J Vis (2024) 27:1171–1193
https://doi.org/10.1007/s12650-024-01001-0

REGULAR PAPER

Patrick Gralka · Christoph Müller · Moritz Heinemann · Guido Reina · Daniel Weiskopf · Thomas Ertl

Power overwhelming: the one with the oscilloscopes

Received: 29 February 2024 / Accepted: 10 April 2024 / Published online: 10 August 2024
© The Author(s) 2024

Abstract Visualization as a discipline has to investigate its practical implications in a world steadily moving toward greener computing methods. Quantifying the power consumption of visualization algorithms is thus essential, given the ever-increasing energy needs of GPUs. Previous approaches rely on integrated sensors or invasive methods that require modifications and special test setups. However, they still suffer from imprecision from low sampling rates and integration over time. Using a high-precision, high-frequency setup via steerable oscilloscopes, we can objectively measure the resulting quality of previous approaches. This is essential to establish a ground truth, pave the way for improved modeling of power consumption in general, and enable better estimates based on the output of lower-quality sensors. We finally discuss benefits that can be drawn from the additional insight of the higher-precision setup and which additional use cases can justify the incurred costs.

Keywords Green visualization · Power measurement · Sphere rendering

1 Introduction

Graphics processing units (GPUs) have become the single most power-hungry component of modern computers, and new hardware standards like ATX 3.0 (Intel 2023a) raise the limit of what future hardware generations are allowed to draw from the power supply unit (PSU) even more. These circumstances have recently sparked interest (Müller et al. 2022) in following up previous research (Johnsson et al. 2012) trying to quantify the effects of software on energy consumption in the areas of visualization and computer graphics. However, the sensors used in this context often aggregate data over time. This is particularly true for the management libraries provided by GPU vendors to read out onboard sensor data from the GPU. For

P. Gralka (✉) · C. Müller · M. Heinemann · G. Reina · D. Weiskopf · T. Ertl
Visualization Research Center, University of Stuttgart, Allmandring 19, 70569 Stuttgart, Germany
E-mail: patrick.gralka@visus.uni-stuttgart.de

C. Müller
E-mail: christoph.mueller@visus.uni-stuttgart.de

M. Heinemann
E-mail: moritz.heinemann@visus.uni-stuttgart.de

G. Reina
E-mail: guido.reina@visus.uni-stuttgart.de

D. Weiskopf
E-mail: daniel.weiskopf@visus.uni-stuttgart.de

T. Ertl
E-mail: thomas.ertl@vis.uni-stuttgart.de

instance, NVIDIA's NVML (NVIDIA Management Library) yields a new value approximately every 100 ms using an averaging window of approximately 1 s (Müller et al. 2022), which makes it impossible to look into a single frame of an interactive visualization application.

Oscilloscopes, in contrast, are designed to sample data at very high rates, but they come at a hefty price point. In addition, the measurement period is very limited by the internal storage size, as oscilloscopes are typically used to analyze the waveform of periodic signals in contrast to the automated evaluation of the energy consumption of GPUs. Therefore, it becomes increasingly important to synchronize the interesting behavior of the software under investigation with the measurements performed on the instruments.

We see various reasons one would still want to embark on such a journey. First, it is desirable to calibrate the results we can obtain with cheaper and easier-to-use sensors with ground truth from the best possible sensor that can even look into different phases within a rendered frame. Only this way can we make an informed decision about whether one sensor is sufficient to answer a question. Likewise, ground truth is also necessary if we want to develop a model of the energy consumption of visualization software that could replace actual measurements. Finally, we see the potential that precise in-frame data on the energy consumption of the GPU can guide the optimization of visualization algorithms. For instance, two different methods might yield the same result when measurements are averaged, as is the case for NVML. However, the cost might be incurred by entirely different phases of the algorithm. Thus, finding the hot spot that requires optimization might not be possible.

Hence, we will subsequently describe a setup that allows for automated measurement of the power consumption of GPUs (and other system components) at the high sampling rate only available using oscilloscopes (Sect. 4). We specifically address how we integrated this measurement process into an existing real-world scientific visualization application and how we synchronize the frames rendered by the software with the instruments. Using this setup, we sampled the energy consumption of different algorithms to render large numbers of spherical glyphs, namely two methods based on OpenGL and an implementation performing ray tracing using OptiX (Parker et al. 2010) (Sect. 5). By comparing the results with the simultaneously acquired measurements from less accurate but at the same time less complex methods, we provide an objective measure for how each of the methods performs, can reason about their accuracy, and make informed decisions on the cost and benefit of each setup. Furthermore, high-resolution power measurements, such as the ones we present, are becoming increasingly prevalent in the high-performance computing (HPC) community to argue about the energy efficiency of software on specific hardware platforms. However, such measurements are currently lacking in the visualization community. Having such data is required to model the effect of software on energy consumption accurately. Understanding these effects is key to improving and optimizing software, reducing its environmental footprint, and a step toward green visualization and computer graphics.

2 Related work

Our work follows up on the one by Müller et al. (2022), who investigated the energy consumption of scientific visualization algorithms and web-based information visualization using different techniques. These were primarily the use of software sensors provided by GPU vendors via their management APIs[1] and the use of shunt resistors and microcontrollers to obtain data at a higher rate than what software can provide. Their work only briefly touches on methods providing even lower sample rates (e.g., external power meters) or higher sample rates as they are achievable using oscilloscopes with a combination of voltage probes and current clamps for each power rail of the system—the latter being the primary focus of this work. Another difference to our work is that we are integrating the power measurements with a real-world visualization application in contrast to measuring a controlled workload, which accounts for limited sensor bandwidth by averaging over time.

A decade earlier, Johnsson et al. (2012) already investigated how software influences power consumption and examined several OpenGL- and OpenGL ES-based rendering algorithms using a custom sensor board equipped with Hall-effect sensors (Ramsden 2006) and shunt resistors. A follow-up publication (Johnsson and Akenine-Möller 2014) focused on measuring per-frame consumption on a CPU-integrated GPU and an iPhone 4 S. To do so, they segmented the power samples obtained at regular distances and integrated the energy between the frame marks. In open-source applications, this segmentation would happen by instrumenting the source code. In contrast, in closed-source applications, they would inspect

[1] NVIDIA Management Library (NVML): https://developer.nvidia.com/nvidia-management-library-nvml, last accessed 2023-11-23; AMD Display Library (ADL): https://gpuopen.com/adl/, last accessed 2023-11-23.

characteristic features of the curve, namely a sharp rise of the power curve at the beginning of each frame after the buffer swap that manifests as a low-power plateau. For mobile devices, Heinemann et al. (2017) investigated how different parameters of a volume raycaster and the rendering APIs used influenced the device's power consumption.

Overall, research on the energy consumption of graphics and visualization algorithms is limited. In contrast, most of the research on the power consumption of GPUs has been done in the HPC and machine learning fields. In this context, large numbers of GPUs in clusters do not only scale up the computing power but also their energy consumption, making the energy efficiency of algorithms an exciting area of research. Mittal and Vetter (2014), as well as Bridges et al. (2016), surveyed power efficiency in the GPGPU context, while the former also include suggestions for improving software in this respect and the latter venture into the area of modeling (Ma et al. 2009; Hong and Kim 2010; Nagasaka et al. 2010; Luo and Suda 2011) and simulating (Sheaffer et al. 2004; Ramani et al. 2007; Lim et al. 2014) the power consumption of GPU clusters. Other optimizations to reduce the power draws include dynamic voltage and frequency scaling (DVFS), which adjusts the GPU clocks and voltage while trying to minimize the performance impact (Abe et al. 2012). Mei et al. (2013) reported savings of up to 20 % using this technique. The models investigated by Bridges et al. derive their data mostly from low-level hardware counters, e.g., by accumulating the number of shader invocations or texture lookups instead of measuring the actual power draw using specialized sensors. While the authors see a strong correlation between these numbers and energy consumption, they acknowledge that the rising complexity of modern GPUs makes this kind of modeling increasingly harder.

In recent years, computer vision (Qasaimeh et al. 2019) and artificial intelligence (Jahanshahi et al. 2020; Hu et al. 2021) came into the focus of researchers interested in the power draw of GPUs. The goal for machine learning workloads is to fully saturate the GPUs because it best balances performance and energy consumption. At the very end of the visualization pipeline, there is work on reducing the display's power consumption by optimizing color mappings (Chuang et al. 2009).

For obtaining power readings, many works rely on software sensors like the NVML, which provides relatively accurate numbers from onboard sensors starting with the *Kepler* architecture. Burtscher et al. (2014) developed an approach to compute a value they consider more accurate from these readings. However, they base their technique solely on theoretical power profiles rather than measured ground truth. NVML power readings have also been combined with micro-benchmarks to derive energy consumption models. Arafa et al. (2020) measured kernels with only low-level instructions like addition and division for this purpose. Their comparison of the numbers sampled using NVML and the high-level, vendor-independent Performance Application Programming Interface (PAPI) (Weaver et al. 2012) with numbers from reference measurements are relevant for all kinds of GPU applications. For NVML, the authors found that sampling NVML in a dedicated thread yields the most accurate results. Their reference data came from a custom riser card with shunt resistors to measure the power drawn from the PEG slot and an oscilloscope with a voltage probe and a power clamp for the PCIe cables, similar to our setup. However, running micro-benchmarks enabled Arafa et al. (2020) to decide how long they measured and thus averaged the measurements over time, as Müller et al. (2022) did for visualization, removing the need to match the measurement to a specific frame. Collange et al. (2009) obtained their power readings for similar CUDA micro-benchmarks using only an oscilloscope because software sensors were not available then. In contrast, Fahad et al. (2019) used power meters between the wall socket and the power supply to obtain measurements for the whole machine. As the latter addressed heterogeneous compute nodes of a cluster, they also obtained power numbers for the CPU using RAPL (Running Average Power Limit) registers (Intel 2023b).

Specifically in HPC, where huge numbers of CPUs scale the energy consumption, RAPL is a widely used technique to monitor and optimize the energy consumption of simulation codes. Various studies were performed to evaluate the accuracy of RAPL for such purposes. Khan et al. (2018) found that the CPU generations following Intel's Sandy Bridge, which introduced the registers, dramatically improved in what RAPL reports compared to external measurements. Hackenberg et al. (2013) criticized RAPL for reporting energy instead of power and completely lacking timestamp information. They concluded that measuring AC power in the power distribution units of a cluster provided sufficiently accurate data for energy-based accounting of HPC jobs. Finally, Hähnel et al. (2012) used RAPL to determine the energy consumption of code, specifically the decoding of H.264 video streams—a goal similar to ours, albeit on the CPU instead of on the GPU. The results we present are the first holistic view of energy consumption of a whole visual computing system that compares various GPU (vendor libraries) and CPU (RAPL) sensors against a ground truth provided by an oscilloscope.

3 Overview

The goal of our setup is to measure high-frequency changes in the power draw of the components in a PC running scientific visualization applications—most notably of the graphics card, which is known to regulate voltage and power at very short intervals—and to compare the results with sensors having only a low sampling rate.

3.1 Measuring power

As all of the components of interest are on the low-voltage DC side of the power supply, power is $P\,[\mathrm{W}] = U\,[\mathrm{V}] \cdot I\,[\mathrm{A}]$. Therefore, we need to measure the voltage U and the current I for each power rail of interest. Previous work relied on deriving the latter from measuring the voltage drop across shunt resistors added into the power rails via off-the-shelf microcontrollers (Müller et al. 2022) or on a combination of shunt resistors and Hall-effect sensors (Ramsden 2006) on a custom printed circuit board (Johnsson et al. 2012; Johnsson and Akenine-Möller 2014).

