Energy Measurements of High Performance Computing Systems: From Instrumentation to Analysis

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Thomas Ilsche

Gutachter:
Prof. Dr. rer. nat. Wolfgang E. Nagel
Technische Universität Dresden

Prof. Dr. rer. nat. Martin Schulz
Technische Universität München

Tag der Einreichung:
3. März 2020

Tag der Verteidigung:
7. Juli 2020

Abstract

Energy efficiency is a major criterion for computing in general and High Performance Computing (HPC) in particular. When optimizing for energy efficiency, it is essential to measure the underlying metric: energy consumption. To fully leverage energy measurements, their quality needs to be well-understood. To that end, this thesis provides a rigorous evaluation of various energy measurement techniques. I demonstrate how the deliberate selection of instrumentation points, sensors, and analog processing schemes can enhance the temporal and spatial resolution while preserving a well-known accuracy. Further, I evaluate a scalable energy measurement solution for production HPC systems and address its shortcomings.

Such high-resolution and large-scale measurements present challenges regarding the management of large volumes of generated metric data. I address these challenges with a scalable infrastructure for collecting, storing, and analyzing metric data. With this infrastructure, I also introduce a novel persistent storage scheme for metric time series data, which allows efficient queries for aggregate timelines. To ensure that it satisfies the demanding requirements for scalable power measurements, I conduct an extensive performance evaluation and describe a productive deployment of the infrastructure.

Finally, I describe different approaches and practical examples of analyses based on energy measurement data. In particular, I focus on the combination of energy measurements and application performance traces. However, interweaving fine-grained power recordings and application events requires accurately synchronized timestamps on both sides. To overcome this obstacle, I develop a resilient and automated technique for time synchronization, which utilizes crosscorrelation of a specifically influenced power measurement signal. Ultimately, this careful combination of sophisticated energy measurements and application performance traces yields a detailed insight into application and system energy efficiency at full-scale HPC systems and down to millisecond-range regions.

Contents

List of Figures

List of Tables

1 Introduction

In the Information Age, scientific computing has become an essential resource for discovery. Simulations with increasing accuracy and size continue to push the tremendous demand for hardware resources, and thus the need for High Performance Computing (HPC) systems. Traditionally, performance is the main criterion for HPC systems. In the more recent past, however, energy efficiency has emerged as another crucial measure in this context. "The Energy and Power Challenge" was first identified as a major limiting factor for increasing performance by the *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems* [Kog+08].

On a large scale, facility operators struggle to realize increasing and highly variable power demands to their data centers [Bat+14]. At a much smaller scale, the power consumption of processors limits their delivered performance [Sch+16b; Hac+15]. Moreover, this trend is motivated by the increasing environmental awareness and electricity costs [Auw+14]. To that end, energy efficiency research strives to improve the ratio of delivered performance to consumed energy. Figure 1.1 illustrates the impact of energy efficiency on processor performance. On this exemplary contemporary processor, the achievable frequencies range from 1.7 GHz to 3.8 GHz depending on the executed workload and number of active cores. The limiting factor for the frequency is power consumption and the resulting heat dissipation. Hence, contemporary processors use complicated control systems to achieve maximum performance under constrained power consumption [GSS15, pp. 53–56].

Energy efficiency research and power capping have one thing in common: They rely on *accurate* energy measurements. An optimization that saves 4 % energy becomes meaningless if the measurement error exceeds this improvement. For power capping, measurement accuracy even affects performance.

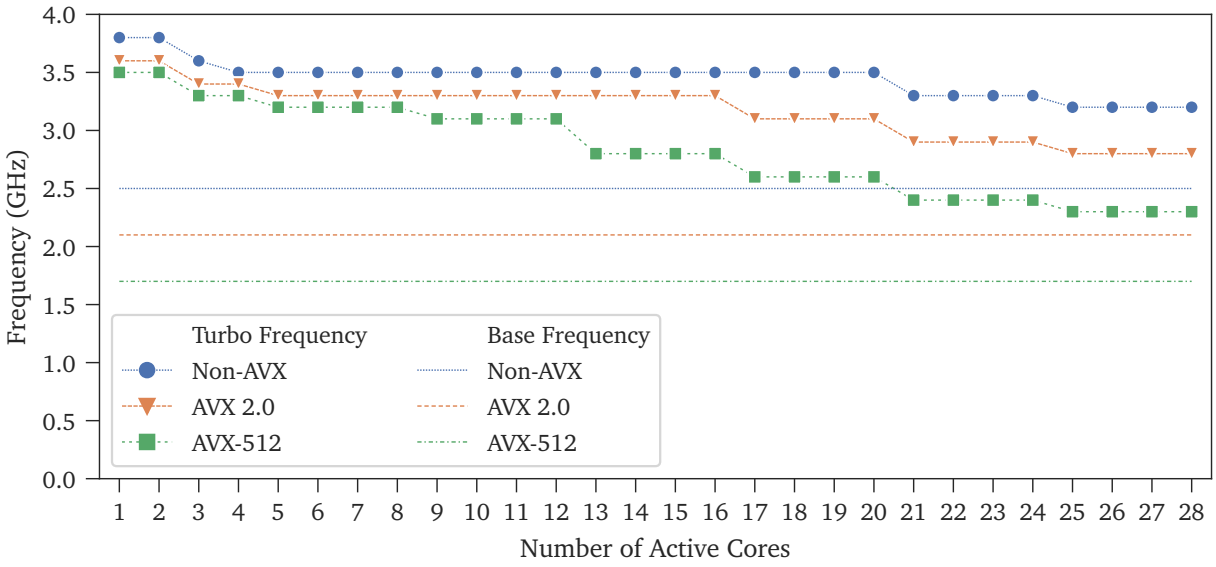


Figure 1.1: The base and turbo frequencies of an Intel Xeon Platinum 8180 depending on the number of active cores and the workload. The underlying data is documented in [Int17b].

Considering a processor whose power consumption must never exceed a specific limit for safe operation, if the measurement to observe this limit had an uncertainty of 10 %, the mechanism would need an equal margin. This margin translates to a significant loss of performance even if the measured value was accurate. Moreover, it is not sufficient for a measurement to be accurate on average or for a limited set of scenarios.

Conventional power monitoring used in computing systems does not provide the necessary quality for energy efficiency research [Hac+13a]. Although metrology offers well-established solutions for energy measurement in general, the specific properties of computing systems have to be considered. A core challenge of measuring computing systems is the high variability of executed workloads. In consequence, the power consumption exhibits an almost arbitrary signal, impeding the definition of assumptions for measurements. Furthermore, understanding the impact of short-time workload changes requires a measurement with a *high temporal resolution*.

Another distinct characteristic of computing systems is the diversity of components with their individual power consumption and the resulting complex power delivery. For energy efficiency research it is vital to isolate specific components, e.g., processor and memory. In particular, bottom-up energy models require a deep understanding of the different components. Therefore, energy measurements should have a *high spatial resolution*. When instrumenting a computing system with an energy measurement, the choice of measurement point and sensor affect the possible temporal and spatial resolution.

Moreover, High Performance Computing concerns systems with thousands of computing nodes, challenging the scalability of energy measurement solutions. Notably, the vast amount of data generated by high-resolution energy measurements from large-scale systems creates demanding performance requirements for further data handling. This includes data collection, processing, and consumption, but also persistent storage and retrieval as well as analysis. While scalable data center monitoring solutions are available, they typically operate at temporal resolutions of one second or longer [Pel+15; Age+14; Bra+09].

Finally, all measurements and data processing are of no avail without an analysis that yields insight and knowledge. Depending on the specific use case, there is a wide range of options to analyze the measurements. For example, energy measurements alone can be plotted in charts to show the variation over time. Nonetheless, a combined analysis of energy measurements with application or system events can leverage more information. The connective factor for this combination is time. However, the energy measurements and application events typically originate from separate systems with different clocks, which, at high temporal resolutions, can create ambiguity.

This thesis addresses the three challenges for energy measurements of HPC systems: 1) implementing accurate and high-resolution measurements, 2) processing the large volume of measurement data, and 3) leveraging the measurements for energy efficiency analysis. To begin, Chapter 2 provides an overview of the existing research on the different aspects of this topic.

Chapter 3 then delves into the proposed improvements for energy measurements. The investigation puts a particular focus on increasing the temporal resolution while retaining a well-known accuracy. As the results reveal, the choice of the measurement point and sensor has a significant impact on the achievable resolution. Chapter 3 further includes a thorough evaluation of a scalable energy measurement solution embedded in a production HPC system as well as a detailed discussion on using CPU energy counters as alternatives to dedicated measurements.

Subsequently, Chapter 4 describes the design and implementation of a scalable solution for processing metric data. A key element of this infrastructure is the concept of Hierarchical Timeline Aggregation (HTA), a novel storage scheme for time series metric data that enables both continuous insertion at high data rates and efficient retrieval of aggregate values and timelines. The discussion of this infrastructure includes a comprehensive performance evaluation, which demonstrates that it satisfies the specified performance requirements of large-scale, high-resolution energy measurements.
Based on power measurement data from the proposed measuring systems and processing infrastructure, Chapter 5 introduces different ways to analyze energy measurements. The focus is set on application power traces, a combination of energy measurements and monitored application and system events, which retains full temporal information. To that end, this thesis contributes a reliable technique to correlate power measurements and application events. This technique uses synthetic workload kernels and power measurements to create a hidden channel between the system under test, which executes the application, and the energy measuring system. The presented method then performs a crosscorrelation on recorded information from both systems, which provides the basis for synchronizing the different timestamps. A set of specific use cases demonstrate the practical applicability of the contributions of this thesis. These use cases include an investigation of a power anomaly that prevented processors from using energy-efficient sleep states. This investigation eventually led to a fix in the officially released version of the Linux operating system kernel that saves $\approx 10\,\%$ of the idle power consumption on affected server systems. Finally, Chapter 6 gives a summary and discusses future work.

2 Background and Related Work

With the ExaScale Computing Study [Kog+08], the energy consumption and hence energy measurements of HPC systems have entered the focus of scientific research. Energy measurement, in general, has a long and established tradition, which I discuss in Section 2.1. In Section 2.2, I then survey a range of approaches to determine the energy consumption of computing systems with a focus on HPC systems and their components. Section 2.3 covers concepts and software for processing and storage of measurement data. Subsequently, Section 2.4 describes factors that impact the power and energy consumption of computing systems and efforts to improve their energy efficiency. In Section 2.5, I discuss common techniques for performance analysis and present ways to combine them with power measurements. Finally, in Section 2.6, I provide a summary of the related work and the aspects that this thesis addresses and improves.

2.1 Basic Concepts of Energy Measurements

Electrical energy is equivalent to the integral of power over time:

$$E = \int P(t)\,\mathrm{d}t \tag{2.1}$$

Conventional electricity meters use electromechanical induction to facilitate the integration of power over time [Ler12, Sec. 8.3]. This approach is not applicable to modern systems that strive for a detailed recording of energy consumption over time. Therefore, electrical energy is not measured directly but digitally calculated as the integral over a series of power measurements. In turn, power is computed from voltage and current measurements [Ler12, Sec. 11.10]:

$$E = \int V(t) \cdot I(t)\,\mathrm{d}t \tag{2.2}$$

While there is no strict distinction, this thesis uses the term *energy measurement* when there is a focus on obtaining an accurate energy value for a given time interval, and *power measurement* if the focus is on the dynamic of power consumption over time. As I discuss in Section 2.1.3 and Section 3.5.2, determining an accurate energy value requires more than just accurate power samples.

Traditional electrical components, such as ohmic resistances or electric motors, have well-understood properties concerning the dynamic properties of the measurement signal. In contrast, the power consumption of computer components can change at the scale of 10 µs depending on the executed workload [Ils+15a]. The arbitrary dynamic power consumption patterns hinder general assumptions that would simplify the measurement and need to be considered throughout the measurement chain.

2.1.1 Basics of Metrology

Terminology

The International Vocabulary of Metrology (VIM) [JCG12] describes the fundamental terminology of metrology. Formally, it defines a *measurement* as the "process of experimentally obtaining one or more quantity values that can reasonably be attributed to a quantity". Moreover, the *measurand* is the "*quantity* intended to be measured" and the *measurement result* comprises the "set of *quantity values* being attributed to a *measurand* together with any other available relevant information". The central outcome of a measurement is the "quantity value representing a *measurement result*" referred to as *measured quantity value* or short *measured value*.

A *sensor* is an "element of a *measuring system* that is directly affected by a phenomenon, body, or substance carrying a *quantity* to be measured". A *transducer* is a "device, used in *measurement*, that provides an output quantity having a specified relation to the input *quantity*". The sensor is a part of the transducer. Moreover, the *measurement signal* refers to the temporal behavior of a physical quantity within the measuring system [Par10].

Extending on this vocabulary, this thesis uses the term *metric* similarly to measurand (see also Section 4.2). In layered measuring systems, in which the measured value is processed further, *readout value* refers to the measured value that is available to the user or analysis as opposed to measured values that are used internally in a measuring system.

Error and Uncertainty

There are different approaches to formally describe the correctness of measured values. The traditional error approach assumes a single true value to which a measured value differs by a *measurement error*. This error can be distinguished into a *systematic error*, which remains constant or predictable across replicate measurements, and a *random error* that varies unpredictably across replications. In practice, an upper limit of the observed error is sometimes referred to as *uncertainty* [JCG12]. A criticism of this approach is that the error of a measurement is unknowable in practice. Corrections for known effects, e.g., calibration, contribute to the error in ways that are difficult to quantify. Further, there may even be a distribution of true values due to insufficient definitions of the measurand.

Contrary, the uncertainty approach offers a formalized definition of uncertainty as well as a process to determine it. The *Guide to the expression of uncertainty in measurement* [JCG08] defines *uncertainty* as "parameter, associated with the result of a measurement, that characterizes the dispersion of the values that could reasonably be attributed to the measurand". In the formulation stage of an uncertainty evaluation, the measurand (Y) is defined and all quantities on which it depends (X_i) are identified. Moreover, a *measurement model*, which relates the measurand to the input quantities ($Y = f(X_1, \ldots, X_N)$) is developed. Then, probability distributions are assigned to the input quantities. In the subsequent calculation stage, the model is used to propagate the probability distributions from the input quantities to the measurand. This calculation results in the standard uncertainty as well as a coverage interval with a specified coverage probability. An uncertainty evaluation can either be based on statistical analysis of a series of observations (Type A evaluation) or on other information, e.g., known uncertainty of input quantities (Type B evaluation) (see [JCG08; JCG09]).

2.1.2 Measuring Voltage, Current, and Power

This thesis focuses exclusively on digital measuring systems, which are based on capturing a voltage with an analog-to-digital conversion. This means that only the voltage as a measurand can be measured directly, but all other quantities must be converted to a voltage before an analog-to-digital conversion [Mal11, p. 122]. Therefore, a current signal is converted to a voltage signal using a current transducer, both voltage signals are captured digitally, and power is computed digitally from both [Web04, pp. 3–11].

Current Transducers

In order to convert the current signal to a voltage signal, a transducer is used. There are two commonly used principles to implement current transducers[1]. *Shunts* are resistors with a low, calibrated value that are inserted between the component under test and its power supply. The current going through the resistor causes a voltage drop proportional to the value of the current. The resistance of the shunt presents a compromise: Small resistances result in a small voltage drop and thus limit the achievable precision from the voltage measurement. Larger resistances cause higher heat dissipation and thus the resistance changes with a rising temperature. Furthermore, the measured component must still function properly with the reduced voltage (see also [Web04, p. 2-5]). The current at a shunt is computed by the following equation:

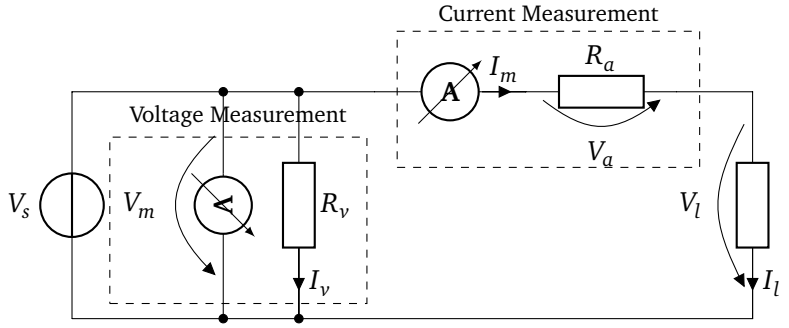
$$I = \frac{V_{\text{shunt}}}{R_{\text{shunt}}} \tag{2.3}$$

Hall effect sensors use the magnetic field to transduce current. In principle, they can operate without contact to the measured system. This property is utilized in contact-less current clamps. In integrated Hall effect transducers, magnetic shielding is used to increase accuracy. These integrated components are inserted between the power supply and the measured component similarly to shunts. To compute the measured current, Hall effect sensors come with a documented response given in V/A. The advantage of Hall effect sensors over shunts is that they have less influence on the measured system and provide galvanic isolation. Integrated Hall effect sensors require an active power supply for amplifying the signal. In return they provide a higher voltage signal that is easier to capture than the low voltage drop of shunts. Further, Hall effect sensors respond differently to high frequencies (see also [Web04, p. 2-10]).

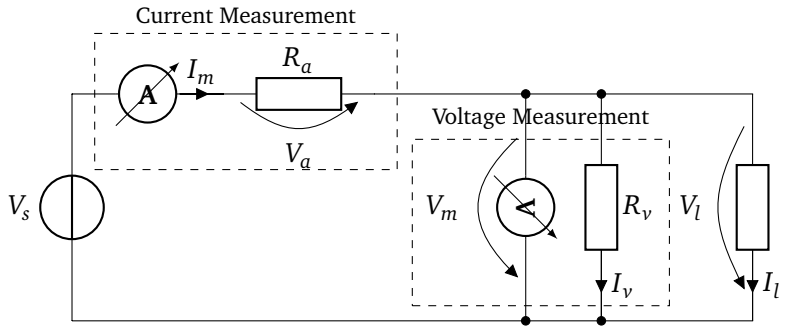
DC Power Measurement

There is a general distinction between measuring direct current and alternating current. *Direct current* (DC) is commonly defined to be “a current that flows only in one direction” [AA02, p. 96]. However, it is often also assumed that in a DC circuit, both the current and the voltage do not vary over time, at least with respect to the timeframe for determining the power consumption [Web04, p. 3-1]. By contrast, *alternating current* (AC) is “a current which periodically reverses its direction, varying sinusoidally with time about a mean value of zero” [AA02, p. 6].

[1] Current transducers are also referred to as current sensors.



(a) Power measurement with proper measurement of load current and source voltage



(b) Power measurement with proper measurement of load voltage and source current

Figure 2.1: Different combinations of voltage and current measurement devices

DC power can be measured by multiplying measured voltage and current values:

$$P = V \times I \tag{2.4}$$

However, since both voltage and current measurements have an influence on the circuit, their impact on each other should be considered. Figure 2.1 shows the two ways to apply voltage and current measurements with their replacement circuits. These two ways focus on either a proper current or a proper voltage measurement. For the circuit in Figure 2.1a, the source power P_s and load power P_l can be computed as follows:

$$P_s = V_m I_m + \frac{V_m^2}{R_v} \tag{2.5}$$

$$P_l = V_m I_m - I_m^2 R_a \tag{2.6}$$

Similarly, for the circuit in Figure 2.1b, power is computed as:

$$P_s = V_m I_m + I_m^2 R_a \tag{2.7}$$

$$P_l = V_m I_m - \frac{V_m^2}{R_v} \tag{2.8}$$

These equations show, that there is a discrepancy between the source or load power and the measured power that only considers measured voltage and current values, i.e.,

$$P_m = V_m I_m \tag{2.9}$$

This difference, referred to as insertion error, depends on which circuit is chosen and which power is to be measured. Ideally, the insertion error is negligible, so the simple computation for P_m can be used in practice. Generally, for setups with high voltage and low current, proper current measurement is preferred, whereas for high current and low voltage, proper voltage measurement of is more accurate. The optimal solution depends on the measured voltage and current as well as the impedance of the measurement devices (see [Mal11, pp. 141 sq.], [Web04, pp. 3-1 sq.]).

AC Power Measurement

The above discussion relates to the power consumption in DC circuits. For varying voltages and currents, the function

$$P(t) = V(t) \times I(t) \tag{2.10}$$

yields the *instantaneous power*. For AC circuits, the *active power*, which is the mean power of one period T is defined as:

$$P = \frac{1}{T} \int_0^T P(t)\,\mathrm{d}t \tag{2.11}$$

For digital measurements, *sampling wattmeters* use discrete instantaneous power samples to compute average the power as per:

$$P_{\text{avg}} = \frac{1}{N} \sum_{k=0}^{N-1} P(k) = \frac{1}{N} \sum_{k=0}^{N-1} V(k) I(k) \tag{2.12}$$

For AC measurements, N is the number of samples in one period and $X(k)$ is the k-th sample of one period (see [Web04, pp. 3-3, 3-11]). Power measurements for AC and constant DC offer simplified measurement approaches (e.g., [Ler12, p. 143]). Computing the power consumption of (non-sinusoidal) variable current can be done similarly to AC with (2.10) for instantaneous power values or (2.11) and (2.12) for average power values with respect to a timeframe T. However, as discussed in the following, the sampling approach must be appropriate for the specific variability of the analog signal.

2.1.3 Measurement Signal Conditioning and Analog-to-Digital Conversion

In order to accurately capture measurement signals, it can be necessary to condition the analog signal. Moreover, the analog-to-digital conversion highly depends on the characteristics of the signal.

Signal Amplification

When using shunts as current transducers, the voltage drop is often very small. Capturing this voltage directly can have a negative impact on accuracy. Measurement amplifiers scale the measurement signal proportionally to a higher amplitude. The following properties are important for a measurement amplifier [Ler12, p. 171]:

- low feedback on the measurand,
- linearity,
- low noise and low distortion at high amplitudes,
- sufficient bandwidth,
- load independent output signal.

There are further analog signal processing techniques such as to determine certain characteristics or statistics in the analog domain. However, this thesis focuses on digital measurement processing, which nowadays enables complex transformations without dedicated hardware.

Anti-Aliasing Filters

The Nyquist–Shannon sampling theorem states that the sampling rate must be larger than two times the maximum frequency contained in the analog signal:

$$f_s > 2B \tag{2.13}$$

This criteria ensures that the original continuous signal can be perfectly reconstructed from the samples. An actual reconstruction is typically not implemented by processing software and a digital system cannot represent a time-continuous signal. Nevertheless, the criterion ensures that measured power allows a correct computation of energy given a duration with sufficient sample count. Violations of the criterion cause the aliasing effect, which prevents a signal reconstruction and thus the average power or energy consumption cannot be determined accurately. To ensure a sufficient sampling rate and prevent aliasing, an analog low-pass (anti-aliasing) filter is often applied before the analog-to-digital conversion (see [Ler12, pp. 331, 435 sqq.]).

Analog-to-Digital Conversion

After the analog processing, the measurement signal is converted to a digital series of values. There are various different ways to implement analog-to-digital conversion aimed at different sampling

rates and precision [Ler12, p. 360]. There are integrated data acquisition solutions that provide the measurement values to a computer. This approach allows for digital measurement processing in software without developing digital hardware and drivers.
An example of such a data acquisition device is the National Instruments NI-6255 [Nat16b]. It can sample up to 80 input channels with an maximum aggregate sampling rate of 750 kSa/s at a value resolution of 16 bits. The device specification includes a detailed equation for accuracy which is split into gain error (relative to the measured value), offset error (relative to the measurement range), and noise uncertainty. The accuracy also depends on temperature and on regular calibration. The high amount of channels in this device is achieved through multiplexing one analog-to-digital converter between multiple channels. This multiplexing introduces a settling error for multichannel measurements [Nat16a; Nat16b]. Another data acquisition card is the NI-6123 [Nat15]. This device has 8 independent differential input channels with a maximum sampling rate of 500 kSa/s for each channel. Both the NI-6255 and NI-6123 are available as PCI devices for mounting in a standard PC and PXI for integration in modular laboratory instruments. The NI-6255 is also available as USB device. It illustrates that there are many aspects to consider when choosing a solution for data acquisition. Further interfaces that can be used for capturing digital measurement data are described in [Ler12, pp. 515–640]. Details for building automated measurements systems based on computers and data acquisition hardware are discussed in [Par10, pp. 185–190].

2.2 Power Measurements for Computing Systems

Measuring the energy consumption of computing systems has always been a prevalent topic for mobile systems where battery life is essential. With the growing importance of energy efficiency and limitations due to power consumption, energy measurements have now also become an important aspect of HPC and other data center systems. Together with my co-authors, I discussed a comparison of existing approaches for measuring energy in HPC systems in [Ils+15a, Sec. II] and [Ils+18c, Sec. 2]. This includes a list of five key criteria for evaluating and comparing power measurement techniques:

1. *Temporal resolution.* The power consumption of a processor highly depends on the executed task, which can change in short time intervals in the order of microseconds. To accurately analyze the impact of such short workloads on energy consumption, it is necessary to provide a high temporal resolution. As discussed in Section 2.1.3, a high sampling rate is already necessary for accurate energy values. However, the frequent samples are not always available as external readout values, but rather used internally to achieve a high accuracy. The typical temporal granularity of measurements for computer systems is between 1 s and 1 ms. A detailed discussion of temporal measurement granularity is given in Section 3.5.

2. *Spatial resolution.* In order to understand the power consumption of a system during a workload in detail, it is often beneficial to measure components in the system separately. For instance, distinguishing the power consumption of the CPU and memory subsystem will yield additional insight about the resource utilization of an analyzed workload. Section 3.2 provides a detailed discussion of instrumentation points on different systems under test.

Table 2.1: Comparison of different energy measurement approaches.

Measurement Type	T. Resolution	Spatial Resolution	Accuracy	Scalability	Cost
External Power Analyzers at AC Input					
LMG450 [ZES]	50 ms	node	0.07 %+0.04 %[a]	(–)	(–)
LMG600 (A1) [ZES16]	50 ms	node	0.015 %+0.01 %[a,b]	(–)	(––)
WT1800E [Yok16]	50 ms	node	0.05 %+0.05 %[a,c]	(–)	(––)
Custom Solutions					
PowerPack [Ge+10]	< 1 s	components	"verified"	(+) 9 nodes	(o)
PowerMon2 [Bed+10]	1 ms	components	6.8 % (I)	(+)	$150
PowerInsight [LPD13]	1 ms	components	avg. 1.8 % (I)	(+) 104 nodes	(o)
ArduPower [Dol+15]	0.17 ms	components	< 1.5 %	(+)	€ 100
PowerSensor 2 [RV18]	0.116 ms	GPU	3.7 % (I)	(o)	(o)
DiG[Bor+18; LBB18]	1 ms / 20 µs[d]	node	< 1 % (σ)	(+) 45 nodes	$90
Integrated Measurement Solutions					
PDU (typical) [Hac+13a]	1 s	node	heavy aliasing	(++)	(+)
Cray XC30 [MK14]	100 ms	node, GPU	aliasing	(++)	(+)
Cray XC40 [MRK15]	100 ms	node, components	calibrated	(++)	(+)
IBM AMESTER [IBM18]	0.25 ms	node, components	undocumented	(++)	(+)
Atos HDEEM [Ils+18c]	1 ms / 10 ms	node, CPU, DRAM	3 %	(++) 1456 nodes	(+)
CPU Counters					
AMD's APM[e] [Hac+13a]	10 ms[f]	per socket	systematic errors	(+)	(++)
RAPL SNB [Hac+13a]	1 ms[f]	cores, mem, pkg	systematic errors	(+)	(++)
RAPL HSW+ [Hac+15]	1 ms[f]	cores, mem, pkg	no systematic errors	(+)	(++)

[a] reading + range
[b] additional terms apply, see (2.14)
[c] additional terms apply for current ranges above 5 A, see (2.15)
[d] 1 ms available in centralized monitoring, 20 µs available for internal edge analysis at the embedded monitoring nodes
[e] tested on the Bulldozer processor generation
[f] in-band readouts are typically not performed in minimal intervals due to imposed perturbation

3. *Accuracy.* It is important to have a clear understanding about the accuracy of energy measurements. Otherwise any analysis and conclusions based on the measurements carries the unknown uncertainty. I evaluate the accuracy of several measuring systems in Section 3.4

4. *Scalability.* An energy measurement solution for parallel workloads running on HPC systems or in large data centers, needs to be scalable. Instrumentation for scalable measurements is introduced in Section 3.2.5. The necessary infrastructure to process the high measurement data rates is presented in Chapter 4.

