives us a simple formula:

$$E = \frac{f}{T}[\%] \tag{4.1}$$

Where:

E – Error of reading $[\%]$
f – Lowest sampling rate of the measurements tool $[s]$
T – Execution time of benchmark $[s]$

Based on that information, it has been tentatively accepted, that the maximum error of power draw reading should be lower than 0.5%, to be considered satisfactory. With lowest possible sampling period of 0.1 second, the execution time of any benchmark should be at least 20 seconds. To make sure, that this constraint is satisfied at all times, the preliminary tests should be done utilizing all of the resources on gives device, i.e. CPU benchmarks should be run on all logical processors available in a parallel manner. At last, benchmark kernel should be also short enough in order to avoid excessively long execution time.

4.2. PRELIMINARY TESTS

The goal of the set of preliminary tests is to choose three, best suitable benchmarks from each implementation, based on the metrics stated earlier.

- Implementation: **OMP-CPP**
  - Benchmarks available: IS, FT, EP, CG, MG, LU, SP & BT
  - Class sizes to choose from: B, C & D
  - 24 different combinations of benchmarks and class sizes in total
  - All tests were run on 2 CPUs and 40 logical processors in parallel
- Implementation: **OMP-CUDA**
  - Benchmarks available: IS, FT, EP, CG, MG, LU, SP & BT
  - Class sizes to choose from: C, D & E
  - 24 different combinations of benchmarks and class sizes in total
  - All tests were run on a single GPU with the same grid sizes each time
- Implementation: **MPI-Fortran**
  - Benchmarks available: IS, FT, EP, CG, MG & LU
  - Class sizes to choose from: B, C & D
  - 18 different combinations of benchmarks and class sizes in total
  - All tests were run on 2 CPUs and 32 logical processors in parallel
- Implementation: **Horovod-Python**
  - Benchmark available: XCeption
  - Number of training epochs: 1, 3 & 5
  - 3 different combinations of model training parameters to choose from
  - All tests were run in configurations with 1, 2 & 4 GPUs, to check training behavior

Every configuration stated above has been conducted 3 times in order to obtain an average results, which are presented in the tables below.

Table 4.1. Execution times of OMP-CPP benchmarks

Benchmark	Class size	Execution time [s]	Benchmark	Class size	Execution time [s]
IS	B	0.35	FT	B	2.92
IS	C	1.25	FT	C	16.14
IS	D	37.53	FT	D	371
EP	B	3.06	CG	B	7.77
EP	C	11.96	CG	C	27.2
EP	D	177	CG	D	(freezed)
MG	B	2.32	LU	B	9.88
MG	C	17.459	LU	C	34.88
MG	D	185	LU	D	1111
SP	B	14.48	BT	B	15.82
SP	C	(freezed)	BT	C	55.182
SP	D	(core dumped)	BT	D	1185

According to tests, three CPUs benchmarks has been chosen as the most suitable for the main experiments on *sanna.kask* later. Those benchmarks are:

- **bt.C** – Block Tri-diagonal solver, class size 'C'
- **is.D** – Integer Sort, class size 'D'
- **lu.C** – Lower-Upper Gauss-Seidel solver, class size 'C'

Table 4.2. Execution times of OMP-CUDA benchmarks

Benchmark	Class size	Execution time [s]	Benchmark	Class size	Execution time [s]
IS	C	(too short)	FT	C	6.63
IS	D	15.33	FT	D	(unsuccesful)
IS	E	(not defined)	FT	E	(unsuccesful)
EP	C	(too short)	CG	C	(too short)
EP	D	27.16	CG	D	(freezed)
EP	E	442.79	CG	E	(freezed)
MG	C	(too short)	LU	C	16.89
MG	D	(unsuccesful)	LU	D	300.74
MG	E	(unsuccesful)	LU	E	(unsuccesful)
SP	C	10.39	BT	C	12.19
SP	D	220.86	BT	D	(unsuccesful)
SP	E	(unsuccesful)	BT	E	(unsuccesful)

According to tests, three GPUs benchmarks has been chosen as the most suitable for the main experiments on *sanna.kask* later. Those benchmarks are:

- **lu.D** – Lower-Upper Gauss-Seidel solver, class size 'D'
- **sp.D** – Scalar Penta-diagonal solver, class size 'D'
- **ep.D** – Embarassingly Parallel, class size 'D'

Table 4.3. Execution times of MPI-Fortran benchmarks

Benchmark	Class size	Execution time [s]	Benchmark	Class size	Execution time [s]
IS	B	0.18	FT	B	3.11
IS	C	0.8	FT	C	13.83
IS	D	17.02	FT	D	360.15
EP	B	1.11	CG	B	3.33
EP	C	4.39	CG	C	8.57
EP	D	71.16	CG	D	761.02
MG	B	0.48	LU	B	10.97
MG	C	4.5	LU	C	41.81
MG	D	88.34	LU	D	770.52

According to tests, three CPUs benchmarks has been chosen as the most suitable for the main experiments on *vinnana.kask* later. Those benchmarks are:

- **ep.D.x** – Embarassingly Parallel, class size 'D'
- **lu.C.x** – Lower-Upper Gauss-Seidel solver, class size 'C'
- **is.D.x** – Integer Sort, class size 'D'

Table 4.4. Execution times of Horovod-Python benchmarks

Number of GPUs used	Number of training epochs	Execution time [s]	Test accuracy [%]
1	1	124.2	88.99
1	3	326.7	91.30
1	5	528.6	91.59
2	1	72.2	87.9
2	3	176.2	90.72
2	5	280.9	92.46
4	1	48.5	86.81
4	3	101.1	91.59
4	5	155.3	93.91

According to tests, the best number of epochs for the training of 'XCeption' model is 1. Test accuracy of model is high and the execution time is satisfactory.

What is worth noting in terms of GPUs benchmarks is the fact, that while utilizing between 90∼100% fo the GPUs resources, various kernels results in different power draw values. The averages of those values are listed below and may provide an additional insight on the future experiments:

- **lu.D** – Power draw fluctuates between 200∼230 [W]
- **sp.D** – Average power draw of 200∼210 [W]
- **ep.D** – Average power draw of 155∼160 [W]
- **XCeption** – Average power draw of 180∼185 [W]

4.3. OVERVIEW ON ALL FINAL CONFIGURATIONS

The measurements will be conducted on many different setups, therefore it is important to explain throughly the entire hierarchy of the configurations.

4.3.1. SERVERS

The tests will be conducted on two computational nodes: *sanna.kask* and *vinnana.kask*. Both of those machines have two Intel Xeon CPUs and several NVIDIA Quadro RTX GPUs. Paired with huge amount of RAM, these servers are suitable choices in terms of choosing the testbed machines.