Oscilloscopes are designed to measure voltage over time. Their advantage is that some of them can do so with a temporal resolution of up to several GHz, which is much higher than what the sensors mentioned above can achieve. While the sampling rate is typically adjustable, the memory of digital oscilloscopes is finite, wherefore an increase in rate decreases the time span that can be observed. Often, oscilloscopes can be connected to a computer, allowing for downloading the waveforms recorded on the instrument for automated analysis. Typical instruments come with two or four channels that can be captured simultaneously. The acquisition on all of them is started by a trigger, which can be defined on one of the channels under observation (for instance, voltage rising above a given threshold), on an external source, or after a certain timeout. Finally, users can also manually trigger the oscilloscope.

The input to each channel is provided by a probe. Voltage probes (see Fig. 1, top) measure voltage against a common ground of the instrument or between two probing points. To do so, they must be attached parallel to the circuit under investigation. As the oscilloscope only measures voltage, but we need to know voltage and current, we need a special kind of probe that converts current into voltage. Current clamps serve this purpose (see Fig. 1, bottom). They rely on the Hall effect (Ramsden 2006) and allow for non-invasive measurements as the probe clasps the conductor in a coil to measure current via the induced magnetic field.

3.2 Experimental setup

For our experiment, we replicated the setup by Müller et al. (2022) regarding used sensor types, adding RAPL sensors and oscilloscopes to the experiment. Like Müller et al., we inserted Tinkerforge Voltage/Current 2.0 bricklets into every relevant power rail in the system. Most importantly, that includes the PCIe/12VHPWR auxiliary power lines to the GPU and the power the GPU can draw from the PCI Express Graphics (PEG) slot. The probing points of an Adex Electronics PEX16IX riser card provide access to the latter. We subsequently refer to the Tinkerforge bricklets, the oscilloscopes, and the power meter at the wall socket as *hardware* sensors. While the current probes can be non-invasively clamped around power cables, voltage probes must contact the wires. One option would be using the probing point of the riser for the PEG slot and pressing the probes into the connectors in case of the PCIe auxiliary cables such that they touch the point where the cable and the plug are crimped. However, as we have already routed all power rails through

Fig. 1 Top: One of the voltage probes we use in our experiments. Bottom: One of the current clamps used in the experiments. The current probes are clasped around the power lines to measure current by utilizing the Hall effect

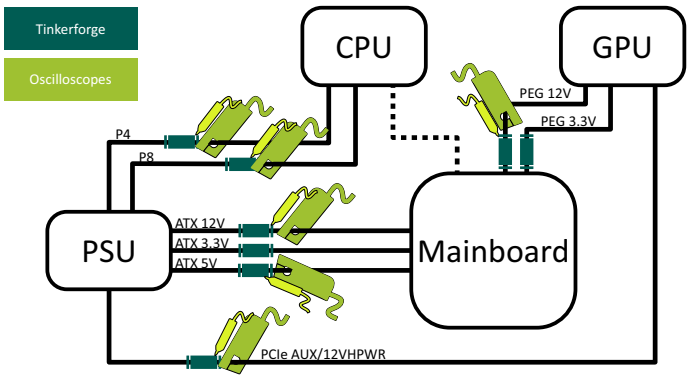


Fig. 2 Schematic of the power rails we probed during our experiment. The dark green shunt resistors indicate the power lines measured by Tinkerforge bricklets. The lines additionally measured by the oscilloscopes are marked with two probes for voltage and current

the Tinkerforge bricklets, where they are easily accessible, we probe voltage at these locations (see also Fig. 2).

Additionally, we include all *software* sensors used in the previous work, namely NVML and the AMD Display Library (ADL). These provide readings from the GPU board itself, albeit at a low rate. We have added a software sensor for the power draw of the CPU that uses the RAPL (Intel 2023b) Model-Specific Registers (MSRs). These registers can be read by means of the `rdmsr` compiler intrinsic, but only in kernel mode due to the possibility of side-channel attacks (Lipp et al. 2021). Windows 11 exposes the data via the Energy Meter Interface (EMI),[2] but previous versions require a device driver to access them. On Linux, they are exposed via device files, albeit there are discussions to remove access to them from non-privileged users.

3.3 Application under test

One of the goals of our experiment is to investigate under which conditions it is beneficial or even necessary to use an oscilloscope (or a set of oscilloscopes) to capture power consumption with an accuracy sufficient for subsequent reasoning about the energy efficiency of a visualization or rendering technique. The visualization framework MegaMol (Grottel et al. 2015) serves as the "application under test" in this experiment. We chose this framework as it provides various rendering algorithms for the same class of rendering primitives, including ones that fully utilize ray tracing (RT) cores in modern NVIDIA GPUs. This variety allows us to measure and compare waveforms from different algorithms for the same data set. Consequently, this experiment focuses on the power draw of the GPU.

The waveforms from the software sensors and the Tinkerforge bricklets are captured simultaneously. This process introduces a computational load and, therefore, also an additional power draw. Decoupling the measurement overhead from the inspected software is desirable. However, the measurement processes would have to be on a different computer, requiring communication and synchronization over the network. This would also not work for software sensors, which have to be sampled on the computer running the application. Therefore, we argue that the benefits of an external measurement driver do not outweigh the additional effort for time synchronization, especially when sampling at high rates, and because the accuracy of the synchronization is essential to retrieve meaningful intra-frame measurements. In our specific experiment, the application under test, MegaMol, is responsible for the setup and configuration of the measurement, as well as the retrieval and storage of the waveforms from all sensors. However, this is a choice and not a must, mainly motivated by the convenience of having everything set up and the results collected automatically from a single point. It would be equally possible to control the measurements externally except for synchronizing the high-frequency measurements using oscilloscopes with the frame borders. Although not impossible without access to the source code of the application under test, synchronization pulses are more easily and reliably integrated with the source.

[2] Microsoft Energy Meter Interface: https://learn.microsoft.com/en-us/windows-hardware/drivers/powermeter/energy-meter-interface, last accessed 2023-11-23.

4 Implementation

As mentioned before, the need for precise time synchronization led us to embed power measurements in the visualization software, in this case, MegaMol. The software needs to be extended to read different sensors and requires the integration of logic that conducts the measurement for a specific test configuration. A measurement consists of five basic steps:

1. Setup of the visualization pipeline.
2. Reading and applying sensor configurations.
3. Setup of measurement (buffer allocation, etc.).
4. Starting the measurement by broadcasting a trigger.
5. Collecting and storing the measured data.

The first step relies on MegaMol's existing functionality. We aim to design the measurement logic in such a way that it requires only minimal changes in the core logic of the application under test, so no adaptations are required in the setup of the visualization pipeline or in the algorithms that are part of it.

The remaining steps are the extended functionality described in this section. For the second step, we extend the scripting interface of MegaMol to enable the setup and configuration of the measurement. The computational logic responsible for the measurement is completely encapsulated and separated from the rest of MegaMol and can be removed entirely at compile time. The data collection from the software sensors and the Tinkerforge bricklets is done during the measurement. The collection from the other hardware sensors is done after the measurement. Especially the oscilloscopes produce significantly more data (several magnitudes in size), so postponing their data collection and storing all data from all sensors to disk after the measurement minimizes the impact of the power draw on the measurement. Otherwise, the visualization pipeline runs unimpeded except for frames marking at frame buffer swaps and partial call stack tracking with the frame profiler Tracy[3] to relate intra-frame results to high-level code.

4.1 Oscilloscope setup and configuration

We use one oscilloscope (or several chained together) to capture power draw on different power rails as accurately as possible to establish baselines for comparison (see Fig. 3). Modern digital oscilloscopes can be set up, configured, and steered remotely via a serial interface over a USB or Ethernet connection. In this experiment, we use oscilloscopes from Rohde & Schwarz (Models RTB2004 (Rohde & Schwarz GmbH & Co. KG 2022b) and RTA4004 (Rohde & Schwarz GmbH & Co. KG 2022a)). Their remote instrument control interface is based on the Virtual Instrument Software Architecture (VISA) API (VXIplug&play Systems Alliance 2022), a standardized communication protocol maintained by the IVI Foundation. The mental model of this interface is that of a user pushing the device's physical buttons. The virtual interaction is facilitated by verbose text-based commands that are exchanged over the serial interface. While the exchange protocol is standardized, most commands encapsulated in the messages are vendor or instrument-specific.

We implemented an abstraction layer on top of the VISA interface and integrated it into the Power Overwhelming library.[4] The abstraction layer hides the intricacies of the interface from the user and establishes a common API that allows for the integration of other vendors or instrument types in the future. A channel in the oscilloscope represents a physical probe attached to a circuit. The setup and configuration of an oscilloscope requires the following steps:

1. Setup of the horizontal scale for all channels.
2. Setup of the individual vertical scale for each channel.
3. Definition of the trigger condition.
4. Setup of the data acquisition.

A trigger initiates the acquisition of samples from each channel to measure, in our case, the power draw on each attached power rail.

[3] Tracy Code Repository: https://github.com/wolfpld/tracy, last accessed 2024-04-13.

[4] Power Overwhelming Code Repository: https://github.com/UniStuttgart-VISUS/power-overwhelming, last accessed 2023-12-12.

Fig. 3 Our setup with four oscilloscopes measuring in parallel with various other sensors, most importantly the Tinkerforge Voltage/Current 2.0 bricklets. The voltage probes of the oscilloscopes are attached to the bricklets. Some of the current clamps are visible in the foreground and on the right of the test bench

4.1.1 Horizontal/vertical scale

The horizontal scale is the time scale that applies to all channels. The range of this scale is the length of the measurement. The resolution of this scale limits the features in the signal we can reconstruct from the acquired samples (see the sampling theorem by Shannon (1949)). However, the resolution limits the maximal length of the time scale due to the limited sample storage in the oscilloscope and the connection to the PC not having sufficient bandwidth for streaming the samples. Therefore, we have to reason before the measurement about the proper resolution depending on the circuits under test.

The focus of this experiment is on the power draw of the GPU to evaluate rendering algorithms. Therefore, we need to factor in the characteristics of how the GPU is supplied with power. For instance, the specification of the new PCIe 12VHPWR auxiliary connector (Intel 2023a) allows up to 55 A sustained current over this connector. However, the specification also allows exceeding this current limit for a short period of up to 100 µs or any time period below that. Consequentially, we decided to acquire 1 000 000 samples per second, resulting in a 1 µs resolution, which allows for both detecting even small bursts and sampling over a reasonable time scale of 6 s on all available oscilloscope types (maximal 20 s on the RTB2004 at the selected resolution). In future implementations of the specification, the 12VHPWR connector will be replaced by the PCIe 12V-2x6 auxiliary connector. However, to the best of our knowledge, the same current limits apply to the replacement connector.

As mentioned, a digital channel in the oscilloscope represents a physical probe connected to a power rail measuring voltage or current. We have to set per channel whether a voltage or current probe is connected and the range and offset of the vertical scale. Since the analog-to-digital conversion resolution is limited (in the specific oscilloscopes to 10 bit), the vertical range should be chosen as tightly as possible without risking the clipping of samples at the range boundary, and the offset should reflect the expected value.

4.1.2 Trigger condition

The waveforms captured with all software and hardware sensors must be synchronized in time to allow a direct comparison. The acquisition of samples from the configured channels in an oscilloscope is started with a trigger. We use the timestamp of this trigger to synchronize all sensors. There are four ways to trigger an oscilloscope: It can be triggered manually, which is impractical in our use case. This “manual trigger” can also be sent over the software interface, which makes it a viable candidate. An event in the signal from a configured channel can trigger it. Finally, it can be triggered by an event in the signal of an external source (basically an additional channel). It is imperative for the measurement that the different sensors are synchronized as accurately as possible. We argue about the delay of each of these trigger types at the end of this section.

As mentioned, the trigger starts the acquisition of samples on the oscilloscope. To synchronize with the sensors sampled in the application under test, we have the same application send the trigger to the oscilloscope and create a timestamp simultaneously. Note that the trigger is the reference point for the oscilloscope to set up the waveform around it, but it is not necessarily the time point of the first acquired sample. It is beyond the scope of this paper to describe the timeline setup in the oscilloscope in detail; we refer the reader to the manuals.