5. *Cost.* A low cost makes energy measurements more accessible to a wide range of users and thus increases the chance for reproducibility of experiments. Further, cost also influences the scalability — an expensive solution may be feasible for individual systems, but hardly for a production-scale HPC system with thousands of compute nodes.

Table 2.1 summarizes different energy measurement approaches according to these criteria. A critical aspect for any measurement solution is the selection of a measurement domain which also involves choosing an appropriate point of instrumentation. As shown in Figure 2.2, the power distribution within a computing system forms a graph in which power conversion units (e.g., power supply unit (PSU) and voltage regulator (VR)) are nodes and cables or other electric connections are edges. There are many possible points of instrumentation within that graph. The power consumption of a

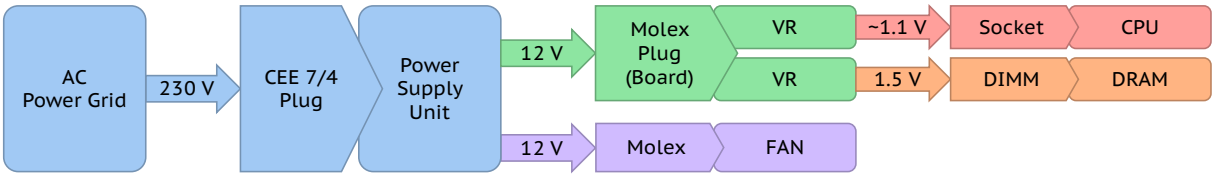


Figure 2.2: Simplified overview of power transmission and conversion for selected components of an exemplary computing system (see also [Ils+15a]).

computing system as a whole, as measured outside of the PSU, is typically characterized as AC and thus measured accordingly[2]. In contrast, the power consumption of components within a computing system is typically DC at lower voltages. Specifically, the involved voltages, supplied by a PSU or VR, vary only in a limited range, but the current varies arbitrary. While this constitutes a DC measurement, the varying nature of the current and power needs to be considered.
Certain power conversion devices have internal measurements and provide them as digital or analog interfaces, e.g., the ON Semiconductor NCP8125 Voltage Regulator includes an analog output current signal [ON 15]. Edges can be instrumented with special measurement adapter cables or riser cards for sockets, e.g., Adex Electronics[3] manufactures a range of riser cards that include current shunt options. In general, instrumenting closer to the component provides a better spatial resolution and a more narrowed down measurement domain. However, since only a small part of components are captured with each instrumentation point, more instrumentation points are needed to get a holistic view of the system. While an instrumentation close to the component offers more insight into the inner energy usage of the components, a more coarse grained instrumentation closer to the power supply provides better information about the actual energy costs of the system.
Hsu [HP11] lists possible measurement domains from the point of view of a large HPC center:

a) a site,

b) an HPC facility,

c) an HPC machine,

d) a cabinet (or server rack),

e) a server (compute node),

f) components inside a server.

In the context of the Jaguar supercomputer, the site and facility levels are used for revenue metering and PUE monitoring. Measuring at the HPC machine level can cover large-scale systems with little effort, but does not provide a fine-grained resolution. Measurements at the level of cabinets and compute nodes level can utilize capabilities of modern PDUs, e.g., Megware ClustSafe [Meg], without requiring additional instrumentation. Some PSUs also provide measurement capabilities, for example through the standardized PMBus interface [Sys15]. This thesis focuses measurements at the level of compute nodes and discusses different approaches for them in the following sections.

[2]Large HPC systems use more complex power delivery schemes and may supply multiple compute nodes with DC power.
[3]Adex Electropnics, Bus Extenders and Risers: `https://www.adexelec.com/extenders.htm`

2.2.1 Measuring Compute Nodes using External Power Meters

There is a wide range of external power meters that are often used at the power inlet of a compute node. Commodity devices such as the "Watts Up? Pro"[4] are popular due to its low price and ease of use and were particularly often used when energy efficiency became an important topic for HPC, but specialized solutions were not yet available [HP11; Ge+10; Dol+10; Jia+10; Gra+17]. This entry level device provides a temporal resolution of at best 1 s and an accuracy of ±1.5 % + 3 × value resolution [Ver15]. However, below 60 W the accuracy further decreases and the minimum measurable power is 0.5 W at which the accuracy is given at ±0.3 W. The device is also limited to 120 V, 60 Hz AC input with up to 1.8 kW[5]. Consequently, such a device cannot be used in data centers with DC power input.

More professional power meters provide improved accuracy and capabilities. For example the Yokogawa WT1800E series [Yok16] specifies the following accuracy for typical AC power measurements:

$$\text{accuracy}_{\text{WT1800E}} = \pm\left[0.05\,\%\text{ of reading} + 0.05\,\%\text{ of range} + 2\,\mu\text{A} \times V\right] \tag{2.14}$$

The ZES Zimmer LMG600 series offers even more accurate readings with the corresponding L60-CH-A1 channels. For typical AC readings using the internal sensors, its accuracy is specified as follows:

$$\text{accuracy}_{\text{LMG600},\geq 10\,\text{A}} = \pm\left[0.015\,\%\text{ of reading} + 0.01\,\%\text{ of range} + \frac{30\,\mu\text{A}}{\text{A}^2} \times I_{trms}^2 \times V_{trms}\right] \tag{2.15}$$

For current ranges below 5 A, the last term disappears:

$$\text{accuracy}_{\text{LMG600},\leq 5\,\text{A}} = \pm\left[0.015\,\%\text{ of reading} + 0.01\,\%\text{ of range}\right] \tag{2.16}$$

For both the WT1800E and the LMG600, this excellent accuracy is coupled to a number of conditions. The voltage and current input should be a sine wave, the temperature needs to be around 23 °C and the device needs to be calibrated in regular intervals. For higher frequency input waveforms, the accuracy drops significantly or even ceases to be specified (see [Yok16; ZES16]). These conditions can be satisfied for the AC input of compute nodes. However, it is also possible to apply these devices to various DC measurement domains within a compute node. The dynamic nature of compute workloads can result in almost arbitrary waveforms at the DC measurement sensors, making it much more difficult to satisfy the conditions for a high accuracy. Section 3.5 evaluates the effect of dynamic workloads on the actual power consumption signal at different DC measurement domains.

The WT1800E uses an internal sampling rate of approximately 2 MSa/s. The rate can be switched between three values to avoid aliasing [Yok16, pp. 8–9]. However, the readout values in "High Speed Data Capturing" mode are only provided at 20 Sa/s [Yok16, p. 16-1]. The LMG600 uses internal sampling rates of up to $1.\overline{21}$ MSa/s. In normal operation, it measures in cycles of at least 30 ms ($33\frac{1}{3}$ Sa/s), but it also provides a scope mode to access the measured values at its internal sampling rate. The possibility to continuously record a power trace at microsecond-resolution makes this device particularly well suited for analyzing the influence of short compute workloads on power

[4] This particular product is discontinued.

[5] There is an international versions with different voltage and frequency specifications.

consumption. In Chapter 3, I use a LMG670 device with six L60-CH-A channels. Both the WT1800E and LMG600 devices include different filter configurations that can be used on the input signal. The LMG600 family further offers a DualPath mode, which allows to simultaneously capture the input signal with two analog-to-digital converters: One wideband signal that contains the full dynamics and frequency spectrum and one filtered narrowband signal that uses an anti-aliasing filter for ensuring accuracy [ZES15]. The L60-CH-A channels uses a wideband sample rate of $1.\overline{21}$ MSa/s at an unfiltered bandwidth of 10 MHz or with an optional 145 kHz analog anti-aliasing filter. The narrowband signal is sampled with $151.\overline{51}$ kSa/s and uses an unconditional anti-aliasing filter with a bandwidth of 14.5 kHz [ZES16].
While the external power analyzers can provide an excellent accuracy, the cost for professional devices is high, and it does not offer scalable solutions for HPC systems. High-end devices can work either on AC or DC measurement points, so the spatial resolution depends on the specific implementation.

2.2.2 Custom Solutions for Measuring Compute Node Power

There are several products and research projects that provide power measurements of compute nodes that have no or insufficient power measurement support on their own. Typically small add-on components are used such that the additional hardware is integrated into the standard compute nodes.
PowerMon2 [Bed+10] uses a form factor that fits into a 3.5 ” hard drive slot. This can be applied to general purpose server designs that feature a hard drive bay, but is not easily applicable for denser, specialized HPC nodes. Measurements are taken at a rate of up to 1024 Sa/s for a single channel or up to 8 measurement channels at a shared rate of 3072 Sa/s. The accuracy is documented at ±0.9 % for voltage measurements and −6.6 % / 6.8 % as worst-case for current. Its predecessor PowerMon costs less but only provided a maximum sampling rate of ∼ 50 Sa/s and used a larger form factor. The measurement units can be fabricated at a cost of less than $150 per device which makes it feasible to apply them to larger systems. In the exemplary measurements of a commodity system, six channels are measured: two 12 V, one 5 V, one 3.3 V channel from the power supply unit as well as a 12 V and 5 V rail from a hard disk. Measurement data is transferred to the compute node under test itself using USB.
PowerPack [Ge+10] is a more abstract framework for power measurement. It describes a set of tool kits including hardware and software with support for various measurement devices and a user-level API. An implementation of PowerPack uses “Watts Up? Pro” power meters for AC power and shunt resistors in combination with National Instruments data acquisition hardware for measuring DC power within the compute node. This implementation would inherit the limitations of external measurements in terms of scalability and cost. While redundant sensors are used to verify each other, no specific accuracy is documented for the exemplary implementation. Examples for PowerPack show a resolution of $\ll$1 s, but no specific information about sampling rate is provided.
Penguin Computing provides a commercial HPC compute node power measurement device called PowerInsight [LPD13]. A BeagleBone board is used as the core for PowerInsight, providing a full Linux with floating point computing capability, basic ADCs, as well as good connectivity. A carrier board — the PowerInsight cape — provides additional ADCs and a total of 15 connectors to sensor modules. Different harnasses (sensor modules) can be connected to the carrier board, e.g., a standard

motherboard connector or PCIe risers. Sensor modules contain a Hall effect sensor and a voltage divider for each channel. The validation of this works only shows average errors for current and voltage of 1.8 % and 0.3 %, respectively. The effective sampling rate is limited by the software overhead to $\sim$ 1 kSa/s. Both, USB communication with the compute node under test, as well as a global LAN interface are available.
The ArduPower [Dol+15] wattmeter uses an Arduino Mega 2560 board as base and adds an ArduPower sensing shield on top of it. ArduPower offers a sampling rate up to 5880 Sa/s and 16 channels. Each channel uses an ACS713 Hall effect sensor with an error of $\pm$1.5 %. Further verification is performed using an external precision power analyzer. While the power traces match visually, no quantitative evaluation of the total error is performed. With a low production cost of approximately € 100, the ArduPower can be used in medium to large-scale systems analogous to PowerMon2.
PowerSensor 2 [RV18] is a similar project with the goal of providing power measurements for GPUs with a high temporal resolution at a low cost. It uses a PCIe riser card, ACS712 Hall effect current sensors, and an Arduino Leonardo. The current measurement has a documented uncertainty of 3.7 %, but the authors further report that “with proper calibration, our measurements are typically within 1 % of built-in GPU power meters and lab equipment”. The accuracy of power consumption is further limited since the voltage is not measured. PowerSensor 2 provides instantaneous power samples at 8620 Sa/s.
DiG (Dwarf in a Giant) [LBB18] leverages a custom power monitoring system for HPC systems based on BeagleBone Black embedded boards. The system can use different configurations of the power sensing module. One configuration uses a current mirror in combination with a shunt resistor for measuring current whereas a voltage divider conditions the voltage level. This configuration is used in the 45-node cluster D.A.V.I.D.E. [Bor+18]. An alternative configuration uses Hall effect sensors and voltage dividers. The authors recognize that ”due to the high operating frequencies of HPC nodes, the power consumption is highly dynamic, and therefore an anti-aliasing filter is required“ [LBB18]. In fact first-order low-anti-aliasing filters are used before analog-to-digital conversion by the BeagleBone Black. The internal sampling rate is 800 kSa/s, which is averaged down to 50 kSa/s in hardware. The measurement software further aggregates measurement data before being it sends it to a message broker at 1 kSa/s and 1 Sa/s for two separate data queues. Since the BeagleBone Black itself is an open platform, it can perform any kind processing on the measurement data at the intermediate 50 kSa/s. The authors demonstrated this by performing FFTs to create power spectral density and Continuous Wavelet Transform plots from high-resolution power measurements.

2.2.3 Measurement Solutions of System Integrators

Several system integrators have adopted power measurement as a system feature. Since the XC30 architecture, Cray provides dedicated power monitoring facilities that are available to users both in-band through sysfs-files as well as out-of-band using the Power Management Database (PMDB) [MK14]. There is also the application library `pm_lib`, which is based on these sysfs-files [Har+14]. On XC30 systems, the implementation is limited to the granularity of nodes and accelerators (i.e., GPU and MIC) and values are updated at 10 Sa/s. In [Har+14], I have shown that the initial implementation is affected by aliasing, which limits the accuracy. With the Cray Advanced Platform Monitoring and Control (CAPMC) [MRK15] for XC40, the power measurement has been improved significantly. By

using a 12-bit ADC instead of an 8-bit ADC, the accuracy was increased. Additionally, hardware averaging over 1 kSa/s prevents aliasing issues [Mar+17]. The XC40 power measurements also include data from VRs which increases the spatial resolution. Measurements for all 12 voltage lanes are available using the out-of-band data collection whereas in-band access provides two new aggregated measurement domains for CPU and memory in addition to total compute node power and accelerator power.

IBM provides sophisticated power measurements with their recent POWER9 systems [IBM18]. The measurements are controlled by the On Chip Controller (OCC), specifically by AMESTER (Automated Measurement of Systems for Temperature and Energy Reporting). AMESTER allows a fine-grained collection of sensor data, including power measurements at 250 µs intervals. An optional Analog Power Subsystem Sweep (APSS) module further provides sophisticated power measurements at the voltage rails. The OCC interface features derived sensors which accumulate power readings to energy and include an update tag that counts the number of values in the accumulation.

In their Bull supercomputer systems, Atos offers an energy measurement infrastructure named HDEEM (High Definition Energy Efficiency Monitoring) [Hac+14; Ils+18c]. HDEEM was created in a cooperation between Atos (formerly Bull) and TU Dresden. Throughout Chapter 3, I describe HDEEM in detail, evaluate it, and discuss improvements based on the evaluation.

All integrated vendor solutions are generally scalable to the size of the full system. The scalability of HDEEM has been specifically demonstrated on a 1456-node production system with a detailed application-power-trace from a run on 1024 nodes (see Section 5.3.3). The power measurement features from vendor solutions are typically included in the purchase of such systems at no explicit cost. However, additional calibration may be necessary and impose additional cost. As an exemplary cost estimation, the Cray XC40 power measurement uses two embedded Texas Instrument devices, a LM5056A and a LM5066I, which are available for approximately € 5 each. Consequently, it can be assumed that the additional hardware cost less than € 100 per compute node, even for a high quality power instrumentation. This does, however, not account for the cost of development as well as including calibration in the manufacturing process. The Cray XC40 architecture leverages a per-unit factory-calibrated power measurement chip within the IVOC to perform a run-time-calibration of the other power measurement device on a compute node (see. [Ils+18c, Sec. 5.2], [Mar+17]).

The aforementioned solutions are specialized and typically use vendor-specific interfaces such as libraries, device drivers, or sysfs-files. An established interface to provide generic access to node-level power measurements is IPMI (Intelligent Platform Management Interface) [Int+13]. Power measurements are only a small part of IPMI, it is used for other kinds of measurements such as temperatures and fan speeds as well as control purposes. A wide range of servers support IPMI, not only in HPC and there are freely available libraries and tools for accessing IPMI devices. However, the generic interface comes with limitations that I discuss in Section 3.5.2. The Redfish Scalable Platforms Management API [DTM18] was specified as a standard to supersede IPMI. Redfish provides a very generic interface using JSON for data. In contrast to IPMI, the JSON format allow for higher precision values and timestamps on metric values. However, the verbose text format makes it unsuitable for scalable measurements at high update rates. Due to the relatively recent introduction and adoption, the practical experience with Redfish for energy measurement is limited.

2.2.4 CPU Energy Counters

With the growing power consumption of processors, power measurements become relevant within individual processors. Therefore contemporary CPUs use measurements for power limiting and also expose them to the user. This new possibility provides distinct advantages: Those power or energy values are readily available — even on commodity systems — without requiring additional measurement hardware or costs. Especially for x86-processors, the homogeneity of processors results in a good availability of particular energy counters across a wide range of desktop, workstation, server, and high-performance systems. This low entry threshold has made CPU energy counters very popular.
Intel's RAPL (Running Average Power Limiting) provides not only controls for power limiting, but also MSRs containing the consumed energy [Dav+10; Rot+12]. For each processor package, RAPL offers four measurement domains (see also Section 3.6). This spatial resolution is another advantage compared to many other measurement approaches. Desktop and mobile systems since the Skylake generation include a platform (PSys) measurement domain which covers the system on chip [Wea18]. For server systems, no measurement domain covering the entire compute node is available.
In the Bulldozer processor family, AMD introduced Application Power Management (APM) [Adv13, Sec. 2.5.2.1.1], which consists of Core Performance Boost and TDP limiting. This also includes a MSR that provides the "current amount of power being consumed by the processor" [Adv13, p. 574]. "APM monitors core activity at the ApmSampleTimer rate", which defaults to 10.24 µs on an Opteron 6274. On this processor, the power is then averaged for a total of ≈ 10.5 ms. For the Zen microarchitecture, AMD switched form APM to RAPL, establishing compatibility with Intel x86 processors.
AMD's APM as well as early generations of RAPL were based on a model, which caused systematic inaccuracies for different workloads, HyperThreading usage, and C-state configuration [Hac+13a]. Since the Haswell-EP processor generation, RAPL is based on physical measurement and no longer affected by theses systematic errors [Hac+15]. DRAM measurements of RAPL have an error of up to 20 % [DPW16]. In Section 3.6, I present an in-depth evaluation of the accuracy of RAPL.

2.2.5 Using Models to Determine Energy Consumption

An alternative way to determine the energy consumption of computer systems is modeling. For this thesis, I classify energy models into two types based on their primary purpose: On the one hand, models for *estimation* are designed to *substitute or complement measurements on existing systems*. They estimate the energy consumption of an actual execution through indirect measurements and can provide power and energy values without additional costs of instrumentation. As an enhancement to physical measurement, models can go beyond the spatial or temporal resolution of what is feasible with direct measurements. Estimation models are often based on measurements of architectural/performance events. On the other hand, models for *prediction* help to determine the energy consumption of *future systems*. They are often part of, or based on, more complex performance models of these systems. Prediction models are particularly useful in co-design to improve the energy efficiency by evaluating design alternatives, when measurements are not possible. With such models, it is also possible to predict the energy consumption of existing systems under new configurations without actually running them in these configurations. In the following, I primarily discuss *estimation models* as a possible substitute to energy measurements.

A *model* is described as an alternative to experimenting with the actual system. The model is built "as a representation of the system and [studied] as a surrogate for the actual system" [Law06]. In the context of energy estimations, I consider quantitative *mathematical models* that lead to a numerical result, similarly as a measurement of an actual system would.

A central method for building models from observation is regression analysis. Regression is used to predict or estimate the value of a *dependent variable* y based the values of J *independent variables* x_j. The general form of the estimated value $\hat{y}$ is as follows:

$$\hat{y} = f(x_1, ..., x_j, ..., x_J) \tag{2.17}$$

The most basic form of regression is a simple linear regression with one independent variable that uses the following model:

$$\hat{y} = b_0 + b_1 x \tag{2.18}$$

A standard approach to estimate the regression parameters b is the *method of least squares*. Regression can use different model formulations and optimization techniques as discussed for example in [Bac+08, pp. 51 sqq.] and [Rya08].

There is no single distinct criteria for evaluating the quality of models. A core metric in statistical modeling is the *coefficient of determination* defined as

$$R^2 = \frac{\sum_i (\hat{y}_i - \overline{y})^2}{\sum_i (y_i - \overline{y})^2} \tag{2.19}$$

where y_i is the observed data, $\hat{y}_i$ the modeled value, and $\overline{y}$ the mean of the observed data. R^2 represents the fraction of the variability within the observed values that can be explained by the model. It is an indicator for the goodness of fit for a regression model. However, it depends on the use case what value of R^2 would constitute a good or bad fit [Rya08, pp. 14 sq.]. While R^2 indicates the fit for the approximation, it does not attest whether the model was specified correctly in the first place. A model based on an insufficient training set may not have a statistical significant correlation despite a high R^2. To avoid this, more sophisticated analysis is required [Bac+08, pp. 71 sqq.].

Due to the large numbers of individual residuals, average errors are often used to quantify the overall quality of a model in a single number. The mean absolute percentage error (MAPE) is commonly used in literature. There is also the mean absolute error (MAE) which is more susceptible to different scales as well as the mean squared error (MSE) and root mean squared error (RMSE) that are more heavily influenced by outliers [Rya08, p. 274]. This focus on averages is in contrast to measurements, which often use an upper bound or distribution of the measurement error (compare Section 2.1.1). Averaging quality metrics prevents conclusions about worst-case performance of a model.

For regression models, there are several common undesirable patterns of the residuals such as heteroscedasticity and autocorrelation. A scatter plot of the residuals and the predicated or true value can help to identify such issues, select the appropriate regression model, and verify the respective assumptions [Bac+08, pp. 80 sqq.].

Approaches to Modeling Computing Systems

There are two opposite ways to build a model: One approach is to build it based on a large number of observations of the system as a whole — this is called a *black-box* or *top-down* model. The other approach is to use an understanding of the interior mechanisms within the system, or to compose smaller models of system components — this is called a *bottom-up* model [Ber+12; Wal+17].
The top-down approach is inherently limited to existing systems and therefore used to build models for estimation. The system is mostly treated as black box, and only observations of inputs and outputs are used. Statistical methods or machine learning techniques build the model based on measurements and very general assumptions. Bottom-up approaches do not require existing hardware, but can be used from a design specification. They make use of design-space exploration frameworks such as McPAT [Li+09]. This approach often combines models of individual components within a system.
The choice between bottom-up and top-down models is not binary: For example, a bottom-up approach can use specific inner knowledge of a system to formulate a general model but then train it as a whole, or a top-down approach can utilize expert knowledge for selection of regressors. To avoid ambiguity regarding in classifying energy models, I will focus on describing how much expert knowledge is utilized in creating a particular model and how much the model reflects the internals of a computing system.

Energy Estimation based on Performance Metrics

The early work by Bellosa [Bel00] has established event counters as basis for an energy accounting model called *Joule Watcher*. As target system, a Pentium II PC with a single core and no DVFS was used. The model maps closely to this relatively simple architecture. Counters of retired microinstructions, floating point operations, L2 cache references, and main memory references are used as inputs for the model. Separate micro-benchmarks are executed at different configurations for each event. A simple multimeter measuring the entire system with a 1 W resolution is used for calibration, i.e., determining the model coefficients. The resulting model uses fixed energy values for each of the operations. Scatter plots confirm the linear correlation between operation rate and power. On this system, the event rates correlate linearly with the power consumption resulting in an approximate energy cost for each event. The paper provides only a vague verification describing the results as "within the resolution of our external multimeter for both synthetic and real-world applications".
While initial work was focused on the CPU itself, Economou et al. [Eco+06] introduced a full-system power modeling approach. Their model is based on more system components, i.e., memory, disk and network in addition to CPU. It is trained for total system power as measured with an AC power meter. While the model calibration is performed using a workload that stresses the different components, the optimization with a "linear program" chooses the model parameters to minimize a prediction error. For one of the two analyzed systems, the off-chip memory access count is multiplied with a negative factor. This is a clear contradiction to the assumption of having independent variables that represent separate system components, which cannot consume a negative amount of power. A verification was done with a different set of workloads. The errors are reported as average for all samples within each workload as well as a 90th percentile error. One particular 90th percentile error was above 20 %, the average errors range up to 15 %.

Bircher and John [BJ07] presented a systematic approach for full-system power estimation, which models each subsystem separately. Again, this model is based on detailed architectural knowledge and subsystem-specific workloads for regression. In the model, five subsystems are considered: memory, chipset, I/O, disk, and microprocessor. Performance events that are propagated between the subsystems, are used as model parameters. The authors make the important observation, that "it is necessary to have sufficient variation within the workload for training of the models". For generating training data, the authors separately measured each subsystem using shunt resistors at 10 kSa/s. Mean absolute percentage errors for each combination of subsystem model and workloads are given with a worst-case of 17.51 %. When averaged over all workloads, the error is below 9 % for each subsystem. With growing complexity of microprocessors, bottom-up power models also increased in complexity as highlighted by the model by Bertran et al. [Ber+10].

A more recent bottom-up approach is presented by Goel et al. [GM16]. They focus on the CPU, but model core and uncore separately as well as dynamic and static power. The latter distinction makes the model easily aware of dynamic voltage and frequency scaling. However, turbo boost and simultaneous multithreading are not supported. For static power, temperature is included as a variable and even varied using a hot-air gun. Dynamic core power is further split into memory, non-memory, and L2 cache components. Expert knowledge about the architecture is gained from experimentation and used in the model, e.g., the fact that the uncore component on this system is neither clock-gated nor power-gated in idle. The performance events as independent regression variables are also chosen using expert knowledge. The level of architectural detail considered for the model makes it one of the most sophisticated bottom-up power estimation models. Training is performed separately for the individual components using especially created microbenchmarks. Each model fit is carefully analyzed both using R^2 and scatter-plots. The evaluation is performed for a varying number of threads and frequencies. While a wide range of benchmarks are used for validation, only mean errors and standard deviation are given for each combination of benchmark group, thread count and frequency. The worst configuration has an average error of $\approx 6\,\%$, the average error over all configurations is 3.14 %. Due to the wide range of tested configuration, the use of many variables, and the relatively low error, this presents significant advancements over previous work.

Variable selection for counter-based energy models has primarily been based on expert knowledge. With increasing complexity of performance monitoring events on modern systems, it has become a significant challenge to select the right events - a process that has to be repeated for each architecture. Walker et al. [Wal+17] introduced a statistical event selection algorithm. They follow a forward selection approach [Nor98], which has been criticized because it only considers a small subset of possible combinations thus not providing an optimization guarantee [Rya08, p. 270]. The performance monitoring event selection algorithm of Walker et al. incrementally selects the event leading to the highest R^2 of the intermediate model. In order to quantify multicollinearity, they evaluate the variance of inflation (VIF) for added performance monitoring events. The VIF is computed by building a different model that estimates the added independent variable from the variables already considered for the primary model. The resulting R^2 is used in the formula for the VIF:

$$\mathrm{VIF} = \frac{1}{1-R^2} \tag{2.20}$$

The VIF indicates whether the added variable is independent (VIF = 1) or whether multicollinearity is an issue (VIF > 5). The authors argue that multicollinearity decreases the stability of the resulting models by making it overly sensitive to changes in the input. In consequence, they apply a manual transformation to the selected events, e.g., subtracting two events to create an independent input variable, hence reducing multicollinearity. For the final run-time power estimation, they include a platform-specific voltage model. This combination of generic statistical methodology for variable selection and low-level expert knowledge offers an improved accuracy and stability.
In [Wal+17], Walker et al. applied this methodology to embedded ARM Cortex-A7 and Cortex-A15 CPUs which results in an average error of 3.8 % and 2.8 % respectively. The model is trained with 20 diverse workloads and evaluated with 60 workloads. The authors offer detailed statistical results including a maximum error of 13 %. In [Cha+17], my co-authors and I adapted the model proposed by Walker et al. to x86 systems. We showed that the accuracy of this modeling approach on the Intel platform is worse with an average error of 7.54 %.