Fig. 4.1. sanna.kask

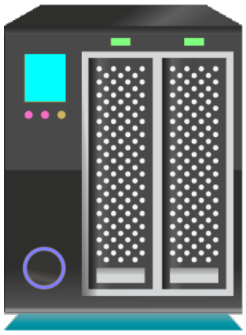

Fig. 4.2. vinnana.kask

Despite the similarity between those two servers, their components slightly differs from each other. A detailed comparison of their parts is listed in the table below:

Table 4.5. Overview on components of the servers used in experiments

	sanna.kask	**vinnana.kask**
OS	Linux Ubuntu 22.04 LTS	Linux Ubuntu 22.04 LTS
CPUs	2 × Intel(R) Xeon(R) Silver 4210 CPU 2.20GHz, TDP: 10/10 85 [W]	2 × Intel(R) Xeon(R) Silver 4210 CPU 2.20GHz, TDP: 10/10 85 [W]
GPUs	8 × NVIDIA Quadro RTX 6000, VRAM: 24 GB, power draw range: 100-260 [W]	4 × NVIDIA Quadro RTX 5000, VRAM: 16 GB, power draw range: 100-230 [W]
Motherboard	Inspur YZMB01130107	Inspur YZMB01130107
RAM	384 GB, DDR4, 2400 (MHz)	384 GB, DDR4, 2400 (MHz)
PSUs	4 × Delta Electronics DPS-2200AB-2	4 × Delta Electronics DPS-2200AB-2

4.3.2. DEVICES

Next, benchmarks will be organized based on devices, that will be utilized during tests. On this level, we distinguish three configurations: *CPUs*, *GPUs* and the *Hybrid*, when both are utilized during experiments.

Fig. 4.3. CPUs benchmarks, utilizing the Intel Xeon Silver CPUs

Fig. 4.4. GPUs benchmarks, utilizing the NVIDIA Quadro RTX GPUs

Fig. 4.5. Hybrid configuration, utilizing both CPUs and GPUs

4.3.3. IMPLEMENTATIONS

After receiving information about used server and devices, the correct implementations are used. For the first node, *sanna.kask* we distinguish *OMP-CPP* for CPUs, *OMP-CUDA* for GPUs and *OMP-CPP+OMP-CUDA* for Hybrid. For the second node, *vinnana.kask* we have *MPI-Fortran* for CPUs, *Horovod-Python* for GPUs and *MPI-Fortran+Horovod-Python* for Hybrid.

Fig. 4.6. CPU benchmarks implementation used on *sanna.kask*

Fig. 4.7. GPU benchmarks implementation used on *sanna.kask*

Fig. 4.8. CPU benchmarks implementation used on *vinnana.kask*

Fig. 4.9. GPU benchmarks implementation used on *vinnana.kask*

Table 4.6. Overview on implementations used in tests

	sanna.kask	vinnana.kask
CPUs	OMP-CPP	MPI-Fortran
GPUs	OMP-CUDA	Horovod-Python
Hybrid	OMP-CPP+OMP-CUDA	MPI-Fortran+Horovod-Python

4.3.4. BENCHMARKS

Based on the server and implementation currently used, benchmarks are chosen by a scheduler script to run. The main task the benchmarks are responsible is to utilize the devices' resources for the purpose of the tests.

Fig. 4.10. Benchmarks evaluation example

In the tables below are listed the benchmarks chosen for final tests:

Table 4.7. Overview on benchmarks used in tests on sanna.kask

	OMP-CPP	OMP-CUDA	OMP-CPP + OMP-CUDA
Benchmark #1	bt.C	lu.D	bt.C+lu.D
Benchmark #2	is.D	sp.D	is.D+sp.D
Benchmark #3	lu.D	ep.D	lu.D+ep.D

Table 4.8. Overview on benchmarks used in tests on vinnana.kask

	MPI-Fortran	Horovod-Python	MPI-Fortran + Horovod-Python
Benchmark #1	ep.D.x	XCeption	ep.D.x+XCeption
Benchmark #2	is.D.x	–	is.D.x+XCeption
Benchmark #3	lu.C.x	–	lu.C.x+XCeption

4.3.5. CONFIGURATIONS

Configurations are based on currently used implementation and defines, which devices' resources should be allocated for the purpose of the current benchmark kernel. Configuration vary heavily between each other and the comprehensive data sheet is listed in the tables below:

Table 4.9. Configurations used on node *sanna.kask*

sanna.kask				
Device	Implementation			
	OMP-CPP			
1 CPU	1 Thread	5 Threads	10 Threads	20 Threads
2 CPUs	2 Threads	10 Threads	20 Threads	40 Threads
	OMP-CUDA			
GPUs	1 GPU & 1 Thread	2 GPUs & 2 Threads	4 GPUs & 4 Threads	8 GPUs & 8 Threads
	OMP-CPP + OMP-CUDA			
Hybrid	1 CPU & 4 Threads 1 GPU & 1 Thread	1 CPU & 8 Threads 2 GPUs & 2 Threads	2 CPUs & 16 Threads 4 GPUs & 4 Threads	2 CPUs & 32 Threads 8 GPUs & 8 Threads

Table 4.10. Configurations used on node *vinnana.kask*

vinnana.kask				
Device	Implementation			
	MPI-Fortran			
CPUs	4 Processes	8 Processes	16 Processes	32 Processes
	Horovod-Python			
GPUs	1 GPU	2 GPUs	4 GPUs	–
	MPI-Fortran + Horovod-Python			
Hybrid	8 Processes & 1 GPU	16 Processes & 2 GPUs	32 Processes & 4 GPUs	–

It is worth noting, that the OMP-based implementations that has been prepared to run on *sanna.kask* node has been pinned carefully to respective CPUs logical processors and GPUs, based on input configuration values. Therefore, using a CPUs monitoring tools, such as *top* or *htop*, one can see the utilization of individual logical processors. Such graphical representation has been collected and prepared for inspection in the set of figures below. The MPI-based implementations were run without manual pinning of the processes to threads, leaving load balancing to the framework itself.