Besides using the software interface, we consider sending a pulse signal to the external channel of the oscilloscope as a trigger because these are the only methods that allow associating a timestamp with the trigger. We investigated two possibilities for creating a signal from the computer running the application under test: A digital signal using the parallel port and an analog wave from the audio output (either square or sine wave). In both cases, the oscilloscope is set up to trigger if the voltage on the signal exceeds a threshold (2.5 V) at the rising edge.

The following presents the results of a small test to determine a method to trigger an oscilloscope with the least delay available to us. The parallel port can be accessed as a virtual file in an application on Windows and Linux operating systems. Subsequently, one can issue byte-wise writes to address one of the eight data pins of the parallel port. However, the port driver also checks the state of the attached device, failing a write attempt if the driver deems the device is not ready to receive further data. This error state can happen if the "Busy" (11) and the "Out-of-Paper" (12) pins are floating. Therefore, we attach both pins to the ground to ensure error-free writes on the parallel port.[5] Then, we can set and reset a specific bit with two writes to the parallel port to produce a square wave signal.

The audio trigger is implemented using the Windows Audio Session API (WASAPI), one of the audio APIs on Windows that achieves a relatively low latency. WASAPI audio clients can operate in two modes: In shared mode, multiple applications can stream to the same device, and the operating system acts as a mixer, producing the final audio buffers sent to the hardware. This feature adds latency to the output. An alternative exclusive mode gives an application sole control over the hardware. An application must continuously fill the audio ring buffer in exclusive mode as the device reports that a previous section has been played. We implemented both modes for comparison. For that, we pre-compute a buffer with a sine wave or a square wave at the required sampling rate of the device. Whenever the application requires a trigger, we copy this wave into the next free buffer instead of marking the buffer as silent, which can be achieved without copying data. Therefore, the audio trigger's latency is at least the length of the buffer that has already been scheduled for playback when the trigger signal is requested, which is hardware-dependent as the minimal size of an audio buffer is hardware-dependent.

Table 1 shows the results of the test. This test does not measure the round trip time but the relative delay between the two trigger methods. Method A is used to trigger the oscilloscope, and method B is attached to one input channel so that we can measure the distance between the pulse of method B and the trigger position. The parallel port is the trigger method with the least delay. A signal sent over the parallel port reaches the oscilloscope around 6 ms earlier than the audio signal and still around 1.5 ms earlier than the software trigger (see Fig. 4).

We configure all oscilloscopes to accept a trigger signal at the external channel to chain several oscilloscopes together to measure several power rails simultaneously. Additionally, all oscilloscopes are configured to send a pulse over their auxiliary port when they detect a trigger. This feature allows us to send the trigger signal only to a single oscilloscope while the trigger will be relayed over the auxiliary ports through the chain.

4.2 Software/hardware sensors

The other sensors are the software sensors (NVML, ADL, and RAPL), the Tinkerforge bricklets, and the wall socket power meter Rohde & Schwarz HMC8015. The latter is addressed over the same VISA software interface as the oscilloscope. It samples at a constant rate of 100 ms the total power consumption of our test computer running the application under test at the wall socket. The results are collected as a comma-separated table in internal storage and are downloaded by the test application after completion of the measurement. In contrast, the samples of the software sensors and from the Tinkerforge bricklets have to be collected during the measurement by the application under test.

[5] Native Instruments LabVIEW Programming Reference Manual: https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/functions/configuring-serial-and-parallel-ports-with-visa.html, last accessed 2023-12-11.

Table 1 The trigger delays between the trigger methods

	Parallel port	Audio port	Software
Parallel port	0	6.6 (36.5)	1.5
Audio port	−5.9 (−36.9)	0	−5.7 (−36.1)
Software	−1.6	5.1 (36.8)	0

The leftmost column states which method was executed first in the application code and, at the same time, is used as a trigger in the oscilloscope to set zero in the timeline. Timings are in milliseconds. The time in parentheses corresponds to shared mode on the audio device. The trigger signal is a square wave where applicable

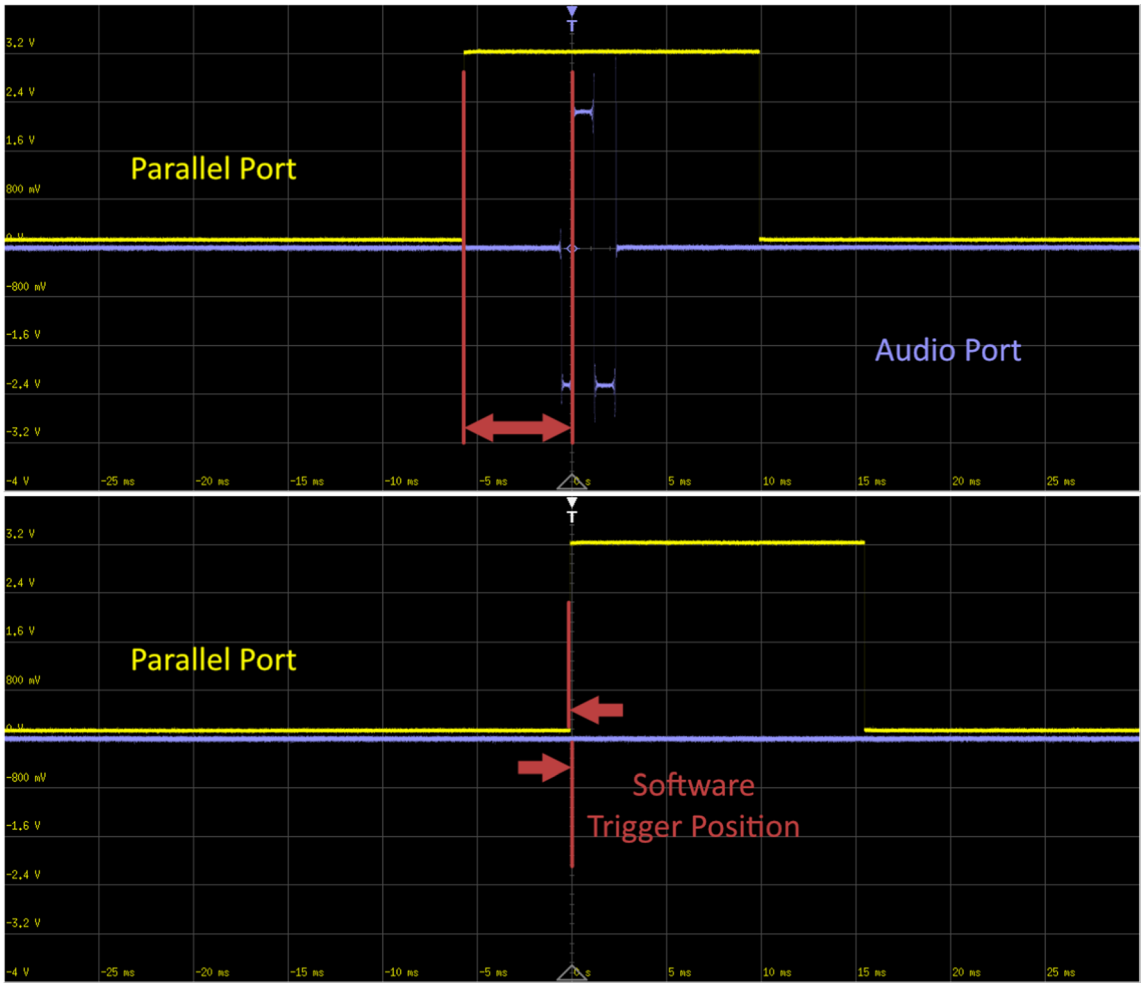


Fig. 4 Top: Delay between audio port with exclusive access to the audio device and the parallel port. Bottom: Delay between the software trigger and the parallel port

While the software interface for the NVML and ADL sensors was already available in the Power Overwhelming library by Müller et al. (2022), we implemented a kernel-mode driver to access the RAPL-related registers on Windows 10 and integrated it into the library.

The Tinkerforge bricklets offer an API to receive measurement values via callbacks asynchronously. Müller et al. (2022) already provided the interface with the Power Overwhelming library. However, we use a modified bricklet firmware to avoid shortcomings we identified to reduce measurement bias. The bricklet uses the INA226 sensor chip to measure voltage and current and a microcontroller for communication. By default, the firmware just stores a value presented by the INA226 chip. However, the communication part uses an independent timer for sending values at a configured sampling rate. As there is no synchronization between these two mechanisms, it is unknown how long the measured value is stored before it is sent to the

host. We modified the firmware to send a value directly whenever the INA226 presents a new measurement without delay.

Furthermore, Müller et al. (2022) assigned a timestamp to each sample based on the time the host PC received the value. In contrast, our modified firmware adds a timestamp to each measured value directly in the firmware of the bricklet to improve the time accuracy further. While this avoids artifacts from network latency, it generates an additional requirement to synchronize clocks between bricklet and host PC. This synchronization is especially challenging because the bricklets use internal clocks to measure time, which are affected by clock drift relative to the host PC clock. The drift is regularly reevaluated and compensated by the interface library.

All sensors captured on the host PC are associated with a dedicated buffer that stores the samples and their timestamps. These buffers are pre-allocated because it is known a priori how long a measurement will take and what sampling resolution will be used by each sensor. This pre-allocation avoids costly reallocations (regarding runtime and energy). The buffers are not large and thus do not interfere excessively with the running application. Suppose we measure a segment of 6 s. Then, the Tinkerforge bricklet with a 10 ms resolution is expected to produce 600 samples. A sample consists of a *float* value representing power and a 64-bit *integer* representing a timestamp.

Where available, we use an asynchronous sampling interface to register a callback, pushing the samples into the respective buffer. For sensors that require us to pull the samples (such as NVML or ADL), we create our own thread that asynchronously pulls samples according to the fixed sampling rate. The sensors are grouped according to their sampling rate such that we have a separate thread for each sampling rate instead of a thread for each distinct sensor. Note that the actual sampling rate in the results also depends on the behavior of the operating system's scheduler.

4.3 MegaMol integration

In our setup, the responsibility of the application under test, MegaMol, is to configure the sensors, start the measurement with a trigger signal, and finally collect the measurement data from all sensors. MegaMol is typically configured via its integrated Lua interface (Gralka et al. 2019). It is used to set up the components of a visualization pipeline, their parameterization, and how these components are interconnected. We exposed the parts of the Power Overwhelming library responsible for configuring sensors and measurements over the same Lua interface. This allows a user to embed the configuration of a measurement in the same file responsible for setting up and configuring the application under test. We exposed additional commands in the interface that control the start and end of a measurement, as well as the output of the collected data. These commands allow for defining an entire measurement sequence, such as a parameter sweep, in which a visualization pipeline is continuously measured, reconfigured, and measured again in a single script.

While the setup and the configuration of the software sensors and the Tinkerforge bricklets finish basically immediately, the commands sent to the oscilloscope take time to execute. The hardware requires time to configure itself; sometimes, hardware features such as relays and switches are set and have to settle. So, the application has to delay the measurement until the oscilloscopes are fully ready. The oscilloscopes in this experiment signal their readiness by setting a specific bit in a status register that we continuously pull in a separate thread. After that, the actual measurement is started.

Once the measurement is finished, we collect the acquired samples from external hardware sensors, i.e., the oscilloscopes and the wall socket power meter. For the software sensors and Tinkerforge bricklets, per-sample timestamps in the time reference of the computer running the application have already been created during buffering. The time reference is *FILETIME*,[6] represented as a 64-bit integer with a 100 ns base resolution. The wall socket power meter provides a UTC timestamp for each sample that we can directly convert into FILETIME. The samples from the oscilloscopes come without per-sample timestamps but with the start and end time of the measured timeline relative to the reference point that we configured to be the received trigger signal. Therefore, simply adding the trigger timestamp yields a timeline synchronized with the other sensors (except for the signal delay between the computer and oscilloscopes).

