








perun: Benchmarking Energy Consumption of High-Performance Computing Applications

Juan Pedro Gutiérrez Herмосillo Muriedas¹(✉) , Katharina Flügel^{1,2},
Charlotte Debus^{1,2} , Holger Obermaier¹ , Achim Streit¹ ,
and Markus Götz^{1,2} 

¹ Steinbuch Centre for Computing (SCC), Karlsruhe Institute for Technology (KIT),
Hermann-von-Helmholtz Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{juan.muriedas,katharina.fluegel,charlotte.debus,holger.obermaier,
achim.streit,markus.goetz}@kit.edu
² Helmholtz AI, Karlsruhe, Germany

Abstract. Looking closely at the Top500 list of high-performance computers (HPC) in the world, it becomes clear that computing power is not the only number that has been growing in the last three decades. The amount of power required to operate such massive computing machines has been steadily increasing, earning HPC users a higher than usual carbon footprint. While the problem is well known in academia, the exact energy requirements of hardware, software and how to optimize it are hard to quantify. To tackle this issue, we need tools to understand the software and its relationship with power consumption in today's high performance computers. With that in mind, we present *perun*, a Python package and command line interface to measure energy consumption based on hardware performance counters and selected physical measurement sensors. This enables accurate energy measurements on various scales of computing, from a single laptop to an MPI-distributed HPC application. We include an analysis of the discrepancies between these sensor readings and hardware performance counters, with particular focus on the power draw of the usually overlooked non-compute components such as memory. One of our major insights is their significant share of the total energy consumption. We have equally analyzed the runtime and energy overhead *perun* generates when monitoring common HPC applications, and found it to be minimal. Finally, an analysis on the accuracy of different measuring methodologies when applied at large scales is presented.

Keywords: Energy Benchmarking · High-performance Computing · Artificial Intelligence · Distributed Memory System

1 Introduction

High-performance computing (HPC) is a key technology to tackle an increasing amount of complex computational problems in science and industry. Example

applications include fluid dynamics [1], molecular biology [2], and quantum chromodynamics [3]. Recently, HPC has accrued particular interest due to the large computational demands and data quantities of artificial intelligence workloads. With this paradigm shift, the utilized hardware has simultaneously changed towards heterogeneous architectures with local storage, significant main memory, and accelerators like GPUs.

A commonly neglected conundrum of using such heterogeneous HPC systems is their massive energy consumption. Modern supercomputers have a power draw of up to 30 MW [4]. While the efficiency of individual hardware components has improved over time, it has enabled manufacturers to pack transistors and components more densely, to increase the number of computational processing units and nodes as well as to expand auxiliary infrastructure. In turn, the increased power consumption for large-scale computational tasks on HPC systems are outpacing individual hardware efficiency gains [5].

Due to the environmental impact of the corresponding energy generation technologies, recent research has focused on estimating the carbon footprint of compute-intensive workloads. A strong emphasis has been put on training and inference with deep learning models on single nodes with multiple accelerators. The overall conclusion: the utilized hardware, training time, and location are the main factors contributing to carbon dioxide and equivalent gas emission (CO₂e).

Yet, several unexplored research questions remain. How reliable are hardware performance counters when estimating application power draw compared to the actual consumption? Are non-compute components like memory, storage, and network sufficiently taken into account? How reliable are current estimation techniques when applied to distributed applications?

In an attempt to provide answers to the above questions, our contributions are as follows:

- A novel MPI-parallelized Python package called *perun*¹ facilitating energy benchmarking on HPC systems. *perun* can utilize both estimates based on sampling hardware performance counters and precise read-outs from energy sensors.
- The assessment of the power estimation and measurement gap.
- An analysis of the power consumption of multi-node applications based on different estimation methodologies, including scaling artifacts for selected benchmark programs with an emphasis on data-intensive deep learning workflows.
- A quantification of the measuring overhead created by *perun*.

2 Related Work

Interest in energy-efficient computing is not novel. For example, the Green500 list [6] ranks the most energy-efficient supercomputers and HPC systems. Its goal was to discourage the performance-at-any-cost design of supercomputing systems by introducing the FLOPs-per-Watt (FLOPs W⁻¹) metric. Yet, the

¹ <https://github.com/Helmholtz-AI-Energy/perun>.

determination of the energy consumption is non-standardized and may vary significantly based on the tooling used.

In recent years, several tools have appeared to aid researchers in compiling carbon footprint statements. Carbontracker [7], one of the most widely used, is a Python package primarily aimed at machine learning. It samples energy data from hardware libraries for a single training epoch to extrapolate the energy consumption during actual training, but is limited to a single device. Similar tools include *experiment-impact-tracker* [8] and *CodeCarbon* [9], collecting information using the same API endpoints. However, they do not predict the expected total consumption but are meant to monitor the application throughout its execution.

Outside the Python ecosystem is the *Machine Learning Emissions Calculator* [10], which targets users of Cloud Service Providers (CSP) such as AWS or Google Cloud. Users may input the training wallclock time, the used CSP, hardware, and geolocation. In turn, this data is used to gather information from public APIs to provide the estimated CO₂e emissions. *Green algorithms* [11] is a similar website targeted to both CSP and personal computer users, with a more extensive set of configuration options.