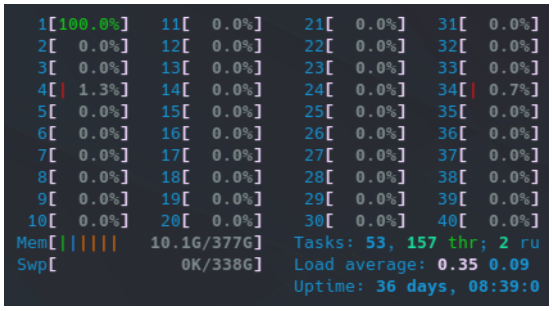

Fig. 4.11. 1 CPU, 1 Thread

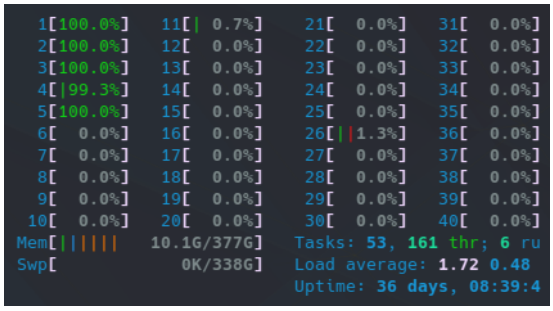

Fig. 4.12. 1 CPU, 5 Threads

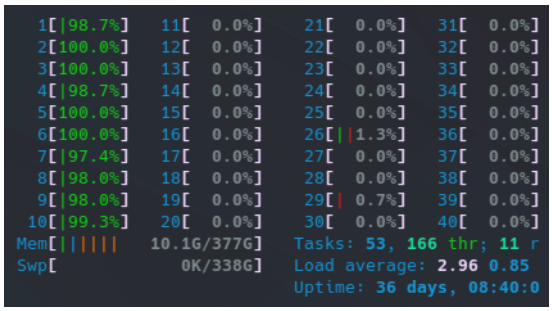

Fig. 4.13. 1 CPU, 10 Threads

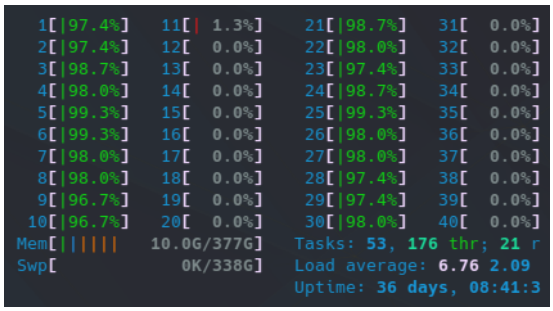

Fig. 4.14. 1 CPU, 20 Threads

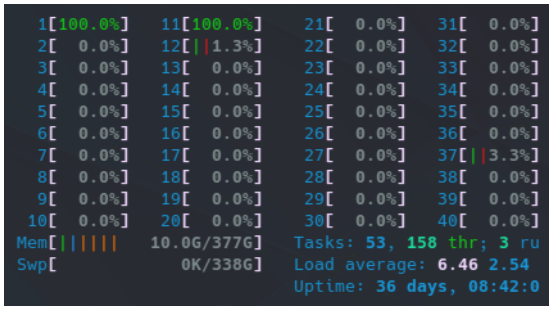

Fig. 4.15. 2 CPUs, 2 Threads

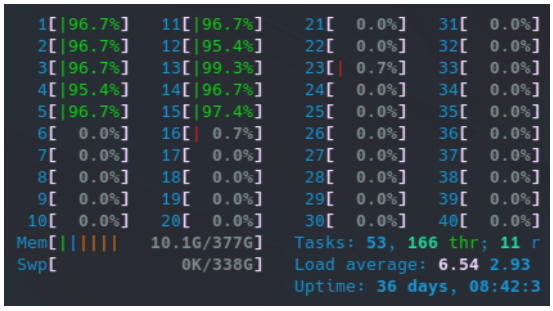

Fig. 4.16. 2 CPUs, 10 Threads

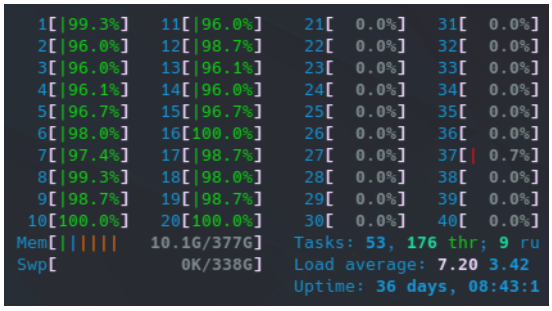

Fig. 4.17. 2 CPUs, 20 Threads

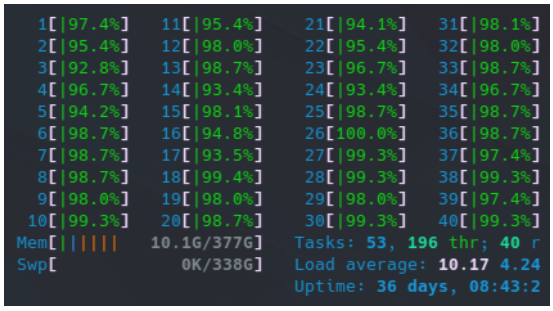

Fig. 4.18. 2 CPUs, 40 Threads

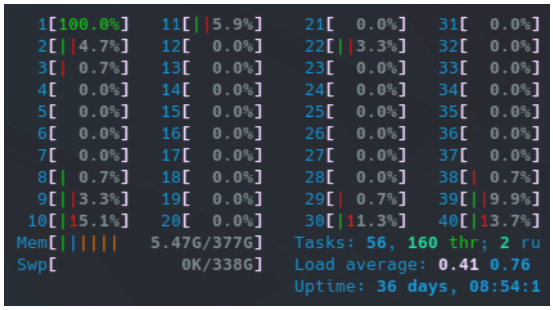

Fig. 4.19. 1 GPU, 1 Thread

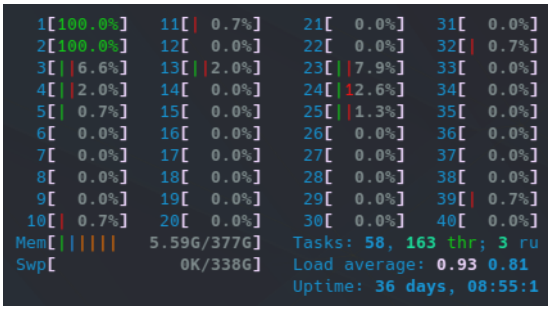

Fig. 4.20. 2 GPUs, 2 Threads

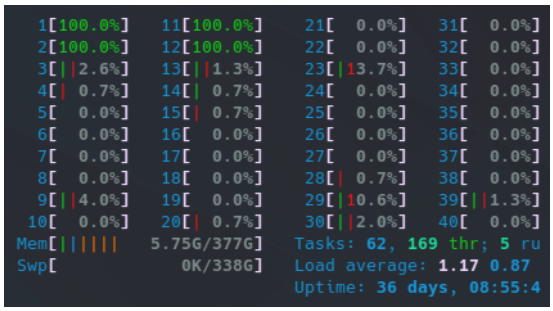

Fig. 4.21. 4 GPUs, 4 Threads

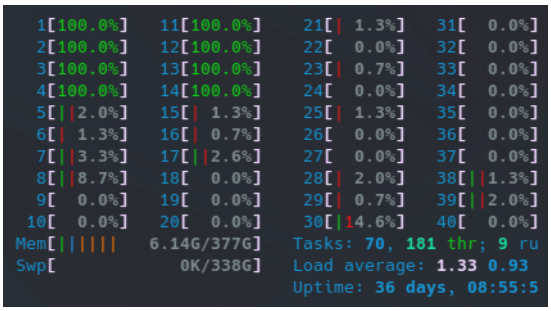

Fig. 4.22. 8 GPUs, 8 Threads

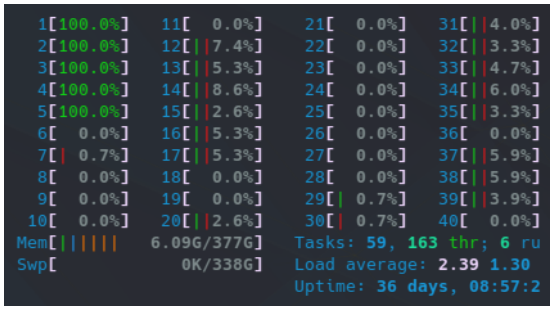

Fig. 4.23. 1 CPU, 4 Threads + 1 GPU, 1 Thread

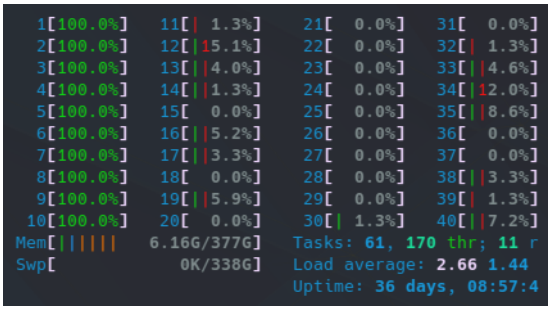

Fig. 4.24. 1 CPU, 8 Threads + 2 GPUs, 2 Threads

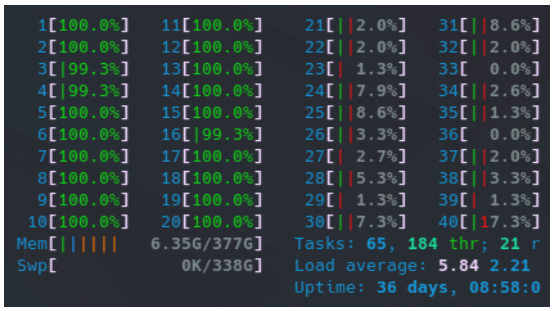

Fig. 4.25. 2 CPUs, 16 Threads + 4 GPU, 4 Threads

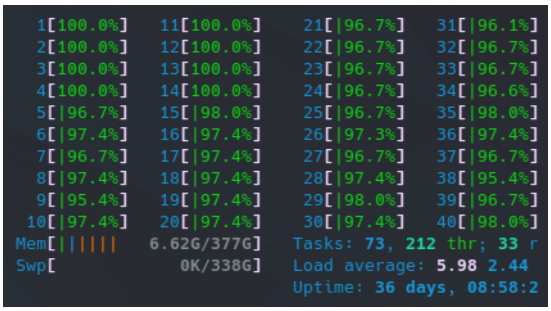

Fig. 4.26. 2 CPUs, 32 Threads + 8 GPUs, 8 Threads

5. EXPERIMENTS

5.1. PROPOSED WORKFLOW AND METHODOLOGY

In order to obtain the details of how system-level physical measurement estimates energy consumption by a component (such as a CPU or a GPU) during an application execution, several steps must be taken:

1. Exclusive reservation of the entire computational node.
2. Observation of the disk consumption and network usage before and during tests.
3. Monitoring of the CPUs and GPUs utilization before and during tests.
4. Running the benchmark kernels on an abstract processor only. Abstract processor comprises of the multicore CPU processor consisting of a certain number of physical cores and DRAM.
5. Gathering of the power measurements.
6. Verification of the accuracy and reliability of the software measurements tools, based on ground truth results.

One of the notable mentions that could be done in order to reduce the amount of uncertain power draw measurements done by the background components is setting the fans to full speed. This solution have a potential of reducing power draw fluctuations, especially during higher workloads, iėwhen running benchmarks kernels utilizing maximum amount of GPUs or running Hybrid configuration, where power draw of the entire nodes are very high. This could not be implemented, however, as the administrator of the department's servers stated, that interference in servers fans could be crucial for the nodes stability.

For Intel RAPL / NVIDIA Management Library:

1. Obtain base power of idle CPUs / GPUs.
2. Obtain execution time of benchmark application.
3. Obtain total energy consumption of the CPUs / GPUs during tests.
4. Calculate dynamic energy consumption by subtracting base energy from total energy used during run.

For Yokotool:

1. Obtain base power of idle CPUs.
2. Obtain execution time of benchmark application.
3. Obtain total energy consumption of the CPUs during tests.
4. Calculate dynamic energy consumption by subtracting base power from total energy used during run

In addition to the main experiments workflow, another methodology must be adapted – the data collection methodology. In order for the results to be properly comparable, several point have to be met:

1. Tests environment must be identical in every case, to eliminate discrepancy of the results.
