Abstract—Efficient use of energy is essential for today’s supercomputing systems, as energy cost is generally a major component of their operational cost. Research into “green computing” is needed to reduce the environmental impact of running these systems. As such, several scientific communities are evaluating the trade-off between time-to-solution and energy-to-solution. While the runtime of an application is typically easy to measure, power consumption is not.

Therefore, we present the Power Measurement Toolkit (PMT), a high-level software library capable of collecting power consumption measurements on various hardware. The library provides a standard interface to easily measure the energy use of devices such as CPUs and GPUs in critical application sections. Index Terms—CPU, Efficiency, GPU, HPC, Performance

I. INTRODUCTION

Contemporary scientific applications such as particle hydrodynamics [1], radio-astronomical imaging [2] and Earth digital twin simulation [3] have demanding compute requirements, in the order of the ExaFlops or more. Moreover, top supercomputers [4] aim to reach Zettascale [5] performance in the current decade. To achieve this goal, increasingly more powerful hardware is used. However, due to the end of Dennard’s Scaling law, these advances in compute power come at the cost of a higher power consumption [6].

Energy efficiency is a critical challenge in HPC (High-Performance Computing) as data centers are struggling to operate within stringent power budgets [7] and also try to reduce their carbon footprint [8]. Therefore, it is crucial to monitor the power consumption of HPC applications in order to evaluate trade-offs between achieved performance and energy efficiency [9].

In this context, we present the Power Measurement Toolkit (PMT), a lightweight high-level library capable of collecting power measurements on various architectures, such as CPUs and GPUs. PMT can easily be used to probe specific application regions to measure power consumption and evaluate energy efficiency.

II. POWER MEASUREMENT TOOLKIT

PMT’s structure is reported in Fig. 1. The library is written in C++ and is Linux-only; it interfaces with hardware vendors’ API to collect power consumption. More precisely, for CPUs it uses NVML [10] for NVIDIA and rocm-smi [11] for AMD. CPUs are monitored through the RAPL (Running Average Power Limit) [12] interface, or through LIKWID [13]. PMT can profile other architectures, such as Xilinx FPGAs, and it can be easily extended to support new vendors’ hardware. Some other architectures expose their power usage information through files in sysfs (the /sys folder). PMT also has an interface to physical power sensors such as PowerSensor2 [14].

PMT library’s core consists of a background thread to the profiled application that communicates and gathers power consumption information from the selected back end, such as NVML. PMT sampling frequency is dependent on the hardware and back end. For instance, NVML is able to sustain up to 10 ms and RAPL up to 500 ms.

PMT can be integrated into C++ and Python applications. In C++, the code must be instrumented as shown in Listing 1. Using PMT in a C++ application is a three-step process: 1) include the PMT header; 2) initialize PMT; 3) surround the region of interest with PMT activation and deactivation calls. This can be done similarly in Python by using Python bindings. However, we provide a simpler interface using Python decorators (see Listing 2). As shown in Listing 2, the Python code only requires an import statement of the PMT library and the PMT decorators. In the case mentioned, we are collecting the measurement by using the NVML back end for NVIDIA GPUs and RAPL for AMD CPUs. The library support multiple decorators at the same time (see Listing 2).

Typically, PMT has a small overhead in the order of 1 ms in C++ and 10 ms in Python. This overhead is cumulative when multiple decorators are used.

The Listings 1 and 2 show the above-mentioned measurement-mode and how to programmatically query the measurement after the region of interest. Some changes are needed to activate the dump-mode. In Python, we just need to rename the decorator with dump. Similarly, in C++,
#include <iostream>
#include <pmt.h>
#include <unistd.h>

int main() {
    std::unique_ptr<pmt::PMT> sensor(pmt::nvml::NVML::create());
    pmt::State start, end;
    start = sensor->read();
    sleep(5);
    end = sensor->read();
    std::cout << sensor->joules(start, end) << " [J]" << std::endl;
    std::cout << sensor->watts(start, end) << " [W]" << std::endl;
    std::cout << sensor->seconds(start, end) << " [S]" << std::endl;
}

Listing 1. PMT example usage in C++ with NVML.

```python
import time
def my_application():
    time.sleep(5)
    @pmt.measure("rapl")
    @pmt.measure("nvml")
    def my_application():
        time.sleep(5)
        if __name__ == "__main__":
            measures = my_application()
            for m in measures:
                print(m)
```

Listing 2. PMT decorator example usage with NVML and rapl.

Fig. 2. Benchmarking simple GPU kernels using PMT: GPUs power consumption (NVIDIA in green and AMD in red) is stacked on top of CPU one (in blue).

we do not have to instantiate the start and end states. To start and stop this mode, we just need to surround some code with the respective APIs, e.g., start_dump_thread. In both Python and C++, the user must provide the filename where we would like to store the power measurements.