Caspart *et al.* [12] collected energy measurements with physical sensors. The data was used to compare the efficiency of CPUs and GPUs for single-node, multi-accelerator machine learning applications. Hodak *et al.* [13] used a similar setup, but instead focused on which hardware settings significantly reduce the power draw without meaningfully increasing training time.

In the context of machine learning (ML), Strubell *et al.* [14] are among the first to look at the environmental impact of natural language processing (NLP) models. CO₂e emissions are calculated as the sum of the energy consumption of all CPUs and GPUs throughout training, multiplied by the data center’s power usage effectiveness (PUE) and carbon efficiency in the data center location. In that, the PUE is the ratio between the energy used by compute components and the energy used for the entire data center infrastructure [15], and carbon efficiency is the ratio of carbon and equivalent gas emissions in tonnes per kilo Watt hour (t CO₂e/(kW h)). While PUE has widespread use in the industry, it has been critiqued because of the lack of data supporting published numbers, working more as a publicity stunt than a relevant metric [16]. Patterson *et al.* [17] analyzed modern NLP models based on the reported training time, hardware, data center power efficiency, and energy supply mix. They highlighted the importance of hardware and data center choice, as they have the highest impact on CO₂e emissions. At the same time, it showcased the energy efficiency of sparse large-scale models like switch-transformers [18] when compared to a densely activated model such as GPT-3 [19]. PaLM 540B [20], another representative of large language models, is one of the first and few works that includes a carbon footprint statement, though it lacks a clear electrical footprint report.

3 Energy Benchmarking in High-Performance Computing

3.1 Background: Determining Energy Consumption

In the following, we will provide a brief overview of common methods used to obtain energy consumption readings. Generally, we distinguish between *measuring* energy, i.e., the process of using physical sensors connected to the compute nodes or other components to monitor the hardware, and *estimating* energy, i.e., using indirect methods to approximate the energy consumption. Leveraging sensors to measure power draw has the highest accuracy, but requires the hardware to be either equipped with additional monitoring devices or manually tapped. Practically, this may hinder sensor use due to additional costs or access restrictions. Which components may be monitored depends on the computing hardware. Past works have focused on individual components like power source, CPU, cooling and memory [13], the individual components of internode communication switches [21], or the consumption of entire nodes [12].

In contrast, energy estimation utilizes indirect, albeit more accessible, tools. An increasingly common way is using the software interfaces provided by hardware manufacturers. The data made available through these interfaces maps to hardware performance counters, special registers embedded in the hardware with the specific purpose of monitoring the device. The energy consumption of hardware components is then estimated by regularly sampling the hardware counters while the monitored device or application is running. These data samples are then aggregated in a post-processing step. Overall, the accuracy of the complete estimation is bound by the registers' resolutions.

An example of such a hardware monitoring interface is `Nvidia Management Library` (NVML) [22], making the power draw of their GPUs available. Figure 1 illustrates an example of the data obtained through its management interface. Similarly, Intel provides access to the accumulated energy usage through the `Running Average Power Limit`² (RAPL) interface. It keeps track of the total energy used on a socket granularity for CPU and DRAM, which can be used to calculate the power draw. Which components may be monitored depends on the individual hardware manufacturers and their interfaces. Additionally, access to these interfaces is usually restricted to privileged users.

If no hardware monitoring interfaces are accessible, a rough estimate can be made using the specifications of the utilized hardware. This method assumes that each component requires the same amount of power throughout the runtime of an application. Practically, the constant power draw is an unrealistic assumption, leading to significant over- or underestimation and should be avoided if possible.

Regardless of the method used to obtain the power draw of individual components, the energy consumption of the application running on a single node can then be calculated by integrating the power draw P over the total runtime T for each component in a computational node and summing up all of them up to obtain the total energy of the node:

² <https://github.com/powercap/raplcap>.

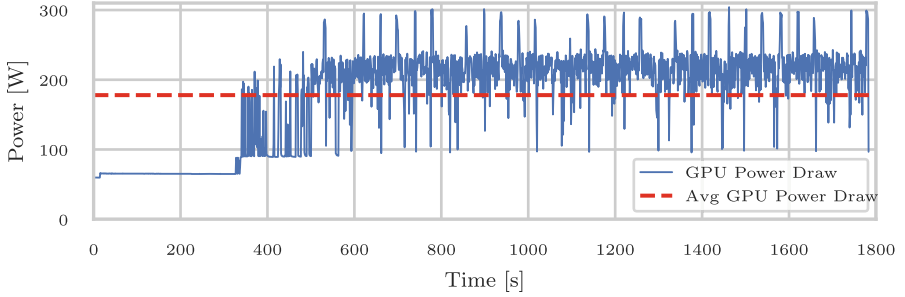


Fig. 1. Energy consumption of a single Nvidia A100 gathered from Nvidia-SMI running the OC20 [23] MLPerf benchmark.

$$E_{\text{node}} = \int_0^T P_{\text{cpu}}(t) + P_{\text{gpu}}(t) + P_{\text{ram}}(t) + P_{\text{others}}(t) dt, \quad (1)$$

where E_{node} is the consumed energy of the subscript components. When running on multi-node applications, the energy of all the individual nodes has to be aggregated and multiplied by the power usage effectiveness (PUE) of the system to obtain the energy consumed of the application:

$$E_{\text{total}} = PUE \cdot \sum_i^{\#\text{nodes}} E_{\text{node}}^{(i)}. \quad (2)$$

PUE is a factor obtained by dividing the total equipment consumption of the data center by the power consumption of the compute equipment. This is a common metric used to compare the efficiency of data centers around the world, with the global average in 2020 being 1.58 [24] and some of the most efficient centers having a 1.02 PUE [25].