2. The results of the power draw reading must be properly compared for the device only measurements (Intel RAPL / NVML) and the measurements of the entire node (Yokotool).
3. Experiments should be conducted on different nodes that utilizes different hardware, in order to state repeatability of tests. [TO BE REDACTED]
4. Experiments should be conducted, using different benchmark kernels or application, to remove the possibility of bias of the results, due to poor diversification of test cases. [TO BE REDACTED]
5. Test runs must be repeated many times.

5.2. WORKING ENVIRONMENT – SERVERS DETAILS

[PLACEHOLDER] NOTE: Probably this section will be cut, because it has been covered in previous chapter

5.3. MAIN TESTS

5.3.1. OVERVIEW ON THE SCHEDULER SCRIPT

In order to automate the experiments, an entire scheduler script had to be created. Its has two main tasks: first is to store the information about the configurations and run the benchmarks according to the presets. The second is to run the measurements softwares and save all the results in the ordered manner.

Both of these goals are reached by using dictionaries with key-value pairs. That created a tree-like dependencies between the corresponding layers of configurations. Finally, that solution works for both choosing the right config and providing the path to save measurements.

2. Section that run CPUs benchmarks, GPUs benchmarks, perf, nvml, yoko 3. Functions that check if the benchmarks are still running 4. Function that cleans-up every process after tests

5.3.2. MAIN AUTOMATION FUNCTION – SCHEDULER()

The entire scheduler script consists of classes and functions that are explicitly designated for their purposes in the code. Overview of them is as follows:

- **class 'Config'** – Contains all the informations about the servers, devices, implementations, benchmarks and configurations, handles the relations between them and provides correct pathes to the corresponding measurements directories.
- **class 'Benchmark'** – Defines functions responsible for executing CPUs and GPUs benchmarks, as well as the measurements softwares: Linux Perf, NVIDIA Management Library and Yokotool
- **class 'Execution'** – This class contains functions tasked with calling the benchmark kernels and checking their status, if they are still running, for the purpose of ending the measurements. Since the measurements softwares are highly dependent on benchmarks being run, they are executed directly from the *main()* function and the *Execution* class only has the functions tasked with the proper termination of them.
- **function 'scheduler()'** – This major function triggers secondary functions from *Execution* class and watched their status.
- **function 'main()'** – Runs every configuration sequentially, based on the lists of presets. Additionally, repeats every ten times in order to achieve repeatability of the experiments.

In order to visualize the entire workflow of the scheduler, as well as the workings of the individual processes, two charts has been created:

- **General Flowchart** – This chart (**Fig. 5.1**) describes the relations between the currently used configurations and the instructions executed based on those conditions.
- **Processes Flowchart** – This chart (**Fig. 5.2**) shows the working priciples of the *runner()* subfunction, which shows the benchmarks and measurements softwares are started on a high level of abstraction.