2.3 Processing of Power Measurement Data

Most digital power measurement instruments have integrated displays showing the measurement result. A range of digital signal processing techniques can expose more specific information, e.g., filters or Fourier transforms [Ler12, p. 436]. This processing can be implemented in a power analyzer along with more sophisticated analysis functions such as harmonic measurements or motor evaluation. Further, modern power analyzers provide remote control software to operate the device from a PC. This software can enable additional analyses such as CE conformity testing (see [ZES16; Yok17]).
For more specific analyses, however, the fixed functionality of measurement devices is not always sufficient. Consequently, the measurement results must be available through an application interface. Such an interface is the basis for combining energy measurements with application and system monitoring data. Moreover, monitoring large-scale HPC systems requires many instruments whose data needs to be aggregated. Given that measurements at high resolution and large scale involve large volumes of data, their processing can be challenging.
LabVIEW is a popular software package by National Instruments, which allows the design of *virtual instruments*. It interfaces with a variety of measurement components and provides a graphical programming environment to process and visualize them [Ler12, pp. 651–655]. Keysight VEE and DIAdem are similar systems focused on measuring systems [Par10, pp. 206–207]. With the addition of a Data Acquisition Toolbox and Instrument Control Toolbox, the numerical computing platform Matlab can also be used for processing measurement data [Ler12, pp. 657–659].
With respect to management, storage, and retrieval of data from sophisticated power measurements, there is a large overlap with general monitoring data. The distinction between measurement and monitoring data is not always clear: In this work I refer to monitoring if the focus is a *continuous* observation of a *readily available* metric, whereas measurement is used for the process of *obtaining* the value. As I show in Chapter 5, the use cases for power measurements are diverse. In general, energy efficiency research leverages time-delimited experiments, while operational data analytics requires continuous monitoring. In practice, there are many intermediate scenarios for both power measurement goals.

One particular aspect of handling measurement results is persistent storage for continuous recording. Power analyzers themselves are typically limited to recordings of short experiments in proprietary binary or CSV files (e.g., [ZES16; Yok17]).

2.3.1 Time Series Databases

General purpose database management systems and relational databases are not suitable to efficiently handle large volumes of metric data. In contrast, time series databases, also referred to as time series management systems, are specifically tailored towards sequences of data points with a corresponding and monotonically increasing time (see [JPT17]).
Nowadays, several advanced time series database solutions are in widespread use. This includes open-source systems, such as InfluxDB, KairosDB, OpenTSDB, and TimescaleDB as well as commercial solutions, such as Kdb+ [sol19]. Some multi-model databases also include time series databases as secondary models, e.g., IBM Informix[6]. The focus on time series data allows databases to make certain trade-offs: For instance, existing data is rarely updated or deleted — restricting this functionality allows for a better query and write performance. Furthermore, new data usually carries recent timestamps. This ascending order can be exploited to improve performance for the majority of inserts [Inf19a, InfluxDB design insights and tradeoffs]. In the following, the widely popular InfluxDB serves as an example to discuss typical techniques and aspects of modern time series databases.

Time Series Storage Concepts

Logically, data in InfluxDB is organized by *measurements*, which are identified by strings. Each data record (*point*) for a measurement is further identified by a set of tags, containing metadata as strings. A point comprises a single timestamp and a set of *fields*, each with a *key* and *value*. The tags and field keys can be freely chosen for each new point, making InfluxDB a schemaless database. A series of data is a sequence of values defined by the measurement, tag set, and field key. Moreover, InfluxDB separates data into shards by coarse grained blocks of time to facilitate efficient deletes of older data. The Time Series Index (TSI) allows efficient access to high-cardinality data, i.e., large number of time series.
Primarily, InfluxDB uses a simple text-based line protocol for inserting or updating data points. Moreover, it supports the protocols of CollectD, Graphite, OpenTSDB, and Prometheus. The Influx Query Language (InfluxQL) is a SQL-like language for retrieving data and metadata from InfluxDB. InfluxDB uses a storage engine based on Time-Structured Merge Trees (TSMs), similar to Log-Structured Merge Trees (LSM-trees) [ONe+96]. New or modified data points are collected in a write ahead log (WAL), which is persistent but does not support reads. In parallel, incoming data points are kept accessible in an in-memory cache for reads. When the WAL exceeds the maximum size, its data is flushed to the actual TSM files. The flushed data points are partitioned by non-overlapping time ranges across multiple TSM files, which provide read-only indexed access. Every TSM file consists of multiple blocks of ordered data, each with separate storage for timestamps and values. Regular background compaction processes are responsible for merging several TSM files of one level to one larger TSM file of the next level of the tree. Following the assumptions about time series workloads,

[6] `https://www.ibm.com/products/informix`

this storage scheme provides efficient append-only insertion using the WAL while retaining older data in the compressed and indexed TSM files. However, changes to older data or retroactive inserts are costly as they require rewriting of TSM files (see [Inf19a]).
TimescaleDB follows a different approach, by storing time series data on top of the established relational database management system (RDBMS) PostgreSQL. The most significant difference is to split logical tables (called *hypertables*) into chunks that are implemented as actual PostgreSQL tables. This approach exploits the property of temporal locality of data points. The smaller index structures of active chunks can fit into memory more easily, thus speeding up inserts and queries. From the user perspective, only the logical table is visible. By relying on a traditional RDBMS, TimescaleDB inherits features such as a fully featured query language with joins and user management (see [Fre17]).

Multiresolution Databases

Another prevalent management system for time series databases is RRDtool. This solution has pioneered the use of round robin buffers, hence the name Round Robin Database, to limit the amount of required disk space for time series data. RRDtool aggregates multiple primary data points into consolidated data points by applying a consolidation function. Possible consolidation functions are average, minimum, maximum, and the last value. For computing averages, RRDtool assumes that all values are rates. The different aggregation levels in RRDtool, called round robin archive, are individually defined by the user. There is a constant number of values within a round robin archive, which are overwritten in a circular fashion. A significant limitation of RRDtool for high-resolution measurements is the use of timestamps of one-second granularity (see [Oet17]).
Serra et al. [LVE16] introduced a formalism for a time series database that uses concepts similar to RRDtool. Their approach is to organize the data in an aggregated way such that a time series is stored at different temporal resolutions, hence they name this aspect *multiresolution*. This acts as a *lossy storage* solution. The emphasis of this formal model is to offer a high degree of generality and also consider irregular samples in time series. In addition to the formalism, which is based on relational algebra, they offer a reference implementations in Python. A library named `pytsms` implements the basic concepts and another library named `roundrobinson` implements the multiresolution time series model. However, neither libraries are actively maintained[7].
Andersen et al. [AC16] designed a storage system for time series data named BTrDB. The key data structure of BTrDB is a time-partitionend, multiresolution, version-annotated, copy-on-write (COW) tree. Each node in the tree includes statistical aggregates which enables efficient statistical queries over arbitrary request intervals. Data in BTrDB is ordered by *streams*, which are identified by a UUID. BTrDB allows unordered insertion and coalesces incoming data points before being merged in the COW tree. The COW tree guarantees consistency of both raw streams and derived analytics without the need for journaling. The data of the COW tree itself is compressed block-wise and stored in a Ceph storage pool. The authors demonstrate this concept with an implementation in Go on a demanding use case of synchrophasor telemetry. This use case comprises a deployment of 35 microsynchrophasor devices, each providing 12 streams of power measurements at 120 Sa/s. The system satisfies statistical queries on one year worth of collected data, i.e., 2.1 trillion data points, in less than 200 ms. With an additional throughput benchmark, the authors show that an instance of BTrDB on a four-node cluster

[7] `https://github.com/allusa/tesi/tree/master/src`, no commits since October 2015

can handle up to 53 million inserted values per second. The authors provide their "quasi-production implementation" as free software, however development appears to stagnate[8].

InfluxDB also supports automatic downsampling in order to limit storage requirements [Inf19a, Downsampling and data retention]. On the one hand, a continous query periodically aggregates data from one measurement to another with a lower temporal resolution. On the other hand, retention policies define a duration after which data points of a database are deleted automatically. However, this has to be manually defined for each measurement and the retention policy and down-sampled measurement identifier has to be chosen manually in queries. TimescaleDB uses special precomputed views named *continuous aggregates* to speed up aggregate queries with similar restrictions [Tim19, API reference, Continuous Aggregates].

Time Series Compression Methods

Pelkonen et al. [Pel+15] introduced Gorilla[9], an in-memory time series database that scales to billions of unique time series and implements advanced compression techniques. For timestamps, the authors observed that the majority of data points arrive at fixed intervals, affected by a limited amount of jitter. Hence, timestamps are encoded as delta-of-delta, i.e., $D = (t_n - t_{n-1}) - (t_{n-1} - t_{n-2})$. This yields the jitter values, which are stored with a variable-length encoding that uses less storage for smaller values. However, Gorilla only supports second-precision timestamps, which can be encoded very efficiently but limit the applicable use cases. For values, Gorilla uses double-precision floating point numbers and applies another encoding scheme. This scheme exploits common cases where the exponent and the first few bits of the mantissa of successive values are identical.

Other databases have adopted part of this encoding approach for compression. InfluxDB, for example, uses the same encoding for floating point values, while timestamps are encoded with a combination of delta encoding and scaling [Inf19a, In-memory indexing with TSM]. BTrDB uses a similar encoding based on delta-of-delta applied to the timestamps as well as separately handling the exponent and mantissa of floating point values [AC16, Sec. 5.4].

Approximate Query Processing

Requests that cover large amounts of data, such as computing the average of a high-resolution time series over a long period of time, can be difficult to answer efficiently. To overcome this performance challenge, several techniques for approximate query processing (AQP) have emerged. Executing queries on samples of the full data set can speed up the processing at the cost of reduced accuracy [Ach+99; BCD03]. Chakrabarti et al. [Cha+01] suggested to use wavelets to build an approximate representation of large data sets that can provide more accurate approximate answers than sampling. Time series data is a good candidate to apply such signal processing techniques [Cha+11]. Perera et al. [Per+15] described a general architecture that uses models of time series data for AQP. Their models allow for average and sum queries with given error bounds and include aggregation over time.

[8] `http://btrdb.io/`, `https://github.com/BTrDB/btrdb-server`, no commits since February 2019

[9] Later, Gorilla was renamed to Beringei.

2.3.2 Data Center Monitoring Systems

Persistent storage is only one part of managing power measurement data. Leveraging measurement data for a comprehensive analysis also requires data collection, reporting, processing and visualization. In that regard, power measurements are similar to metric data monitoring in general and data center monitoring in particular. Data center monitoring serves as a foundation for operational data analytics and often includes distributed power measurements.

Prevalent Open-Source Monitoring Solutions

The InfluxDB software ecosystem leverages the TICK stack comprising Telegraf, InfluxDB, Chronograf, and Kapacitor. *Telegraf* is an agent for collecting and reporting metric data centered around computing systems. Due to its plugin architecture, it can collect a wide range of metrics with a particular focus on statistics of server software. *Kapacitor* provides functionality for real-time data processing such as anomaly detection and alerting. Finally, *Chronograf* serves as a front-end for visualization (e.g., real-time dashboards) and configuration (e.g., setup of monitoring and alerting) of the TICK stack (see [Inf19b]).
Prometheus[10] is another widely-used open-source monitoring toolkit. In a Prometheus system, *Exporters* expose metrics from within server applications. Each measurement point contains a millisecond-resolution timestamp and a double-precision floating point value. The central Prometheus server regularly pulls metrics from Exporters via a text-based protocol on top of HTTP. A push model is indirectly supported by the use of *Pushgateways*. Prometheus includes a local on-disk time series database, but can also use other dedicated time series databases such as InfluxDB or TimeScaleDB. The community around Prometheus also develops an enhanced set of components called Thanos[11], which adds support for automatic down-sampling to Prometheus. Based on the pulled metric data, the Prometheus server can push alerts to an *Alertmanager*. Stored data is accessed via the Prometheus Query Language (PromQL), which also enables the Prometheus web UI as well as visualization with Grafana.
Grafana[12] is an open-source platform for analyzing and visualization of monitoring data. It can be extended with data source plugins that leverage many different databases, e.g., InfluxDB or Graphite. Grafana is used for metric data visualization, in particular for building dashboards by many of the monitoring solutions mentioned in the following paragraphs.

Custom Monitoring Solutions and Use Cases with Research Focus

Libri et al. [LBB18] described a monitoring solution called DiG (Dwarf in a Giant) (see also Section 2.2.2). They aim to provide scalable high-resolution monitoring for data center analytics, automation and control. As discussed in Section 2.2.2, DiG leverages sophisticated node-level power measurements. DiG provides out-of-band data collection of power measurements and other telemetry. By performing analysis functions within the data collection nodes itself, DiG reduces the data volume that needs to be passed to centralized monitoring. This edge analytics is demonstrated by performing

[10] Prometheus — From metrics to insight: `https://prometheus.io`
[11] Thanos — Open source, highly available Prometheus setup with long term storage capabilities: `https://thanos.io/`
[12] Grafana — The open platform for analytics and monitoring: `https://grafana.com/`

a Fourier analysis as an example of feature extraction. DiG leverages ExaMon [Lib+18a; Ben+17] for node-level data collection and centralized data analytics. ExaMon includes a *collector* daemon that covers CPU performance and energy counters. These node-level metrics, and pre-analyzed measurements from the DiG agents are published to MQTT message brokers. An MQTT subscriber implementation then inserts data into a KairosDB time series database on top of Cassandra. In the D.A.V.I.D.E. cluster, this centralized monitoring solution collects a total of 14 940 metrics at an aggregate rate of ≈ 47 kSa/s [Bor+18]. The ExaMon software is available under an open-source license[13].

Netti et al. proposed the concept and implementation of DCDB (Data Center Data Base) [Net+19]. DCDB includes data sources on compute nodes and from facility sensors. *Pushers* collect measurements and publish them via MQTT. *Collect Agents* subscribe to these measurements and write them to *Storage Backends* based on Apache Cassandra databases. Regarding performance, the authors focused on evaluating the impact of the in-band data collection overhead on compute nodes. For the Collect Agent performance, the authors reported a CPU usage of 900 % in a scenario with an aggregate sensor reading rate of 500 kSa/s (50 pushers reporting data for 10 000 sensors at 1 Sa/s each). These numbers are based on a synthetic pusher plugin and do not include persistent data storage. In addition to publishing the measurement data, Pushers also provide a REST API through HTTPS. This REST API can be used to access local sensor caches and configure the pushers. The authors further described a DCDB installation at the Leibniz Supercomputing Centre (LRZ) and a use cases for evaluation of the data center cooling system. In this installation, sophisticated infrastructure sensors are collected out-of-band and then refined into aggregated metrics using *virtual sensors*. These system-level metrics provide information about the total efficiency of heat removal in the liquid-cooled HPC system. DCDB is publicly available as open-source software[14].

Bautista et al. introduced OMNI (Operations Monitoring and Notification Infrastructure) [Bau+19]. OMNI is deployed at the National Energy Research Scientific Computing (NERSC) center at Lawrence Berkeley National Laboratory and uses RabbitMQ, Logstash, Elasticsearch, and Grafana to process and visualize metric data from a heterogeneous set of distributed sources. The authors described that the system can currently ingest data up to an aggregate rate of 25 kSa/s. The OMNI installation at NERSC provided valuable insight for multiple use cases. One such use case was the provisioning of power infrastructure for a future HPC system based on long-term recordings of actual power requirements. Figure 2.3 shows an example Grafana Dashboard based on OMNI.

Vazhkudai et al. showcased GUIDE (Grand Unified Information Directory Environment) [Vaz+17] and its deployment at the Oak Ridge Leadership Computing Facility (OLCF). GUIDE ingests metric data and syslog streams from storage systems, schedulers, the interconnect, and compute nodes of HPC systems. In a pre-processing stage, logs are cleansed and metric data is statistically analyzed and categorized. The variety of data is then ingested in a central Splunk instance, which provides federated storage, indexing, querying, and visualization. The authors described the operational impact by the means of several use cases covering storage systems, resiliency, scheduling, interconnect usage, and archival storage.

[13] `https://github.com/EEESlab/examon`
[14] `https://dcdb.it/`

Figure 2.3: Partial display of a power dashboard for the Cory system based on OMNI [Bau+19, Fig. 6].

Dataheap [KHN12] is a distributed system for processing metric data. While Dataheap is aimed at general performance data from computer systems, it can also handle measurements from physical instruments — such as power-meters. Dataheap is capable of handling metric data in the order of millions of updates per second and persistent storage as well as access libraries to expose live and historic data to general purpose programming languages. Derived metrics can be defined in the system, allowing combinations such as GFLOP/s per watt from current and voltage measurements as well as application-based performance data. A fundamental limitation of Dataheap comes from the use of timestamps in millisecond resolution. Among many others, the author of this thesis was also involved in the development of Dataheap.

Applicability of Time Series Databases and Monitoring Solutions for Power Measurements

On the one hand, OMNI and GUIDE represent site-specific configurations and combinations of generic software. On the other hand, DCDB, ExaMon, and Dataheap are publicly available monitoring software frameworks that build on existing open-source software. All of the discussed custom monitoring solutions are used in part for processing data from power measurements — even DCDB, which is more focused on in-band node-level measurements. These HPC data center monitoring solutions, as well as the more general open-source monitoring frameworks and time series databases, all demonstrate that they can handle a high cardinality of data. The predominant monitoring use case requires large numbers of measurands, typically in the range of tens of thousands or even up to billions with Gorilla. However, none of the readily available solutions actively address the processing and storage of high-resolution measurements beyond 1 Sa/s. For example Prometheus only supports timestamps at millisecond resolution and its pull-based architecture is not well-suited for collecting continuous high-resolution sensor data. Other solutions support high-resolution measurement in general, but are limited by performance. As an exception, BTrDB is specifically designed for higher-resolution measurements at 120 Sa/s and demonstrates efficient insertion and queries. However, BTrDB is not widely adopted or actively developed.

2.4 Influences on the Energy Consumption of Computing Systems

Performing computations with as little energy as possible has challenged system designers, operators and researchers for a long time. A classic example are all forms of mobile computers that have to make due with batteries where a reduced power consumption translates to a longer device operation. With increasing frequencies, energy efficiency has also become an important aspect for the design and operation of high performance systems. There is a strong link between the energy-efficient operation of computer systems and their energy measurement and monitoring. Optimized execution needs metrics to identify inefficient operations and as confirmation feedback for tuning. The increasing focus on energy efficiency, energy proportionality, and adaptivity yields complex, sophisticated systems with interactions that are challenging to understand and model accurately.

2.4.1 Processor Power Consumption Breakdown

A fundamental model for CMOS processor power consumption is given by Weste and Harris in [WH11, Sec. 5]. The total processor power comprises dynamic and static power:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} \tag{2.21}$$

Dynamic power is dissipated by the switching of transistors and short phases of short circuit during the transition:

$$P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short circuit}} \tag{2.22}$$

The switching power depends on the total load capacitances C that need to be charged and discharged during switching. However, not all transistors are always switching with the frequency f. Thus an activity factor α between 1 (a clock) and 0 (a clock-gated circuit) is introduced:

$$P_{\text{switching}} = \alpha C V^2 f \tag{2.23}$$

Most notably, switching power scales quadratically with voltage — this is a strong incentive to keep voltage minimal for reducing power. Many factors impact the value of α for active compute components in a system, e.g., the types of executed instructions, the location for data transfers, and even the number of bits within processed values [Mol+10; Sch+19]. Additional dynamic power is dissipated due to phases of short-circuits, which is normally less than 10 % of dynamic power.

$$P_{\text{short circuit}} = I_{\text{short circuit}} V \tag{2.24}$$

Static power is dissipated independently of the frequency and switching activity. It comprises different leakage currents (subthreshold, gate, and junction) as well as the contention current in ratioed circuits.

$$P_{\text{static}} = (I_{\text{leak}} + I_{\text{contention}}) V \tag{2.25}$$

This breakdown serves as a basis for reducing processor power consumption as discussed in the following section.

2.4.2 Energy-Efficient Hardware Configuration

Different techniques for reducing the energy usage in contemporary systems tackle the different components of dynamic and static power (2.21). The following coarse classification in sleeping, power, performance, and throttling states is given by the Advanced Configuration and Power Interface (ACPI) standard [Uni16]. The different kinds of states typically align with certain power reduction techniques.

G-states Global system states include different power-off states that remind us that it is always an option to turn off a computer when not in use. However, all state is lost in G2/G3 off states and a large latency is required to return to a working state. Reducing transition (boot) times may facilitate to shut down a computer more often.

S-states Sleeping and soft-off states are global to the system and provide a very coarse grained control to stop operation of a computer without necessarily losing all context. S-states drastically reduce power consumption by powering off many components in a system. There is overlap between G-states and S-states in terms of the soft off state G2/S5. Sleep states or power-off states can by used by high-performance clusters that use resource managers that can put unused compute nodes to sleep and wake them up accordingly [Sch19; Pur+18; Mai+18]. The energy savings can be improved by consolidating idle resources and powering down unneeded rack-level components [PC11].

C-states Processor power states provide a more fine-grained control. C-states are used when individual cores of a multi-core system are not actively used for certain periods of time. Resuming the active operation from C-states is fast, i.e., in the order of 1 µs to 100 µs. C-states reduce dynamic power by disabling the frequency connection for parts of a chip (*clock gating*). Some C-states further reduce static power by disabling the power input for parts of a chip (*power gating*). In order to use C-states efficiently, and model them in terms of performance and power consumption, one needs to know the power consumption of the processor during the states and during transitions as well as the latency for transitions. Schöne et al. [SMW14] showed that practical latencies differ significantly from the specifications. While ACPI defines C0 (active) and C1 through C3 (inactive with increasing latency and decreasing power consumption), contemporary processors offer more C-states and distinguish between core C-states and package C-states [GSS15]. The latter are used when all cores of a processor package are idle and provide increased power savings. The specific used package C-state depends on the lowest of its core C-states. Particularly for contemporary systems, core C-states and package C-states provide vastly reduced power when a processor or individual cores are not in use. In Section 5.3.1 I analyze an anomaly in which an incorrect use of C-states causes a significant increase in energy consumption. Exploiting C-states is important as many processors do not always have work scheduled in practice. Even in parallel HPC applications, tasks can wait on communication, on other tasks during load-imbalances, and on I/O. Due to the transition cost, C-states are not applicable to very short inactivity times, e.g., less than 1 µs for a core waiting for data from memory.

P-states Processor and device performance states control performance and power consumption of active processors, processor cores, or devices. ACPI P-states are defined to be generic for devices — examples include the monitor brightness level and the maximum audio volume. The most significant P-states are those for processors. Processor performance levels are implemented using dynamic voltage and frequency scaling (DVFS). This technique reduces power consumption as per (2.21) by changing frequency and voltage. The necessary minimum voltage for stable operation depends on the selected frequency. Therefore typically only the frequency is actively varied while the appropriate voltage is selected by the firmware.

DVFS is widely used to improve the energy efficiency of applications, and is also the underlying technique for GPU P-states [GSS15]. P-states are used when a processor is continuously active, thus C-states are not applicable, but the full performance is not necessary. A common use case are memory-bound workloads on systems with a frequency-independent memory performance [SHM12]. In this case, reducing the processor frequency through P-states does not decrease performance, but reduces power consumption and thus consumed energy. Due to the quadratic impact of voltage on dynamic power (2.23), the fastest performance state often uses disproportionately more power given their performance. Thus, slower performance states often require less total energy to complete a task.

As with C-states, contemporary systems have more complex P-states than specified by ACPI. P-states are selected for individual cores or hardware threads, but some processors will apply the lowest of all per-core P-states to the entire processor package. Hackenberg et al. [Hac+15] described the change from per-package to per-core P-states in the Intel Haswell processor generation. While per-core P-states allow more flexibility, they introduced a significantly increased variable latency for P-state transitions. Contemporary processors also expose uncore P-states, which the frequency that is applied to shared resources and for instance affects shared cache and memory performance [KHL18].

Originally, P-states are used to select processor frequencies between the nominal and minimum frequency. Contemporary processors, however, also allow frequencies above the nominal frequency, e.g., with Intel's Turbo Boost Technology or AMD's Turbo Core. In these turbo modes, an internal controller dynamically decides on the actual frequency based on a number of factors. The actual frequency is chosen to not exceed the TDP, except for a short time, although there may be further exceptions. Further, the maximum possible frequency depends on the number of active cores, i.e., in a certain C-state. Beginning with the Haswell-EP microarchitecture, high-performance Intel processors further use different sets of frequencies depending on the usage of AVX instructions [Int17a]. For example, an Intel Xeon Platinum 8180 uses a nominal frequency of 2.5 GHz. For AVX-512 workloads, however, the guaranteed frequency is as low as 1.7 GHz, while the turbo frequency for non-AVX workloads on up to two active cores goes up to 3.8 GHz (see also Figure 1.1). Another impact of this development affects large clusters of processors that execute a parallel application: Manufacturing variability that previously caused power variations at constant performance, now lead to a variation in performance at constant (peak) power [Sch+16b].

In particularly power constrained environments, DVFS can also be used to limit the power consumption below the TDP. Power limiting is controlled via the RAPL interface (see also Section 2.2.4) rather than P-states. This can be exploited to achieve good performance for hardware that is over-provisioned with respect to power consumption [Pat+13]. The Global Extensible Open Power Manager (GEOPM) provides global rebalancing of power budgets within globally power-limited HPC systems [Eas+17].

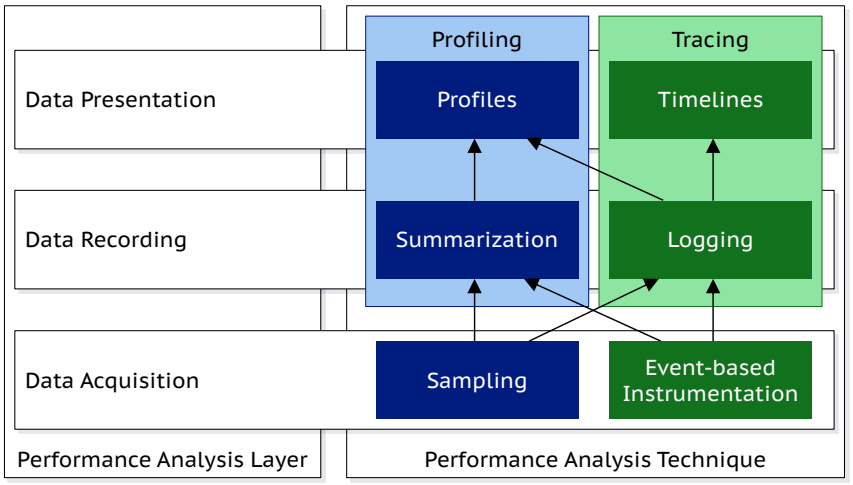


Figure 2.4: Classification of performance analysis techniques (based on [Juc12]).

T-states Processor throttling states present an additional mechanism to reduce processor power consumption during active operation. T-states are typically implemented using clock modulation. With clock modulation, the effective clock signal is partially disabled in repeating intervals. In [Sch+16a], my co-authors and I provide an in-depth evaluation of T-state implementations on contemporary processors. A particular finding is that certain processors can use DVFS rather than clock modulation to implement T-states. Further, we provide a detailed breakdown of the transitions and states for active clock modulation.

2.5 HPC Performance and Energy Analysis

Performance analysis is a crucial aspect of High Performance Computing to ensure that expensive hardware is used efficiently. It is essential to consider the existing approaches for performance analysis in order to best exploit the information from energy measurements.

2.5.1 Performance Analysis Techniques

The following section uses the classification of performance analysis techniques that I have proposed together with my co-authors in [Ils+15b]. An overview of the techniques along theses stages is given in Figure 2.4. The core process of performance analysis consists of three stages: *data acquisition*, *data recording*, and *data presentation* [Jai91].

Data Acquisition

The first step for performance analysis is to acquire information about the execution of an application or a running system. This data acquisition can be done in two ways: *event-based instrumentation* and *sampling*.