All acquired samples are written to disk as *Parquet* files.[7] The *Parquet* library is part of Apache's *Arrow* package. It is a lightweight representation of tabular data that natively supports lossless compression. We embed metadata in each *Parquet* file containing the sensor configurations, computer properties, etc., for

[6] FILETIME Reference: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime, last accessed 2024-04-13.

[7] Apache Parquet Documentation: https://parquet.apache.org/docs/, last accessed 2023-12-12.

Algorithm 1 Simplified measurement sequence

```
C ← ReadConfigs();
O ← FindScopes();
A ← ∅;
foreach o ∈ O do
    SyncClock(o);
    Reset(o);
    ApplyConfig(o, C);
    A{o} ← StartAcquisition(o);
end
while WaitingOnTrigger(O) do
    Sleep(100 ms);
end
foreach o ∈ O do
    D ← FetchData(o);
    T ← GenerateTimeline(D, A{o});
    Write(D, T);
end
```

reproducibility. The Python *pandas* package supports the direct conversion of a *Parquet* file into a data frame, simplifying analysis.

Algorithm 1 illustrates our measurement sequence implemented in MegaMol. While we utilize existing facilities in MegaMol, such as scripting, for ease of use, nothing related to the actual measurement sequence depends on MegaMol itself. On the contrary, the same measurement sequence could be integrated into any other visualization software using our additions to the standalone Power Overwhelming library available on GitHub.

5 Results

The main goal of our experiment is twofold: First, we want to compare the aggregated estimate of energy consumption from all sensors on the low-voltage side of the power supply between each other and with the total number measured at the wall socket. This should allow us to disregard the load-dependent transformation loss at the transition between AC and the DC rails used in computers. Second, we want to compare the patterns that each type of sensor can detect. Here, we focus on the ones measuring the GPU, which is the part of the system mainly affected by scientific visualization applications. Accordingly, we are interested in intra-frame information that can help us optimize parts of a visualization technique with regard to power consumption. The data corresponding to the presented results can be found in (Gralka et al. 2024).

5.1 Hardware

We tested our setup on a PC equipped with an AMD Ryzen 9 5900X CPU and 64 GB DDR4 RAM and varied only the GPU for testing different vendors and generations (see Table 2). The system was running Windows 10 (build 19041). The oscilloscopes used were three Rohde & Schwarz RTB2004 and one RTA4004, equipped with RT-ZP03 voltage probes (PEG 12 V, EPS P4, EPS P8), an RT-ZPR20 voltage probe (PCIe 12 V), RT-ZC03 current clamps (EPS P4, EPS P8, ATX 12 V, ATX 5 V), and a Chauvin-Arnoux E27 current clamp (PEG 12 V). For the auxiliary PCIe connectors and the 12VHPWR connector, we used an RT-ZC10B to account for the current exceeding the limit of the RT-ZC03 clamp. Additionally, we inserted Tinkerforge Voltage/Current 2.0 bricklets in the following power rails: ATX 3.3 V, ATX 5 V, ATX 12 V, PEG 3.3 V, PEG 12 V, PCIe 12 V, EPS P4, and EPS P8 (see also Fig. 2). The external power meter measuring the power draw between the system and the wall socket was a Rohde & Schwarz HMC8015.

5.2 Software

Our application case is the rendering of spherical glyphs, commonly used for visualizing results of molecular dynamics or galaxy formation simulations. There are different ways how this can be implemented to achieve interactive frame rates for hundreds of millions of particles. We selected three renderers, two based on OpenGL and one based on OptiX (Parker et al. 2010). The OpenGL renderers use a raycasting technique utilizing the rasterizer to render implicit spheres via billboards, in line with Gumhold (2003). Both use Shader Storage Buffer Objects (SSBOs) to store the sphere data on the GPU. The difference between the two is that one fits the entire data statically in VRAM (labeled as *static* in the plots later in the paper), and the other uses SSBOs to continuously stream the data to the GPU in chunks of 32 MB (labeled as *streaming*). The *OptiX* implementation is a ray tracer utilizing an accelerated bounding volume hierarchy to render the entire data set at once. It comprises two stages: First, it renders the data into internal data structures. Then, it blits the color and depth buffers into an OpenGL framebuffer for presentation.

5.3 Procedure

Our measurements are performed over a time span of 6 s, during which the software repeatedly renders the same frame. We consider this time span long enough to allow averaging over external influence, for instance, from short-term background tasks of the operating system, but at the same time short enough to avoid thermal throttling of the GPU. We tracked the throttling states with external tools in separate measurement runs to ensure this assumption holds.

The oscilloscopes are configured to acquire samples at a rate of one million samples per second. After receiving a trigger signal from the software, they start their acquisition, which is sent via the parallel port. The external power meter is instructed to record data to its internal memory every 100 ms. The data from these externally recorded sensors are downloaded from the instruments after the 6 s test run has been completed. All other sensors are sampled in-process while rendering. The Tinkerforge API spawns its own communication thread. We instructed it to asynchronously deliver a sample every 10 ms as higher sampling rates exhibited an increasing number of lost samples. The ADL and NVML sensors are sampled every 10 ms and RAPL every 1 ms. The sampling threads are managed by the Power Overwhelming library that fetches the sensor data according to the selected sampling rate.

The first step to preparing the collected data for analysis is to determine the timestamps of the first and last fully captured frame and the number of frames between these timestamps. While one data pin of the parallel port is used to trigger the acquisition of samples on the oscilloscope, our setup uses another data pin to send a pulse at every buffer swap in the application under test. The waveform of this "frame signal" is also collected from the oscilloscope and used to find the necessary frame boundaries on the timeline. The timestamps of the first and the last full frame are used to align all collected waveforms in a common timeline.

Since the other sensors capture data with a significantly lower resolution than the oscilloscopes, the start and end timestamps will almost always lie between two samples. So, to align the other sensors' waveforms with the waveforms from the oscilloscopes, we resample each signal on the timeline of the oscilloscopes using piece-wise linear interpolation. The Tinkerforge bricklets have a relatively low current limit, so we sometimes have to use several bricklets to sample a single power rail. One such example is the 12VHPWR connector that is wired through six bricklets. Resampling enables us to combine the results of these bricklets with a simple discrete addition. Once all waveforms are aligned, we integrate the signals with the trapezoid rule between the start and end timestamp and get the energy transported on each powerline in Watt-seconds Ws or Joule J.

Table 2 Key specifications of the GPUs used in our benchmarks

Model	Architecture	VRAM [GB]	Shading units	FP32 peak [TFlops]	TDP [W]
NVIDIA GeForce RTX 4090	Ada Lovelace	24	16,384	82.58	450
NVIDIA GeForce RTX 3090 Ti	Ampere	24	10,752	40.00	450
AMD Radeon RX 7900 XTX	RDNA 3	24	6144	61.42	355
AMD Radeon RX 7900 XT	RDNA 3	20	5376	51.48	315
AMD Radeon RX 6900 XT	RDNA 2	16	5120	23.04	300
Intel Arc A770	Alchemist	16	4096	19.66	225

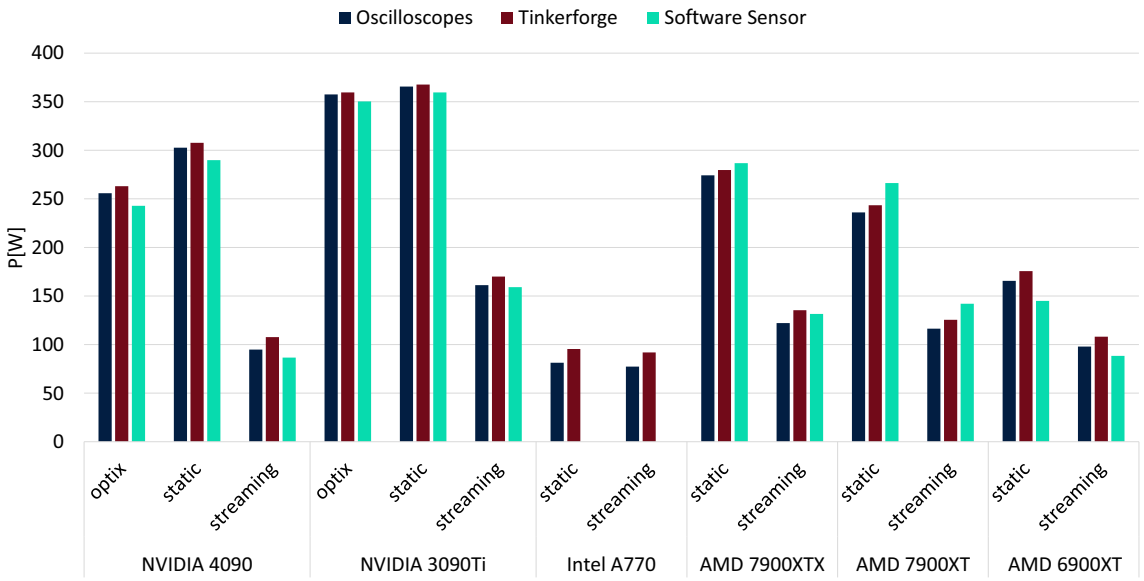


Fig. 5 Aggregated power consumption in Watt of the GPU between the first and last full frame recorded in the waveforms (results normalized by timeline length). The software sensor is either NVML for NVIDIA GPUs or ADL for AMD GPUs. We do not capture a software sensor for the Intel GPU

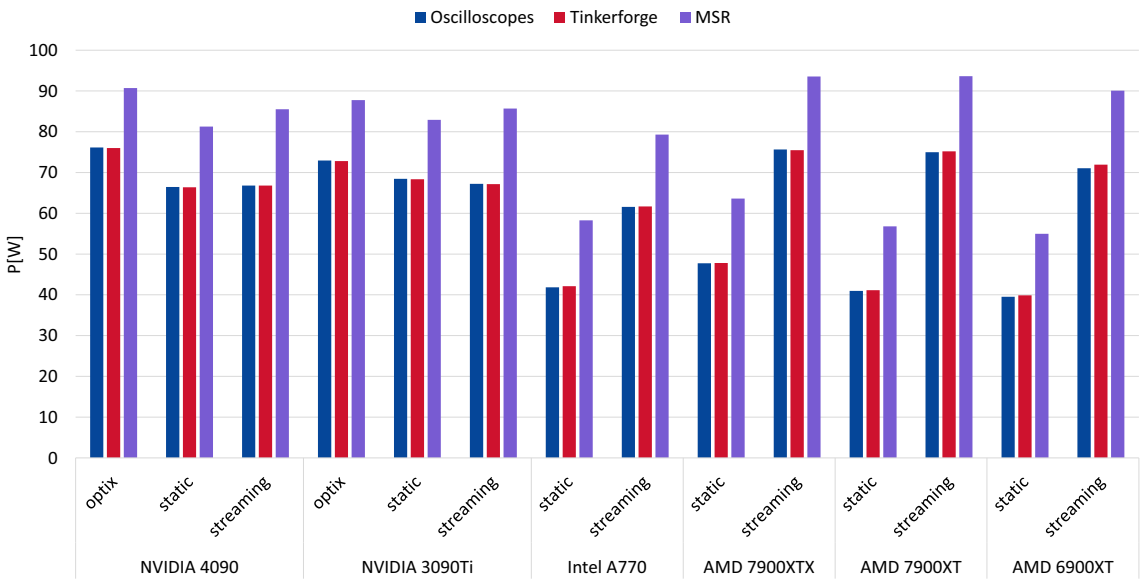


Fig. 6 Aggregated power consumption in Watt of the CPU between the first and last full frame recorded in the waveforms (results normalized by timeline length)

5.4 Aggregated power draw

First, we look into the aggregated results separated between GPU (see Fig. 5), CPU (see Fig. 6), and total power draw (see Fig. 7). We define the GPU power draw as the sum of the 12 V power draw from the PEG slot and the auxiliary PCIe 12 V or 12VHPWR connectors. The GPU results show a significant disagreement between the different sensor types. Tinkerforge sensors generally report an overall higher power consumption than the oscilloscopes. The NVML software sensor for NVIDIA GPUs always reports a lower power consumption than all other sensors. The relative results for ADL—the software sensor for AMD GPUs—differ between GPU generations. While ADL underestimates the power consumption of a 6900 XT compared to the other sensors, it overestimates significantly for the newer 7900 XT and 7900 XTX.