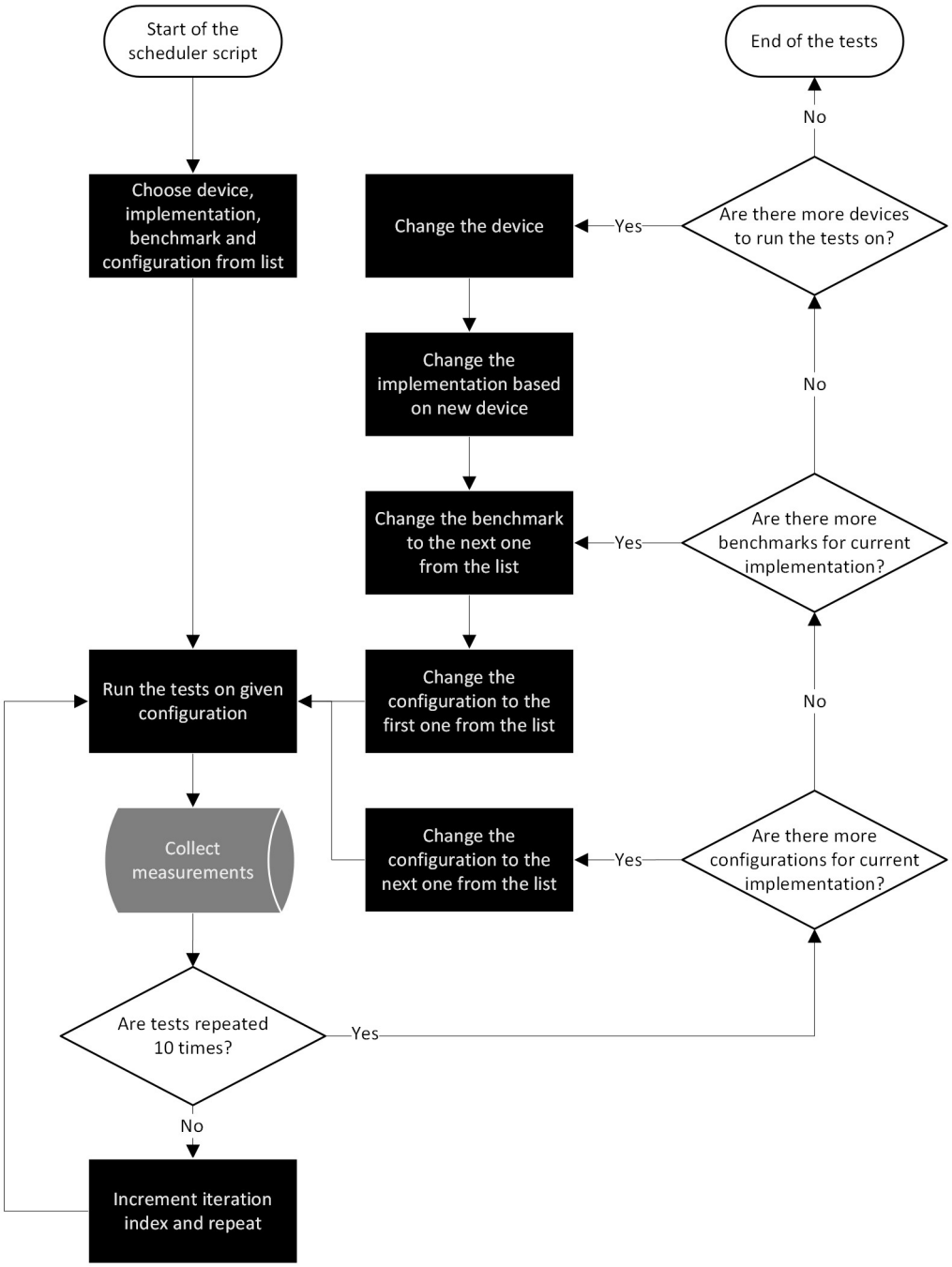

Fig. 5.1. General Flowchart

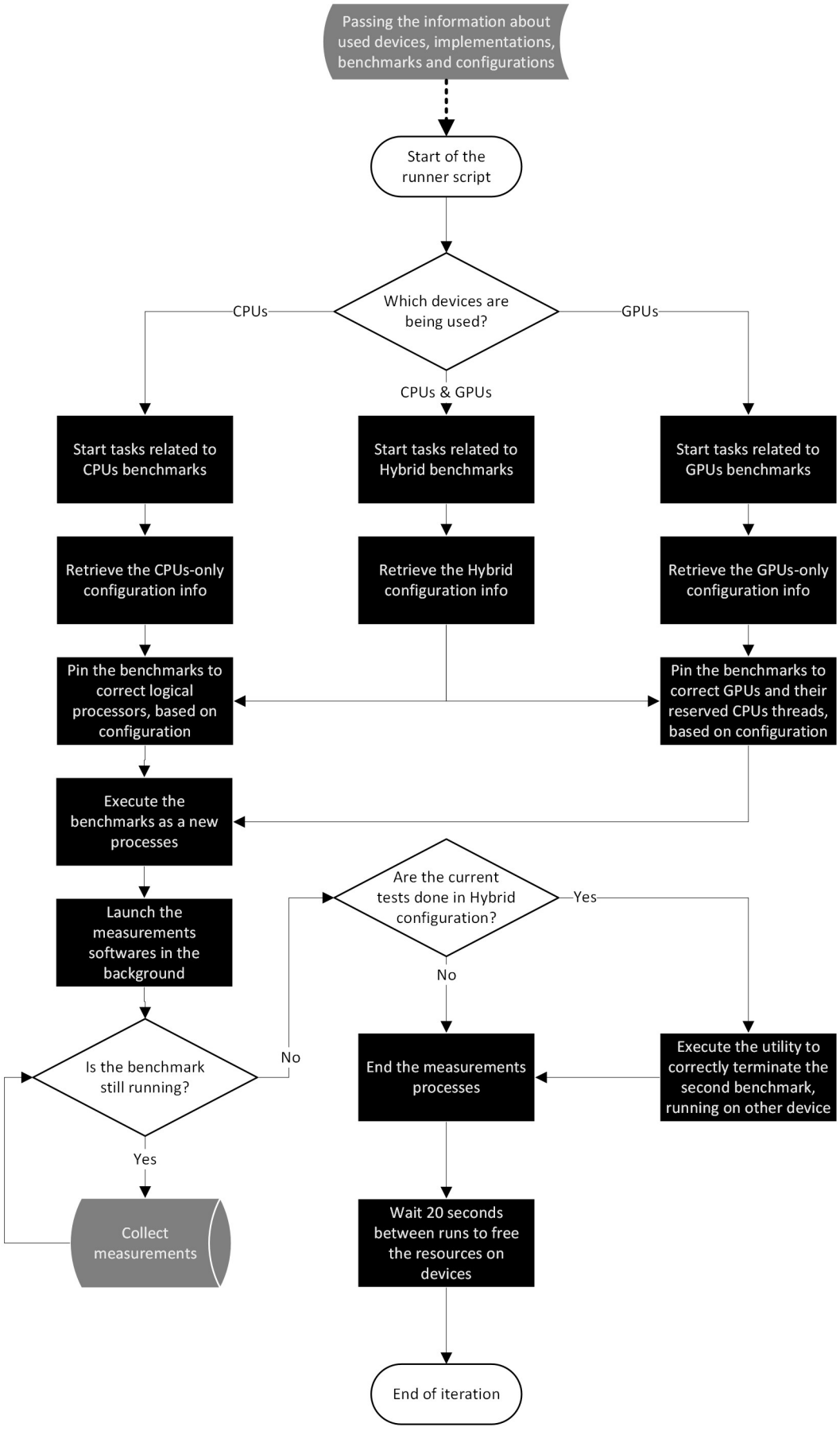

Fig. 5.2. Processes Flowchart

5.3.3. THREADS PINNING AND KERNELS EXECUTION – CPU_BENCHMARK()

This function is responsible of executing CPUs benchmarks, based on the given configuration. It creates a separate processes by utilizing the Python *subprocess* module.

```
cpu_benchmark = subprocess.Popen(
    [
        "taskset --cpu-list <T> <P> <B> > /dev/null 2>&1"
    ],
    shell=True
)
return cpu_benchmark.pid
```