For the purpose of this work, Eq. (2) is exhaustive to determine the energy consumption of software running on HPC systems. If needed, the corresponding carbon footprint can be derived by multiplying the resulting total energy E_{total} with an energy-to- CO_2e conversion factor r_e . r_e signifies the emission rate of the energy generation technologies. Just like PUE, this number depends on the location where the software is being run, but additionally changes over time depending on weather, seasons, and energy grid usage, as those factors have a great effect on the available energy sources. The total monetary cost may be derived similarly by replacing r_e with the energy-to-cost conversion ratio.

Both emissions and price fluctuate continuously, whereas it is reasonable to expect that an algorithm would require a (close to) constant amount of power if executed on the same or similar hardware.

3.2 *perun*

To allow users to gather energy measurements of their Python applications on multi-node environments, we have developed a Python package called *perun*. It works by sampling information while a Python application is running. The background process periodically collects information by querying hardware APIs. If the right setup is available, it is capable of incorporating sensor readings into the estimation process, including the power consumption of a single compute node or the complete rack. As of the time of writing, we are not aware of any other energy benchmarking tool capable of incorporating hardware performance counters and sensor measurements of MPI distributed Python applications.

Currently, *perun* supports Intel-RAPL to sample CPU and DRAM information, NVML for GPU information, and `psutil`³ to sample information on network and filesystem I/O. Sensor data may be additionally collected using Lenovo XClarityController⁴, a hardware management system provided by the manufacturer. `psutil`, Intel-RAPL and the hardware sensors report the energy consumption of the whole system. To get more representative results, *perun* works best when there are no other programs running in the system.

To handle distributed Python applications, *perun* makes use of the Message Passing Interface (MPI). MPI defines a communication protocol standard for parallel computers. When using MPI applications, *perun* has a coordination step where each individual rank communicates its host name and the visible devices to all other ranks. Based on this information, the first rank in each host is selected to spawn the process and monitor the visible devices. This coordination step ensures that only one monitoring process is spawned per host and that each device is only accounted for once, keeping the overall overhead of *perun* low. Synchronization between the main and monitoring process is handled by using the `multiprocessing` events from the standard library.

All the raw data gathered during the execution of the monitored application is saved in a HDF5 file, along with metadata about the individual devices, nodes, and environment. *perun* then processes the data using Eqs. (1) and (2) and returns a summarized report. All the results can be exported to human as well as machine-readable formats, like JSON and YAML.

To facilitate usage in different environments, *perun* provides a command line interface as a replacement for the Python command. Alternatively, a `monitor` decorator can be used to target specific functions, as shown in Listing 1. *perun*'s behavior can be modified using a configuration file, command line arguments, decorator arguments, or environmental variables.

While most of the interfaces and software features described during this and previous sections can be applied similarly to other programming languages, due to the way *perun* manages the primary Python process when started from the command line, its functionality is as of the time of writing limited to Python applications.

³ <https://github.com/giampaolo/psutil>.

⁴ <https://www.lenovo.com/in/en/data-center/software/systems-management/XClarity-Controller/p/WMD00000367>.

```
import perun

@perun.monitor(data_out="results/", format="json")
def expensive_computation(input_args):
    pass
```

Listing 1: Example decorator usage

4 Experimental Evaluation

The goal of the following experiments, as described in Sect. 1, are the following: quantify the runtime and power overhead caused by sampling performance counters using *perun*, determine the accuracy of the available performance counters when compared to measurements provided by hardware sensors embedded in the system, observe the power consumption of non-compute components and the impact they have on the overall system consumption, and compare different energy estimation methodologies when applied at scale.

The following sections describe the different use cases used for the analysis, and the system where the experiments were implemented.

4.1 Application Use Cases

As a calibration measure, the energy consumption of an idle node with and without the monitoring software is compared. Based on sensor data obtained during both types of execution, the runtime, and average power usage difference between monitored and non-monitored applications can be used to get an estimate on the overhead caused by *perun*.

Two single-node use cases are considered, one running on CPU and the other running on four GPUs. First, we apply *perun* to monitor *Black-Scholes* [26] option pricing, an operation commonly used in finance, that can be computed in an embarrassingly parallel way, as a common benchmark in the HPC community. We monitor the energy consumption of solving one billion *Black-Scholes* operations using 76 CPU threads as a single-node high resource utilization example. As a second example, we fine-tune the NLP model BERT [27] on the QUAD 1.2 question-answering dataset using a multi-GPU implementation based on the *huggingface*⁵ libraries.