On our test system, we access the RAPL values through MSRs and report the energy consumption of the whole CPU package. The overview in Fig. 6 shows that the results of the Tinkerforge sensors and the oscilloscopes roughly agree. For these sensors, the CPU power draw corresponds to the sum of EPS P4 and EPS P8. However, the MSRs in the 5900X consistently overestimate the power draw compared to the other sensors.

Regarding the total power draw aggregated in Fig. 7, the Tinkerforge bricklets measure all power rails except those connected to the hard drives in our test system, whereas the oscilloscopes are additionally missing PEG 3.3 V and ATX 3.3 V, which might supply, for instance, the RAM. Unlike the other sensors, the HMC instrument measures the power draw at the wall socket and not on the DC side of the PSU. So, we expect the HMC to report a higher total power draw that includes the transformation loss in the PSU. The HMC instrument includes an internal integrator we enabled to verify our sampling results. The output of the internal integrator confirmed our integration of the samples. The results reported by the other sensors are lower but in a similar range relative to the HMC instrument, such that the difference can be explained by the missing power rails and the transformation loss.

Overall, the aggregated results from the hardware sensors agree within a reasonable margin. However, the quality of the software sensors depends on the vendor and even the generation of a product. To investigate this further, we compare the aggregated measurements from the hardware and software sensors to the ground truth provided by the oscilloscopes. Figure 8 (for the GPU) and Fig. 9 (for the CPU) show the relative difference aggregated over all GPUs and rendering methods. The Tinkerforge bricklets for NVIDIA GPUs and the NVML sensor have the overall smallest variance and the lowest difference to the oscilloscope results, as can be seen in Fig. 8. We observe the largest variance in the streaming case, most likely due to the high-frequency patterns originating from the multi-pass rendering scheme. Tinkerforge has a smaller difference and variance compared to ADL for AMD GPUs. For the ADL sensor, we observe both a larger variance and an overall higher difference to the ground truth. On older AMD GPUs, the ADL sensor underestimates; on newer AMD GPUs, it overestimates the power draw. In comparison, the NVML sensor produces reasonable aggregated results in both absolute and relative terms compared to the other software sensors, although it has the worst temporal resolution.

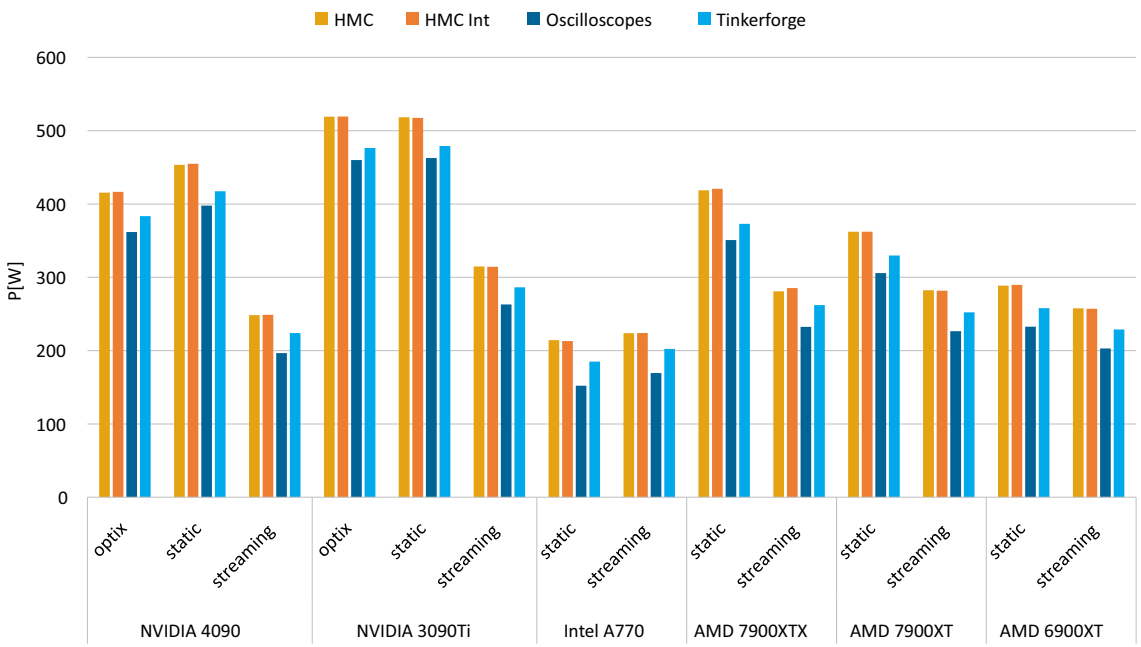


Fig. 7 Aggregated total power consumption in Watt of the test system between the first and last full frame recorded in the waveforms (results normalized by timeline length). Note that HMC measures at the wall socket, whereas the oscilloscopes and the Tinkerforge bricklets measure at the DC side of the PSU. Also note that the series "HMC" refers to individual samples that we integrated ourselves, whereas "HMC Int" is derived from the internal integrator of the instrument. Also, neither Tinkerforge bricklets nor the oscilloscope measure all power rails

In the case of the CPU, depicted in Fig. 9, the variance of the Tinkerforge sensors compared to the ground truth is negligible. Regarding the MSR sensor on the AMD platform, we see similar effects to those on ADL in that the behavior of the sensor changes with CPU generations. On the older 5900X, the sensor overestimates the energy consumption, whereas it underestimates the energy consumption and shows a lower variance on the newer 7950X3D. Underestimating energy consumption aligns with the assumption that a CPU-internal sensor omits the conversion loss of external voltage transducers. Such conversion loss is, however, included in the results of the oscilloscopes and the Tinkerforge bricklets.

5.5 Detailed intra-frame results

The intra-frame results present three frames from the middle of the captured waveforms. The test cases are configured to process the same commands in the same order in every frame. Consequently, we expect the waveforms to be qualitatively the same. Any deviation from the repetitive pattern has a high chance of originating from external influence, such as the operating system's scheduler switching to other workloads. The oscilloscope allows the detection of such deviations. One example can be seen in the waveform recorded for ray tracing on the GeForce RTX 3090 Ti (see Fig. 10). We could find interferences at such regularity and distance on the time scale that this pattern originates most likely from the screen refresh of the attached display.

Other than that, the frame pattern is always qualitatively the same within the same GPU. It is separated into two distinct parts. The first, longer segment has a roughly constant and high power draw, indicating good GPU utilization during the ray tracing operation in OptiX (CUDA). The remainder of the frame shows a repeating pattern of an idle state followed by a short power spike and another idle state. This pattern could be explained by the renderer copying the rendering result to the window context provided by OpenGL because this particular implementation of a sphere ray tracer in OptiX registers an OpenGL texture as the rendering surface. To write to this texture, it must be mapped to the OptiX/CUDA context. Therefore, to copy the rendering result, the renderer unmaps the texture, which implies a synchronization point (the first idle state). Then, the texture content is rendered into the OpenGL framebuffer (a GPU operation causing the power spike).

Finally, the texture is mapped back to the OptiX/CUDA context to render the next frame. Again, this is an implicit synchronization point (the second idle state). The recorded CPU power consumption further supports this assumption. It spikes during the idle states of the GPU, possibly due to a spin lock in the driver waiting for the synchronization. However, this pattern on the GPU signal could also originate from the general framebuffer presentation or be caused by the context switch between CUDA and OpenGL. Note that this power draw pattern is not visible in any of the signals recorded by the MSR, Tinkerforge, or NVML sensors. Without an accurate event trace from the GPU, providing a confident assessment of the source for such patterns is impossible. Unfortunately, not all APIs allow capturing event traces with sufficient accuracy.

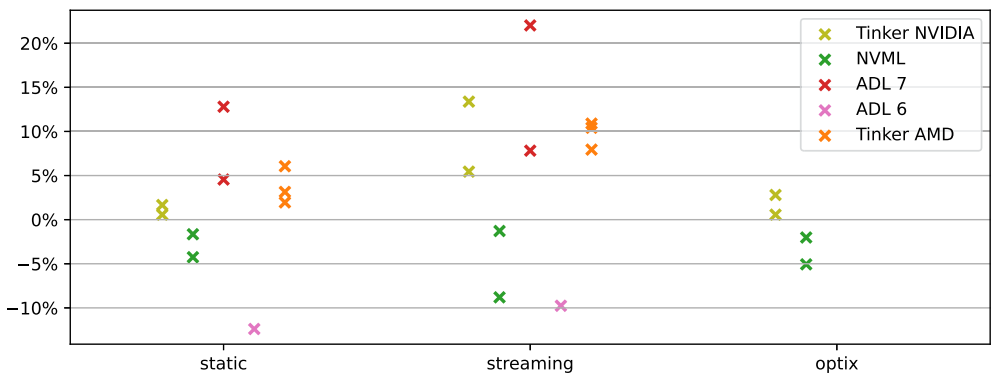


Fig. 8 The relative deviation of the sensors compared to the oscilloscope results regarding the GPU measurements. The color encodes the type of sensor. Considering the NVIDIA GPUs, the Tinkerforge bricklets have the smallest difference and variance. NVML behaves similarly regarding difference and variance. The older generation of ADL underestimates the power draw, whereas the newer generation overestimates it. The Tinkerforge bricklets report for the AMD GPUs results with a higher difference compared to the NVIDIA cases but with a significantly smaller variance in the streaming case. Additionally, they perform better than ADL