Here is an explanation of every part of the command:

- **cpu_benchmark** – A variable of type *<class 'subprocess.Popen'>* is created mainly in order to retrieve PID later on.
- **subprocess.Popen** – The underlying process creation and management is handled by the Popen class. Its function is to execute a child program in a new process.
- **taskset** – This command is used to set or retrieve the CPU affinity of a running process given its pid, or to launch a new command with a given CPU affinity.
- **–cpu-list** – This option interprets mask as numerical list of processors instead of a bitmask. Numbers are separated by commas and may include ranges. For example: 0,5,8-11.
- Variables that dynamically changes based on the configurations:

  T – Logical processors indexes
  P – Specified absolute path to the correct measurements folder
  B – Currently used benchmark kernel

- **> /dev/null 2>&1** – Redirecting *stderr* containing error messages from the executed command or script to *stdout* to the output of the command. Both are, in fact, redirected then to the so-called *null device*. The result of that action is suppression of all messages printed by the benchmark kernels. It is useful when collecting logs from the terminal, that is running the entire scheduler script, without unnecessary messages.
- **shell=True** – Invokes the program as 'shell'

Finally, the function returns the PID of newly created process as an integer value. It is done for the purpose of terminating the benchmark in a Hybrid configuration.

5.3.4. GPUS AND THREADS MANAGEMENT – GPU_BENCHMARK()

This function consists of two parts: first part is responsible for executing the Horovod-Python benchmark and the second part is responsible for running the OMP-CUDA benchmark.

```
gpu_benchmark = subprocess.Popen(
    [
        "mpirun -np <N> --map-by socket -x NCCL_DEBUG=INFO \
        python3 <P>+"Xception.py > /dev/null 2>&1"
    ],
    shell=True
)
return gpu_benchmark.pid
```

Here is an explanation of every part of the command:

- **gpu_benchmark** – A variable of type *<class 'subprocess.Popen'>* is created mainly in order to retrieve PID later on.
- **subprocess.Popen** – The underlying process creation and management is handled by the Popen class. Its function is to execute a child program in a new process.
- **mpirun** – This command is used to execute serial and parallel jobs. It will run X copies of specified program in the current run-time environment and scheduling (by default) in a round-robin fashion by CPU slot.
- **-np** – This option specifies, how many processes will be started.
- Variables that dynamically changes based on the configurations:

  N – Number of GPUs used in training
  P – Specified absolute path to the correct measurements folder

- **–map-by socket** – Map to the specified object, such as slot, hwthread, core, socket, numa, board, node and more. In this particular case, benchmark application is mapped by 'socket', allowing to utilize multiple GPUs for training in a ditributed manner.
- **-x** – Export the specified environment variables to the remote nodes before executing the program.
- **NCCL_DEBUG=INFO** – This flag is used for debugging. In case of NCCL failure, you can set NCCL_DEBUG=INFO to print an explicit warning message as well as basic NCCL initialization information.
- **python3** – Specify the use of Python interpreter when executing the script of deep neural networks model training.
- **Xception.py** – Name of the script file.
- **> /dev/null 2>&1** – As mentioned in the previous subsection, this command suppresses the output from the terminal, in order to avoid the unnecessary messages.
- **shell=True** – Invokes the program as 'shell'

Finally, the function returns the PID of newly created process as an integer value. It is done for the purpose of terminating the benchmark in a Hybrid configuration.

The second part of the function that runs the OMP-CUDA benchmarks is as follows:

```
list_of_gpu_benchmarks = []
value = Config.taskset_gpu[configuration_gpu]
for i in range(0, len(value), 1):
    gpu_benchmark = subprocess.Popen(
        [
            "taskset --cpu-list <I> <P+I> <B> > /dev/null 2>&1"
        ],
        shell=True,
    )
    list_of_gpu_benchmarks.append(gpu_benchmark.pid)
return list_of_gpu_benchmarks
```

Here is an explanation of every part of the command:

- **gpu_benchmark** – A variable of type *<class 'subprocess.Popen'>* is created mainly in order to retrieve PID later on.
- **subprocess.Popen** – The underlying process creation and management is handled by the Popen class. Its function is to execute a child program in a new process.
- **taskset** – This command is used to set or retrieve the CPU affinity of a running process given its pid, or to launch a new command with a given CPU affinity.
- **–cpu-list** – This option interprets mask as numerical list of processors instead of a bitmask. Numbers are separated by commas and may include ranges.
- Variables that dynamically changes based on the configurations:

  I – Allocation of individual logical processors to the GPUs, based on the index number from special dictionary
  P+I – Specified absolute path to the correct measurements folder, modified by the number of total GPUs used
  B – Currently used benchmark kernel

- **> /dev/null 2>&1** – As mentioned in the previous subsection, this command suppresses the output from the terminal, in order to avoid the unnecessary messages.
- **shell=True** – Invokes the program as 'shell'
- **list_of_gpu_benchmarks.append(gpu_benchmark.pid)** – Filling the list with PIDs of benchmarks for later termination.

Finally, the function returns the PID of newly created process as an integer value. In this particular case, if there are more than one GPUs used in tests, the PIDs of all spawned processes are parsed as a list to the function that is responsible in orderly terminating all the kernels.

5.3.5. MEASUREMENTS WITH YOKOTOOL SOFTWARE – YOKO()

This function utilizes the high-level Python wrapper for Yokogawa WT310E Power Meter, the *Yokotool*.

Syntax is similar to the previous examples:

```
yokotool = subprocess.Popen(
    [
        "yokotool read T,P -o <P+N> > /dev/null 2>&1 &"
    ],
    shell=True
)
return yokotool.pid
```

Here is an explanation of every part of the command:

- **yokotool** – Yokotool's command line interface is based on commands and sub-commands, similar to git and many other tools. This invokes the Yokotool wrapper for use.
- **read** – Read measurements data.
- **T,P** – Specifies, what data we want to read. In this particular case it is time **T** measured from the start of the epoch (on 'UNIX time' is starts at 00:00:00 UTC on 1 January 1970) and power **P**, measured in Watts [W]. Output is separated with comma for easy manipulation of data after tests.
- **-o** – This flags redirects the output from the measurements to a file for later analysis.
- Variables that dynamically changes based on the configurations:

  P+N – Specified absolute path to the correct measurements folder, modified by the current number of iterations

- **> /dev/null 2>&1** – As mentioned in the previous subsection, this command suppresses the output from the terminal, in order to avoid the unnecessary messages. In this case, and additional & is placed at the end, which means that the entire command is put as a background process.
- **shell=True** – Invokes the program as 'shell'

Finally, the function returns the PID of newly created process as an integer value. It is done for the purpose of terminating the measurements by the designated function.