With (event-based) instrumentation, an application is modified to emit certain events during its execution. These events can be generic such as function calls and returns, or specific such as communication or I/O operations. The specific events often carry additional semantic information, e.g.,

communication partners. There are different levels at which the instrumentation can be applied. Some instrumentation techniques require recompilation of an application while others work dynamically during execution. Event-based instrumentation is also applicable at a system level, e.g., to expose system calls or power state changes.
Contrary, sampling exposes the state of an application or system at specific intervals. On the one hand, sampling intervals can be time-based, e.g., one sample every 10 ms. On the other hand, sampling also uses fine-granular countable events, e.g., one sample every 1 million executed instructions. Sampling typically considers the current instruction pointer or call-path as an exposed state from applications. At the system level, hardware performance counters or state such as the current processor frequency is typically collected during a sample.
While instrumentation exposes every occurrence of a certain state change, sampling only provides a statistical aspect of the state. However, the overhead of instrumentation depends on the actual rate of events that are occurring. This event rate is intrinsic to the observed workload itself. Sampling, in contrast, allows a predictable and controllable overhead by means of selecting the sampling interval.

Data Recording

Logging is a data recording approach that preserves the full information when recording performance data. In addition to the event or sampled state, the log also includes a timestamp. Logging at a high event rate requires either a substantial amount of memory or causes disk I/O that can perturb the execution.
Instead of collecting all events and samples, the exposed information can also be recorded in a summarized form. For example, *summarization* counts the number of times a function has been entered, the total time it was executed, or the number of times it was on the call-stack during a sample. Summarization requires less memory for collecting the reduced information during execution.

Data Presentation

For presenting the recorded data to a user, there are also two basic variants. *Timeline* displays use a linear time axis, typically the x-axis, at which the collected events and samples are displayed. There are many variants to display events and samples alongside the time axis. For example, the executed functions can be shown color-coded with the y-axis being used for either the call-stack or level of parallelism. Timeline displays can also show metric data with quantitative y-axes or as value-color-coded heat maps. Due to the variety of displayed information, timeline charts often have multiple y-axes.
Alternatively, basic *profiles* show summarized information with no temporal context. For example, a profile can display the number of invocation or the execution time grouped by functions within an application. More sophisticated variants of profiles allow grouping by the call path and include metric information. At a system level, a profile can, for example, list the residency times of each C-state.
On the one hand, timelines require logging for data recording. This combination is referred to as *tracing*. On the other hand, summarization always results in a profile. Hence, summarization is often simply referred to as *profiling*. However, a profile can also be generated from logging by summarizing the data during the presentation step or a post-execution analysis step.

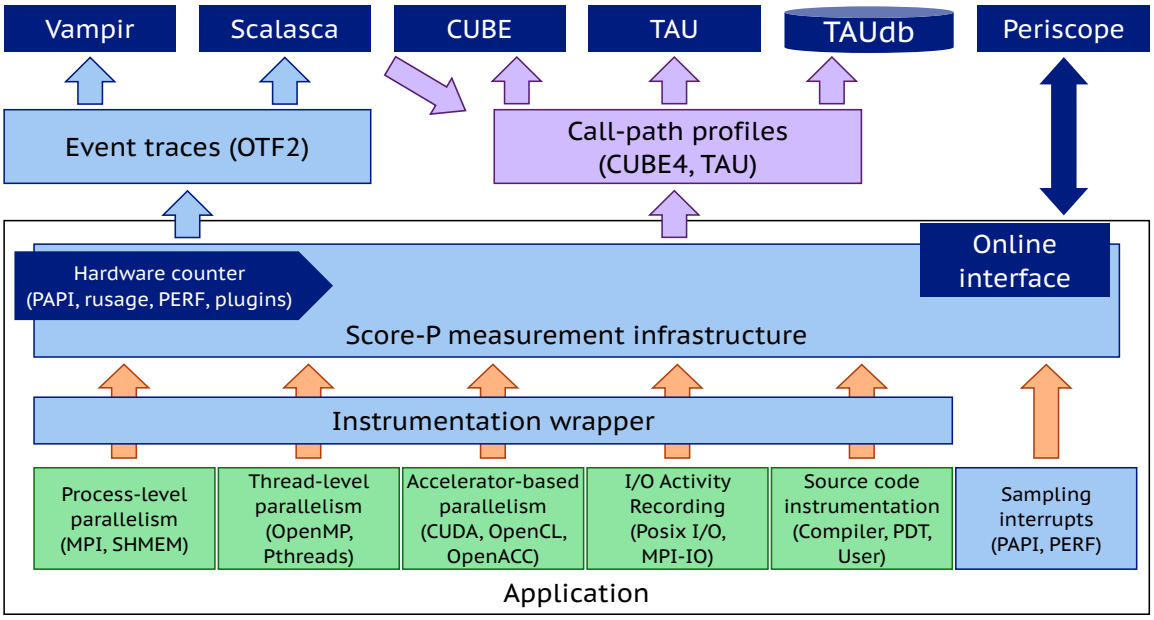


Figure 2.5: Overview of the Score-P architecture, interfaces, and connection to tools for data presentation (adapted from [RWT+19]).

2.5.2 HPC Performance Analysis Tools

There are numerous tools available that implement the previously mentioned techniques, often as a combination of several methods. The following performance analysis tools focus on HPC use cases. Therefore, all of these tools support parallel applications using shared memory and message passing, at least with OpenMP and MPI, respectively.

TAU (Tuning and Analysis Utilities) [SM06] implements all stages of performance analysis. TAU primarily uses instrumentation but also supports a hybrid data acquisition mode that combines sampling and instrumentation [Mor+10]. While TAU primarily uses summarization, it can also create traces. ParaProf supports the visual analysis of TAU profiles. The Extrae[15] software package also implements several instrumentation techniques as well as sampling based on interval timers and hardware counter overflows. To enhance the resolution of information about repeated code regions, Servat et al. [Ser+12] suggested folding of samples from multiple instances of a code region. Traces collected by Extrae can be visualized with Paraver[16]. HPCToolkit [Adh+09] leverages sampling with refined stack unwinding to collect information from parallel applications. It focuses on profiling but also supports tracing, each with a respective interactive visualization tool [Tal+11].

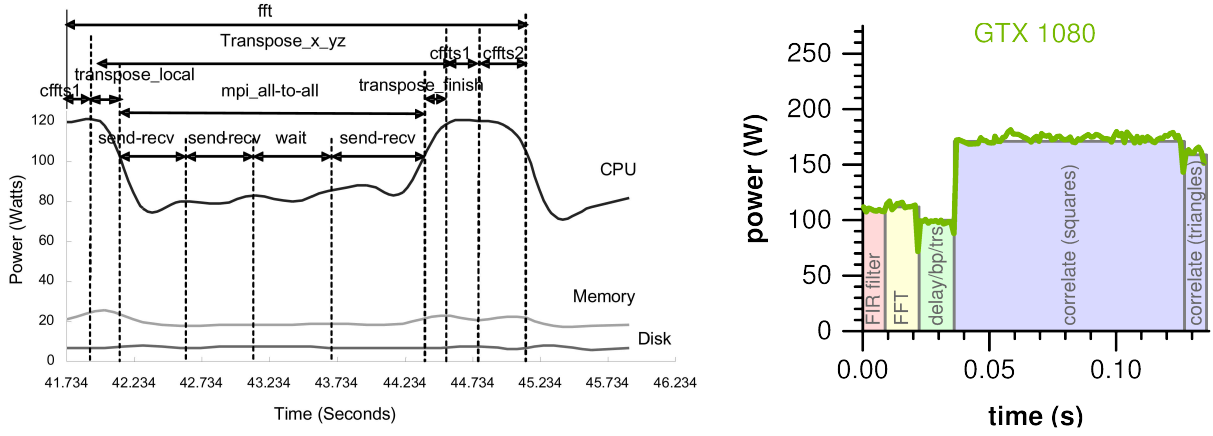
Another performance monitoring infrastructure with a wide range of applications is Score-P [Knü+12]. Figure 2.5 shows the modular design of Score-P, which allows various instrumentation techniques and passes all events to logging for traces, to summarization for call-path profiles, or both. Score-P also supports sampling based on PAPI or perf overflow interrupts as well as collecting metric information from several interfaces. On the one hand, the Vampir [Knü+08] visualization tool provides interactive visualization of the OTF2 traces generated by Score-P both as timelines and as profiles. On the other hand, CUBE [Sav+15] can visualize the call-path profiles that are generated by Score-P itself.

[15] `https://tools.bsc.es/extrae`
[16] `https://tools.bsc.es/paraver`

To complement the existing tools, I developed `lo2s`(Linux Otf2 Sampling) together with my co-authors. `Lo2s`primarily leverages the perf infrastructure of the Linux kernel, which provides sampling capabilities and exposes a wide range of system-level events. In general, the Linux kernel logs the exposed information independently into a user-provided memory buffer. Given sufficient hardware support, the processor itself performs the logging. This shift from the logging in userspace to the kernel or processor helps to reduce the perturbation of monitoring. Whenever the memory buffers are full, `lo2s`flushes the collected traces to OTF2. Thus, `lo2s`can be used in conjunction with OTF2 compatible trace visualization tools, in particular, Vampir. Contrary to the other discussed tools, `lo2s`operates only on a node level. While it supports thread or process parallelism, it is not aware of OpenMP or MPI. However, `lo2s`adds the crucial system aspect to performance analysis that is not well supported by other tools. For instance, `lo2s`can natively record the selection of C-states and P-states by the operating system (OS), which, as I described in Section 2.4, has a significant influence on computing system power consumption. Other system-oriented tools lack parallel logging mechanisms and scalable visual analysis (see [Ils+17]).
In Chapter 5, I leverage Score-P and `lo2s`for combining energy measurement and application monitoring as well as Vampir for visualizing resulting traces. Together, these tools cover a comprehensive range of performance analysis scenarios of massively parallel application to sophisticated system event logging. Further, both monitoring frameworks are extensible to integrate power measurements. OTF2 as a format and Vampir for visualization support storage and analysis of generic metric data and, therefore, power measurements.

2.5.3 Combining Application and Power Measurements

Section 2.2 discussed how power measurements are commonly used in an isolated form, often in terms of logging and plotting timeline charts. The combined information from application performance recordings and power measurements can yield more insight.
Grant et al. [Gra+17] classified HPC related power measurements on different levels. Level 1 describes a summarization on a job-level granularity without any insight into the temporal variations. Further, they classify isolated periodic sampling of power consumption and timelines with only power consumption as level 2. At level 2, phases in the power consumption can be identified but not easily correlated with non-power information. An important distinction is whether the sampling is performed in-band or out-of-band. In-band sampling on the system under test itself can negatively impact the system performance and energy consumption. Due to synchronization within parallel applications, even a small local impact can cause more significant overall slowdowns. Level 3 leverages application instrumentation — either manually or by leveraging automated tools. At each event triggered by the instrumentation, power or energy measurements are collected. This approach can exhibit a high overhead if the instrumented regions are short. Finally, level 4 combines and correlates information from the previous levels. This level includes logging power consumption samples out-of-band while recording application events in-band and correlating the two logs by timestamps.
In [Ils09], I categorized three different approaches to include external power measurements in application traces. This work is based on an existing, sophisticated application monitoring system and then integrates power measurement samples. In particular, I use VampirTrace [Knü+08] for application monitoring and Dataheap [KHN12] for managing power measurement data, but the



(a) Trace from PowerPack [Ge+10].

(b) Trace from PowerSensor 2 [RV18].

Figure 2.6: Timeline presentations from combined application and power measurement.

classification is general. The *pull* technique follows the typical approach for reading hardware performance counters whenever the monitoring system records an event in the application. With the *push* technique, incoming power values are sent to the measuring system asynchronously for being recorded into the trace. The *post-mortem* approach buffers measured values in an intermediate system and transfers the recorded power measurement after the experiment. The latter approach has several advantages over the others: Most importantly, by fully leveraging out-of-band measurement and data recording, there is no additional computation on the system under test during the experiment. Thus, there is no perturbation that scales with the amount of measurement data. Only the *post-mortem* approach allows high measurement sampling rates without compromising the accuracy of the application measurements. The *push* and *pull* approaches often utilize the clock of the system under test for adding timestamps to power measurements. This delayed timestamping introduces an error due to network latency and jitter (*push* and *pull*) and the age of the most recent measurement (*pull*). With *post-mortem* integration, timestamps from the power measuring system are used, which requires synchronization of the respective clocks (see also Section 5.2). In terms of the classification by Grant et al., the *push* and *post* techniques correspond to level 3, while *post-mortem* corresponds to level 4. Dolz et al. [Dol+15] presented timeline charts recorded with ArduPower, although they refer to it as power profiles. Their charts include multiple power measurements of components of a computing system on a shared power-axis. In these charts, coarse-grained executed applications, which run in regular intervals, are marked. As shown in Figure 2.6, both PowerPack [Ge+10] and PowerSensor 2 [RV18] allow similar timeline power charts but augment them with application phases. With custom APIs, users can manually instrument application code in order to expose the application phases for logging. PowerSensor 2 collects GPU power measurement results in-band by logging time and power consumption to an ASCII file. Contrary, PowerPack logs power data out-of-band to avoid perturbing the system under test. The application events are collected in a separate log. A post-mortem data analysis software merges both logs with the help of the collected timestamps. The authors also use data collected with PowerPack to create Energy profiles grouped by different process configurations and degrees of parallelism. PMLib [Alm+18] further leverages Extrae for automated application instrumentation. Using the trace visualization tool Paraver [Pil+95], users can create combined timelines, which include application events and power samples.

2.6 Conclusion

In summary, the challenge of measuring the energy consumption of HPC systems can be approached from two sides. On the one hand, there are well-established practices for power measurements and metrology in general. However, the general measurement approaches do not consider the specific properties of computing components as power consumers. On the other hand, contemporary measurement approaches specific to HPC systems and their components provide only limited resolution and are often not well understood in terms of their uncertainty.

With this dissertation, I address this gap by designing energy measurement solutions with a particularly high temporal resolution. In Chapter 3, I use these measurements to reveal the properties of the power consumption signal at different possible instrumentation points. This understanding is crucial for the design of new power measurement solutions for computing systems and validating assumptions about existing ones. I target both measurements of single nodes as well as a scalable HPC measurement solution. To further strengthen the quality of HPC energy measurements, I provide a rigorous evaluation of all discussed measurements.

With respect to processing and storing measured values, existing infrastructures and time series databases are well-suited for high-cardinality data, i.e., many different measurands. High-resolution measurement data, however, is not well supported. In addition to ingesting a high rate of measured values, the retrieval of information covering long time intervals presents a performance challenge for contemporary monitoring infrastructures. Approximate query processing techniques can improve performance but reduce accuracy. Consequently, in Chapter 4, I present a novel infrastructure for processing power measurement results, which I designed for both high-resolution and large-scale measurements of HPC systems. This infrastructure includes a time series database that allows efficient ingestion and retrieval. For this database, I exploit a new hierarchical storage concept that improves on existing techniques for down-sampling.

The state of research offers established concepts for performance analysis of HPC applications and systems alongside a range of tool implementations. These concepts provide a solid foundation for introducing energy measurements. However, the temporal resolution and scalability of current solutions is limited. Moreover, current approaches for combined application and power measurement often require manual application instrumentation. Hence, in Chapter 5, I leverage existing tools for application performance analysis and integrate power measurements to provide fine-grained and scalable energy efficiency analysis. In this context, I focus on tracing rather than profiling, as it preserves the crucial temporal information and benefits particularly from high temporal resolutions.

3 Evaluating and Improving Energy Measurements for Computing Systems

Energy measurements are a crucial prerequisite for energy efficiency analysis, energy-efficient operation, energy accounting, and power limiting. All of these use cases rely on the accuracy of provided energy and power values. Particularly, energy-optimization decisions, both at run-time and design time, may be wrong if the underlying measurements are flawed. In this chapter, I present a systematic approach to evaluate energy measurements for computing systems along various aspects. For each aspect, I discuss improvements over the state of the art and present practical results.
In Section 3.1, I introduce various systems under test, ranging from single node server systems to a large-scale production HPC system. These systems are instrumented with different energy measuring systems, which I describe and evaluate throughout this chapter. I discuss the different aspects for measurements roughly following the measurement signal from analog instrumentation to digital analysis. Section 3.2 describes the first parts — possible instrumentation points and sensors for energy measurements of the aforementioned systems under test. The next step, processing of the analog power measurement signal and its digitization, is covered by Section 3.3. In Section 3.4, I present an in-depth accuracy evaluation of the different measuring systems covering theoretical as well as practical aspects. Section 3.5 discusses various aspects of enabling a high temporal resolution for power measurements. Such a high resolution increases the information detail in measurement results and reveals dynamic behavior of computing systems down to tens of microseconds. Temporal resolution of measurements is also closely related to the question whether a power measurement can be used to compute accurate energy values. Finally, in Section 3.6 I discuss and evaluate CPU energy counters, specifically RAPL, as an alternative to dedicated power measurements.

3.1 Description of the Systems Under Test

I discuss the different aspects of energy measurements on the basis of various systems under test. All of the systems under test use x86-64 processors and are focused towards high performance, either as single node server systems, or being HPC clusters. The hardware specifications of the different systems are listed in Table 3.1 and Table 3.2 for the single-node and HPC systems respectively.
My contribution to the different measuring systems are as follows. The measuring system for *apollo* and *artemis* was developed by a third party whereas I present an in-depth evaluation regarding accuracy and temporal behavior of the different sensors. My experiences with *apollo* also influenced the design of the follow-up measuring system *artemis* for which I specified the requirements and instrumentation points, while the assembly of instrumentation harnesses and measurement amplifiers was done by a contractor. I initially published the description and evaluation of both *apollo* and *artemis* in [Ils+15a] and [Ils+18c]. The modular measuring system of *artemis* was further transferred

Table 3.1: System specifications for the single-node systems under test.

	apollo	*artemis*	*diana*	*ariel*
Processor Model	AMD Opteron 6274	Intel Xeon E5-2690	Intel Xeon E5-2690v3	Intel Xeon Gold 6154
Microarchitecture	Bulldozer	Sandy Bridge-EP	Haswell-EP	Skylake-SP
Processor TDP	115 W	135 W	135 W	200 W
Cores	$3 \times 16 = 48$[a]	$2 \times 8 = 16$	$2 \times 12 = 24$	$2 \times 18 = 36$
Memory	48 GiB	64 GiB	256 GiB	384 GiB

[a] Since one of the four hardware sockets was damaged, the system is running in a thee-socket configuration.

Table 3.2: System specifications for the HPC multi-node systems under test.

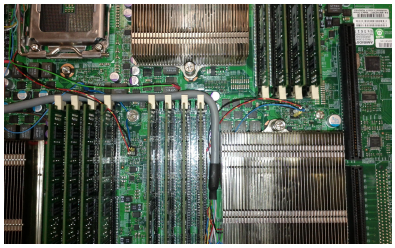
	taurus (phase 1)	*taurus* (Haswell)
Processor Model	Intel Xeon E5-2690	Intel Xeon E5-2680v3
Microarchitecture	Sandy Bridge-EP	Haswell-EP
Processor TDP	135 W	120 W
Cores per Node	$2 \times 8 = 16$	$2 \times 12 = 24$
Memory per Node	32 GiB to 128 GiB	64 GiB to 256 GiB
Nodes	270	1456

to a new measuring system — *diana*. Therefore, some of the evaluations for this measuring system are performed on *artemis*, while others are performed on *diana*. The most recent system under test is *ariel*, for which I designed the measurement harnesses. A workshop at TU Dresden built the harnesses to my specification and the sensors and digitization are part of an integrated commercial power analyzer.

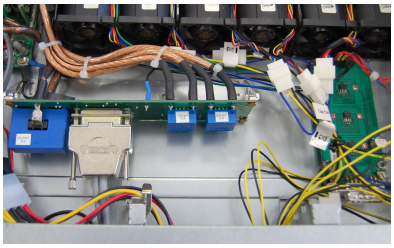
The High Definition Energy Efficiency Monitoring (HDEEM) project is a vendor collaboration between BULL (now part of Atos) and TU Dresden to enable high quality energy measurements for production HPC systems. HDEEM is available on the *taurus* HPC system deployed at TU Dresden. The *taurus* phase 1 was an earlier smaller system which employed Intel Sandy Bridge generation processors and less sophisticated energy measurement capabilities. While *taurus*, as a whole, includes phase 1 and also other types of specialized nodes, references to *taurus* in this chapter refer to its Haswell nodes with full HDEEM capabilities unless noted otherwise. My contribution to HDEEM is in designing, implementing, conducting, and analyzing the measurement evaluation. This includes accuracy, energy correctness, and functional aspects that were also part of the procurement acceptance test. Further, my evaluation and specific feedback enabled the vendor to incrementally improve the measurement solution. I described the initial prototype and general approach in [Hac+14], whereas I covered HDEEM and its production implementation in [Ils+18c].

3.2 Instrumentation Points and Measurement Sensors

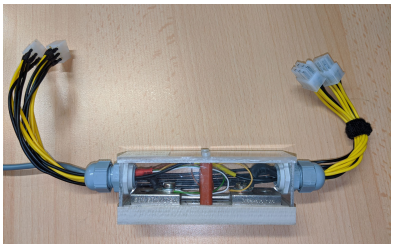
A fundamental choice for any measurement is the point of instrumentation in the power conversion chain (see also Figure 2.2). The choice of the instrumentation point also implies the measurement domain and has some influence on the applicable power measurement sensors. While the theoretical background is given in Section 2.1.2, I describe the practical aspects of several instrumentation approaches in the following sections. First, Sections 3.2.1 through 3.2.3 consider custom single-node instrumentation with a focus on high temporal resolution, while Section 3.2.5 describes HDEEM as an example of a scalable instrumentation solution for many-node production systems.



(a) Instrumentation at VRs



(b) Hall effect sensor board



(c) Shunt-based harness

Figure 3.1: Different instrumentation approaches used in the *apollo* and *artemis* systems under test.

3.2.1 Analog Measurement at Voltage Regulators

The *apollo* system uses two levels of instrumentation described in this and the following section respectively. The first level is close to the consuming components and utilizes the voltage regulators of the mainboard. Instrumenting at the level of voltage regulators allows access to more fine-grained power domains, i.e., core, northbridge, and DRAM for each processor package as well as one power input that is shared across packages. However, not all consumers are covered by this measurement approach, e.g., on-board Ethernet and DRAM termination voltage are not measured.
The mainboard uses several different voltage regulators[1] that are supplied by the 12 V input and provide lower voltages to the components, e.g., 1.5 V for memory. Theses chips use internal current measurements over an inductor to perform their function. Each voltage regulator consists of multiple circuits that are responsible for one phase. These phase circuits provide a shared output signal to a control circuit that corresponds to the total current. For the *apollo* measurements, the summary signal of each voltage lane is captured and used for the external measurements. While the calculation for power is described in the datasheet of the IC [Int], the accuracy is not documented. The measurement is performed on the consumer side and does not include the losses of the voltage regulator circuits, which are also typically non-linear [Haj+16, Sec. 1.2.3.2]. Depending on the goal of the measurement this can be an advantage — when targeting power of a specific component, or a disadvantage — when optimizing total system power.
This instrumentation approach is extremely specific to a particular system. It also requires detailed knowledge of the wiring on the mainboard to locate the right points for instrumentation. Adding the probes for the current sensing and component voltage signals is an intricate process. Figure 3.1a shows the mainboard with attached probes and wires. With the probe wires connected, the sensing signals are susceptible to electromagnetic interference, which can affect the operation of the system.

3.2.2 Instrumentation with Hall Effect Transducers

The mainboard of the *apollo* system uses ATX input voltages (12 V, 5 V, 3.3 V) supplied by the PSU[2]. This connection constitutes the other instrumentation point of this system. Since the 12 V lane covers a majority of total power consumption, and is also the most variable, it is of primary interest. However, this instrumentation point provides only a coarse grained resolution with most components of interest

[1] IR3521 control IC with IR3529 phase IC and ISL6566 controller
[2] In addition to the standard ATX plug, there are three additional 8-pin connectors supplying 12 V to the mainboard. In the following, I consider the combined 12 V lane from all connectors.

summed up — similarly to a full-system measurement. The other two voltage lanes are measured for completeness and comparison with AC power measurements.
The PSU is connected to the board with very short ATX cables. In order to enable the instrumentation, the connecting cables were cut and rerouted to pass them through a sensor board with three Hall effect transducers[3]. The sensor board, shown in Figure 3.1b, provides a connector with the current and voltage signals. This connector also supplies the operating voltage for the Hall effect transducers.

3.2.3 Modular Instrumentation of DC Consumers

For the instrumentation of the *artemis* system, the design goal was portability in the sense that the instrumentation can be transferred to another system. This helps keeping up with advancements in energy-efficient hardware development without the need to regularly build new instrumented systems. Further, the instrumentation should allow separate measurement of components at least at the level of processor packages. All DC consumers with a separate connection to the PSU were used as measurement domains, i.e.,

- two separate 8-pin 12 V connectors, where each connector supplies power to one of the two processor packages and its memory,
- the ATX mainboard power supply with the 12 V, 5 V, and 3.3 V lanes,
- the 12 V and 5 V lanes of a SATA power connector,
- the 12 V power supply for all fans in the system at a 4-pin connector,
- a joint measurement of two 6-pin 12 V auxiliary power connectors as additional power supply for a graphics processing card,
- the 12 V and 3.3 V lanes of a PCIe card — this power domain is already included in the respective voltages of the main ATX power supply,
- the main voltage lane of one DDR3 memory module — this power domain is already included in the respective 8-pin 12 V package power input.

Most of the power connectors use widely adopted Molex plugs. This allows an instrumentation by using Molex extenders with an embedded shunt. One such shunt is shown in Figure 3.1c. In addition, the main PCIe power as well as the DDR3 DIMM power is measured using riser cards. Those riser cards contain small measurement shunts and allow access to the voltage drop (current) signal and the component voltage.
While the modular measuring system was initially used with the *artemis* system, it was later moved to the more recent *diana* system. This shows that, by using removable, standardized connectors for instrumentation, the measurement setup can be easily migrated to other systems under test. However, some server systems, particularly compact rack-mounted devices, use proprietary connectors, so mainly workstations or desktop computers can benefit from the portability.

[3] LEM LA 100-TP for 12 V and LEM HXS 20-NP for 5 V and 3.3 V lanes

Table 3.3: Comparison of the error introduced by different wirings of power measurements for *diana* and *ariel* main power domains (processor + DRAM)

					Proper Load Current Wiring		Proper Load Voltage Wiring	
System	Load	V_s (V)	I_l (A)	P_l (W)	P_e absolute (W)	P_e relative (%)	P_e absolute (W)	P_e relative (%)
diana	full	12	17	204	5.8×10^{-1}	2.8×10^{-1}	7.2×10^{-10}	3.5×10^{-10}
diana	idle	12	1.3	15.6	3.4×10^{-3}	2.2×10^{-2}	7.2×10^{-10}	4.6×10^{-9}
ariel	full	12	17	204	2.9	1.4	3.0×10^{-5}	1.5×10^{-5}
ariel	idle	12	0.5	6.0	2.5×10^{-3}	4.2×10^{-2}	3.1×10^{-5}	5.2×10^{-4}

For the *ariel* system, I followed a similar approach of instrumenting Molex connectors. The measurement domains are therefore similar. However, shunts are no longer included in the adapters. For each measurement domain, the adapters expose three measurement connectors with 4 mm test leads. The adapter allows passing the current of each measurement domain through a power analyzer with internal shunts. The third measurement connector provides ground reference for measuring voltage.

3.2.4 Optimal Wiring for Shunt-Based Measurements

As introduced in Section 2.1.2, there are two different ways to connect voltage and current measurements of a power measurement such that either voltage or current is measured correctly. Depending on the selected wiring, there is a specific error between measured power[4] and actual load power consumption: $P_e = P_m - P_l$. To understand which wiring results in the smaller error, and to estimate the magnitude of the introduced error, I evaluate the choices for a selection of practical measurement domains. The examples given are for the per-package measurements of *diana* (including DRAM) and *ariel* (excluding DRAM). For the custom-built measurements of *diana*, the resistance of the shunt is $R_a = 2\,\mathrm{m\Omega}$ and the input impedance of the amplifiers for the voltage measurements is $R_v = 200\,\mathrm{G\Omega}$ [Lin]. The integrated measuring system for *ariel* specifies the resistive impedances as $R_a = 10\,\mathrm{m\Omega}$ and $R_v = 4.59\,\mathrm{M\Omega}$ [ZES16]. For each measurement domain, I use full load and idle workloads to cover the full range of operations. This calculation considers only direct current and resistive impedances. In practice, the voltage is regulated, but there can be variations in the current. Table 3.3 shows a comparison of the error of load power measurement for different wirings as presented in Figure 2.1. The error of load power with proper load voltage wiring is orders of magnitude lower than the error for proper load current wiring in all configurations. Given the high currents and low voltages in compute node DC measurements, this conclusion can be expected. For the *artemis*/*diana* measuring system, however, one goal was to measure source power correctly for a PSU efficiency evaluation. Thus, proper load current wiring was chosen. The resulting relative error for load power measurement of at most 0.28 % remains smaller than other sources of uncertainty as I show in Section 3.4.2. For *ariel*, I configured the connectors to use proper load voltage wiring although they can be reconfigured just by re-plugging. At scales of 0.0005 % relative / 30 µW absolute in all configurations, the resulting error does not contribute significantly to the overall uncertainty of the load power measurement.