As a large-scale, multi-node example, we evaluate *perun* on two tasks from the MLPerf HPC benchmark suite [28]. The BERT use case was also scaled to two nodes, i.e. eight GPUs. *DeepCam* [29] is an image segmentation model trained on the CAM 5 weather simulation dataset. OpenCatalyst 2020 (OC20) [23] dataset consists of molecular simulation data and defines three supervised learning tasks, where attributes of individual molecules are recreated based on the initial structure. Both models are trained to a pre-defined performance threshold. The imple-

⁵ <https://huggingface.co>.

mentation used is based on the closed submission by the Helmholtz AI group⁶. The OC20 benchmark was run using up to 128 nodes, each one with four GPUs, to observe how scaling affects the overall energy consumption of the application. DeepCam was run using the same hardware setup. All single node programs were run a total of 20 times, half using *perun*, and the other half without. MLPerf programs were run only six times, five times using *perun* and once without.

4.2 Hardware Environment

The use cases were run using two different hardware configurations of the HoreKa supercomputer at the Karlsruhe Institute for Technology: CPU-only nodes, which have no accelerated hardware, and GPU nodes, which include four Nvidia A100-40 GB GPUs. Each node in both partitions has two Intel Xeon Platinum 8368 processors, a 960 GB SSD, and is interconnected with an InfiniBand HDR fabric. The CPU-only nodes have 256 GB, while the accelerated nodes have 512 GB of main memory.

Each node includes special hardware that gathers power measurements from different components, including the CPU, GPU, and the entire node. This information is consistently being collected via the Lenovo XClarity Controllers Redfish REST API and is transferred to an InfluxDB time series database. According to the server documentation⁷, XClarity Controller measurements for GPU and CPU has an 97% accuracy at a 100 Hz sampling rate. The system uses an energy-efficient hot water cooling system, making it a highly efficient HPC system with a PUE of 1.05.

4.3 Software

Two different stacks were used to run the different use cases: a container-based environment, used for both MLPerf use cases, and a *native* environment, which was used for the rest. The native environment makes use of Python 3.8.6, OpenMPI 4.1.3, mip4py 3.1.4, and pytorch 1.12.1 with CUDA 11.6. The base container used for the MLPerf Benchmarks is `pytorch:22.08-py3-devel` from the Nvidia Container Registry. It contains Python 3.8.12, OpenMPI 4.1.2, mip4py 1.13 and pytorch 1.13 with CUDA 11.7. The *perun* version used at the time of the experiments is 0.1.0b16. All jobs were scheduled using SLURM⁸.

5 Results

5.1 Monitoring Overhead

First, we measured the overhead that running an application with *perun* entails using hardware sensor data. Table 1 shows differences in runtime, average power draw per node, and the total energy consumption for runs with and without *perun*. Column N indicates the number of nodes used for each use case.

⁶ <https://mlcommons.org/en/training-hpc-20/>.

⁷ <https://lenovopress.lenovo.com/lp1395-thinksystem-sd650-v2-server>.

⁸ <https://slurm.schedmd.com/overview.html>.

Table 1. Runtime and average node power consumption comparison between software run with and without monitoring software, aggregated over multiple runs.

| Name | N | Runtime | | | | | \bar{P}_{node} | | | | |
|---------------|-----|-------------|-------|-----------|--------|----------|-------------------------|--------|-----------|--------|----------|
| | | Unmonitored | | Monitored | | Δ | Unmonitored | | Monitored | | Δ |
| | | [s] | \pm | [s] | \pm | Δ | [W] | \pm | [W] | \pm | Δ |
| idle | 1 | 78.58 | 1.73 | 81.71 | 1.64 | 3.13 | 293.67 | 12.53 | 292.26 | 13.34 | -1.42 |
| idle gpu | 1 | 81.10 | 0.57 | 86.92 | 2.07 | 5.82 | 571.20 | 15.36 | 564.75 | 13.20 | -6.45 |
| black-scholes | 1 | 1998.00 | 29.36 | 2085.56 | 150.11 | 87.56 | 618.55 | 18.44 | 634.35 | 31.27 | 15.80 |
| BERT | 1 | 1180.27 | 3.93 | 1190.15 | 18.10 | 9.88 | 1319.53 | 29.97 | 1301.74 | 43.31 | -17.79 |
| BERT | 2 | 970.60 | 12.40 | 975.75 | 7.07 | 5.15 | 1058.19 | 58.22 | 1079.33 | 51.90 | 21.14 |
| OC20 | 64 | 2542.00 | - | 2428.51 | 84.16 | -113.49 | 1305.48 | 25.71 | 1300.41 | 31.74 | -5.06 |
| OC20 | 128 | 1752.00 | - | 1785.00 | 53.60 | 33.00 | 1096.76 | 33.16 | 1106.29 | 31.09 | 9.53 |
| DeepCam | 128 | 526.00 | - | 484.60 | 33.03 | -41.40 | 1030.50 | 120.24 | 995.39 | 116.58 | -35.11 |

The high variance makes identifying a clear trend from these results difficult, as the overhead caused by *perun* is often in the same order of magnitude as the variance. The variance in the software’s runtime seems to have the biggest impact and makes it difficult to discern the effect running *perun* has on the monitored software runtime and power consumption. From the execution time of the use cases idle and BERT, we can expect an increase of 5s to 10s of execution time on average. The results of OC20 and DeepCamp have low statistical relevance, as those use cases were run only once without *perun*. Even then, *perun* seems to have a small enough impact that some monitored applications had shorter execution times than the monitored ones.

perun has the biggest impact on the runtime of the *Black-Scholes* use case. As it is a CPU-intensive benchmark compared to the others, the extra processing load from the monitoring process hurts the overall performance. Like the runtime, the average power draw per node has a similarly high variance, often larger than the difference between monitored and unmonitored runs. The high variance can be explained in part by small changes in the hardware itself, as the software was run on different nodes with the same hardware, and there were no warm-up runs before running the use cases, putting the nodes in different stages of idleness/activity when the software started.