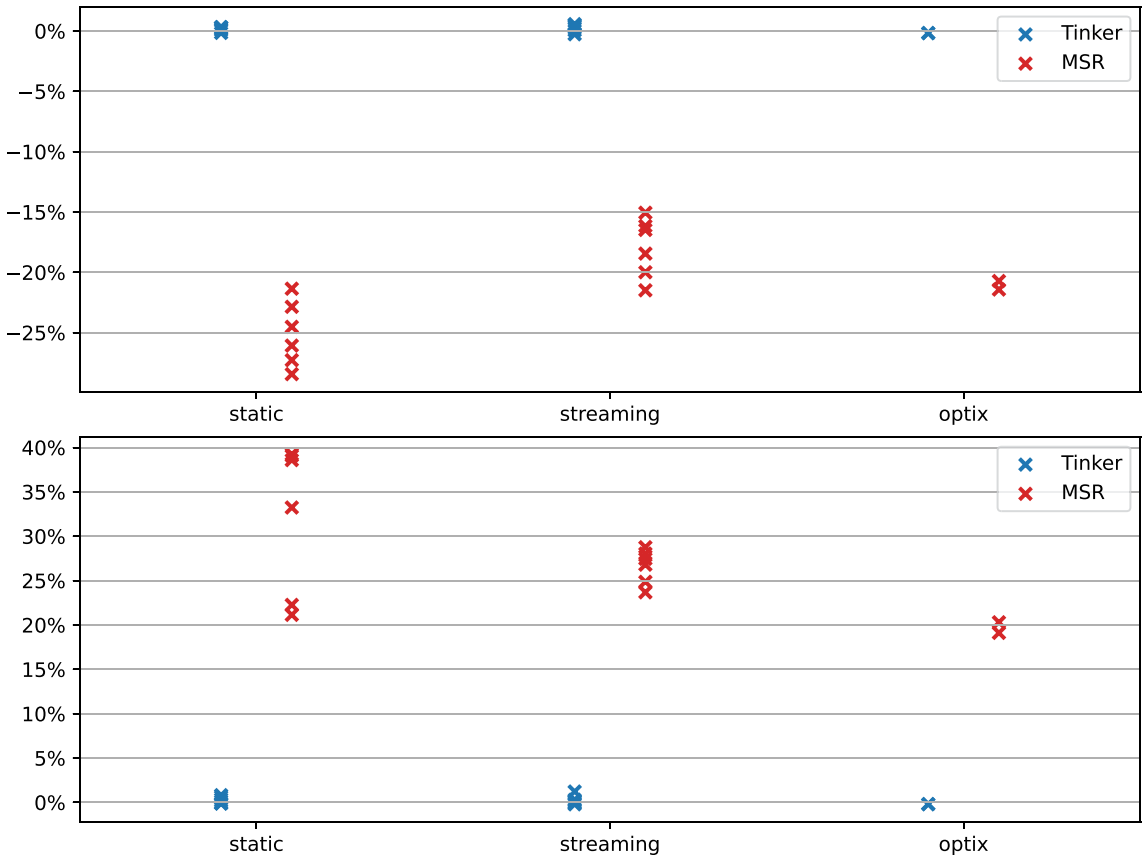


Fig. 9 The relative deviation of the sensors compared to the oscilloscope results regarding the CPU measurements. The color encodes the type of sensor. Top: Results obtained from an AMD Ryzen 9 7950X3D. Bottom: Results obtained from an AMD Ryzen 9 5900X. Like the ADL sensor, the MSR sensor shows different behaviors depending on CPU generation. The 7950X3D underestimates its energy consumption. The 5900X overestimates its energy consumption and has an overall higher variance

Additionally, this example shows that proper fencing of the API commands is mandatory to analyze the intra-frame power draw. Figure 11 depicts the effect of an additional *glFlush* and *glFinish* inserted before the framebuffer swap. It properly aligns the power draw curve with the frame markers, such that the power spike attributed to the texture copy between the OptiX/CUDA and OpenGL context is at the end of the frame, as expected. Without flushing, we see a rising edge toward the power plateau attributed to the actual ray tracing in the OptiX/CUDA context. As a consequence, all results were captured with the additional flush.

The Tinkerforge sensors have a configured sampling frequency of 10 ms. The ADL and NVML sensors have the same configured sampling frequency (although NVML produces a new sample roughly every 100 ms). For the following results, we post-processed the oscilloscope-related plots where necessary to reduce clutter and allow a direct comparison with the other sensors. First, the input waveforms were binned into a histogram with a bin width of roughly 10 ms. Then, the standard deviation was computed. Instead of the original input waveform, we plot the mean signal as a line and the standard deviation of the signal as a transparent band. The Tinkerforge results are qualitatively similar to the smoothed oscilloscope signals. This is especially visible in the streaming case for the AMD GPUs (see Fig. 13). In general, Tinkerforge is qualitatively in line with the oscilloscope signal but reports a higher overall power level. This is particularly noticeable in the streaming case for the 4090 (see Fig. 12). A drill down to the raw oscilloscope signal in Fig. 12 shows repeated idle states due to the data streaming method. It seems that the bricklets cannot detect these and, therefore, average a higher power draw than in reality.

Comparing the results of the 7900 XTX and the 6900 XT (see Fig. 13) shows a qualitative difference in the signals derived from the ADL samples. The ADL sensor reports a significantly smoothed signal in the case of the 6900 XT, showing no obvious qualitative similarity with the resampled reference signal collected from the oscilloscopes. In contrast, the ADL signal in the 7900 XTX (and 7900 XT) case is

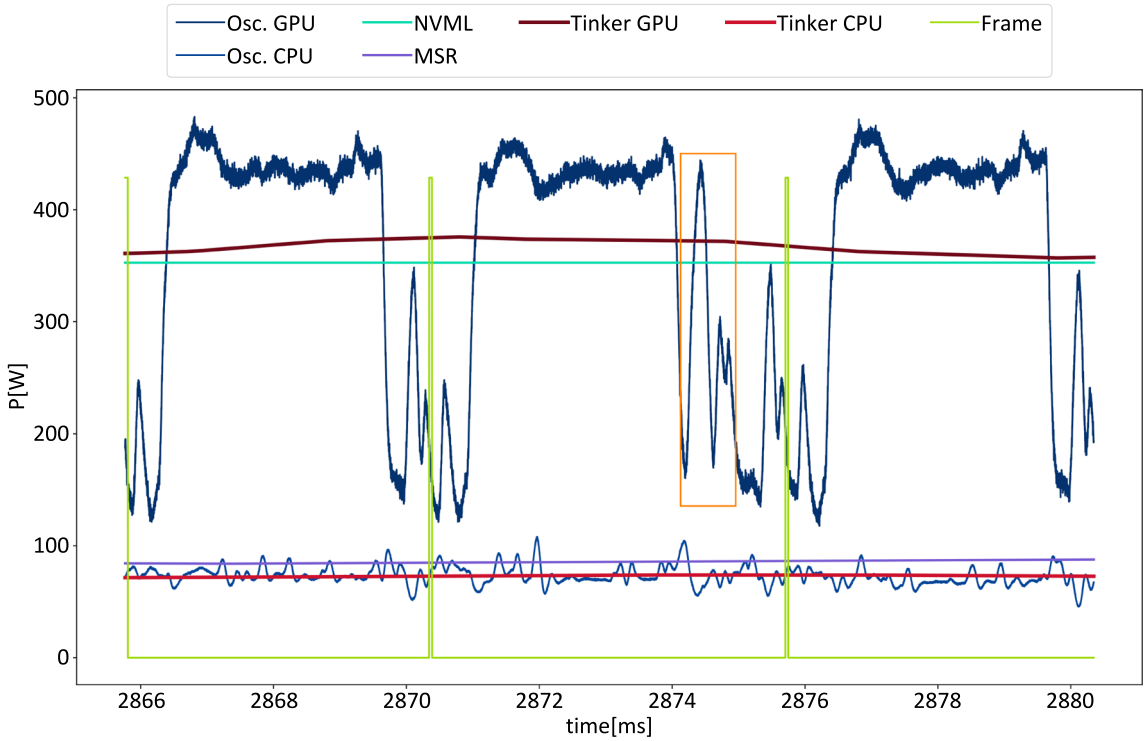


Fig. 10 Results for NVIDIA 3090 Ti sphere ray tracing. This detail view illustrates an additional interference most likely from the screen refresh that can be seen in the middle frame (orange box). Otherwise, the pattern of the two-stage rendering is visible. The plot contains the raw oscilloscope data (including the green line for the frame markers injected into the oscilloscope via the parallel port) and results from NVML, MSR, and Tinkerforge sensors

qualitatively similar to the oscilloscope reference. The difference is also reflected in the aggregated results in Fig. 5. This indicates that AMD has improved the accuracy of their internal energy sensors in the new GPU generation.

The NVML sensor provides no significant insight for intra-frame analysis, neither for older generation GPUs nor for current generation GPUs. It appears to average the input signal over a 1 s sliding window and adds a new sample every 100 ms to this averaging window. Previous work by Müller et al. (2022) explicitly tested this behavior.

The MSR sensor follows roughly the same pattern as the smoothed CPU signal from the oscilloscope but at a higher power level. The quality differs between generations, similar to the ADL case, as discussed in Sect. 5.4.

5.6 Stable power state

The previous sections show the results for the default behavior of GPUs. However, with modern rendering APIs, like Direct3D 12, GPUs can be configured to use a so-called *stable power state*. It effectively disables dynamic frequency scaling and typically fixes GPU clock frequencies at relatively low rates such that throttling limits are never reached.[8] We analyze the effect of this setting on aggregated and intra-frame results. Figure 14 shows that the aggregated power draw of the GPUs is reduced in stable power state across the board. For NVIDIA GPUs, the power draw measured by each sensor is around 30 % lower in stable power state than under normal operating conditions. The reduction is less consistent for AMD GPUs except for energy per frame. In the static rendering case, power consumption is reduced by more than 50 %, but in the streaming case, only by about 20 %, although all sensors agree on this difference. A drill-down into the streaming case (see Fig. 13) reveals that the variance (represented by the band) of the power draw in stable power state is significantly lower than in the standard operating state.

[8] Microsoft Stable Power State: https://learn.microsoft.com/en-us/windows/win32/api/d3d12/nf-d3d12-id3d12device-setstablepowerstate, last accessed 2024-02-23.

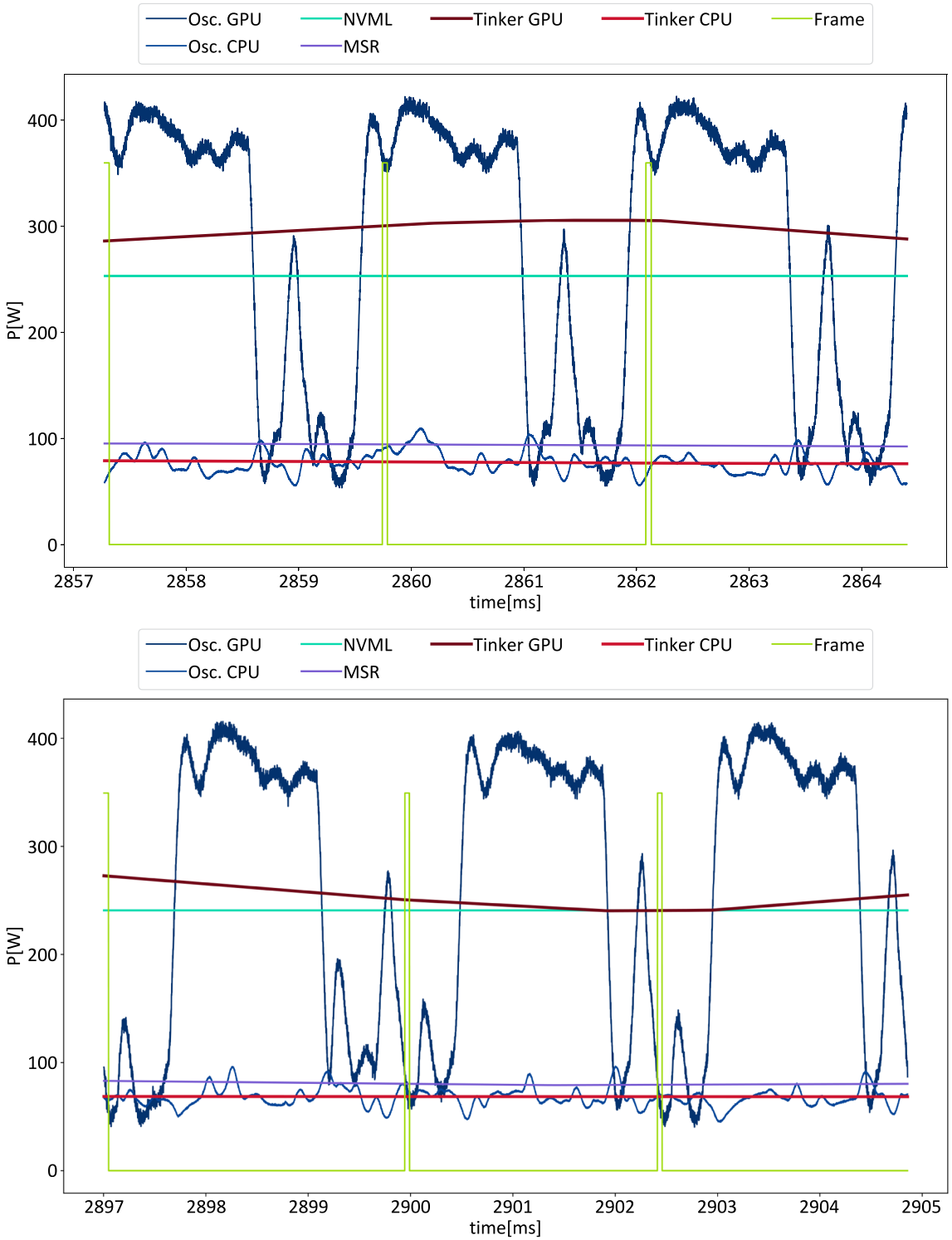


Fig. 11 Results for NVIDIA 4090 sphere ray tracing. Top: Without additional *glFlush* and *glFinish* before the framebuffer swap. Bottom: With additional *glFlush* and *glFinish*. The difference shows that proper fencing of rendering commands is necessary for an intra-frame analysis of the power draw. The plot contains the raw oscilloscope data (including the green line for the frame markers injected into the oscilloscope via the parallel port) and results from NVML, MSR, and Tinkerforge sensors