[4] I consider measured power as computed by $P_m = V_m I_m$. The actual load power can be computed as per (2.5) and (2.7). This does, however, require precise quantities of the involved resistances.

3.2.5 Node-Level Instrumentation for HPC Systems

Like the instrumentation of the *apollo* test system, HDEEM is also applied on two levels. A full-node measurement uses a dedicated power and current monitoring chip at the DC/DC conversion of the compute nodes[5]. This measurement domain is referred to as "blade" by the vendor, even though there are two separately-measured nodes per physical blade. The monitoring circuit provides an analog current and power signal. Moreover, six power domains are measured at the voltage regulators (VRs): two processor packages and four groups of DRAM modules. These measurement domains are referred to as "CPU0" and "CPU1" as well as "DDR AB" through "DDR GH" by HDEEM. In contrast to the *apollo* VR measurements where the analog current signal is captured, in HDEEM the VRs provide the measurement signal digitally. Together with co-authors from the vendor, I described further details of HDEEM in [Ils+18c].

3.3 Analog Signal Conditioning and Analog-to-Digital Conversion

Analog measurement signals are typically conditioned with amplifiers and low-pass filters. The next step after the conditioning is the analog-to-digital conversion, in which the analog signal is captured digitally (see Section 2.1.3). In the following, I discuss the analog signal conditioning and analog-to-digital conversion aspects of the presented measurement setups. For the measurements on *apollo* and *artemis*, I consider the entire measurement chain. Contrary, for the integrated measurement systems of *ariel* (ZES LMG670) and *taurus* (HDEEM), some aspects are not publicly documented.

3.3.1 Signal Amplification

As described in the previous sections, the measuring systems for *apollo* and *artemis*/*diana* feature several different sensors. Table 3.4 shows a summary of those sensors and their various analog output signal characteristics. The analog signal from the voltage regulators of *apollo* has a relatively low-voltage measurement signal, which is also not isolated from the operating of the VRs. These signal properties need to be considered for data acquisition. Further, the load voltages at the VR are around 1 V. The Hall effect sensors provide higher and isolated current measurement voltages. At their measurement points, the load voltages are up to 12 V. Shunt-based measurements, as shown in *diana*, can have even lower current measurement voltages than the the signal from voltage regulators. These low voltages are difficult to acquire, requiring the sophisticated signal amplification. Higher measurement voltages from shunts with larger resistance, however, are not feasible as this would significantly reduce the voltage available to the consumer. For example, the processor and DRAM connector is supplied with 12 V from the PSU. At the maximum current of 17 A, the available load voltage is reduced by 34 mV or 0.28 % (the measurement voltage). ATX specifies a tolerance of ±5 % for 12 V voltage rails [Int13][6]. This voltage drop is within tolerance, larger resistors would increase this voltage drop and could start to affect system stability. The PSU may introduce a variance that has to be accounted for in the tolerance. Further, a larger measurement voltage would imply a higher power measurement error as well as thermal dissipation that could increase uncertainty.

[5] The compute nodes use an input voltage of 54 V DC supplied by PDUs in the rack.
[6] The P8 12 V package connectors are not included in the ATX specification, tolerances refer to other 12 V rails.

Table 3.4: Overview of the analog measurement signal ranges.

System	Sensor Type	Measurement Domain	Voltage (V)	Current[a] (A)	Measurement[b] (mV)
apollo	VR	core	1.0	120	230
apollo	VR	northbridge	1.2	23	220
apollo	VR	DRAM	1.5	8.7	8
apollo	Hall effect	board	12.0	42	692
apollo	Hall effect	board	5.0	3.1	386
apollo	Hall effect	board	3.3	0.9	56
diana	shunt	processor + DRAM	12.0	17	34
diana	shunt	board	12.0	0.8[c]	10
diana	shunt	board	5.0	1.2[c]	7
diana	shunt	board	3.3	0.2[c]	1
diana	shunt	GPU	12.0	20[d]	24
diana	shunt	PCIe	12.0	6.5[d]	65
diana	shunt	PCIe	3.3	2.0[d]	20
diana	shunt	SATA	12.0	0.3	15
diana	shunt	SATA	3.3	0.4	2
diana	shunt	fan	12.0	0.6	30

[a] maximum of load current as measured under sustained full load unless noted otherwise
[b] voltage of the current measurement at maximum load current
[c] in a configuration without GPU
[d] estimated maximum based on specification

The current measurement voltages given in Table 3.4 represent values for maximum load. During typical workloads, and especially for idle configurations, the measurement voltages are lower. This poses the challenge to capture both stable voltages of up to 12 V as well as highly varying voltages in the order of 1 mV. To address this, the measuring system for *apollo* and *artemis*/*diana* uses programmable precision instrumentation amplifiers of type LT1167 [Lin], which can be configured dynamically for an amplification factor between 0.05 and 400. This ensures that all signals can be digitally captured with a common voltage range of ±10 V.

3.3.2 Analog Filtering and Analog-To-Digital Conversion

As discussed in Section 2.1.3, it is important that the sampling rate matches the analog signal bandwidth. The measuring system for *apollo* and *artemis*/*diana* uses a flexible data acquisition with two National Instruments cards. For the best temporal resolution, the NI PCI-6123 card captures up to four measurement domains (eight channels with for each of the voltage and current signals) at 500 kSa/s. This high-resolution sampling is used for highly variable measurement signals such as the per-socket power domains of the *diana* system. The majority of channels is covered by the NI PCI-6255 card, which offers 80 signal inputs at a maximum aggregate sampling rate of 1.25 MSa/s (see also Section 2.1.3). However, to avoid cross-talk between signals from different measurement domains, a lower sampling rate of 7 kSa/s per channel (560 kSa/s aggregate) is used in practice. With a configuration covering a lower number of measurement domains, the sampling rate per channel can be increased while retaining the aggregate sampling rate. To accompany this flexible data acquisition setup, the amplifier boards for this measuring system include a low-pass filter. This filter is configurable in order to allow a high temporal resolution measurement when using the respective measurement channels while ensuring accuracy when using lower sampling rates.

The LMG670 measurement device for the *ariel* system uses a sampling rate up to $1.\overline{21}$ MSa/s per channel with static as well as configurable anti-aliasing filters (see also Section 2.2.1). This setup allows to push the boundaries of sampling rate and temporal resolutions for measuring compute power. Moreover, these sophisticated single-node measuring systems can utilize dedicated computing resources posing no limitation of measurement data rate.

Contrary, for a full-scale integrated system with no external hardware to support the measurement, the processing capability can limit the sampling rate. Furthermore, cost has to be considered for scalable production systems with thousands of nodes. The HDEEM infrastructure utilizes an FPGA, which allows end-to-end sampling rates up to 1 kSa/s without compromising node performance. A second-order low-pass filter at 600 Hz is applied to the analog signal. Since this filter does not have an ideal cutoff, the bandwidth for considering the Nyquist criteria is actually higher than 600 Hz. Thus, to ensure accuracy, the sampling rate is 8 kSa/s. The digital signal is immediately processed with another anti-aliasing filter and downsampled to 1 kSa/s. This approach enables the observation of dynamic power close to the readout rate (see also Section 3.5.1, Figure 3.9e) while retaining high accuracy due to internal oversampling. The reduced readout rate of 1 kSa/s relaxes the requirements for further digital processing and enables longer recordings are in the limited memory of the BMC. The measured HDEEM values at VR level are provided digitally by the VRs at a rate of 1 kSa/s. These signals are also digitally filtered and downsampled to 100 Sa/s.

3.3.3 Integrated Solutions for High-Resolution Measurement

The custom solutions of the energy measuring system for *apollo* and *artemis*/*diana* use external cabling, plugs, and screw terminals to connect power lines, sensors, amplifiers, and data acquisition. This poses practical challenges regarding outside interference and long-term stability. For the measurement of the *ariel* system, I selected the following contemporary integrated measurement solution that combines sensors, signal conditioning, and data acquisition in one device. Going with an integrated device reduces the outer complexity of the measurement setup significantly.

The LMG670 power analyzer [ZES16] with six L60-CH-A1 power measurement channels offers a sampling rate of $1.\overline{21}$ MSa/s per channel (see also Section 2.2.1). A crucial feature is the possibility to read measurements at full resolution from the device. This possibility enable high-resolution energy measurements of computing systems. In the scope mode, measurements can be read remotely through the gigabit network interface from both the narrowband and wideband channels at full resolution[7]. This data collection can operate continuously rather than just collecting a fixed number of samples on the device for a limited time. Overall, these features enable the use case of application power traces (see Section 5.1.3).

The aspects of choosing instrumentation points and adapters for instrumentation still apply. For *ariel*, I designed similar DC-Molex adapters as the ones discussed in Section 3.2.3. Other aspects of the measurements are handled by the device itself. Each channel uses internal shunts that are configurable for different current ranges up to 32 A. The different aspects of analog processing are also covered by the device, e.g., configurable filters of the DualPath signals and their data acquisition.

[7] When using all channels simultaneously, the sampling rate is limited by the network bandwidth.

3.4 Accuracy Evaluation and Calibration

Verifying the accuracy of energy measurements of computing system is particularly challenging for several reasons: Firstly, the true value of a measurement is unknowable [JCG08]. The uncertainty approach discussed in Section 2.1.1 provides a systematic way to discuss accuracy in the absence of the true value. Creating the required measurement model, however, requires detailed knowledge about all components of the measurement that contribute to the error. But certain aspects of the discussed measurements are black boxes for this evaluation. Secondly, the actual power consumption of a computing system cannot be precisely controlled for successive measurements. The dispersion of the true value of a measurand are significant and may dominate the dispersion of measurement results, even for repetitions over a short period of time. Therefore, the repeatability conditions for measurements that are integrated in compute systems cannot be guaranteed. This limits the applicability of a type A (statistical) uncertainty evaluation. Consequently, only the error approach remains feasible.
For evaluating the error of the custom-built and integrated measuring systems, I use measured values from a calibrated LMG450 power meter as substitute for the true value (see also Section 2.2.1). The wide range of possible operating points of computer systems with respect to power consumption also presents a challenge to for the reference measurements. While the range of values is known, it is not trivial to induce a set of diverse operating point across this range. Moreover, the signal may contain high frequencies and is neither pure DC nor limited to sinusoidal shape. Effectively, computer systems can exhibit almost arbitrary patterns of power consumption (see Section 3.5.1). Therefore I use the worst documented uncertainty[8] of the reference measurement, which amounts to 0.6 % of measuring value + 0.5 % of measuring range for active power at up to 20 kHz.

3.4.1 Synthetic Workloads for Evaluating Power Measurements

In the following evaluations, I use different sets of workload generators: Firstly, *synthetic workload kernels* provide a specific and diverse utilization of resources, e.g., floating point units or the memory subsystem [Hac+13a; Bie15]. With a framework around the workload kernels, I varied several kernel and system parameters, e.g., the core frequency, the number of threads, and the mapping from threads to hardware threads. Such a variety of workload configurations is important to expose biases of measurements or models towards certain components (see also [BJ07]). This is particularly relevant when evaluating CPU energy counters, which may be implemented with models (see Section 3.6).
A general limitation of the quantitative error evaluation stems from differences in the temporal scope of the evaluated and reference measurement. While the evaluated measurements provide a readout rate of up to 500 kSa/s, the reference power meter LMG450 only exposes readouts at 20 Sa/s, even though it internally samples at a much higher rate. Further, at a sub-millisecond time scale, assigning consistent timestamps to measurements generated by different devices becomes increasingly difficult (see Section 5.2). To overcome the discrepancy in measurement rate and synchronization, for comparisons with the reference measurement all workloads were executed for least 10 s in each configuration. For each time slice, the inner 8 s are used to compute the average power consumption

[8] The documentation [ZES] specifies a “measuring accuracy”, which I assume to be an upper bound for the measurement error.

of both the evaluated and the reference measurement. This averaging does hide the impact of noise (i.e., random error) for the fine granular measurements, but allows an absolute comparison of accuracy, particularly with respect to energy which relates to average power (i.e., systematic error). This applies to the evaluation in Section 3.4.2, Section 3.4.3, and Section 3.6.
For evaluating the accuracy of a measuring system in the presence of rapid and sharp load changes — i.e., the energy correctness evaluation in Section 3.5.2) — I used alternating synthetic workload kernels that exhibit particularly low and high power consumption, respectively. In particular, *FIRESTARTER* [Hac+13b] is a stress test utility designed to maximize power consumption, but can also generate regular power alternations with very high amplitude by alternating between tuned computational parts and idle. A modified version of FIRESTARTER also performs load swings in a cluster that are synchronized using MPI.
While idle presents a particularly low level of power consumption, the transition from execution to idle can take hundreds of microseconds (see Section 5.3.2). The temporal granularity evaluation in Section 3.5.1 required load changes at higher frequencies up to the order of 100 kHz. Thus, I designed and implemented additional custom workload generators that utilize simpler workload kernels and require no C-state transition:

- The square root instruction family is known to exhibit a very low power consumption, on some systems even below the power consumption of idle loops [Mol+10].
- On the Skylake-SP system *ariel*, the square root instruction performs significantly better, which causes a higher power consumption. Therefore, I use a `PAUSE`-loop on this system. The `PAUSE` instruction is documented to reduce the power consumption during spin loops [Int18a, Vol. 2B 4-229]
- A vectorized arithmetic loop with addition and multiplication that operates on a small data set that fits into caches causes a high power consumption. While there are even higher power workloads such as FIRESTARTER, the simpler loop does not depend on memory power consumption and can easily be controlled to run for a specific fine-grained amount of iterations.

To increase the significance of amplitude changes on the measured power domains, multiple threads execute these workloads. This requires synchronization either using barriers, e.g., with OpenMP, or a shared clock. When evaluating measurements, it can be insightful to evaluate specific workload alternation frequencies, such as the analog-to-digital sampling rate or intermediate readout rates. In addition to checking known frequencies, I used workloads that allow varying the workload alternation frequency over time (sweep) or manual interactive selection of workload alternation frequency. The result from an automatic sweep or manual tuning was then used as parameter of a constant frequency workload for further evaluation.

3.4.2 Improving and Evaluating the Accuracy of a Single-Node Measuring System

In the following section, I focus on the accuracy of single-node high-resolution power measurements. For this verification, I utilized the synthetic workload kernels introduced in the previous section and varied the thread count and thread mapping configuration. All comparisons are with respect to average power over time slices due to the differences in sampling rate.

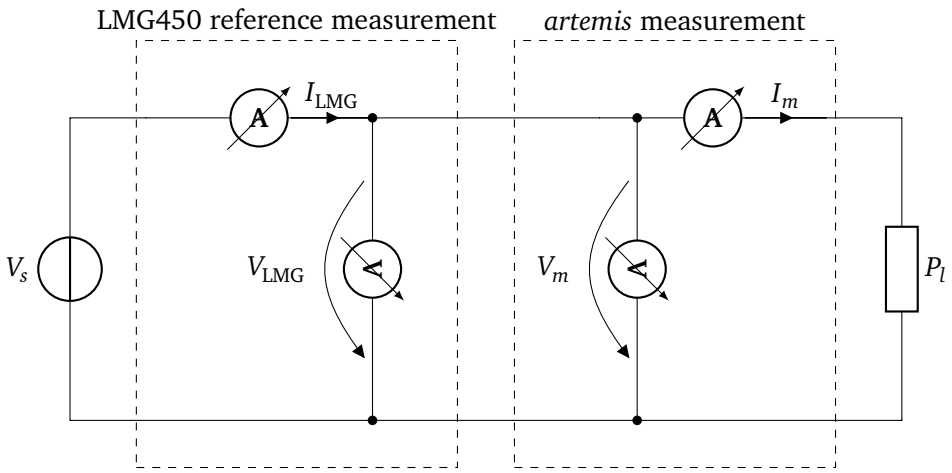


Figure 3.2: The setup for absolute verification of *artemis* measurements, applied to for each of the two processor packages

Calibration of Amplifiers and Shunts

The measuring system for *apollo* and *artemis*/*diana* was calibrated separately at the level of amplifiers and shunts [Ils+15a]. Each amplifier use a digital8-bit calibration factor, which is nominally set to 240. This allows a calibration at a precision of ±0.21 % $\left(= \frac{0.5}{240}\right)$. To calibrate the amplifiers, a stable input signal was provided by a signal generator and measured at the input and output with calibrated reference voltmeters.

For calibrating the shunts, a sliding resistor was used, providing a stable ohmic load during calibration. The reference measurement is used to gather current and voltage in the setup. The calibration revealed differences of up to 16 % compared to the specifications of the current sensing resistors. This can be explained by additional contact resistances of the harness given the small shunt resistances ranging from 1.2 mΩ to 12 mΩ.

Verification of Shunt-Based DC Measurements for *Artemis*

The evaluation of *artemis* focuses on the 12 V per-package power measurements as these power domains consume the majority of variable power. I applied the reference power meter to the same power domain as shown in Figure 3.2. In this configuration, both measuring systems covered the same power domain except for contact resistances and input impedances of both voltage measurements (see Section 2.1.2 and Section 3.2.4). The resulting difference between reference measurement and shunt-based package measurement of *artemis* is shown in Figure 3.3.

The relative error on both packages for all workload configurations is below 1.7 % or 2.3 W absolute. In this measurement setup, the uncertainty of the reference measurement is significant, particularly for low power configurations. Any error within the uncertainty can be explained either by the error of the reference measurement or the error of the evaluated measurement.

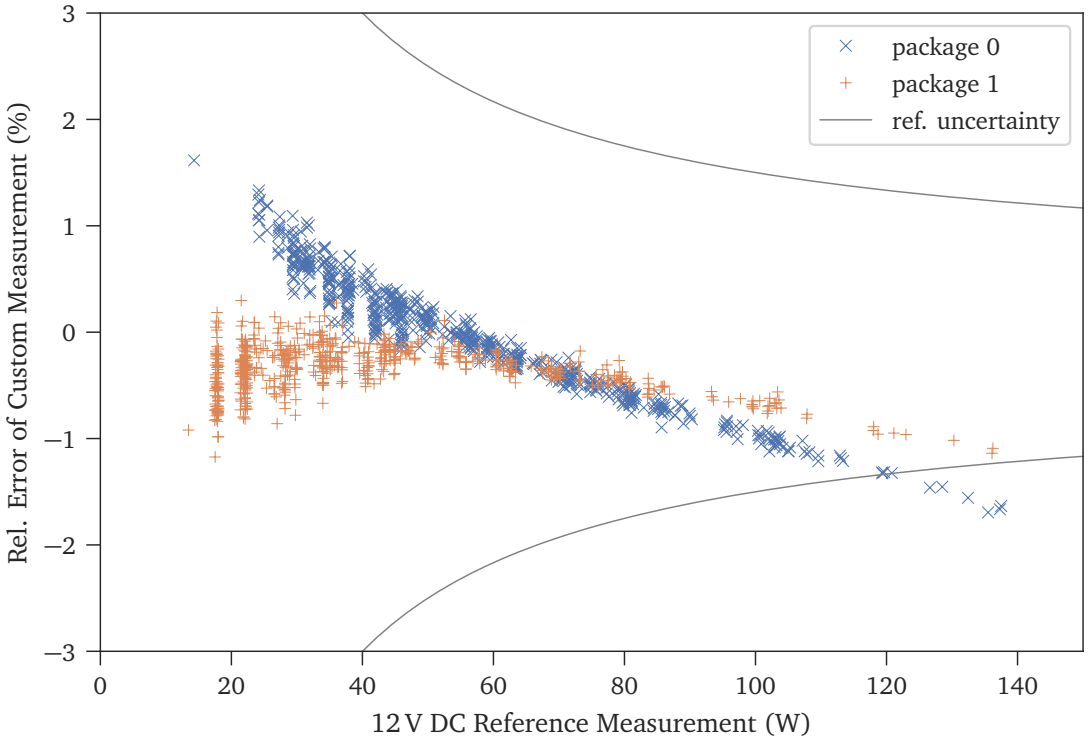


Figure 3.3: Relative differences between the 12 V per-package measurements on *artemis* compared with the reference measurement. The reference uncertainty is included based on the manufacturer accuracy specification.

Compound Verification of Shunt-Based DC Measurements for *Artemis*

As listed in Section 3.2.3, all DC consumers within the *artemis* system are measured individually. This allows an additional comparison between the sum of all DC power consumption (P_{DC}) and the total AC input power to the PSU as measured by the reference power meter (P_{AC}). In this case, however, measurement domains are not identical. Instead, the PSU is between the measurements of DC consumers and the reference AC power measurement. Nevertheless, the two measurements should strongly correlate. A general correlation coefficient would yield a quantitative measure of this correlation, but it does not express the possible error on the measurement scale. Further, the commonly used Pearson correlation coefficient assumes a linear correlation, which does not apply to this case. Rank correlation coefficients such as Spearman's ρ only consider the order of values [HEK09] and cannot detect certain implausible patterns as long as monotony is given. Therefore, I define a specific measure to quantify the *discrepancy* for this use case. To account for the differences in power domains, I model the PSU efficiency. This needs to consider, that the loss of switching mode power supplies includes quadratic components from conduction [Haj+18, Sec. 1.2.3.2]. The following quadratic equation was fit on all measurement data by minimizing the squared error.

$$P_{\mathrm{AC}} = M(P_{\mathrm{DC}}) = 0.00026\,\mathrm{W}^{-1}\,(P_{\mathrm{DC}})^2 + 0.99988 P_{\mathrm{DC}} + 14.7\,\mathrm{W} \tag{3.1}$$

Note that this model is not intended to evaluate the PSU efficiency, but rather to expose noise as well as workload-dependent errors in the evaluated measurement. It is only used for interpolation rather

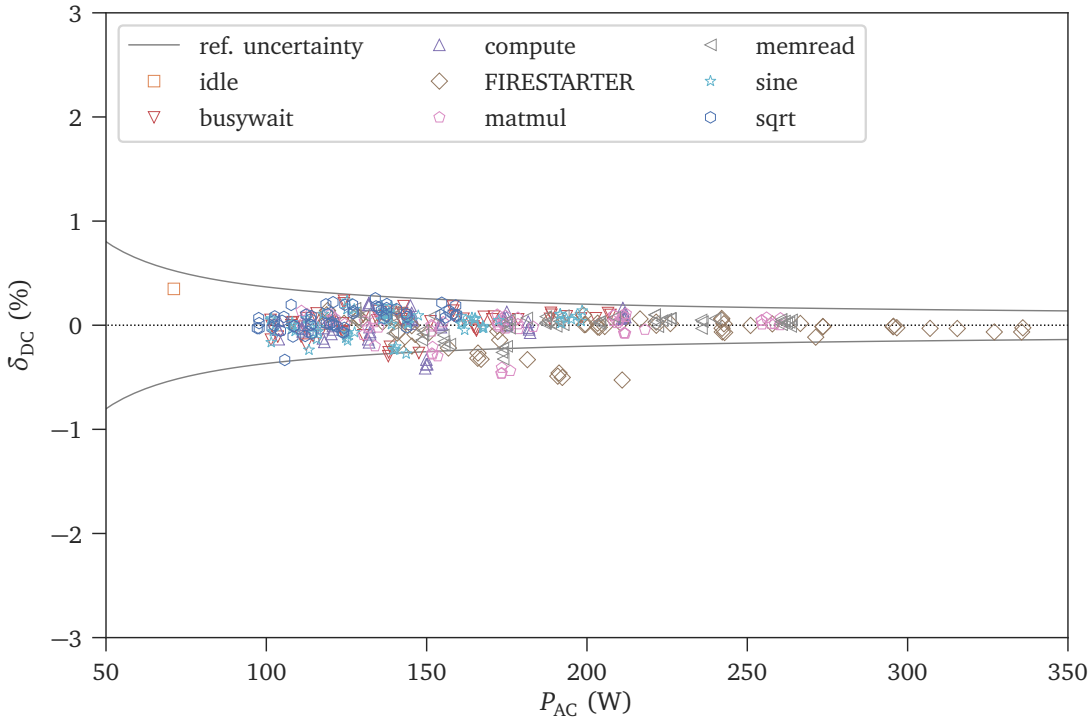


Figure 3.4: Relative discrepancies between the sum of DC shunt measurements and the AC reference measurement on *artemis* under assumption of the PSU efficiency model (3.1). The reference uncertainty is based on the manufacturer accuracy specification.

than extrapolation. Since there are no independent inputs to train the model, the comparison can not reveal an overall bias or calibration error that is common to all evaluated measurements. Instead, this metric exposes discrepancies between multiple evaluated measurements by stressing components differently and shows the random error that remains after averaging the time slices. The absolute and relative discrepancies (Δ and δ respectively) within the DC domain are then defined as follows:

$$\Delta_{DC} = P_{DC} - M^{-1}(P_{AC}) \tag{3.2}$$

$$\delta_{DC} = \frac{\Delta_{DC}}{M^{-1}(P_{AC})} \tag{3.3}$$

Figure 3.4 shows the relative discrepancies between the AC reference power and modeled AC power based on the sum of DC consumers. Due to the AC coupling, the reference power meter is more accurate as reflected by the reference uncertainty bounds. The worst-case observed absolute and relative discrepancies are $|\Delta_{DC}| < 1.0\,\text{W}$ and $|\delta_{DC}| < 0.6\,\%$ respectively for all configurations. The results showed no systematic patterns or biases for certain workloads. Some configurations with high utilization (FIRESTARTER and `matmul` workloads) on a single socket show a slightly higher negative error which is consistent with the package 0 error pattern shown in Figure 3.3. On the other end of the scale, the idle configuration shows a slightly higher positive error.

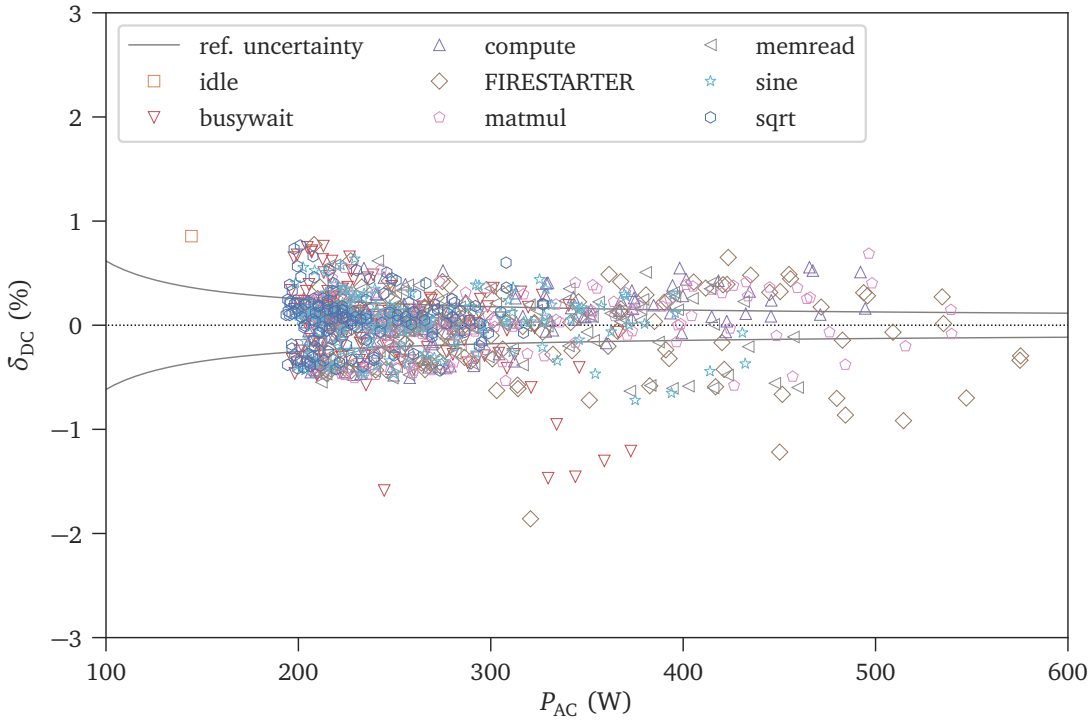


Figure 3.5: Relative discrepancies between the sum of DC Hall effect measurements and the AC reference measurement on *apollo* under assumption of the PSU efficiency model (3.4) The reference uncertainty is based on the manufacturer accuracy specification.