5.3.6. MEASUREMENTS WITH LINUX PERF SOFTWARE – PERF()

Linux Perf is a lightweight profiling tool with performance counters. It utilizes Intel RAPL for measurements of pre-defined events on CPUs. To obtain more informations about what can be measured by Linux Perf, one can use *perf list* command in the terminal.

```
list_of_perf_pids = []
pin_to_cpus = ("0", "10")
idx_names = {"0": "0", "10": "1"}
for i in pin_to_cpus:
    perf = subprocess.Popen(
        [
            "perf stat --event=power/energy-pkg/ \
            --cpu=<C> --delay 100 --interval-print 100 \
            --summary --field-separator , \
            --output <P+N> > /dev/null 2>&1 &"
        ],
        shell = True,
    )
    list_of_perf_pids.append(perf.pid)
return list_of_perf_pids
```

Here is an explanation of every part of the command:

- **perf** – Invocation of measurements tool.
- **stat** – Run a command and gather performance counter statistics.
- **–event=power/energy-pkg/** – Event selector, in this case, the measured physical quantity is the energy usage of selected CPUs, during the benchmark kernel execution.
- **–cpu=<C>** – Targeting of a specific CPUCorrect integer value is based on the output of *lscpu* command and the information about *NUMA node(s)*.
- **–delay 100** – A small delay of 100 [ms] is introduced in order to offset the slight delay of measurements of Yokogawa power meter. This solution has been introduced as a result of an observation during the preliminary tests.
- **–interval-print 100** – Measurements are performed with the same time interval of 100 [ms] set as two others measurements methods.
- **–summary** – At the end of the measurements, an additional information about total energy used during tests, as well as the total measurements time is given. Mainly used during preliminary tests as an additional insight on gathered results.
- **–field-separator** – Sets the output delimiter for easier access in softwares like LibreOffice – every printed value is separated by commas ‘,’
- **–output** – Saves measurements to file. In this case, file names matches the consecutive runs.
- Variables that dynamically changes based on the configurations:

  P+N – Specified absolute path to the correct measurements folder, modified by the current number of iterations

- **> /dev/null 2>&1** – Silences the output, redirecting it to null device and puts the process in the background.
- **shell=True** – Invokes the program as 'shell'

Finally, the function returns the PID of newly created process as an integer value. It is done for the purpose of terminating the measurements by the designated function.

5.3.7. MEASUREMENTS WITH NVML HANDLING FUNCTION – NVML()

The method of gathering the measurements of power draw of GPUs, using NVIDIA Management Library is a little different than in previously shown implementations. No new processes are spawned, due to the fact, that the measurements are handled by a special function already implemented in the scheduler script. This function has two core parts: the first one is responsible of precise executing the measurements every 100 [ms] – it works as an built-in scheduler, and the second part handled the invocation of NVML-specific functions and saving the results to the file.

A short code-snippet below illustrates the practical usage of NVML related function

```
import py3nvml
nvmlInit()
handle_Idx0 = nvmlDeviceGetHandleByIndex(0)
# based on number of GPUs used, there are more variables of that kind
measure_Idx0 = nvmlDeviceGetPowerUsage(handle_Idx0) / 1000.0
# rest of code contains the sub-scheduler routine and saving the output
to file.
```

- **import py3nvml** – Import of a module, that function as a high level Python wrapper for NVML.
- **nvmlInit()** – NVIDIA Management Library initialization. It is mandatory to run before calling any other methods.
- **handle_Idx0 = nvmlDeviceGetHandleByIndex(0)** – create variable *handle_Idx0* and assign the return value of data type *<class 'py3nvml.py3nvml.LP_struct_c_nvmlDevice_t'>*, which is, in fact, a pointer to a memory register containing data about specific GPU.
- **measure_Idx0 = nvmlDeviceGetPowerUsage(handle_Idx0) / 1000.0** – This function return a value of power draw of a specified GPU, in Watts [W].

In terms of proper ending the measurements process, this task is handled quite differently than in previous implementations. The *nvml()* function is started as a parallel thread, using *multiprocessing.Process()* module. The thread is then started as a daemon, collecting measurements and saving them in the background. After the end of benchmark kernel, the function is ended by using innate multiprocessing function – *.terminate()*.

5.3.8. TERMINATION OF BENCHMARKS IN HYBRID CONFIGURATION

[PLACEHOLDER]

5.3.9. CLEANUP OF MEASUREMENTS DAEMONS

[PLACEHOLDER]

5.4. ANALYSIS OF THE RESULTS AND DISCUSSION

[PLACEHOLDER]

Table 5.1. sanna.kask, CPUs, OMP-CPP, bt.C, 1 CPU [POWER DRAW ONLY!!!]

	1 CPU			
Results from 10 runs	1 Thread	5 Threads	10 Threads	20 Threads
Avg. Exec. time [s]	1054.795	216.315	114.315	101.372
Std. dev. of time [-]	0.966	0.243	0.121	0.158
(Yokogawa) Avg. power draw [W]	379.962	402.881	432.171	445.752
(Yokogawa) Std. dev. of avg. power draw [-]	0.605	0.225	0.382	1.007
(CPU: 0) Avg. power draw [W]	33.717	51.274	70.433	77.958
(CPU: 0) Std. dev. of avg. power draw [-]	0.102	0.13	0.085	0.089
(CPU: 1) Avg. power draw [W]	28.98	28.887	28.871	28.874
(CPU: 1) Std. dev. of avg. power draw [-]	0.125	0.062	0.067	0.055
(GPU: 0) Avg. power draw [W]	21.755	21.673	21.604	21.634
(GPU: 0) Std. dev. of avg. power draw [-]	0.343	0.061	0.2	0.05
(GPU: 1) Avg. power draw [W]	25.522	25.534	25.516	25.524
(GPU: 1) Std. dev. of avg. power draw [-]	0.023	0.032	0.052	0.029
(GPU: 2) Avg. power draw [W]				
(GPU: 2) Std. dev. of avg. power draw [-]				
(GPU: 3) Avg. power draw [W]				
(GPU: 3) Std. dev. of avg. power draw [-]				
(GPU: 4) Avg. power draw [W]				
(GPU: 4) Std. dev. of avg. power draw [-]				
(GPU: 5) Avg. power draw [W]				
(GPU: 5) Std. dev. of avg. power draw [-]				
(GPU: 6) Avg. power draw [W]				
(GPU: 6) Std. dev. of avg. power draw [-]				
(GPU: 7) Avg. power draw [W]				
(GPU: 7) Std. dev. of avg. power draw [-]				

6. SUMMARY AND FUTURE WORK

6.1. SUMMARY

[PLACEHOLDER]

6.2. FUTURE WORK

[PLACEHOLDER]

RESEARCH PAPERS

[1] Yehia Arafa et al. “Verified Instruction-Level Energy Consumption Measurement for NVIDIA GPUs”. In: *Proceedings of the 17th ACM International Conference on Computing Frontiers*. CF ’20. Catania, Sicily, Italy: Association for Computing Machinery, 2020, pp. 60–70. ISBN: 9781450379564. DOI: 10.1145/3387902.3392613. URL: https://doi.org/10.1145/3387902.3392613.

[2] Gabriell Araujo et al. “NAS Parallel Benchmarks with CUDA and beyond”. In: *Software: Practice and Experience* 53.1 (2023), pp. 53–80. DOI: https://doi.org/10.1002/spe.3056. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3056.