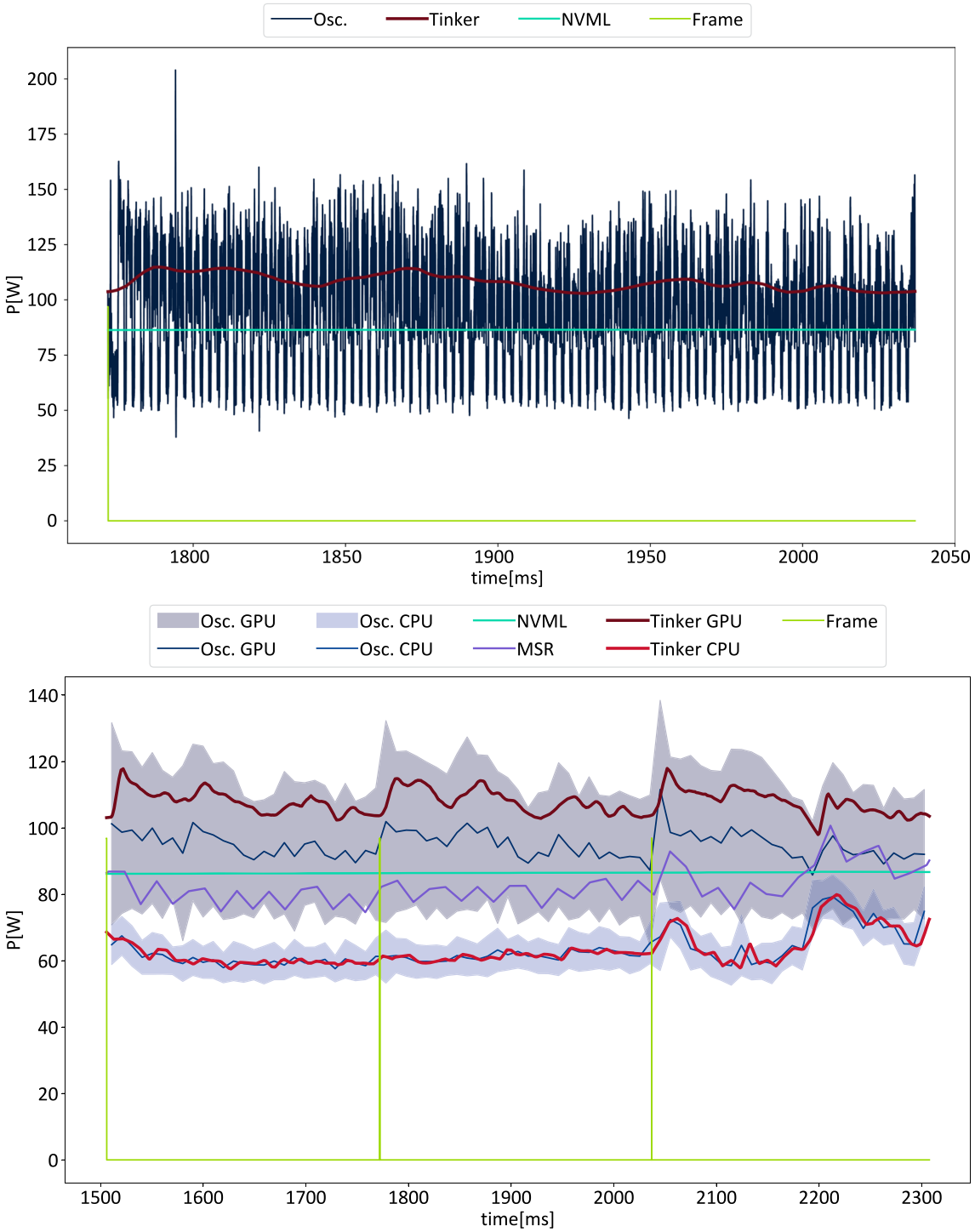
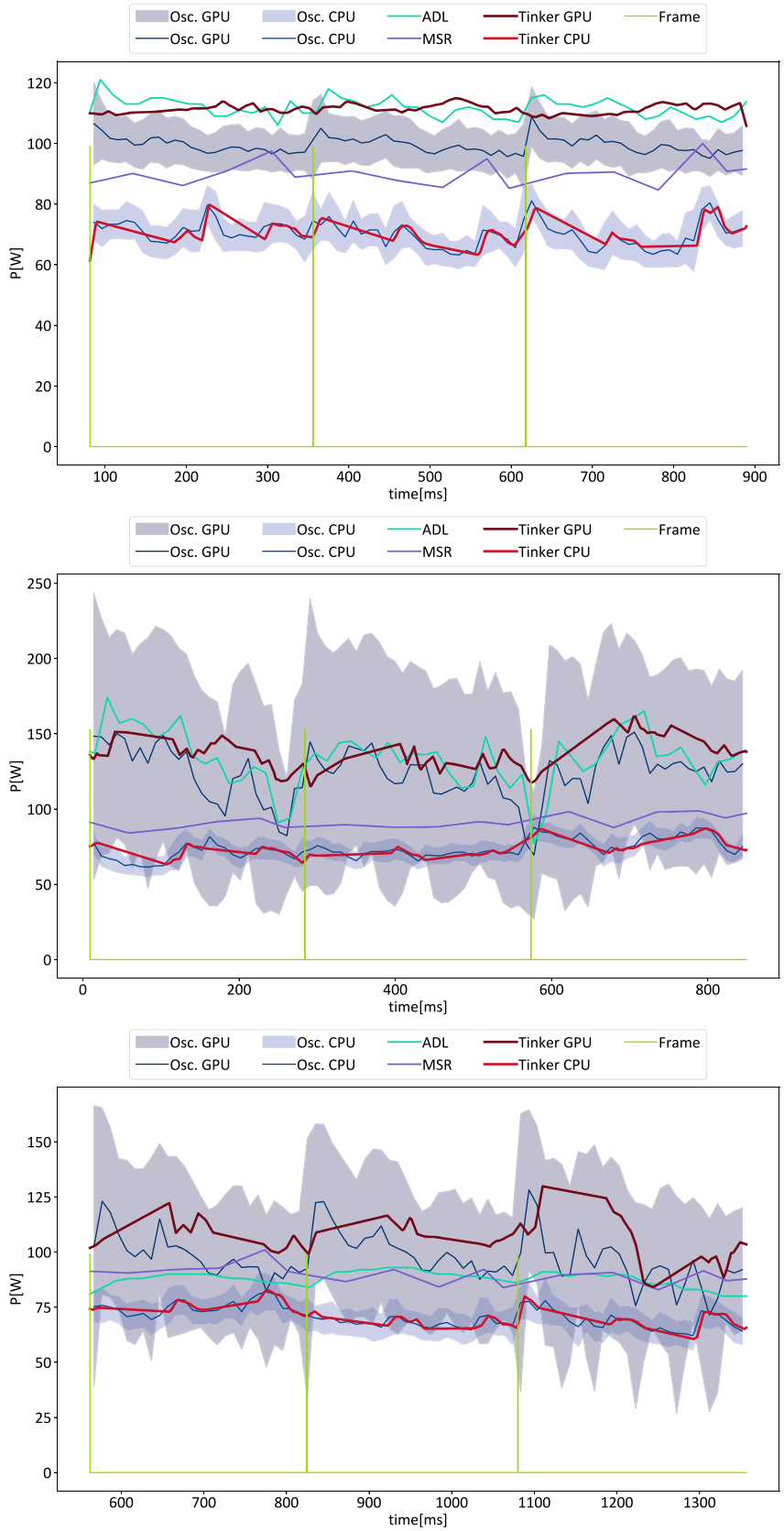


Fig. 12 Results for NVIDIA 4090 OpenGL sphere rendering with data streaming. Top: Detailed view of a single frame clearly showing the data streaming pattern. Bottom: Overview, comparing the quality of the different sensor types. These diagrams contain the results from NVML, MSR, and the Tinkerforge sensors. The top diagram shows the raw results from the oscilloscopes. The bottom diagram shows a subsampled signal from the oscilloscopes to reduce clutter. It is subsampled to 10 ms bins; the plot represents the mean, while the band shows the deviation


Osc. GPU
Osc. CPU
ADL
Tinker GPU
Frame
Osc. GPU
Osc. CPU
MSR
Tinker CPU
P[W]
time[ms]
Osc. GPU
Osc. CPU
ADL
Tinker GPU
Frame
Osc. GPU
Osc. CPU
MSR
Tinker CPU
P[W]
time[ms]
Osc. GPU
Osc. CPU
ADL
Tinker GPU
Frame
Osc. GPU
Osc. CPU
MSR
Tinker CPU
P[W]
time[ms]

◀**Fig. 13** Results for OpenGL sphere rendering with data streaming on AMD GPUs. Top: 7900 XTX with stable power setting. Middle: 7900 XTX without stable power setting. The stable power setting reduces the overall power draw and the variance indicated by the width of the band. Bottom: 6900 XT without stable power setting. While for the 7900 XTX, the ADL signal is qualitatively similar to the smoothed oscilloscope signal, the ADL signal for 6900 XT is not able to capture the high-frequency variations. The quality of the ADL sensor is improved in the newer generation of AMD GPUs. These diagrams contain the results from ADL, MSR, the Tinkerforge sensors, and a subsampled signal from the oscilloscopes. It is subsampled to 10 ms bins; the plot represents the mean, while the band shows the deviation

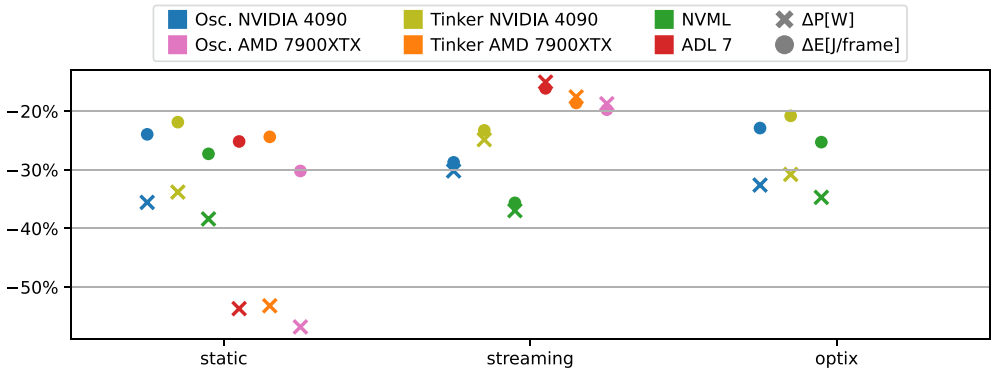


Fig. 14 The relative difference between the measurements performed in stable power state compared to the default operating state as the baseline. The color encodes the type of sensor, ΔP designates the difference in power, whereas ΔE is the difference in average energy consumed per frame

6 Conclusion

We presented a setup for investigating the power consumption of hardware-accelerated visualization algorithms using oscilloscopes. This setup enables researchers to investigate the power draw on individual power rails of the system at a temporal resolution sufficiently high to observe the behavior within a frame. In addition, it addresses how to synchronize the measurements with the application under investigation using a parallel port connector to identify the frame borders within the recorded waveforms. Rendering millions of particles as spheres as an application example, we performed an experiment comparing the high-resolution data obtained from oscilloscopes with lower-resolution sensors based on microcontrollers, with an external power meter, with software sensors provided by most GPU vendors to read out the power draw measured on their boards, and with RAPL MSRs available on all modern x64 CPUs.