Verification of Hall-Effect-Based DC Measurements for *Apollo*

In *apollo*, the DC consumers are measured with Hall effect sensors at the three different voltages (12 V, 5 V, and 3.3 V). Similarly to the previous setup, I performed a reference measurement and applied the following PSU efficiency model to the DC power consumption in order to compare it to the AC reference measurement:

$$P_{AC} = M(P_{DC}) = 0.00011\,W^{-1}(P_{DC})^2 + 1.0015 P_{DC} + 48.3\,W \tag{3.4}$$

Only the sum of all three DC voltages is used input for the model (P_{DC}), even though it is conceivable that the PSU has different efficiencies at the different voltages. This is to avoid overfitting the model, and since there are no programmatic means to cause variations in the 3.3 V and 5 V lanes. The resulting model is consistent in itself, but indicates a significantly lower PSU efficiency than the model for *artemis*.

Figure 3.5 illustrates the resulting measurement discrepancy. The random error of this measurement is significantly higher than the shunt-based measurement for *artemis*. There are also outliers with negative errors of more than −1 %, particularly for the busy waiting and FIRESTARTER workloads. A possible but unconfirmed explanation is a specific characteristic of high frequencies in the power consumption signal of theses workloads that triggers a non-linear response of the Hall effect measurements. Overall the worst case absolute and discrepancies are $|\Delta_{DC}| < 5.0\,W$ and $|\delta_{DC}| < 1.9\,\%$ respectively.

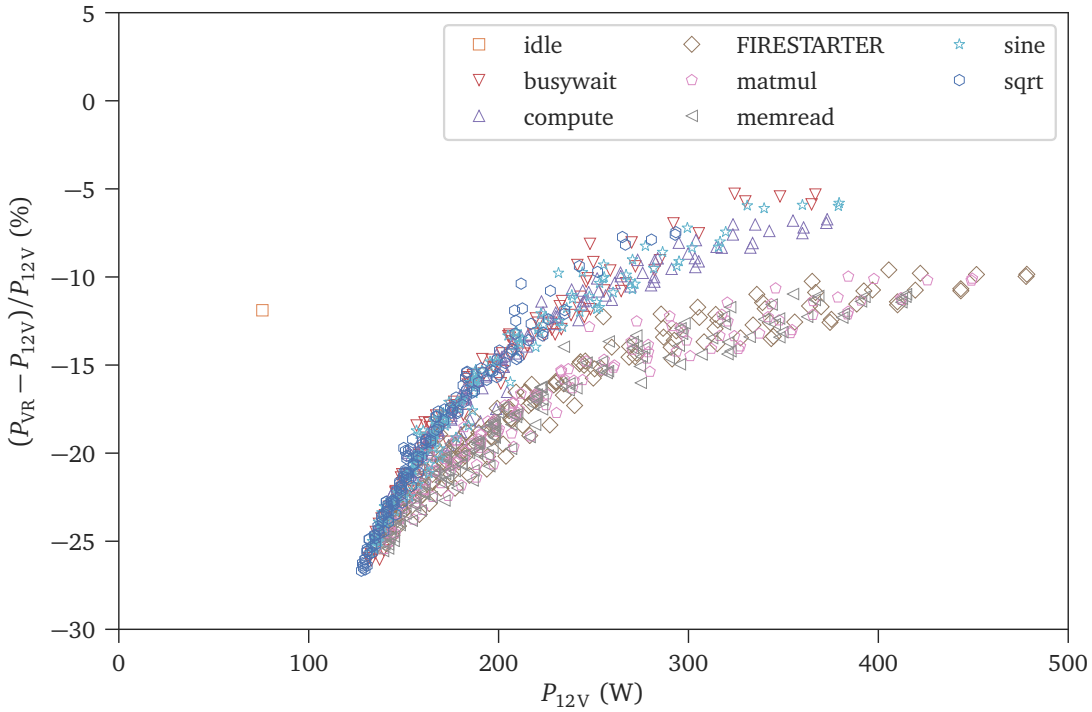


Figure 3.6: Relative differences between the sum of VR power measurements and the shunt-based DC 12 V measurement for different workloads.

Verification of Voltage-Regulator-Based DC Measurements for *Apollo*

The power for the VRs is provided by the 12 V lane. Therefore, I used the 12 V previously evaluated DC measurements (P_{12V}) for further investigation of VR-based measurements. In this comparison, the measurement domains are again different between the evaluated measurements and the reference measurement. The measurement domains of the VRs are on the consumer side (e.g., 1 V to the processor). Therefore, they do not include the loss of the VRs itself. Contrary, the 12 V measurement is located between the VR and PSU. Moreover, some undocumented 12 V might not be included in the sum of VR measurements.

The resulting differences between the sum of VR and 12 V measurement are plotted in Figure 3.6. In contrast to the previous comparisons, no efficiency model is used for the visualization. The difference between measurements is highly correlated with different workloads rather than just random noise. These different workloads utilize the different power domains for core, northbridge, and DRAM at different rations. Due to the workload-specific bias, a model on top of the sum of VR power (P_{VR}), similar to previous models, highly depends on the ratio of workloads used for training. Attempting to generate a model on top of the separate VR power domains resulted in implausible efficiency factors. When applying a simple quadratic model for quantifying the discrepancy within the experiment results, the idle case has the worst $|\Delta_{VR}| = 28.7\,\text{W}$ and $|\delta_{VR}| = 75.7\,\%$. For the remaining configuration the absolute and relative discrepancy remains high at $|\Delta_{VR}| < 18.0\,\text{W}$ and $|\delta_{VR}| < 6.3\,\%$.

It can be concluded, that the VR-based measurements suffer from workload-specific nonlinear errors making them unreliable as basis for quantitative analysis. I observed the best accuracy with shunt-based measurements, which I also evaluated on identical power domains without applying a corrective model. The evaluated Hall effect DC measurement is also accurate but exhibits more noise.

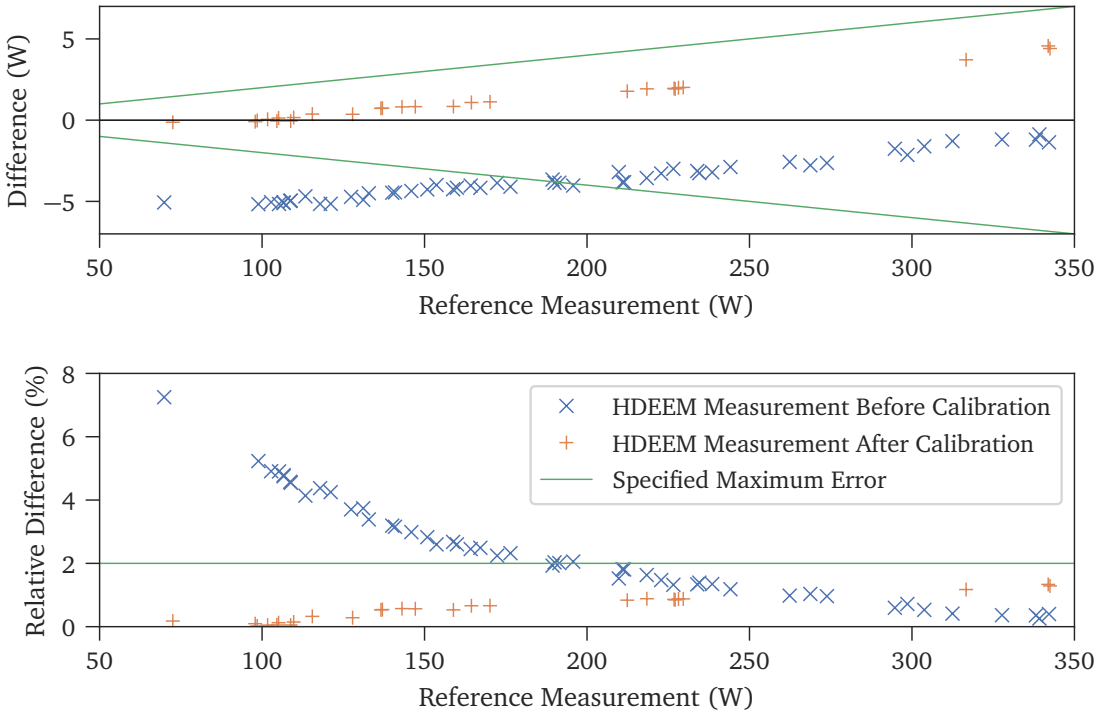


Figure 3.7: The absolute and the relative difference between the HDEEM measurement and the reference measurement (ZES LMG450) on one node of *taurus* before and after the in-situ calibration by the vendor (adapted from [Ils+18c]).

3.4.3 Absolute Accuracy Evaluation of a Many-Node Measuring System

Verifying the absolute accuracy of the HDEEM energy measurements in *taurus* presented several challenges. The measurements are an integral part of a large production HPC system. All nodes are part of integrated, compact chassis — each hosting nine blades with two nodes per blade. While the integrated measurements are located at the input of each node, it is not feasible to apply an instrumentation at this point given the closed chassis setup. Therefore, the 54 V input to the chassis was instrumented at the four connectors to the chassis using custom-built breakout boxes. Further, due to the number of nodes in the production system, it is not feasible to individually verify every node. Consequently, two sample chassis for a total of 36 nodes were selected for verification. This evaluation only concerns the full-node measurements, not the separate voltage regulator-based measurements which are also part of HDEEM. The latter have a lower specified accuracy and do not allow a comparative measurement at a similar measurement domain.

Each chassis does not only contain 18 nodes, but also a chassis management module, fans, and networking components such as an InfiniBand switch. In order to isolate individual nodes, a baseline was measured at which all nodes were disabled.

In the first evaluation, when comparing with the reference power meter, a number of nodes exhibited large relative measurement errors, in some cases exceeding 10 %. This particularly affected low-power states in which small absolute errors result in larger relative errors. The relative error is particularly relevant given that the vendor specifies a maximum error of 2 % for this measurement point.

Due to the revealed errors in the sampled nodes, the system was not accepted initially. The vendor performed an in-situ re-calibration, which I further described in [Ils+18c]. This re-calibration used a

separate measurement setup and was not based on the verification data. After the re-calibration, I repeated the verification. The second verification also used 36 nodes in two chassis. The two chassis selected for the second verification included one chassis that was previously evaluated, for direct comparison of nodes before and after calibration, and one other randomly selected chassis, to ensure that not only the reported faulty nodes were fixed. After calibration, the difference between reference measurement and HDEEM for all sampled nodes under all workloads were between −2.76 % and 2.98 %. It has to be noted that these results are also affected by the uncertainty of the reference measurement which makes it more difficult to provide a very narrow bound on error at low power states. The improvement from calibration is exemplified in Figure 3.7.

3.5 Evaluating Temporal Granularity and Energy Correctness

There are two main reasons for striving towards a high temporal granularity of computing system power measurements. On the one hand, a high temporal resolution is required to understand the dynamic impact of rapid changes in compute workloads and processor states to power consumption. On the other hand, a sufficient temporal resolution of the power measurement is required in order to provide a accurate energy values (see also Section 2.1.3).

It is not trivial to say at what temporal granularity the power consumption of a processor changes when it is executing an application. One might suggest the clock frequency, which is in the order of 1 GHz or 1 ns. However, instructions are not executed one-after-another but rather in a pipeline. Additional features of the microarchitecture, such as out-of-order execution and prefetchers, induce a highly complex interaction between instructions and state of the processor. Thus, an attribution from consumed energy to individual executed instructions is fundamentally impossible on contemporary processors. Considering the statistical instruction mix or density, which can be correlated with power consumption, the time scale of changes vague. The following sections provide an evaluation of how such changes between active workloads impact power consumption at different time scales. Since the most impactful changes in power consumption are caused by changes of hardware states, particularly C-states (see also Section 2.4.2), the time granularity of such changes is of particular relevance for measuring dynamic power. A wake-up from the most shallow C-state can take from 0.5 µs to 10 µs [SMW14]. An ideal power measurement would be able to accurately observe such a transition. This is not possible when using power measuring solutions with readout rates from 1 ms to 1 s (compare Section 5.3.2, Table 2.1).

During early experiments with custom-built power measurements, I have shown that short load changes do occur in practical applications [Hac+13a]. For example, OpenMP programs[9] can exhibit short sub-millisecond phases of synchronization that cause a significant drop in consumed power. As shown in Figure 3.8, this can can only be observed with the appropriate power measurement.

In the following, I evaluate the impact of instrumentation points and sensors on temporal resolution. To understand how short changes in workloads impact the power consumption, I evaluate the shortest possible transitions in as well as the power signal bandwidth based on binary white noise workloads in Section 3.5.1. While I introduced the topic of energy correctness for analog-to-digital conversion in Section 3.3.2, Section 3.5.2 discusses it for digital readouts.

[9] The examples shown are measurements of the applu benchmark from SPEC OMP.

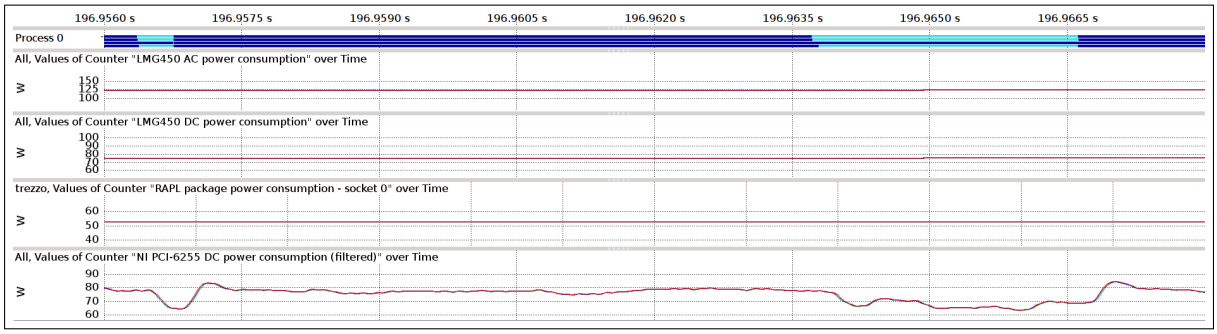


Figure 3.8: Observing the power consumption during short idle phases (cyan) in a parallel benchmark with different measurements. The change in power for <1 ms can only be observed with the bottom-most high-resolution measurement (NI PCI-6255) [Hac+13a].

3.5.1 Measurement Signal Bandwidth at Different Instrumentation Points

Power consumption from transistors does not immediately propagate up the power conversion chain (see also Figure 2.2). Every conversion implies certain capacitances which introduce a low-pass effect. Measurements at the AC-input are further limited by the sinusoidal signal at 50 Hz. Capacitances in the PSU may cause a further limitation of the dynamics in the measurable signal. The DC signal between PSU and VR provides a measurement opportunity with much higher signal bandwidth. The ATX standard [Int13, pp. 14 sq.] specifies maximum DC output transients of 60 % with up to 10 kHz load-changing repetition rate and a load slew rate of 1 A/µs. This gives a coarse notion about the possible order of magnitude of variability in a DC power consumption signal. The power output of the voltage regulators can possibly provide an even more dynamic measurement signal than the DC power signal. Another aspect to consider, is that switching power converters can introduce a high-frequency signal, which may overlap with the actual power load caused by the application running on the processor (see. [Haj+16, Sec. 1.2.3.2]).
To demonstrate the limits of 12 V DC and VR based measurements, I create and observe the shortest possible time for workload changes on *artemis*, *apollo*, and *ariel*. This is complemented by the same measurements on the 54 V-blade and VR-based CPU sensors on *taurus*. However, the temporal limitation on *taurus* measurements do not result from the measurement points or sensors, but rather the analog and digital processing and the sampling rates.
A workload kernel that is used to expose the fastest possible transients should cause a high amplitude change in consumed power over a minimal time. Therefore, I use the synthetic high-frequency load-changing workloads described in Section 3.4.1, which exhibit known low and high power consumptions on the system under test.

Evaluation of Individual Transition Patterns

The workload kernels are alternated as follows: First, the low-power workload kernels is executed for a relatively long time such that its stable amplitude can be clearly determined. Then, the high-power workload kernel is executed for a short time, followed by the low-power workload for a short time. After this, the high-power workload kernel is executed for a relatively long time, again to determine its stable amplitude. Due to the fine temporal granularity, the workload execution time is controlled by the number of loop iterations rather than using a timer. The workload is configured on a per-system

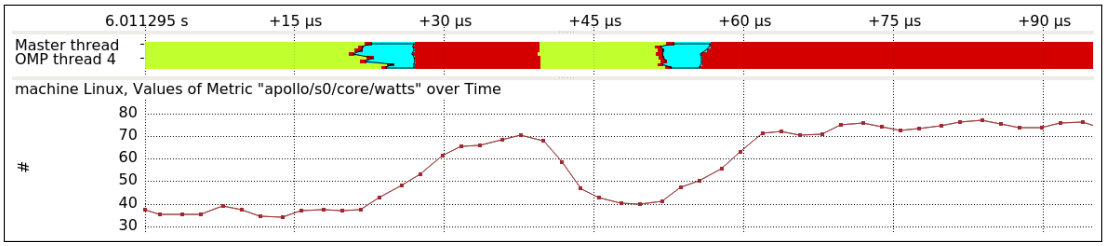
basis to run both the low-power and high-power kernels for approximately the same time. This whole sequence is repeated and the short time is decreased in order to find the lowest time for which the power consumption reaches approximately the stable power consumption level. The experiments on *apollo* and *artemis* used a configuration with simultaneous sampling at 500 kSa/s. I used 8 threads on *apollo* and *artemis* and 18 threads on *ariel* for the experiment. On each system, the threads were pinned to one processor package. This is a compromise between cross-package synchronization jitter and high amplitude power changes. On *taurus*, the full 24 cores of both processor packages were used as there is less relative impact from synchronization at the lower sampling rates of HDEEM.

Figure 3.9 shows the results from the different measuring systems and systems under test at different time scales. I manually selected the state transitions to show the minimal time (t_{min}) of equally short low-power and high-power phases that reach their respective stable power amplitude. Therefore, the workload time differs for each measurement configuration. Note that t_{min} is *half* of a full period when considering continuous load changes, i.e., the maximum load-changing rate is $f_{\text{max}} = \frac{1}{2t_{\text{min}}}$. As shown in Figure 3.9a, the VR-based measurements for *apollo* fully resolve the shortest load change. This is expected as they measure closest to the actual load of all measuring solutions. Low-power and high-power load are each $t_{\text{min}} \approx 12\,\mu\text{s}$ in duration, which corresponds to a load-changing rate of $f_{\text{max}} \approx 42\,\text{kHz}$. The Hall effect measurements on *apollo* resolve down to $t_{\text{min}} \approx 140\,\mu\text{s}$ (Figure 3.9b). Both shunt-based measurements on *artemis* (Figure 3.9c) and *ariel* (Figure 3.9d) exhibit slightly different granularity with $t_{\text{min}} \approx 120\,\mu\text{s}$ and $t_{\text{min}} \approx 160\,\mu\text{s}$, respectively. Overall all three discussed measurements on the 12 V PSU output are similar. The resulting $f_{\text{max}} \approx 3.6\,\text{kHz}$ are also on the same order of magnitude as the maximal load-changing rate of 10 kHz suggested by the ATX specification.

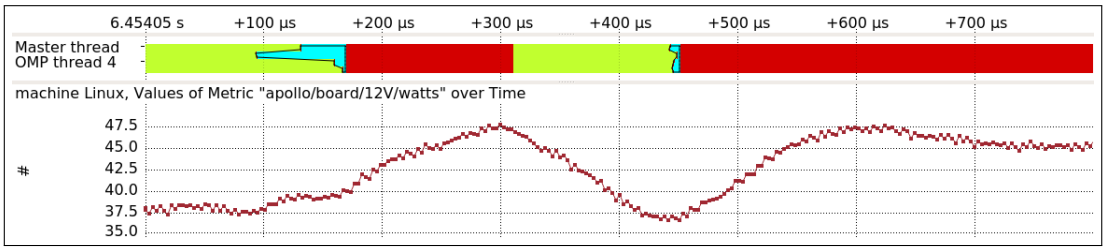
These results are not exact limits, shorter patterns are resolved by the measurement with a dampened amplitude. Noise at the high resolution is also a factor for small amplitude changes, particularly with the Hall effect measurements for *apollo*. The volatile thread synchronization for these short time periods also impacts the accuracy of the determined values for t_{min}. The visual comparison shows that the patterns for these four different measurements are very similar, despite the different time scales. This consistency is particularly interesting given that three different sensor types are used as well as two independent measurement infrastructures with the custom-build amplifiers for *artemis* and *apollo*, and the integrated commercial solution for *ariel*. The choice between the two different instrumentation points at the PSU output and the VR output have the biggest impact on the temporal scale.

The measurements on *apollo*, *artemis*, and *ariel* resolve these short minimal duration load patterns with sufficient resolution. The maximal discernible load-changing rate depends on the measurement signal at the instrumentation points, the sensor, and the measuring system. Therefore, it cannot be trivially determined which of these is the limiting factor. The integrated power meter LMG670, which is used for *ariel*, has a bandwidth of 10 MHz and a sampling rate of $1.\overline{21}\,\text{MSa/s}$ with the utilized wide-band measurements [ZES16]. This provides strong support that the observed load-changing rate of $f_{\text{max}} \approx 3\,\text{kHz}$ is in fact limited by the instrumentation point rather than the measuring system.

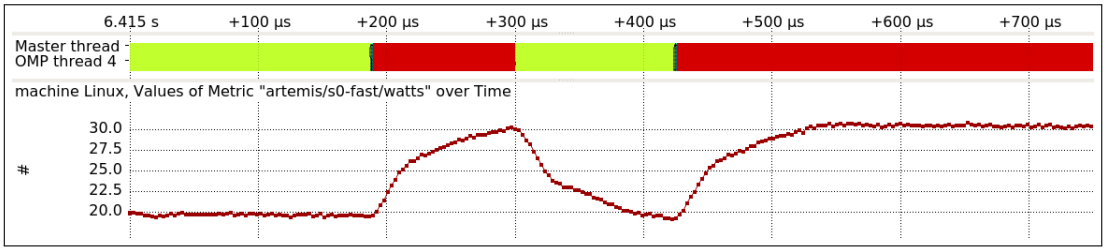
For comparison, Figure 3.9e and Figure 3.9f, show the blade and CPU0 measurements on *taurus*, respectively. In contrast to the previously described measuring systems, the temporal limitations of HDEEM in *taurus* arise from the sampling rate and analog and digital processing described in Section 3.3.2. Therefore, this analysis cannot yield conclusions for their measurement points.



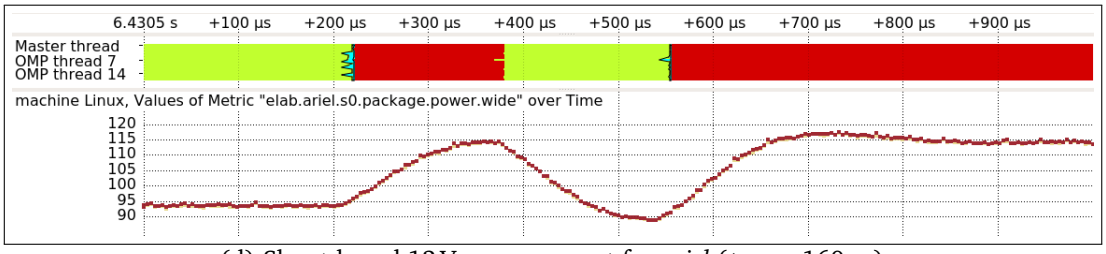
(a) VR-based measurement for *apollo* ($t_{\min} \approx 12\,\mu s$)



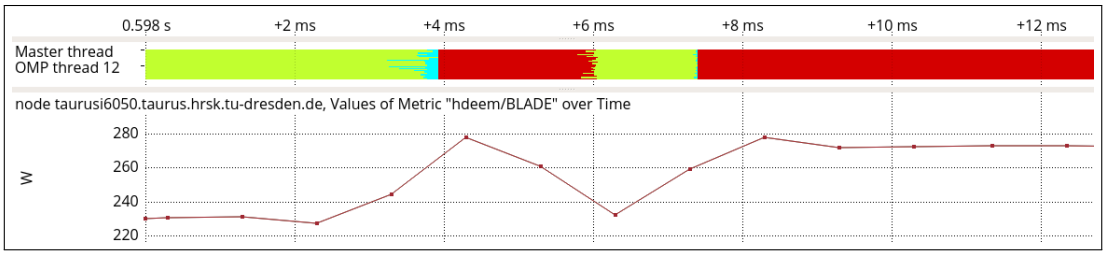
(b) Hall effect sensor 12 V measurement for *apollo* ($t_{\min} \approx 140\,\mu s$)



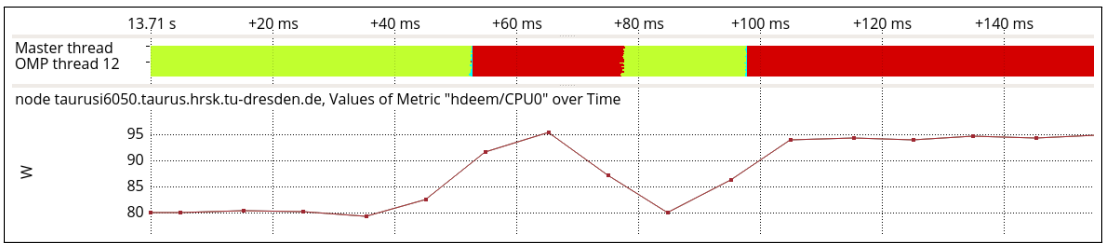
(c) Shunt-based 12 V measurement for *artemis* ($t_{\min} \approx 120\,\mu s$)



(d) Shunt-based 12 V measurement for *ariel* ($t_{\min} \approx 160\,\mu s$)



(e) Blade (one compute node) DC measurement for *taurus* ($t_{\min} \approx 2\,ms$)



(f) VR-based CPU0 (active processor package) measurement for *taurus* ($t_{\min} \approx 20\,ms$)

Figure 3.9: High-frequency variations in the power consumption observed with different measurements. Each chart shows a different time window. The top of each chart shows alternation of low-power load (green), high-power load (red), and synchronization (cyan). The bottom shows the power consumption in watt.

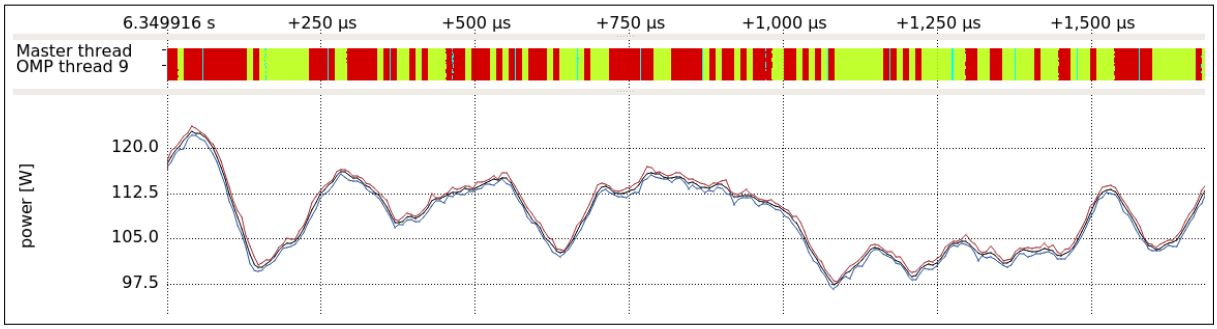


Figure 3.10: Power measurements of binary white noise on *ariel*. The top part shows low-power phases in green and high-power phases in red. The minimum, average, and maximum power is shown in blue, black, and red, respectively.