### III. BENCHMARKING

We monitor the power consumption on a system with an AMD Ryzen Threadripper 3970X CPU and two different GPUs: NVIDIA TITAN RTX and AMD Radeon PRO W6600. We run a set of simple GPU kernels to show in Fig. 2 PMT profiling capabilities: 1) a SLEEP kernel that leaves the GPU in idle; 2) FMA32, that executes many single-precision floating-point fused multiply and accumulate instructions; 3) STREAM, that stresses the GPU’s device memory with a stream add motif; 4) GRIDDER and 5) DEGRIDDER are radio-astronomical griddner and degriddner kernels inspired by [2]; 6) GEMM, 7) JACOBI2D extracted from Polybench [18].

We clearly notice in Fig. 2 that GPUs have low idle power consumption and can consume extremely high power while executing applications.

In real-world scenarios, PMT can be used to assess application energy efficiency. This can be done in several ways. Users can extract measurements with PMT and derive energy efficiency metrics such as energy-delay product (EDP), which is the product of the execution time and the energy consumed, and the FLOPs efficiency, which can be expressed in GFLOPs/W. Note that the last metric requires the number of FLOPs computed; this can be done by hand or using specific tools such as PAPI [19] and LIKWID [13].

### IV. RELATED WORK

Power meters remain the most accurate method to monitor the power consumption of current computing systems [14]. However, it is not always possible and/or desirable to use a physical tool to measure power consumption. For quick and reliable estimates, built-in sensors can be used [20]. Most of the prior art focuses on a limited set of CPU and GPU brands [21]–[24], while PMT has more comprehensive hardware support. PAPI [19] and LIKWID [13] are noteworthy to be mentioned; they support power consumption monitoring for a limited set of architectures.

To the best of our knowledge, PMT is the first library that provides a common interface to measure power consumption on numerous devices.

### V. CONCLUSION

PMT allows power consumption measurements and monitoring with a simple interface on various hardware. PMT’s users can range from HPC application developers, who could employ it to monitor and evaluate the energy efficiency of their code while optimizing it, to basic Python developers that would like to measure the power consumption of their application with a simple tool. We plan to add support for upcoming hardware, such as Intel GPUs. The library will be available at the link: https://git.astron.nl/RD/pmt.

ACKNOWLEDGMENT

We thank anonymous reviewers of SC22 for their feedback and comments. This research was partially funded by the Swiss State Secretariat for Education, Research and Innovation (SERI). We want to thank J. W. Romein (ASTRON) for its PowerSensor2 work, which lays the foundation of PMT.
REFERENCES

[1] A. Cavelan, R. M. Cabezón, M. Grabarczyk, and F. M. Ciorba, “A smoothed particle hydrodynamics mini-app for exascale,” in Proceedings of the Platform for Advanced Scientific Computing Conference, ser. PASC ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3389347.3401855

[2] S. Corda, B. Veenboer, A. J. Awan, J. W. Romein, R. Jordans, A. Kumar, A.-J. Boonstra, and H. Corporaal, “Reduced-precision acceleration of radio-astronomical imaging on reconfigurable hardware,” IEEE Access, vol. 10, pp. 22 819–22 843, 2022.

[3] P. Bauer, B. Stevens, and W. Hazeleger, “A digital twin of earth for the green transition,” Nature Climate Change, vol. 11, pp. 1–4, 02 2021.

[4] J. Dongarra and P. Luszczek, TOP500. Boston, MA: Springer US, 2011, pp. 2055–2057.

[5] X.-K. Liao, K. Lu, C.-q. Yang, J.-w. Li, Y. Yuan, M.-c. Lai, L.-b. Huang, P.-j. Lu, J. Fang, J. Ren, and J. Shen, “Moving from exascale to zettascale computing: challenges and techniques,” Frontiers of Information Technology & Electronic Engineering, vol. 19, pp. 1236–1244, 10 2018.

[6] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in 2011 38th Annual International Symposium on Computer Architecture (ISCA), 2011, pp. 365–376.

[7] P. Heinisch, K. Ostaszewski, and H. Ranocha, “Towards green computing: A survey of performance and energy efficiency of different platforms using opencl,” in Proceedings of the International Workshop on OpenCL, ser. IWOCCL ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3388333.3403035

[8] D. C. Wilson, S. Jana, A. Marathe, S. Brink, C. M. Cantalupo, D. R. Guttman, B. Gelz, L. H. Lawson, A. H. Al-rawi, A. Mohammad, F. Keceli, F. Ardanz, J. M. Eastep, and A. K. Coskun, “Introducing application awareness into a unified power management stack,” in 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021, pp. 320–329.