Given that the difference in power draw between monitored and unmonitored applications is close to zero, it is fair to assume that the background sampling process does not meaningfully raise the power consumption.

5.2 Monitoring Accuracy and Missing Power Consumption

In order to assess the accuracy of *perun*’s estimates based on hardware performance counters, we compare the difference between the power reported by hardware libraries and sensor data for individual devices. Based on the power consumption of the compute components and sensor data from the entire node, the power draw of difficult to access components, e.g., internal fan, storage, networking cards, can be quantified as a group.

Table 2. Average power draw for component x as reported by performance counters $\overline{P}_{x,p}$ and the hardware sensors $\overline{P}_{x,s}$ for each use case.

| Name | N | $\overline{P}_{\text{dram},p}$ | | $\overline{P}_{\text{dram},s}$ | | $\overline{P}_{\text{cpu},p}$ | | $\overline{P}_{\text{cpu},s}$ | | $\overline{P}_{\text{gpu},p}$ | | $\overline{P}_{\text{gpu},s}$ | | $\overline{P}_{\text{node},p}$ | | $\overline{P}_{\text{node},s}$ | |
|---------------|-----|--------------------------------|-------|--------------------------------|-------|-------------------------------|-------|-------------------------------|-------|-------------------------------|-------|-------------------------------|-------|--------------------------------|-------|--------------------------------|-------|
| | | [W] | \pm | [W] | \pm | [W] | \pm | [W] | \pm | [W] | \pm | [W] | \pm | [W] | \pm | [W] | \pm |
| idle | 1 | 16.8 | 1.3 | 16.4 | 1.2 | 195.5 | 11.1 | 198.0 | 12.2 | - | - | - | - | 212.4 | 11.8 | 289.4 | 14.2 |
| idle gpu | 1 | 24.6 | 0.8 | 24.5 | 1.1 | 210.5 | 5.3 | 208.2 | 9.2 | 218.6 | 3.3 | 218.6 | 3.4 | 453.6 | 5.8 | 560.4 | 13.8 |
| black-scholes | 1 | 17.8 | 0.8 | 17.7 | 0.8 | 529.4 | 28.2 | 531.5 | 28.4 | - | - | - | - | 547.1 | 27.8 | 642.7 | 31.4 |
| BERT | 1 | 26.3 | 1.1 | 26.2 | 1.1 | 231.7 | 5.1 | 232.1 | 5.4 | 970.7 | 24.6 | 941.6 | 33.0 | 1228.7 | 24.1 | 1338.5 | 57.4 |
| BERT | 2 | 27.6 | 1.0 | 27.5 | 1.1 | 240.4 | 5.9 | 240.0 | 5.8 | 712.6 | 19.0 | 698.2 | 24.1 | 980.6 | 19.3 | 1115.6 | 63.6 |
| OC20 | 16 | 26.2 | 1.2 | 26.1 | 1.2 | 256.7 | 6.8 | 257.3 | 6.8 | 1035.4 | 14.8 | 1031.4 | 15.5 | 1318.3 | 18.0 | 1473.7 | 22.7 |
| OC20 | 32 | 26.4 | 1.2 | 26.3 | 1.2 | 258.8 | 7.2 | 259.3 | 7.0 | 1027.5 | 18.5 | 1022.4 | 19.6 | 1312.7 | 21.3 | 1465.3 | 29.3 |
| OC20 | 64 | 26.6 | 1.2 | 26.6 | 1.2 | 266.9 | 7.2 | 267.1 | 7.4 | 882.8 | 16.5 | 874.6 | 19.2 | 1176.3 | 18.8 | 1316.2 | 29.5 |
| OC20 | 128 | 26.8 | 1.2 | 27.5 | 23.1 | 268.4 | 7.7 | 269.0 | 8.0 | 692.1 | 13.7 | 686.1 | 19.1 | 987.3 | 17.3 | 1119.4 | 31.9 |
| DeepCam | 16 | 31.1 | 1.2 | 30.6 | 1.3 | 261.4 | 6.6 | 259.6 | 7.7 | 640.0 | 15.5 | 645.2 | 43.0 | 932.5 | 16.9 | 1032.9 | 74.6 |
| DeepCam | 128 | 30.3 | 1.3 | 30.1 | 1.8 | 246.6 | 7.0 | 251.1 | 11.5 | 681.8 | 12.6 | 692.0 | 83.6 | 958.7 | 15.7 | 1026.1 | 124.5 |

Table 2 shows the average power draw measured by each device throughout the execution of the software, averaged over multiple runs. All DRAM, CPUs, and GPUs in a node are grouped and summed together. \overline{P} indicates the average power draw from the compute nodes while the software was executed. The subscript indicates the hardware component and the data source, p for performance counters and s for sensor data.