Still, the results gives insight on the temporal scale of dynamic behavior that can be observed with these measurements. The minimal observed workloads for blade and CPU measurements are $t_{\text{min,blade}} \approx 2\,\text{ms}$ and $t_{\text{min,vr}} \approx 20\,\text{ms}$, respectively. This is close to the limit given by the observable sampling rates of 1 kSa/s and 100 Sa/s; there are only two samples within each high/low plateau. This analysis shows that the sampling rate of a measurement alone does not imply the observable magnitude of temporal granularity, it only represents the upper limit. Hence, well-conceived choices of instrumentation point, sensor, and processing in addition to a sufficient sampling rate are necessary to observe short changes in workloads and their impact on power consumption. For sub-second down to sub-millisecond workloads, DC connections within a system are well suitable instrumentation points. Anything below $\sim 100\,\mu\text{s}$ requires an instrumentation closer to the consumer, e.g., between a consumer and its voltage regulators. If the subsequent analog and digital measurement infrastructure properly handles the high sampling rates, workloads lasting $\sim 12\,\mu\text{s}$ can be observed.

Evaluation of Frequencies within the Power Measurement Signal

To further strengthen the understanding of the measured signal and its bandwidth, I designed a binary white noise workload. This workload consists of the aforementioned low-power and high-power kernels configured for a duration of 10 µs each. Which of the two kernels is executed, is randomly chosen for each period of the workload signal. Therefore, the binary workload signal contains a maximum square wave frequency of 50 kHz. The workload implementation and its configuration is tuned to reduce unwanted influences by anything other than the workload kernels. Every 10 kernel executions, an OpenMP barrier synchronizes the threads. This effectively prevents different threads to drift apart over time at a limited impact of synchronization, which is typically around 2 µs. For selecting the kernel, the implementation uses a PCG random number generator [ONe14] due to its low latency and low memory footprint that could otherwise affect cache of compute kernel data. To expose the running kernel at any given time, I manually instrumented the workload with Score-P in a minimal way. For the following analysis, I used the *ariel* system with single-channel LMG670 measurements at $1.\overline{21}$ MSa/s. I selected this measurement setup because of its well-specified behavior at high frequencies. This allows me to keep the focus on the properties of the measurement signal at the instrumentation point, rather than influences of the sensors and analog measurement processing. The noise workload was executed for 20 s total, a short section of it is shown in Figure 3.10.

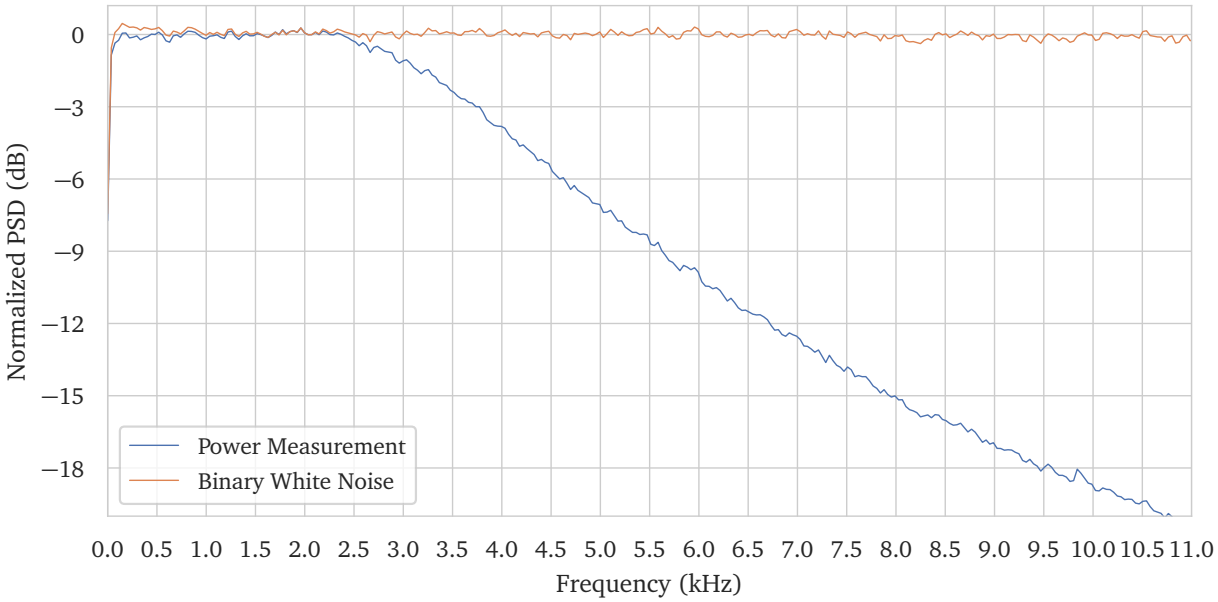


Figure 3.11: The power spectral density (PSD) of a binary white noise input signal (max. 50 kHz) and a power measurement signal on *ariel*.

To compare the workload signal with the power measurements, I applied several post-processing steps: First, I proportionally normalize the measurement signal such that the stable low-power of 92.11 W corresponds to −1 and the stable high-power of 124.92 W corresponds to 1. Initially, the workload state (high or low) is recorded as trace of events for each transition. To achieve a common sample rate, the workload state is sampled at the same time points of the power samples. Similar to the normalized power values, I chose the numerical state values such that the discrete low-power and high-power states correspond to −1 and 1, respectively. After this processing, there are two sequences (representing discrete-time signals) corresponding to the same equidistant timestamps: the normalized power and the discrete workload state. For a theoretical ideal power measurement, which would allow all frequencies, the normalized power signal and the workload signal would be identical. I analyze the resulting signal with Welch's method [Wel67] using the Python package SciPy [Vir+20] and normalize the power spectral density (PSD) according to the known workload frequency range and amplitude. Figure 3.11 shows the PSD power of the binary white noise workload and power measurements. Both signal amplitudes are very similar for frequencies $\lesssim 2.5$ kHz. For higher frequencies, the measurement signal strength decreases. The half-power cutoff frequency (−3 dB) is at ≈ 3.7 kHz. This is consistent with the previous observation, which revealed that short peaks of $t_{\text{min}} \approx 160\,\mu\text{s}$ are at the boundary of what can be clearly resolved by this power measurement.

This knowledge about the frequency spectrum of the effective power consumption signal can be used to chose appropriate sampling frequencies and design analog filters as discussed in Section 3.3.2. However, basing the sampling rate on the −3 dB cutoff frequency itself would still result in a significant aliasing error. For example, when considering a permissible attenuation of −20 dB, the observed power spectral density yields $B = 10.5$ kHz. Thus the minimal sampling rate would be 21 kSa/s. However, higher oversampling or the use of sharper low-pass filters with a smaller transition band can further reduce the error (see also [Web04, p. 22-4]).

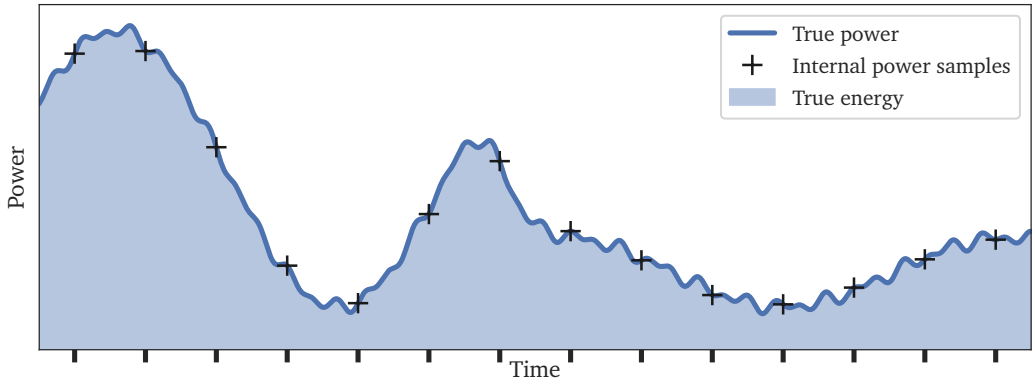
3.5.2 Retaining Energy Correctness During Digital Processing

Aliasing issues do not only have to be considered for analog-to-digital conversion, but also during the digital processing. Further, these issues not only apply to the fine-grained signals at DC and VR level, but also the AC total system power input. Generally, this common issue arises from a disconnect in the control flow and results in a loss of information. For example, an internal program, which runs on a BMC and reads an analog-to-digital converter, has access to the best possible information about the power consumption and energy. The visualization in Figure 3.12a displays the loss of accuracy due to limited sampling rates as discussed previously. The best way to preserve the remaining information would be to use the full trace of measured values and timestamps for all further processing. However, this is not always feasible. For instance, consumers of power measurement data often require a power value that is valid *right now* and implement their own readout routine to read measured values in peculiar intervals. Typical polling monitoring interfaces do not allow the internal readout to trigger an external reaction. The consequences of following an external readout interval are shown in Figure 3.12b. With an external readout rate that is lower than the internal sampling rate, values get lost, causing or amplifying aliasing issues. Typically, the most recently read value is returned from an interface, rather than issuing a new readout at the time of the request. Therefore, the actual (internal) measurement timestamps are lost. Instead external readout timestamps are used, adding a latency error. These effects can be partially compensated by triggering the readout at a higher rate than the internal sampling rate. In general, however, that is not feasible or desirable since it can cause performance issues. Lost samples and lost timestamps result in significant errors, particularly when attempting to compute an energy consumption.

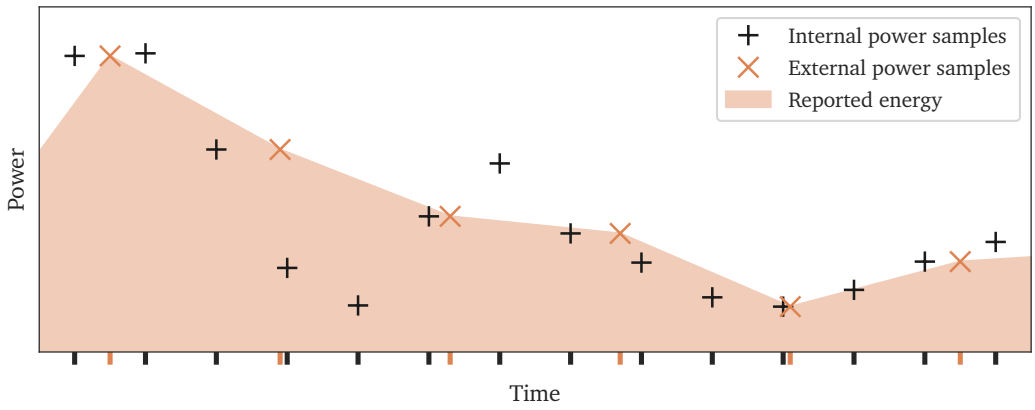
In [Hac+14], together with my co-authors, I introduced a solution by providing an atomic way to read an accumulated energy E_n and the most recent measurement timestamp t_n. This approach works without changing the control flow or using extensive power traces. While some dynamics of the original trace are lost, this information allows an external consumer to accurately determine the energy consumed between two measurements points as well as the average power:

$$P_{\text{average}}(t_{n-i}, t_n) = \frac{E_n - E_{n-i}}{t_n - t_{n-i}} \tag{3.5}$$

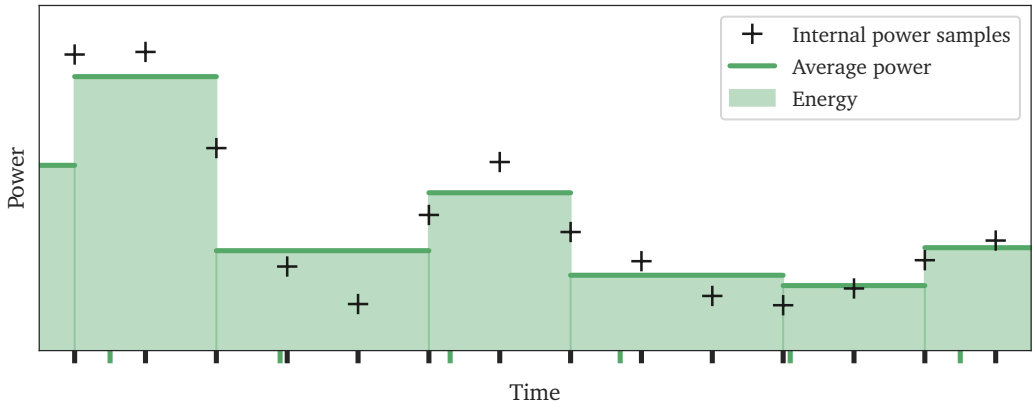
The described issue regarding loss of timestamp information affects several commonly used interfaces that are used for power and energy monitoring (see also Section 2.2.3). For instance, in addition to losing intermediate samples and timestamp information, IPMI suffers from reduced value precision when using its standardized sensor readings. While IPMI offers a sophisticated set of derived units, scaling factors, and nonlinear scales, the actual sensor value is contained in only one byte (see [Int+13, Sec. 36.3, Tab. 35]. The resulting relative value discretization error is $\frac{0.5}{255} \approx 0.2\,\%$ at the maximum of a linear scale and proportionally more for lower values. The aforementioned solution using timestamped accumulated energy values was implemented as IPMI OEM extension.



(a) Internal sampling of an analog signal



(b) External readout at orange ticks with loss of sampling times and intermediate values



(c) Using energy difference and associated sampling timestamps to reconstruct average power. Readouts happen at green ticks.

Figure 3.12: Information processing with internal sampling and external readouts.

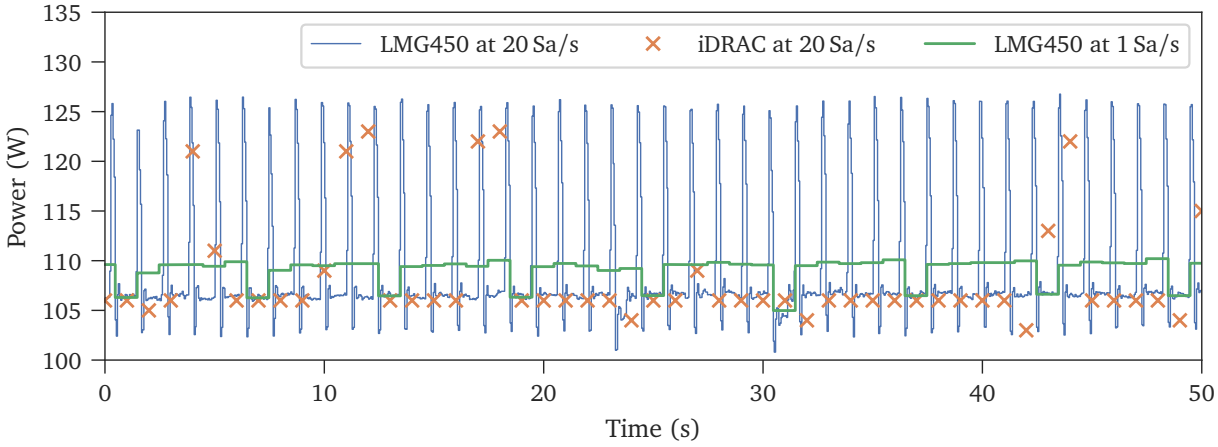
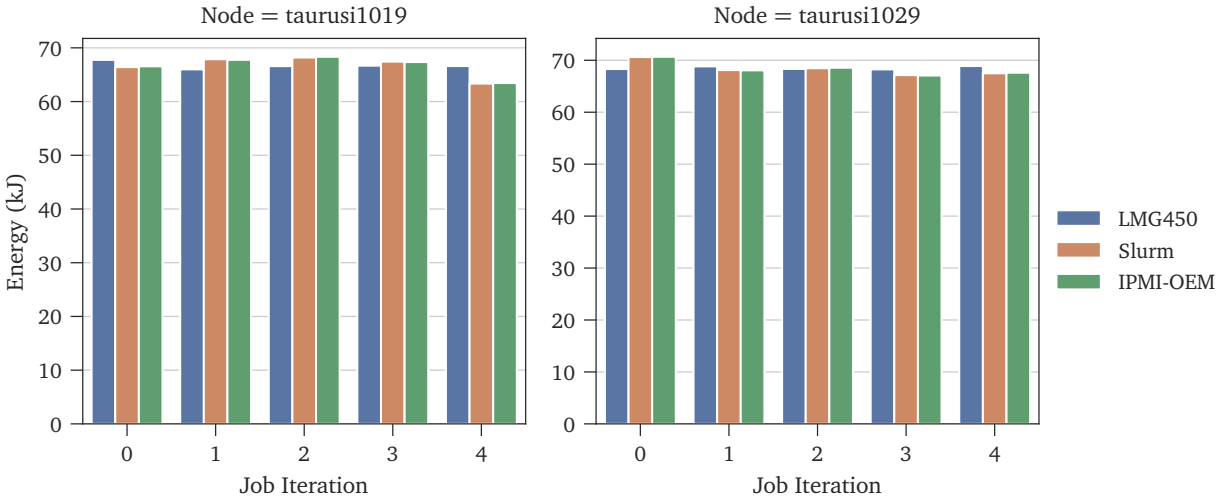


Figure 3.13: Different power measurements of a regular dynamic workload (repeating 1 s low-power and 0.2 s high-power, adapted from [Hac+13a]).

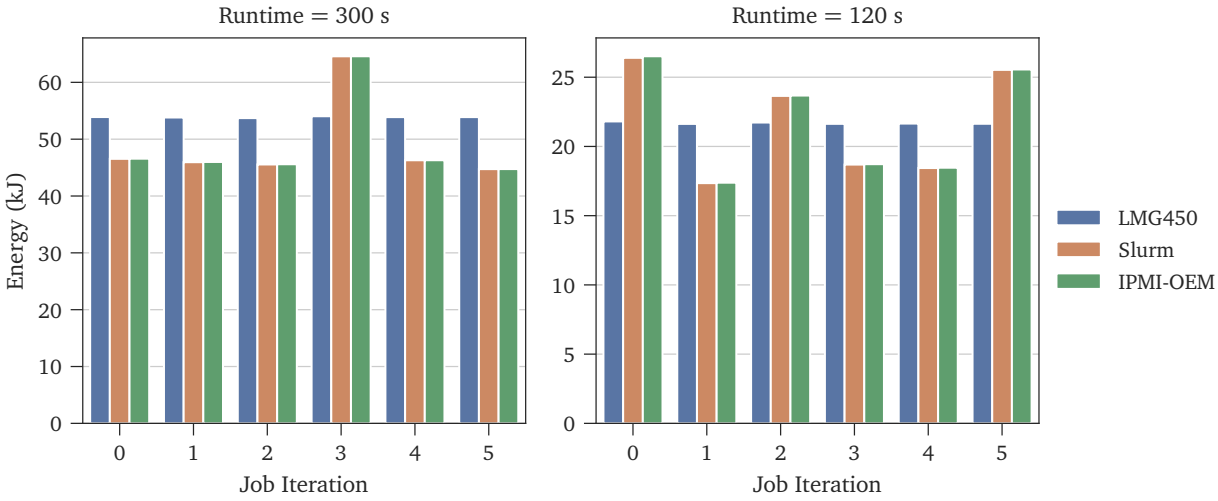
Practical Example of Errors Introduced by Digital Processing

As a concrete example of how these issues impact the accuracy of energy and average power readings, I use PSU-based measurements on a dual-socket Sandy Bridge Dell R720 system[10]. This system utilizes an integrated Dell Remote Access Controller (iDRAC) to provide power measurements via IPMI. DELL describes a "power monitoring accuracy" of 1 % [Del13, Tab. 19]. Using the IPMI sensor readout, however, the value resolution is 14 W, which introduces a discretization error of up to 7 W. In idle, this system consumes ≈ 70 W, resulting in an error of up to 10 %. Moreover, the generic IPMI sensor value is an average over the last minute, so it does not reveal the dynamic power consumption. However, access to the instantaneous value is possible with an OEM-IPMI extension, which returns a multi-byte value with a resolution of 1 W. Still, for idle states, discretization alone causes an error of up to 0.7 %. The returned value changes once per second and no timestamp is provided.
I used the LMG450 power emeter at the AC measurement for comparison (see Section 2.2.1). In a comparison of average power for a series of diverse but stable workloads, the maximum difference between iDRAC and reference measurement was 5 W as detailed in [Hac+13a]. In addition to the static difference, I evaluated the measurement with a synthetic dynamic workload that alternates between 1 s low-power (idle) and 0.2 s high-power kernels. As depicted in Figure 3.13, the measured iDRAC values are significantly affected by aliasing. The reference measurement at 20 Sa/s can clearly resolve the workload pattern, including a drop right after the high-power phase in which the AC power consumption is below the stable low-power. Moreover, I computed one-second power averages (1 Sa/s) based on the reference measurements. The plots of these average power values are not able to resolve the original pattern and show different patterns: While five values contain a short high-power phase, the sixth value contains only a low-power phase. Even though dynamic details are lost, the averaged power values are still correct and a correct energy consumption can be be computed based on them. In contrast, the iDRAC values show an irregular pattern which completely misses many high-power phases as well as some low-power phases. Due to this aliasing effect, it is impossible to compute the accurate average power or energy based on these instantaneous values for dynamic workloads.

[10] I use this additional system because none of the other instrumented SUTs provide PSU measurements.



(a) *taurus* phase 1, 170 ms high/low interval, 300 s runtime on two different nodes



(b) *taurus*, 270 ms high/low interval, 120 s and 300 s jobs on one nodes, before using HDEEM API

Figure 3.14: The job energy of a high/low FIRESTARTER workload, measured with the LMG450 reference power meter, the Slurm workload manager, and the IPMI-OEM extension.

Accurate Energy Accounting on HPC Systems

During the early development of HDEEM and the acceptance phases of *taurus*, several issues concerning aliasing and digital processing emerged. Initially, the system used the standardized IPMI sensor readings, which imposed the aforementioned issues. In order to allow accurate energy readings and energy accounting, the BMC firmware accumulates all power readings in an energy register which can be read through an IPMI-OEM extension. Each energy value is accompanied with a measurement timestamp from an NTP-corrected clock, allowing the average power computation as per (3.5). Contrary, the energy calculation from instantaneous values within the BMC is done based on the strictly-monotonic internal clock.

In the *taurus* phase 1 measurements, the BMC reads the values every ≈ 170 ms from a filtered signal. This allows it to provide low-dynamic but accurate energy values (see also Section 3.3.2). This interface is used by the Slurm workload manager to provide energy accounting on a per-job basis. I performed an acceptance test that included jobs that alternate between high-power and low-power

workloads at the BMC sampling interval of 170 ms. Since this is a relatively large time interval, it was not necessary to use the fine-granular `sqrt` / `compute` kernels in this context. Instead, I used FIRESTARTER [Hac+13b] because it exhibits higher power amplitude differences. I also used this setup for accuracy tests of *taurus* to test for end-to-end errors in energy accounting, which are caused by aliasing errors. In this test, the total job energy consumption reported by Slurm was compared with the energy computed by the LMG450 reference measurement as well as a manual energy computation based on separate IPMI OEM readouts and job start/end times. The extensive acceptance test with various workload frequencies revealed some remaining aliasing at a workload interval of 170 ms. Figure 3.14a shows the reported and reference job energy consumptions for two nodes and five repeated iterations of the same job configuration. The error in this configuration is between −5.0 % and 2.8 %, which is within the specification of the *taurus* phase 1 system. Values from Slurm and manual IPMI-OEM measurements are consistent, which indicates that Slurm introduces no additional error. Further, the energy consumption reported by Slurm varies between repeated job iterations whereas the energy values from the reference measurement are relatively stable. This results in varying errors from individual runs and nodes, likely caused by the remaining aliasing effect.

I re-evaluated the fully deployed *taurus* HDEEM system after the vendor calibrated the measurement as described in Section 3.4.3. To do so, I analyzed various workload frequencies. While the calibration reduced the overall error, dynamic workloads showed larger errors, especially for workload intervals of 270 ms. The results in Figure 3.14b show a significantly higher error ranging from −17.0 % to 21.1 %. This large error was caused by a slow readout interval within the BMC of 270 ms. As described in Section 3.3.2, the system was designed to allow high-resolution measurements at 1000 Sa/s. Thus, a digital polling and processing in the BMC at ≈ 3.7 Sa/s is insufficient. After I discovered this issue, the vendor updated the firmware such that the FPGA preprocesses the measured values at full temporal resolution. This change allows the BMC to keep an accurate accumulated energy value and exposes it with the newly designed HDEEM C API. Once Slurm was updated to exploit the HDEEM API instead of the previous IPMI OEM extension, the aliasing issue is no longer traceable.

3.6 Evaluating CPU Energy Counters

With the introduction of Running Average Power Limiting (RAPL)[11], energy values became widely available without the need for a separate instrumentation. Section 2.2.4 presented a brief introduction to the available interfaces and related work about the topic. In [Hac+13a], I discussed the principal interface of RAPL and its first implementation in the Sandy Bridge micro architecture.

On a low level, RAPL is configured and monitored using model specific registers (MSRs). According to the documentation, the read-only energy value is "updated every ∼ 1 ms" [Int18a, Vol. 3, Sec. 14.9]. Further, the documentation lists several power domains: *package*, *PP0* ("refers to the processor cores"), *PP1* ("may reflect uncore devices", only supported on client/desktop platforms) and *DRAM*. While the documentation states that the *DRAM* domain is supported only on server segment platforms, practically it appears to be available even on client segment systems [DPW16]. With the Skylake architecture, the *PLATFORM* domain was introduced, but is not available on all systems [Haj+16].

[11] RAPL was first available in the Intel Sandy Bridge microarchitecture, which was launched in 2011.

3.6.1 Energy Readouts with RAPL

By providing energy readings directly, RAPL does not exhibit the aliasing issues that prevent the computation of accurate energy values (compare Section 3.5.2). However, this interface still lacks the temporal information of the measurements. Not only does this lack of information prevent accurate attribution of consumed energy to a given period of time, it also impacts the accuracy when computing power. Consider the following example readouts:

$$t_0 = 0\,\text{ms},\ E_0 = 0\,\text{mJ}$$
$$t_1 = 3.5\,\text{ms},\ E_1 = 120\,\text{mJ}$$

Since the true measurement (update) time is unknown, there could be either three or four updates between the two readouts. The true time difference between measurements Δt could be either 3 ms or 4 ms. Consequently, the true average power is either 40 W or 30 W respectively.

There are different ways to mitigate this issue: If fine-granular information is required, one can oversample the registers, i.e., reading in less than 1 ms intervals. This ensures, that every internal update is observed, but also causes a high overhead. Since RAPL is typically read in-band on the system under test itself, this affects the perturbation of the measured application. Hähnel et al. [Häh+12] described an approach to measure the energy consumption for isolated short code paths. Their approach is to temporally align the code execution with the expected measurement updates. The relative error on average power can also be reduced by lowering the readout rate significantly. This comes at the cost of not being able to observe highly dynamic behavior, while also reducing the overhead and perturbation.

3.6.2 Methodology

In order to evaluate the accuracy of RAPL for a wide range of workloads, I leverage the methodology described in Section 3.4.1. The workload generator alternates a set of distinct kernels while varying the number and distribution of active threads as well as the core frequency. This generates a diverse set of operating points for a comparison with reference measurements. The comparison uses average power of of 6 s time intervals such that the error from inaccurate timestamps is negligible.

As a reference for the quantitative comparison of RAPL readouts, I used the modular measuring system of *ariel*, but applied it to *artemis* and *ariel* (see Section 3.1 for system description). In contrast to the original measurements of *artemis*, the *ariel* measurement separates between processor package and memory power consumption. This allows an extended analysis compared to my previous work in [Hac+13a] (Sandy Bridge) and [Hac+15] (Haswell). Moreover, the integrated LMG 670 power measurements are more accurate than the previous measurements of *artemis*.