[9] AMD, “Rpcm system management interface (rcm smi) library,” Aug. 2022. [Online]. Available: https://github.com/RadeonOpenCompute/rcm_smi_lib

[10] J. W. Romein and B. Veenboer, “Powersensor 2: A fast power measurement tool,” in 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2018, pp. 111–113.

[11] T. Gamblin, “spack/spack: v0.17.1,” Aug. 2022. [Online]. Available: https://github.com/spack/spack

[12] G. Becker, P. Scheibert, M. LeGendre, and T. Gamblin, “Managing combinatorial software installations with spack,” in 2016 Third International Workshop on HPC User Support Tools (HUST), 2016, pp. 14–23.

[13] B. Veenboer, M. Nijhuis, and S. Corda, “Schaap-spack,” https://git. astron.nl/rd/schaap-spack/, 2022.

[14] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, “Auto-tuning a high-level language targeted to gpu codes,” in 2012 Innovative Parallel Computing (InPar), 2012, pp. 1–10.

[15] H. Jagode, A. Danalis, H. Anzt, and J. Dongarra, “Papi software-defined events for in-depth performance analysis,” The International Journal of High Performance Computing Applications, vol. 33, no. 6, pp. 1113–1127, 2019.

[16] Y. Araf, A. ElWazir, A. ElKanisty, Y. Aly, A. Elsayed, A.-H. Badawy, G. Chennupati, S. Eidenbenz, and N. Santhi, “Verified instruction-level energy consumption measurement for nvidia gpus,” in Proceedings of the 17th ACM International Conference on
**REPRODUCIBILITY INITIATIVE**

To reproduce our results, you can install PMT yourself and run the benchmarks that we used to demonstrate PMT. In the following subsections, we first list all the relevant software versions, next, we describe how PMT is installed, and we describe how the benchmarks can be executed and how the plots are generated.

A. Software Packages

Software packages can be installed using Spack [15]. A recent version of git and cmake are required. The cuda package is required to use PMT with NVIDIA GPUs. For AMD GPUs, the rocm-smi-lib is needed. Furthermore, the hip package is required to run the provided benchmarks.

We employed the following software versions:

| Software  | Version |
|-----------|---------|
| spack     | 17.1.0  |
| gcc       | 9.4.0   |
| git       | 2.31.1  |
| cmake     | 3.21.4  |
| cuda      | 11.5.0  |
| hip       | 4.3.1   |
| python    | 3.8.12  |
| py-pybind11 | 2.8.1 |
| rocm-smi-lib | 4.3.1 |

B. PMT installation

Make sure to load the environment modules, such as cuda, or install the packages mentioned above system-wide. Then, download and install the default PMT features such as Rapl.

```
spack install pmt
```

The previous command can be enrich with the features that the user would like to add, e.g., `-D BUILD_NVML=ON`.

Otherwise the user can select a custom option such as building NVML or ROCM support by accessing the CMake graphic interface by running (in the build folder):

```
cmake..
```

The previous command can be enrich with the features that the user would like to add, e.g., `-D BUILD_NVML=ON`.

Otherwise the user can select a custom option such as building NVML or ROCM support by accessing the CMake graphic interface by running (in the build folder):

```
ccmake .
```

After selecting the additional features, the library can be rebuilt:

```
make -j install
```

Another installation method is using Spack with the custom recipe in [17], which are developed for the Square Kilometre Array (SKA) project by the Schaap Team. After adding the Spack repository to a new Spack environment by using the command:

```
spack repo add schaap--spack
```

The PMT package can be just installed by running:

```
spack install pmt
```

C. Benchmarks

If PMT is built from the source code, the PMT install folder should be added to the linker path:

```
export LD_LIBRARY_PATH=LD_LIBRARY_PATH:${user_path%/pmt/install}/
```

In case the user would like to use the Python bindings after installing PMT from the source code, the Python library path must be added to the Python path:

```
export PYTHONPATH=PYTHONPATH:${user_path%/pmt/install/lib/python}:
```

Note that the previous two steps are not required if the user has installed PMT using Spack.

We provide the benchmarks employed at this link https://gitlab.com/p7294/pmt-bench.git

The benchmarks can be built using CMake. Otherwise, we prepared installation scripts in the scripts folder such as `install_nvidia.sh` Before running building the benchmarks, the user must source the environment variables in the scripts in the share folder by running:

```
source . /share/setup-env.sh
```

If the user targets an NVIDIA GPU, additional environmental variables are needed:

```
source . /share/setup-hip-nvidia.sh
```

After compiling the benchmark, create a folder called res and run the benchmarks.

D. Plotting the results

We provide a Jupyter notebook (https://gitlab.com/p7294/pmt-bench.git, in the plot folder) to plot the results stored in the res folder. To run the notebook some python packages are required: matplotlib, seaborn, pandas, numpy and scipy.