For DRAM and CPU, we observe almost no difference between the sensor data and performance counters, with a maximum difference of 1 W for DRAM and 0 W to 2 W for CPU power draw. This difference is more pronounced for GPU devices, averaging at 5.68 ± 10.35 W overestimation from performance counters. Data from performance counters and sensors have a higher variance for GPUs than other hardware components, making it harder to approximate. According to the official Nvidia System Management Library documentation [22], the power values returned by the performance counters have an accuracy of ± 5 W. Additionally, the measured sensor includes the power consumption of all components on the GPU board at a higher sampling frequency, not only the GPU and High Bandwidth Memory (HBM) that are measured by the performance counters.

When looking at the aggregated power consumption for the entire node, there is a clear difference between what can be estimated using performance counters and full node sensor data, providing a clearer picture on the power draw of components lacking monitoring support. In this particular setup, this means networking, storage, cooling systems, power supply and motherboard. The power consumption of these components are also application dependent and come with their own variance, adding uncertainty to the estimation process. For nodes without GPUs, we have measured their required power draw to be 78.93 ± 8.39 W, and for nodes with GPUs, the unaccounted power draw is on average 109.37 ± 30.51 W. The previous values can be inserted into Eq. (1) alongside the estimates for CPU, GPU, and DRAM to correct the energy estimation of

individual nodes, thus closing the gap between estimated and measured total energy consumption seen in the last two columns.

In conclusion, the usage of performance counters provide a good estimation of the energy consumption for individual components, reducing the need of dedicated hardware unless precision is of utmost importance. Estimations of the total node power draw can be improved by adding a constant power draw throughout the runtime of the application. Finding an optimal value is difficult without measuring equipment, and will need to be chosen carefully based on the system.

5.3 Impact of Non-compute Devices on the Overall Energy Consumption

Table 3. Power draw percentage per device.

| Name | N | $\overline{P}_{\text{dram}}$ | | $\overline{P}_{\text{cpu}}$ | | $\overline{P}_{\text{gpu}}$ | | $\overline{P}_{\text{rest}}$ | | $\overline{P}_{\text{node}}$ [W] |
|---------------|-----|------------------------------|-----|-----------------------------|------|-----------------------------|------|------------------------------|------|----------------------------------|
| | | [W] | [%] | [W] | [%] | [W] | [%] | [W] | [%] | |
| idle | 1 | 16.4 | 5.7 | 198.0 | 68.4 | – | 0.0 | 75.1 | 25.9 | 289.4 |
| idle gpu | 1 | 24.5 | 4.4 | 208.2 | 37.1 | 218.6 | 39.0 | 109.1 | 19.5 | 560.4 |
| black-scholes | 1 | 17.7 | 2.7 | 531.5 | 82.7 | – | 0.0 | 93.6 | 14.6 | 642.7 |
| BERT | 1 | 26.2 | 2.0 | 232.1 | 17.3 | 941.6 | 70.3 | 138.7 | 10.4 | 1338.5 |
| BERT | 2 | 27.5 | 2.5 | 240.0 | 21.5 | 698.2 | 62.6 | 149.9 | 13.4 | 1115.6 |
| OC20 | 16 | 26.1 | 1.8 | 257.3 | 17.5 | 1031.4 | 70.0 | 158.9 | 10.8 | 1473.7 |
| OC20 | 32 | 26.3 | 1.8 | 259.3 | 17.7 | 1022.4 | 69.8 | 157.4 | 10.7 | 1465.3 |
| OC20 | 64 | 26.6 | 2.0 | 267.1 | 20.3 | 874.6 | 66.4 | 147.9 | 11.2 | 1316.2 |
| OC20 | 128 | 27.5 | 2.5 | 269.0 | 24.0 | 686.1 | 61.3 | 136.7 | 12.2 | 1119.4 |
| DeepCam | 16 | 30.6 | 3.0 | 259.6 | 25.1 | 645.2 | 62.5 | 97.5 | 9.4 | 1032.9 |
| DeepCam | 128 | 30.1 | 2.9 | 251.1 | 24.5 | 692.0 | 67.4 | 52.8 | 5.2 | 1026.1 |

As shown in the previous section, the power draw of non-compute components is not negligible, contributing to a high percentage of the overall energy consumption. Table 3 breaks up the total energy consumption by devices, assigning the remainder to the non-compute components.

We observe that as the CPU and GPU utilization, and with it their power draw, increases, the share of non-compute components in the total energy consumption decreases. However, even under high utilization, non-compute components make up about 15% of the energy consumption of CPU-only nodes and more than 5% of GPU nodes.