The results led to two major observations: First, true frame introspection is only possible with the oscilloscopes. They reveal performance issues, such as the data streaming pattern or the two-stage rendering in the ray tracing case. Second, the oscilloscopes provide the ground truth for assessing the quality of software sensors. The NVML sensor has a too-low resolution to reveal intra-frame patterns over all tested GPU generations but is in good agreement with the ground truth considering aggregated results. In contrast, AMD has significantly improved the quality of the ADL sensor in the current generation so that it can provide a (smoothed) insight into intra-frame patterns. The behavior of the MSR sensor differs between CPU generations. Additionally, even within the same generation, vendor, and type, GPUs (and CPUs) show differences regarding performance and power draw (Sinha et al. 2022), diminishing the transferability of specific results to other hardware instances.

However, we must consider the significant effort involved with setting up the oscilloscopes and conducting tests with them—not only financially but also in terms of manual work and limited possibilities to automate tasks like repositioning the probes. Therefore, we suggest the following workflow to observe and analyze the energy consumption of visualization and rendering techniques. First, the oscilloscopes should be used to assess the usability and quality of available software sensors. Then, these software sensors can be used in a parameter sweep to provide an overview of the energy consumption of the tested algorithms or parameters. This overview can be used to identify interesting parameter sets for a drill-down with oscilloscopes.

Moreover, only knowing the ground truth allows for judging whether any of the more affordable—and also more easy-to-use—sensors provide sufficiently accurate data to answer an energy-related research

question without resorting to oscilloscopes. Additionally, only knowing the ground truth will eventually allow building models to predict detailed consumption behavior from what the software is doing or from low-resolution measurements. As for the HPC area, we are convinced that knowing what is going on in detail will pave the way for optimizing software for energy consumption. We think that energy consumption will become a metric to judge and compare novel algorithms in visualization research that is as important as, for instance, run time.

Acknowledgements The authors thank Tim Schmidt for implementing and testing an early concept of the MegaMol integration.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—Project-ID 251654672—TRR 161, and the Bundesministerium für Bildung und Forschung (BMBF) as part of the WindHPC project (project ID 16ME0608K).

Data and code availability The data for the presented results is available at https://doi.org/10.18419/darus-4256. The code for the Power Overwhelming library can be found at https://github.com/UniStuttgart-VISUS/power-overwhelming. The code for the MegaMol visualization framework can be downloaded from https://github.com/UniStuttgart-VISUS/megamol.

Declarations Not applicable

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Arafa Y, ElWazir A, ElKanishy A, Aly Y, Elsayed A, Badawy A-H, Chennupati G, Eidenbenz S, Santhi N (2020) Verified instruction-level energy consumption measurement for NVIDIA GPUs. In: Proceedings of the 17th ACM International Conference on Computing Frontiers, pp 60–70. https://doi.org/10.1145/3387902.3392613

Abe Y, Sasaki H, Peres M, Inoue K, Murakami K, Kato S (2012) Power and performance analysis of GPU-accelerated systems. In: Proceedings of the Workshop on Power-Aware Computing and Systems

Bridges RA, Imam N, Mintz TM (2016) Understanding GPU power: a survey of profiling, modeling, and simulation methods. ACM Comput Surv 49(3):1–27. https://doi.org/10.1145/2962131

Burtscher M, Zecena I, Zong Z (2014) Measuring GPU power with the K20 built-in sensor. In: Proceedings of the Workshop on General Purpose Processing Using GPUs, pp 28–36. https://doi.org/10.1145/2588768.2576783

Collange C, Defour D, Tisserand A (2009) Power consumption of GPUs from a software perspective. In: Proceedings of the 9th International Conference on Computational Science: Part 1, pp 914–923. https://doi.org/10.1007/978-3-642-01970-8_92

Chuang J, Weiskopf D, Möller T (2009) Energy aware color sets. Comput Graph Forum 28(2):203–211. https://doi.org/10.1111/j.1467-8659.2009.01359.x

Fahad M, Shahid A, Manumachu RR, Lastovetsky A (2019) A comparative study of methods for measurement of energy of computing. Energies 12(11):2204. https://doi.org/10.3390/en12112204

Gralka P, Becher M, Braun M, Frieß F, Müller C, Rau T, Schatz K, Schulz C, Krone M, Reina G, Ertl T (2019) MegaMol—a comprehensive prototyping framework for visualizations. Eur Phys J Spec Top 227(14):1817–1829. https://doi.org/10.1140/epjst/e2019-800167-5

Gralka P, Müller C, Heinemann M, Reina G, Weiskopf D, Ertl T (2024) Supplemental material for "Power overwhelming: the one with the oscilloscopes. https://doi.org/10.18419/darus-4256.

Grottel S, Krone M, Müller C, Reina G, Ertl T (2015) Megamol—a prototyping framework for particle-based visualization. IEEE Trans Vis Comput Graph 21(2):201–214. https://doi.org/10.1109/TVCG.2014.2350479

Gumhold S (2003) Splatting illuminated ellipsoids with depth correction. In: Proceedings of the Symposium on Vision, Modeling, and Visualization, pp 245–252

Heinemann M, Bruder V, Frey S, Ertl T (2017) Power efficiency of volume raycasting on mobile devices. In: EuroVis 2017 – Posters. https://doi.org/10.2312/eurp.20171166

Hähnel M, Döbel B, Völp M, Härtig H (2012) Measuring energy consumption for short code paths using RAPL. SIGMETRICS Perform Eval Rev 40(3):13–17. https://doi.org/10.1145/2425248.2425252

Hackenberg D, Ilsche T, Schöne R, Molka D, Schmidt M, Nagel WE (2013) Power measurement techniques on standard compute nodes: a quantitative comparison. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, pp 194–204. https://doi.org/10.1109/ISPASS.2013.6557170

Hong S, Kim H (2010) An integrated GPU power and performance model. In: Proceedings of the 37th Annual International Symposium on Computer Architecture, pp 280–289. https://doi.org/10.1145/1815961.1815998

Hu Q, Sun P, Yan S, Wen Y, Zhang T (2021) Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. https://doi.org/10.1145/3458817.3476223
Intel: ATX version 3 multi rail desktop platform power supply, Revision 2.1a edn. (2023)
Intel: Intel®64 and IA-32 architectures software developer's manual vol. 3B, Sept 2023 edn. (2023)
Johnsson B, Akenine-Möller T (2014) Measuring per-frame energy consumption of real-time graphics applications. J Comput Graph Tech 3(1):60–73
Johnsson B, Ganestam P, Doggett M, Akenine-Möller T (2012) Power efficiency for software algorithms running on graphics processors. In: Proceedings of the Symposium on High Performance Graphics, pp 67–75. https://doi.org/10.2312/EGGH/HPG12/067-075
Jahanshahi A, Sabzi HZ, Lau C, Wong D (2020) GPU-NEST: characterizing energy efficiency of multi-GPU inference servers. IEEE Comput Archit L 19(2):139–142. https://doi.org/10.1109/LCA.2020.3023723
Khan KN, Hirki M, Niemi T, Nurminen JK, Ou Z (2018) RAPL in action: experiences in using RAPL for power measurements. ACM Trans Model Perform Eval Comput Syst 3(2):1–26. https://doi.org/10.1145/3177754
Lipp M, Kogler A, Oswald D, Schwarz M, Easdon C, Canella C, Gruss D (2021) Platypus: software-based power side-channel attacks on x86. In: Proceedings of the IEEE Symposium on Security and Privacy, pp 355–371. https://doi.org/10.1109/SP40001.2021.00063
Lim J, Lakshminarayana NB, Kim H, Song W, Yalamanchili S, Sung W (2014) Power modeling for GPU architectures using McPAT. ACM Trans Des Autom Electron Syst 19(3):1–24. https://doi.org/10.1145/2611758
Luo C, Suda R (2011) A performance and energy consumption analytical model for GPU. In: Proceedings of the IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp 658–665. https://doi.org/10.1109/DASC.2011.117
Ma X, Dong M, Zhong L, Deng Z (2009) Statistical power consumption analysis and modeling for GPU-based computing. In: Proceedings of the Workshop on Power-Aware Computing and Systems
Müller C, Heinemann M, Weiskopf D, Ertl T (2022) Power overwhelming: quantifying the energy cost of visualisation. In: Proceedings of IEEE Evaluation and Beyond-Methodological Approaches for Visualization, pp 38–46. https://doi.org/10.1109/BELIV57783.2022.00009
Mittal S, Vetter JS (2014) A survey of methods for analyzing and improving GPU energy efficiency. ACM Comput Surv 47(2):1–23. https://doi.org/10.1145/2636342
Mei X, Yung LS, Zhao K, Chu X (2013) A measurement study of GPU DVFS on energy conservation. In: Proceedings of the Workshop on Power-Aware Computing and Systems. https://doi.org/10.1145/2525526.2525852
Nagasaka H, Maruyama N, Nukada A, Endo T, Matsuoka S (2010) Statistical power modeling of GPU kernels using performance counters. In: Proceeding of the International Conference on Green Computing, pp 115–122. https://doi.org/10.1109/GREENCOMP.2010.5598315
Parker SG, Bigler J, Dietrich A, Friedrich H, Hoberock J, Luebke D, McAllister D, McGuire M, Morley K, Robison A, Stich M (2010) OptiX: a general purpose ray tracing engine. ACM Trans Graph 29(4):1–13. https://doi.org/10.1145/1778765.1778803
Qasaimeh M, Denolf K, Lo J, Vissers K, Zambreno J, Jones PH (2019) Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. In: Proceeding of the IEEE International Conference on Embedded Software and Systems, pp 1–8. https://doi.org/10.1109/ICESS.2019.8782524
Ramsden E (2006) Hall-effect sensors. Newnes, Oxford
Ramani K, Ibrahim A, Shimizu D (2007) PowerRed: a flexible modeling framework for power efficiency exploration in GPUs. In: Proceedings of the Workshop on General Purpose Processing on GPUs
Rohde & Schwarz GmbH & Co. KG: R&S®RTA4000 Oscilloscope user manual, Version 09 edn. (2022)
Rohde & Schwarz GmbH & Co. KG: R&S®RTB2000 Oscilloscope user manual, Version 12 edn. (2022)
Sinha P, Guliani A, Jain R, Tran B, Sinclair MD, Venkataraman S (2022) Not all GPUs are created equal: characterizing variability in large-scale, accelerator-rich systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. https://doi.org/10.1109/sc41404.2022.00070
Shannon CE (1949) Communication in the presence of noise. Proc IRE 37(1):10–21. https://doi.org/10.1109/JRPROC.1949.232969
Sheaffer JW, Luebke D, Skadron K (2004) A flexible simulation framework for graphics architectures. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp 85–94. https://doi.org/10.1145/1058129.1058142
VXIplug&play Systems Alliance: VPP-4.3: The VISA Library, Revision 7.2 edn. (2022)
Weaver VM, Johnson M, Kasichayanula K, Ralph J, Luszczek P, Terpstra D, Moore S (2012) Measuring energy and power with PAPI. In: Proceedings of the 41st International Conference on Parallel Processing Workshops, pp 262–268. https://doi.org/10.1109/ICPPW.2012.39

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.