[3] Gabriell Alves de Araujo et al. “Efficient NAS Parallel Benchmark Kernels with CUDA”. In: *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2020, pp. 9–16. DOI: 10.1109/PDP50117.2020.00009.

[4] Maria Avgerinou, Paolo Bertoldi, and Luca Castellazzi. “Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency”. In: *Energies* 10.10 (2017). ISSN: 1996-1073. DOI: 10.3390/en10101470. URL: https://www.mdpi.com/1996-1073/10/10/1470.

[5] David H. Bailey. “NAS Parallel Benchmarks”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1254–1259. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_133. URL: https://doi.org/10.1007/978-0-387-09766-4_133.

[6] Martin Burtscher, Rupesh Nasre, and Keshav Pingali. “A quantitative study of irregular programs on GPUs”. In: *2012 IEEE International Symposium on Workload Characterization (IISWC)*. Nov. 2012, pp. 141–151. DOI: 10.1109/IISWC.2012.6402918.

[7] Martin Burtscher, Ivan Zecena, and Ziliang Zong. “Measuring GPU Power with the K20 Built-in Sensor”. In: *Proceedings of Workshop on General Purpose Processing Using GPUs*. GPGPU-7. Salt Lake City, UT, USA: Association for Computing Machinery, 2014, pp. 28–36. ISBN: 9781450327664. DOI: 10.1145/2588768.2576783. URL: https://doi.org/10.1145/2588768.2576783.

[8] Howard David et al. “RAPL: Memory power estimation and capping”. In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[9] Walter Dehnen. “A Hierarchical (N) Force Calculation Algorithm”. In: *Journal of Computational Physics* 179.1 (June 2002), pp. 27–42. DOI: 10.1006/jcph.2002.7026. URL: https://doi.org/10.1006%2Fjcph.2002.7026.

[10] Muhammad Fahad et al. “A Comparative Study of Methods for Measurement of Energy of Computing”. In: *Energies* 12.11 (2019). ISSN: 1996-1073. DOI: 10.3390/en12112204. URL: https://www.mdpi.com/1996-1073/12/11/2204.

[11] Chao Jin et al. “A survey on software methods to improve the energy efficiency of parallel computing”. In: *The International Journal of High Performance Computing Applications* 31.6 (2017), pp. 517–549. DOI: 10.1177/1094342016665471.

[12] Kiran Kasichayanula et al. “Power Aware Computing on GPUs”. In: *2012 Symposium on Application Accelerators in High Performance Computing*. July 2012, pp. 64–73. DOI: 10.1109/SAAHPC.2012.26.

[13] Hamidreza Khaleghzadeh et al. “Out-of-Core Implementation for Accelerator Kernels on Heterogeneous Clouds”. In: *J. Supercomput.* 74.2 (Feb. 2018), pp. 551–568. ISSN: 0920-8542. DOI: 10.1007/s11227-017-2141-4. URL: https://doi.org/10.1007/s11227-017-2141-4.

[14] Adam Krzywaniak, Jerzy Proficz, and Pawel Czarnul. “Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors”. In: Sept. 2018, pp. 339–346. DOI: 10.15439/2018F177.

[15] Júnior Löff et al. “The NAS Parallel Benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures”. In: *Future Generation Computer Systems* 125 (2021), pp. 743–757. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2021.07.021. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21002831.

[16] Rezaur. Rahman and SpringerLink (Online service). *Intel® Xeon Phi™ Coprocessor Architecture and Tools*. Berkeley, CA : Apress : 2013. URL: https://doi.org/10.1007/978-1-4302-5927-5.

[17] Efraim Rotem et al. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge”. In: *IEEE Micro* 32.2 (Mar. 2012), pp. 20–27. ISSN: 1937-4143. DOI: 10.1109/MM.2012.12.

ADDITIONAL RESOURCES

[18] Artem Bityutskiy, Antti Laakso, and Helia Correia. *Yokotool*. URL: https://github.com/intel/yoko-tool.

[19] Martin Burtscher. *K20Power v1.1*. URL: https://userweb.cs.txstate.edu/~burtscher/research/K20power/.

[20] International Electrotechnical Commission. *IEC 62301:2011 | IEC Webstore | energy efficiency, smart city, standby power*. URL: https://webstore.iec.ch/publication/6789.

[21] The Institute for Development and Resources in Intensive Scientific Computing. *Horovod: Multi-GPU and multi-node data parallelism*. URL: http://www.idris.fr/eng/jean-zay/gpu/jean-zay-gpu-hvd-tf-multi-eng.html.

[22] Intel. *Intel® Xeon Phi™ Coprocessor DEVELOPER'S QUICK START GUIDE*. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf.

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

[24] Michael Kerrisk. *Pthreads - Linux manual page*. URL: https://man7.org/linux/man-pages/man7/pthreads.7.html.

[25] Linux. *"taskset" - Linux manual page*. URL: https://man7.org/linux/man-pages/man1/taskset.1.html.

[26] NASA. *NAS Parallel Benchmarks*. URL: https://www.nas.nasa.gov/software/npb.html.

[27] NVIDIA. *NVIDIA CUDA Compiler Driver NVCC*. URL: https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/.

[28] NVIDIA. *NVIDIA Management Library (NVML)*. URL: https://developer.nvidia.com/nvidia-management-library-nvml.

[29] NVIDIA. *NVML API Reference Guide*. URL: https://docs.nvidia.com/deploy/nvml-api/index.html.

[30] NVIDIA. *PAPI CUDA Component*. URL: https://developer.nvidia.com/papi-cuda-component.

[31] NVIDIA. *Parallel Thread Execution ISA Version 8.2*. URL: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[32] NVIDIA. *System Management Interface SMI*. URL: https://developer.nvidia.com/nvidia-system-management-interface.

[33] OpenMP. *OpenMP Reference Guides*. URL: https://www.openmp.org/resources/refguides/.

[34] ICR Polska. *EnergyStar - ICR POLAND - testing and certification*. URL: https://icrpolska.com/en/energystar/.

[35] The Parliamentary Office of Science and Technology. *Energy Consumption of ICT*. URL: https://researchbriefings.files.parliament.uk/documents/POST-PN-0677/POST-PN-0677.pdf.

[36] SPEC. *SPECpower and Performance Committee*. URL: https://www.spec.org/power/.

[37] iTeh Standards. *Electrical and electronic household and office equipment - Measurement of low power consumption*. URL: https://standards.iteh.ai/catalog/standards/clc/371d2d67-a439-4f20-96f0-02675496fd03/en-50564-2011.

[38] ISS Group at the University of Texas. *LonestarGPU*. URL: https://iss.oden.utexas.edu/?p=projects/galois/lonestargpu.

[39] Yokogawa. *WT300E Digital Power Analyzer*. URL: https://tmi.yokogawa.com/solutions/products/power-analyzers/digital-power-meter-wt300e/#Documents-Downloads.

[40] Yokogawa. *WT300E Series Digital Power Meter*. URL: https://cdn.tmi.yokogawa.com/1/2562/files/BUWT300E-01EN.pdf.

LIST OF FIGURES

LIST OF TABLES