5.4 Scaling Behavior for Multi-Node Applications

Using the OC20 use case, we compare the accuracy of different energy consumption estimation methods on massively parallel applications. We consider

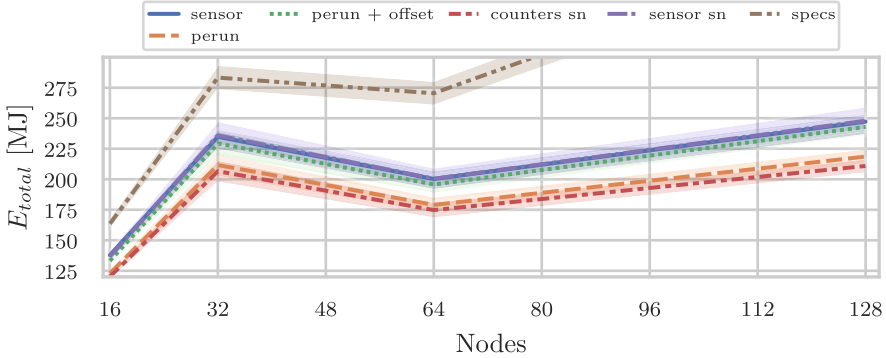


Fig. 2. Reported total energy consumption of different estimation methods on the OC20 tasks as a function of number of compute nodes.

performance counters (*perun*), performance counters including a “correction offset” of 110 W based on the analysis done in Sect. 5.2 (*perun+offset*), hardware measurements (sensors), and a simpler estimation based on the hardware specifications (*specs*) of the CPU and GPU. The analysis will also include estimations based on the data of a single node (*sn*) for both sensors and performance counters, following the methodology described in Green500 benchmark tutorial [30] from 2007.

Figure 2 displays the total energy calculated by the different estimations methods. The transparent area around each line represents the standard deviation. At a first glance, it becomes clear that using the specified Thermal Design Power from the components leads to an overestimation of the consumed energy, with the difference becoming bigger as the number of nodes increases and the utilization of each component decrease. It can work as an upper bound if no other source of information is available.

The results show, that for this particular use case, measuring a single full node and multiplying by the number of nodes provides an accurate approximation of the total energy usage during the application runtime. This might change for different applications, if the load distribution is drastically different in the individual compute nodes. A close second is *perun+offset*, which managed to close the gap between sensors and performance counters by adding a flat offset to the original estimation of *perun*. Estimations based on performance counters (*perun* and *sensor sn*) slowly diverge as the number of nodes increases, with a difference of around 25 MJ on the 128 node configuration. Still, performance counter based estimations provide better results when run on all nodes, with a difference in the order of MJ between multi-node and single-node estimations on all node configurations. This supports the need for a distributed monitoring tool when hardware sensors are not an option.

6 Conclusion

In this study, we introduce *perun*, a novel Python package to facilitate energy benchmarking of MPI-distributed workloads. We analyzed the impact such a tool has on the overall runtime and energy consumption and found that the overhead is negligible.

Making use of *perun*, an analysis of the energy estimations based on hardware performance counters was presented alongside sensor data from those same components and the entire node. The difference in reported power draw from the two sources indicates that CPU and DRAM data matches the sensor readings adequately. A larger distance is observed between power draw estimations and measurements for GPUs.

From these results, an approximation could be made on the power draw of often unaccounted hardware components, which can later be used to correct any power estimations made using only CPUs, GPUs, and DRAM. The data shows that the power of those components entails a non-minuscule percentage of the total power consumption of each node, and its impact should be considered when writing impact statements.

Finally, the difference between different energy estimation methodologies is highlighted using the OC20 benchmark on different hardware configurations. The results highlight the importance of making use of distributed monitoring tools like *perun* and the need to account the power draw of non-compute components, as their impact increases with the number of nodes.

6.1 Limitations

While the individual hardware components and software interfaces are common in other HPC systems, the power measuring equipment is not so, complicating the evaluation of the presented approach in other systems. Similar studies with different hardware and workloads will further aid in understanding the energy consumption of applications with high levels of parallelism.

In the making of this paper, a total of 2136.42 kWh were used, which based on the location and time of our experiments, generated 841.75 kg CO_{2e}.

Acknowledgments. This work is supported by the Helmholtz project HiRSE_PS, the Helmholtz AI platform and the HAICORE@KIT grant.

References

1. Zhang, F., Bonart, H., Zirwes, T., Habisreuther, P., Bockhorn, H., Zarzalis, N.: Direct numerical simulation of chemically reacting flows with the public domain code OpenFOAM. In: Nagel, W.E., Kröner, D.H., Resch, M.M. (eds.) High Performance Computing in Science and Engineering 2014, pp. 221–236. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10810-0_16 ISBN: 978-3-319-10810-0

2. Weiel, M., Götz, M., Klein, A., et al.: Dynamic particle swarm optimization of biomolecular simulation parameters with flexible objective functions. *Nat. Mach. Intell.* **3**(8), 727–734 (2021). <https://doi.org/10.1038/s42256-021-00366-3>. ISSN: 2522-5839
3. Durr, S., Fodor, Z., Frison, J., et al.: Ab initio determination of light hadron masses. *Science* **322**(5905), 1224–1227 (2008)
4. Strohmaier, E., Dongarra, J., Simon, H., et al.: TOP500 (1993). <https://www.top500.org/>. Accessed 20 Feb 2023
5. Patterson, D., Gonzalez, J., Hülzle, U., et al.: The carbon footprint of machine learning training will plateau, then shrink. *Computer* **55**(7), 18–28 (2022). <https://doi.org/10.1109/MC.2022.3148714>. Conference Name: Computer, ISSN: 1558-0814
6. Feng, W.-C., Cameron, K.: The Green500 list: encouraging sustainable supercomputing. *Computer* **40**(12), 50–55 (2007). <https://doi.org/10.1109/MC.2007.445>. Conference Name: Computer, ISSN: 1558-0814
7. Anthony, L.F.W., Kanding, B., Selvan, R.: Carbontracker: tracking and predicting the carbon footprint of training deep learning models (2020). [arXiv: 2007.03051](https://arxiv.org/abs/2007.03051) [cs, eess, stat]
8. Henderson, P., Hu, J., Romoff, J., et al.: Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Mach. Learn. Res.* **21**(248), 1–43 (2020). ISSN: 1533-7928
9. Schmidt, V., Goyal-Kamal, Courty, B., et al.: mlco2/codecarbon: v2.1.4 (2022). <https://doi.org/10.5281/zenodo.7049269>
10. Lacoste, A., Luccioni, A., Schmidt, V., et al.: Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700* (2019)
11. Lannelongue, L., Grealey, J., Inouye, M.: Green algorithms: quantifying the carbon footprint of computation. *Adv. Sci.* **8**(12), 2100707 (2021). <https://doi.org/10.1002/advs.202100707>. ISSN: 2198-3844, 2198-3844
12. Caspart, R., et al.: Precise energy consumption measurements of heterogeneous artificial intelligence workloads. In: Anzt, H., Bienz, A., Luszczek, P., Baboulin, M. (eds.) *ISC High Performance 2022*. LNCS, vol. 13387, pp. 108–121. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-23220-6_8 ISBN: 978-3-031-23220-6
13. Hodak, M., Dholakia, A.: Recent efficiency gains in deep learning: performance, power, and sustainability. In: *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2040–2045 (2021). <https://doi.org/10.1109/BigData52589.2021.9671762>
14. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP (2019). <https://doi.org/10.48550/arXiv.1906.02243>. [arXiv:1906.02243](https://arxiv.org/abs/1906.02243) [cs]
15. ISO/IEC 30134-2:2016. ISO (2016). <https://www.iso.org/standard/63451.html>. Accessed 09 Feb 2023
16. Brady, G.A., Kapur, N., Summers, J.L., et al.: A case study and critical assessment in calculating power usage effectiveness for a data centre. *Energy Convers. Manag.* **76**, 155–161 (2013). <https://doi.org/10.1016/J.ENCONMAN.2013.07.035>
17. Patterson, D., Gonzalez, J., Le, Q., et al.: Carbon emissions and large neural network training (2021). <https://doi.org/10.48550/arXiv.2104.10350>. <http://arxiv.org/abs/2104.10350> [cs]
18. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: scaling to trillion parameter models with simple and efficient sparsity (2022). [arXiv:2101.03961](https://arxiv.org/abs/2101.03961) [cs]
19. Brown, T.B., Mann, B., Ryder, N., et al.: Language models are few-shot learners (2020). [arXiv:2005.14165](https://arxiv.org/abs/2005.14165) [cs]
20. Chowdhery, A., Narang, S., Devlin, J., et al.: PaLM: scaling language modeling with pathways (2022). [arXiv:2204.02311](https://arxiv.org/abs/2204.02311) [cs]

21. Wang, H., Li, Z., Zhao, X., He, Q., Sun, J.: Evaluating the energy consumption of InfiniBand switch based on time series. In: Wong, W.E., Zhu, T. (eds.) *Computer Engineering and Networking*. LNEE, vol. 277, pp. 963–970. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-01766-2_110 ISBN: 978-3-319-01766-2
22. NVIDIA system management interface, NVIDIA Developer (2012). <https://developer.nvidia.com/nvidia-system-management-interface>. Accessed 14 Feb 2023
23. Chanussot, L., Das, A., Goyal, S., et al.: The open catalyst 2020 (OC20) dataset and community challenges. *ACS Catal.* **11**(10), 6059–6072 (2021). <https://doi.org/10.1021/acscatal.0c04525>. arXiv: 2010.09990 [cond-mat]. ISSN: 2155-5435, 2155-5435
24. Lawrence, A.: Data center PUEs flat since 2013. Uptime Institute Blog (2020). <https://journal.uptimeinstitute.com/data-center-pues-flat-since-2013/>. Accessed 31 Jan 2023
25. Miller, R.: Immersion cooling at scale: BitFury pushes density to 250kw per rack. *Data Center Frontier* (2015). <https://www.datacenterfrontier.com/featured/article/11431449/immersion-cooling-at-scale-bitfury-pushes-density-to-250kw-per-rack>. Accessed 31 Jan 2023
26. Black, F., Scholes, M.S.: The pricing of options and corporate liabilities. *J. Polit. Econ.* **81**, 637–654 (1973)
27. Devlin, J., Chang, M.-W., Lee, K., et al.: BERT: pre-training of deep bidirectional transformers for language understanding (2019). <http://arxiv.org/abs/1810.04805> [cs]
28. Farrell, S., Emani, M., Balma, J., et al.: MLPerf HPC: a holistic benchmark suite for scientific machine learning on HPC systems (2021). arXiv:2110.11466 [cs]
29. Kurth, T., Treichler, S., Romero, J., et al.: Exascale deep learning for climate analytics. In: *SC 2018: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 649–660 (2018). <https://doi.org/10.1109/SC.2018.00054>
30. Ge, R., Feng, X., Pyla, H., et al.: Power measurement tutorial for the Green500 list (2007)