Nevertheless, the power domains of RAPL and the reference measurement are not identical. The reference measurement is applied to the input of the VRs (at 12 V). Contrary, it is unspecified whether or not the RAPL domains include the loss of VRs. Therefore, the two values could be different even if

the the RAPL readouts were perfectly accurate[12]. However, I assume that there is at least a strong positive correlation between the reference measurement and RAPL output.
As indicated in Figure 3.15a, the reported RAPL power is consistently below the reference measurement. This suggests, that RAPL values, contrary to the reference measurement, do not include losses of the VRs themselves. Therefore, a direct comparison with the reference measurement would not be meaningful. Similarly to the evaluation in Section 3.4.2, I apply a VR efficiency model M for each of the evaluated RAPL power domains. Unless otherwise noted, I used a quadratic fit for modeling the efficiency. While this enables a quantitative comparison of the workload-specific inaccuracies, the model also hides any absolute calibration differences between RAPL and the true value at its actual measurement domain. Thus, I cannot quantify the absolute accuracy of RAPL with the given data. The following comparison uses the definitions for absolute and relative discrepancy (Δ/δ) from (3.2) and (3.3) respectively. Since δ can be skewed by low values of the `idle` configuration, I further introduce δ^*, which excludes the `idle` measurement point.

3.6.3 RAPL on Intel Sandy Bridge-EP

The Sandy Bridge (SNB) microarchitecture, provides the first implementation of RAPL. Rotem et al. described this implementation, although they do not refer to RAPL by name:

> Sandy Bridge implements architectural power meters. It collects a set of architectural events from each Intel architecture core, the processor graphics, and I/O, and combines them with energy weights to predict the package's active power consumption. Leakage information is coded into the die and is scaled with operating conditions such as voltage and temperature to provide the package's total power consumption. ([Rot+12, p. 22])

The authors further claimed that "the actual and reported power correlate accurately" [Rot+12, Fig. 3]. This claim refers to a timeline chart, which includes measured and predicted power during the course of a "combined CPU and graphics workload". However, such a timeline comparison with a dynamic mixed workload may average out specific inaccuracies. My methodology offers a more comprehensive comparison by quantitatively comparing distinct operating points.

Specific Power Connections of the Test System

For evaluating RAPL on the Sandy Bridge architecture, I measured package and DRAM separately on *artemis*. The specification of the mainboard describes the mapping of 12 V inputs to CPU packages and memory DIMMs: Two 12 V inputs separately provide power to the two processor packages (V_{CPP}, V_{SA}, V_{PLL}, and V_{TT}). The other two 12 V inputs provide power to two groups of four DIMMs each. However, the DIMM groups are intermixed among the processor packages such that the memory attached to each processor package cannot be measured separately at the 12 V inputs. The internal voltages provided to the DIMM groups include voltages for the DIMMs themselves (V_{DDQ} and V_{TT}) as well as one voltage (V_{DDR01} or V_{DDR01}) to the CPU package [Int15, Fig. 67]. It is not clear whether RAPL covers the same load voltages for its DRAM domain. An experimental distinction would require a full set of DDR3 riser measurement cards, which are not available for this system.

[12] Assuming perfectly accurate RAPL measurement would also imply that no error is introduced by the power measurement (compare Section 2.1.2).

Results and Discussion

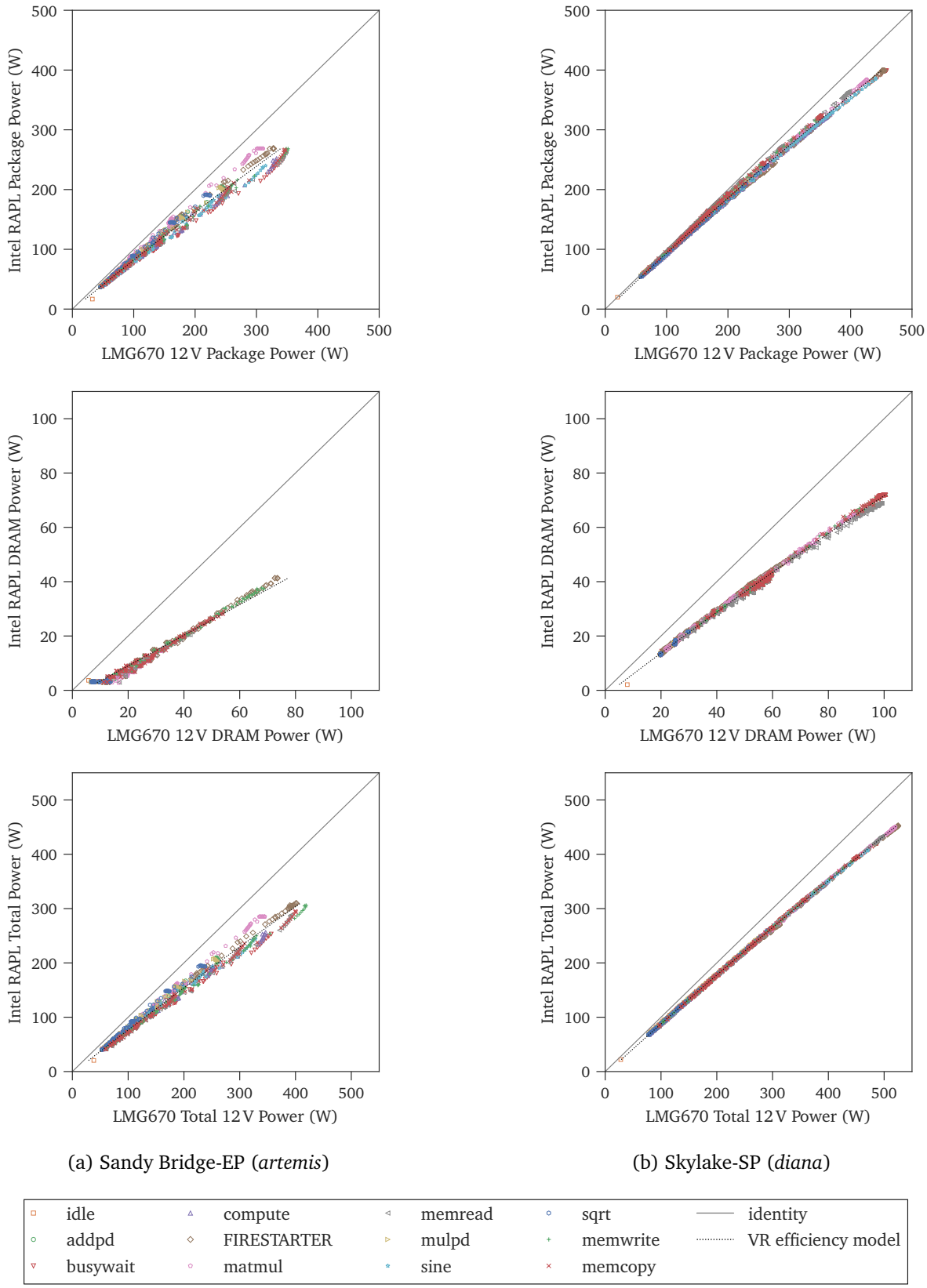
The measurement results are plotted for comparison in Figure 3.15a. For the package domain, the correlation between RAPL and reference power is weak and distinct patterns are visible: RAPL readouts of different workload kernels are consistently higher (e.g., `matmul`) or lower (e.g., `busywait`), even for data points with the same reference power values. This is a contradiction under the assumption that the VR efficiency does not depend on the workload. For the given workloads with a constant power consumption this is a reasonable assumption. Note that technically this domain comprises different internal voltages. To some extend, it is conceivable that the workload stress the internal voltages differently. In combination with different efficiencies at different voltages, this could explain some level of discrepancy. In any case, the results show that RAPL introduces significant errors when considering actual power consumption of the full system, even when accounting for systematic error and power domain differences.

The comparison for the DRAM power domain paints a different picture: While closely correlated at higher power consumptions, there is a significant discrepancy for lower-power configurations. Workloads that do not stress the memory exhibit a constantly low reported RAPL power, while the reference power varies significantly. For example, for most configurations of the `addpd` kernel RAPL reports a power from 3.13 W to 3.18 W, while their respective reference power ranges from 6.83 W to 12.94 W. Consequently, the absolute RAPL discrepancy for the DRAM domain is lower than for the package domain, whereas the opposite is true for the relative discrepancy, even without considering the `idle` configuration (see Table 3.5). Note that fitting this particular VR efficiency model resulted in a negative quadratic factor. Thus, I used a more plausible linear model, which also exposed higher discrepancies.

The comparison of total power consumption (the sum of package and DRAM for both RAPL and 12 V reference measurement) is dominated by the same effects observed for the package domain. Quantitatively, the discrepancies regarding the total power are very similar to the discrepancies for the package power. In comparison to the package domain, only the `idle` case exhibits a significantly reduced δ due to the added base memory power. In the sum, the high relative discrepancy the DRAM domain is masked by the larger absolute values. The observation that the absolute discrepancy on the total power domain is not better than the separated domains, indicates that the covered RAPL load voltages match the ones used of the 12 V reference measurement. Otherwise, the errors from different domains would cancel each other.

The exposed discrepancy limits the applications of RAPL as a replacement for measurements. Particularly the systematic overestimation or underestimation for certain workload kernels prohibits an accurate evaluation of trace-offs between different algorithms. This also confirms, that RAPL uses an internal model rather than an actual measurement (see [Rot+12]). Other models based on performance events show similar characteristics of workload-dependent errors (compare [Bie15]). Consequently, any model built from RAPL readouts will inherit the errors.

Overall, the observation is consistent with my previous results presented in [Hac+13a]. The separation into package and DRAM power yields further insight into different individual inaccuracies. The previous work also highlights specific discrepancies in hyper threading configuration, which are not covered in this thesis [Hac+13a, Fig. 12].



(a) Sandy Bridge-EP (*artemis*)

(b) Skylake-SP (*diana*)

Figure 3.15: Verification of RAPL on different generations of Intel processors.

Table 3.5: The maximum discrepancies between RAPL readouts and the measured reference power under a VR efficiency model.

	Package Max.			DRAM Max.			(Package + DRAM) Max.		
SUT	$\lvert\Delta\rvert$ (W)	$\lvert\delta\rvert$ (%)	$\lvert\delta^*\rvert$ (%)	$\lvert\Delta\rvert$ (W)	$\lvert\delta\rvert$ (%)	$\lvert\delta^*\rvert$ (%)	$\lvert\Delta\rvert$ (W)	$\lvert\delta\rvert$ (%)	$\lvert\delta^*\rvert$ (%)
artemis (SNB)	29.6	36.4	15.5	4.0	126.5	59.6	30.1	27.0	15.1
ariel (SKL)	10.2	23.9	5.7	2.8	51.8	7.1	6.1	3.8	3.3

3.6.4 RAPL on Intel Haswell-EP and Skylake-SP

The Haswell-EP processor generation featured an improved implementation of RAPL, as I described in [Hac+15]. Facilitated by the introduction of fully integrated voltage regulators (IVRs) [Bur+14], this implementation is based on physical measurements rather than an architectural model.

In the following, I evaluate RAPL on the next Intel high-performance processor generation, Skylake-SP (SKL). As system under test, I used *ariel*. The public documentation of its mainboard does not describe the specific electrical connections of the 12 V inputs, but indicates that there are separate power pins for the two processor packages the their associated memory [MiT17, Sec. 2.12].

Results and Discussion

Figure 3.15b reveals a more precise correlation between RAPL readouts and measured reference power, similar to the results on the Haswell system. Table 3.5 quantitatively confirms a significant improvement compared to Sandy Bridge-EP for all measurement domains. However, the discrepancy for separate package and DRAM measurements is noticeably higher than for the sum of power consumptions. This anomaly can be explained by a slight mismatch in how the power consuming components are split between the two RAPL domains and the respective 12 V pins.

It is noteworthy, that Haj-Yahya et al. [Haj+19], citing [Fay+16], report that Intel Skylake processors no longer utilize IVRs. However, this statement may only refer to the mobile and desktop variants, rather than the processors targeting servers. Nevertheless, RAPL continues to show reliable results.

Neither RAPL nor its implementations provide an uncertainty specification, hence the evaluation with reference measurements is necessary to understand the quality of RAPL readouts. While the correlation with a reference measurement clearly improved with contemporary RAPL implementations, this evaluation can not verify their absolute accuracy. Moreover, RAPL does not provide the temporal resolution of a sophisticated measurement setup as described in this section.

Regardless of the accuracy of energy values, it is always important to make a conscious decision of the measurement domain. While RAPL offers separate power domains for the processor package, cores, and memory, it does not provide the full-system power consumption for high-performance systems. Optimizations with the prevailing goal of reducing total energy consumptions are not possible based solely on RAPL. Since power measured by RAPL is not strictly proportional to full-system power, the static consumption from other components outside the covered domains as well as non-linear efficiency losses have to be considered for energy-efficiency optimizations.

3.7 Conclusion

In this chapter, I described several approaches for measuring the power consumption of computing systems with a focus on pushing the boundaries of temporal and spatial resolution. In addition to the instrumentation approaches, I provided rigorous techniques for evaluating uncertainty and actual temporal granularity.

By the means of four systems under test, I discussed three specific power measurement instrumentations in detail. This includes the choice of measurement domain and instrumentation point as well as the possible current sensors. In particular, I designed and implemented several synthetic workload generators to systematically expose errors and aliasing effects. A workload alternating low-power and high-power at varying intervals allows a visual inspection of power transitions at the scale of microseconds. I further introduced a synthetic workload that generates a power consumption signal containing binary white noise to evaluate the frequency spectrum for measurements at a certain measurement point.

The results show that shunts and Hall effect sensors deliver good accuracy and can show details in the order of 150 µs at the spatial granularity of processor packages and memory groups. Measurements at voltage regulators can increase the resolution to $\approx 12\,\mu s$ and separate voltages of a processor, at the cost of higher uncertainty and a more complex instrumentation.

I further characterized and evaluated the scalable measurement solution HDEEM. My evaluation provided valuable feedback for sustained improvements: On the one hand, it led to an in-situ calibration of the measurements. On the other hand, my specifically designed workloads exposed aliasing issues, which facilitated a redesign of software interfaces. My deliberate evaluation of the improved measurements now provides a strong confidence for a wide range of uses.

Finally, I evaluated CPU energy counters as an alternative to sophisticated measurements. I systematically exposed discrepancies of early implementations based on a comparison with a reference measurement for a wide range of workload configurations. In addition to a graphical representation, I quantified the discrepancies by compensating for the divergent reference measurement domains. I further showed the improvements to RAPL implementations in contemporary processor generations.

The measurements described in this chapter provide the foundation for the remainder of this thesis. However, the high resolution and large scale result in a large amount of data, which presents a challenge for processing, storage, and analysis. Consequently, the following chapter introduces a measurement data infrastructure that addresses this challenge and facilitates further analysis.

4 A Scalable Infrastructure for Processing High-Resolution Power Measurement Data

Many traditional power analyzers focus on user interfaces on the device itself rather than a digital read-out. While this is suitable for classical analysis use cases, it does not fulfill the requirements for measuring the power consumption of computing systems. Especially the scale of High Performance Computing and the amount of information from high-resolution measurements provide a challenge to gain actual insight from the large volume of measured values.
Therefore, in Section 4.1, I first discuss the requirements for the processing of power measurement data, particularly considering the measurements presented in Chapter 3. Section 4.2 describes the concepts and implementation of a scalable measurement data infrastructure. This infrastructure comprises scalable distributed services connected by a high-performance message broker. In Section 4.3, I provide an extensive performance evaluation of the proposed infrastructure.

4.1 Requirements for Power Measurement Data Processing

Power measurements present specific and challenging requirements for a data collection and processing infrastructure. The range of possible data rates and cardinalities is large: While some measurement devices provide values at less than 1 Sa/s, other devices go beyond 1 MSa/s. This wide range of readout rates is particularly difficult to cover by with existing solutions (see Section 2.3). A trivial criterion for suitability is the precision of timestamps supported by a measurement infrastructure: A resolution of 1 ms cannot support update rates above 1 kSa/s. In terms of cardinality, a sophisticated instrument observing a single compute node may work with less than 10 channels while an HPC system has several thousand nodes each with multiple measurement points. Measurement data can originate from a variety of different sources, e.g., a single sophisticated measuring system, multiple PDUs, and numerous compute node BMCs. Ideally, a unified measurement infrastructure supports a broad range of configurations to allow the re-use of common software components.
Moreover, there is a range of different use cases for consuming power measurement data. Monitoring typically uses live data, e.g., to provide gauges and live charts with the current power consumption or to check thresholds for triggering alerts. Persistent storage is necessary for long-term analysis, plotting charts of power over time in the past, and energy accounting.
Finally, the infrastructure must support integrating power measurements with application and system measurements. In the following, I refer to this major use case as *application power tracing*, even though this also includes monitoring of system events. Analyzing a delimited experiment is typically a post-mortem process to avoid perturbation from handling power measurements during the experiment (see Section 5.1.3). This requires at least temporary storage of full resolution measurement data by the infrastructure for the duration of the experiment. For some use cases, profiling, i.e., collecting

only aggregate data from both the application and power measurements, may be sufficient. Generally, profiling is less demanding although there can be different challenges: For instance, when creating a profile during the execution of an application, the latency of retrieving the up-to-date measured power value may be critical. I primarily consider the requirements of tracing because traces can be used to generate profiles, but not vice-versa (see also Section 2.5).

As discussed in Section 3.5.2, it is essential to accurately retain the source timestamps whenever processing measurement data. In certain cases, it can be necessary to aggregate measurement data in order to make storage, processing, or visualization of large volumes of measurement data feasible. Still, any aggregation of power measurement data should retain the best possible amount of information. For example, a plot of the average power consumption using correct interval times conveys accurate information of the dynamic energy consumption and is thus more valuable than a plot of sampled instantaneous power values (compare Figure 3.13). If a chart further includes minimal and maximal power consumption for the displayed intervals, it yields more insight. This is especially the case for system operation, where it is important to identify peaks and outliers.

In this thesis, I address the specific challenges related to handling power measurement data. Therefore, there are certain aspects that are not covered in the context of this thesis. While the concept and implementation of the proposed infrastructure can be applied to a broad range of measurement data from different kinds of sensors, I focus on its usage for power consumption metrics. In practice, my proposed infrastructure is also used for various data center metrics such as temperature measurements, fluid flows, and valve positions. In a production environment with many users and actors, authentication and permission management is an important aspect. Although my concept supports authentication and authorization in principle, I do not consider the details and implementation for this thesis. Encryption is considered implicitly at transport level via TLS. In terms of scalability, I evaluate the proposed concept according to the challenging practical requirements for power measurements of a high-resolution power analyzer and a petascale data center. Even further scalability can be achieved by clustering and federation of the message broker. However, since it is not necessary to meet the defined requirements, it is not included in this evaluation. While it is important to also provide metadata for measurements, I only consider this in a very general form, but do not provide a specific taxonomy. Further, facilitating resilience of the system and integrity of data is assumed to be provided by the underlying interfaces and protocols and thus is out of scope for this thesis. Based on the practical use cases, I define the following minimal performance requirements for the measurement infrastructure:

(a) Processing measurement data from one data source providing six metrics with 1 MSa/s per metric (approximately the highest-resolution configuration of an LMG670 power analyzer).

(b) Processing measurement data from 1000 data sources providing six metrics each with 1 kSa/s (approximately the high-resolution data rate of HDEEM on *taurus*).

(c) Processing data of 25 000 metrics at 1 Sa/s (a high cardinality use case, e.g., data center monitoring).

(d) Insertion of measurement data into persistent storage at a total rate of 1 MSa/s with one storage agent (the temporal resolution for persistent storage is limited due to storage space).

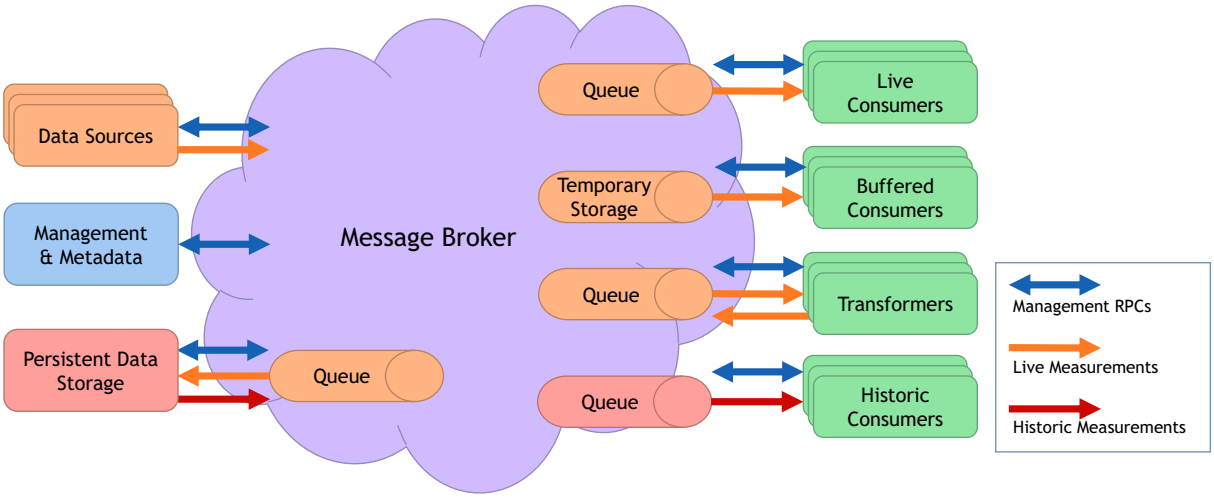


Figure 4.1: Overview of a measurement data infrastructure using distributed services and a message broker.

(e) Collecting measurement data at 1 MSa/s for a 5 min experiment and retrieving (draining) it in < 1 min, thus retrieving at a rate of > 5 MSa/s (application power tracing).

(f) Querying timelines and aggregates for arbitrary random intervals with a maximum response time of 1 s from a persistent data set with six metrics at 1 kSa/s over one year. This facilitates interactive visualization without interrupting *the user's flow of thought* [Nie94].

4.2 Concepts and Implementation of Measurement Data Management

In the following, I describe MetricQ, a concept and implementation of an infrastructure for scalable metric data processing[Ils+19][1]. An overview for this infrastructure is given in Figure 4.1. The following description includes protocols, interfaces, and core components of the infrastructure. MetricQ consists of the following kinds of loosely coupled *agents*:

- *Data sources* provide measurement data for one or multiple metrics. Such a software component could control a measurement instrument, fetch data from a monitoring device, or forward local information from a system under test (e.g., CPU energy counters).
- *Live consumers* receive incoming values of a certain subset of metrics, typically for immediate processing. Examples are dynamic visualization in dashboards or continuous monitoring and analysis to detect anomalies.
- *Buffered consumers* also use live data, but in a time-delimited workflow that uses a temporary recording and then receives all buffered measurement data in bulk. The typical use case is the recording of an *application power trace*.
- *Transformers* preprocess measurement data for further analysis, e.g., by filtering or combining multiple metrics. They act as both live consumers of the raw measurement data and sources for enhanced metrics.

[1] I presented MetricQ in [Ils+19] together with my co-authors. This publication overlaps with Section 4.2 and Section 4.3.7 of this thesis and was written after the respective sections of this thesis were drafted.

- *Persistent data storage* agents act as a consumer for incoming live data and provide recorded measurement data on request of historic consumers.
- *Historic consumers* use persistently stored measurement data. This is often used to generate interactive charts for arbitrary periods of time in the past.
- A *management* agent is responsible for providing the configuration for data sources and orchestrate connections between sources and consumers. The management agent also collects and provides metadata for metrics in the infrastructure, but is not involved in any transfer of measured values, neither live nor historic.

Metrics

Context is essential in order to gain knowledge from a series of measured values. To that end, the infrastructure concept uses *metrics*, where a metric identifies one specific measurand that is repeatedly measured over time. Metrics have a unique name, which is composed of fragments separated by dots. While there is no rigid specification for metric names, each fragment should be self-descriptive. Components of multiple metrics form a tree, presenting a hierarchy with the first component containing the largest set of metrics. By convention, the fragments specify a system component or location, starting with the coarsest location. The last fragment then specifies the name of the measured quantity. An example name would be `elab.ariel.s0.dram.power`, where `elab` is the name of a laboratory, `ariel` a hostname, `s0` short for the first socket, `dram` for the memory within the socket, and `power` is the measured quantity. In addition to the implicit naming scheme, each metric is associated with a set of flexibly definable metadata. Some metadata should always be specified as it is of particular interest to interpret the numerical values, i.e., the *unit* symbol [Int06, Tab. 3], a textual *description*, and the update *rate* (in Sa/s).
The spatial hierarchy in the naming allows to easily refer to a set of metrics, e.g., all metrics within the scope of `elab.ariel`. Regardless of that, measurement data for distinct metrics is technically strictly separate. As I will demonstrate in Section 4.3, this property enables a trivial parallelization of key services in MetricQ on the one hand, and an efficient selection of measurement data on the other hand. However, in some cases, this approach can lead to redundancy. For example, this scheme does not allow grouping of data for multiple metrics from one measurement device or otherwise closely related measurands, which could be processed in bulk messages or share the same timestamps for their measured values. The set of metrics and their metadata is collected by the management agent.

4.2.1 Message-Based Communication between Agents

All communication between the agents in the system is performed using a message broker. This provides several advantages over direct communication and enables crucial features of the infrastructure. Decoupling data sources and consumers allows for a very flexible and dynamic mapping based on the metrics of interest. For example, a data source can provide measurement results for any number of consumers without being aware of the consumers and even without establishing direct network connections. A source is agnostic as to whether its measurement data undergoes live analysis, is used in application power tracing, or stored persistently. This simplifies the implementation of sources for new

measurement devices. In contrast, the previous implementation for high-resolution measurements for a specific data acquisition interface included the temporary storage on the measuring system. This previous approach required a complex monolithic implementation, which sent aggregated data to persistent storage, controls and buffers experiments, and sends high-resolution data for application power traces using a custom protocol. The memory requirements for temporary storage also resulted in an inflexible deployment. Moreover, with the novel approach, consumers can utilize measurements of many sources, or just a subset of the measurements of one source, on a per-metric basis. This means that less software components are required in order to integrate different measurements on a system under test.

Data queues within the message broker allow simple publishing of measured values without waiting for consumers. In the case of application power traces, queues act as temporary storage to collect measurements for the duration of an experiment. Queues also provide decoupling such that all consumers can be restarted or temporarily disconnected without the loss of measurement data. For example, a persistent storage agent can be rebooted and still retain a complete continuous recording of measured values. Management control, configuration, and metadata messages are also transmitted using the message broker, thus providing the same benefits. This approach does introduce a crucial dependency to the message broker. The resulting strict requirements for stability can be fulfilled by modern message broker software and standardized message queuing protocols.

The selective routing by the message broker ensures that the measurement data is only handled by those agents that need it. This is necessary to enable the very high sampling rates and thus data rates of sophisticated power measurement devices. That also means that the management agent does not need to handle any actual measurement data. Further, all consumers receive only the measurement data that is relevant for them on a per-metric basis. This concept for distribution provides an inherent scalability in the infrastructure. The only limiting factor is the message broker, which can exploit federation and clustering for scalability. All agents in the system that handle measurement data can be trivially parallelized on a per-metric basis, unless there is an external requirement for an agent to process a large set of metrics. For example, persistent storage in the infrastructure can be arbitrarily split across multiple agents, which cover disjunct sets of metrics. A parallelization of the management agent is typically not necessary, since it does not handle measurement data. Still, it is possible to perform the management functions by a set of parallel agents as long as they operate stateless or somehow maintain a coherent view of the state of the system.

Since the management agent orchestrates the dynamic configuration of the message broker, it offers the possibility to enforce permissions. For example, if a consumer wants to receive data from a set of metrics, the management agent sets up the necessary queues and routing and could also check if that consumer has permission for these metrics. As mentioned previously, while this possibility is intended in the design, the specifics of it are not in scope of this thesis.

The implementation of this design uses the Advanced Message Queuing Protocol (AMQP) in version 0-9-1. With AMQP, messages are *published* to a specified *exchange* and then delivered to message queues depending on the type of the exchange and the routing configuration. Respectively, consumers then *subscribe* to *queues* and retrieve the messages. Note that in terms of MetricQ, agents are only considered to be consumers if they receive either live or historic metric data. In the sense of AMQP, all

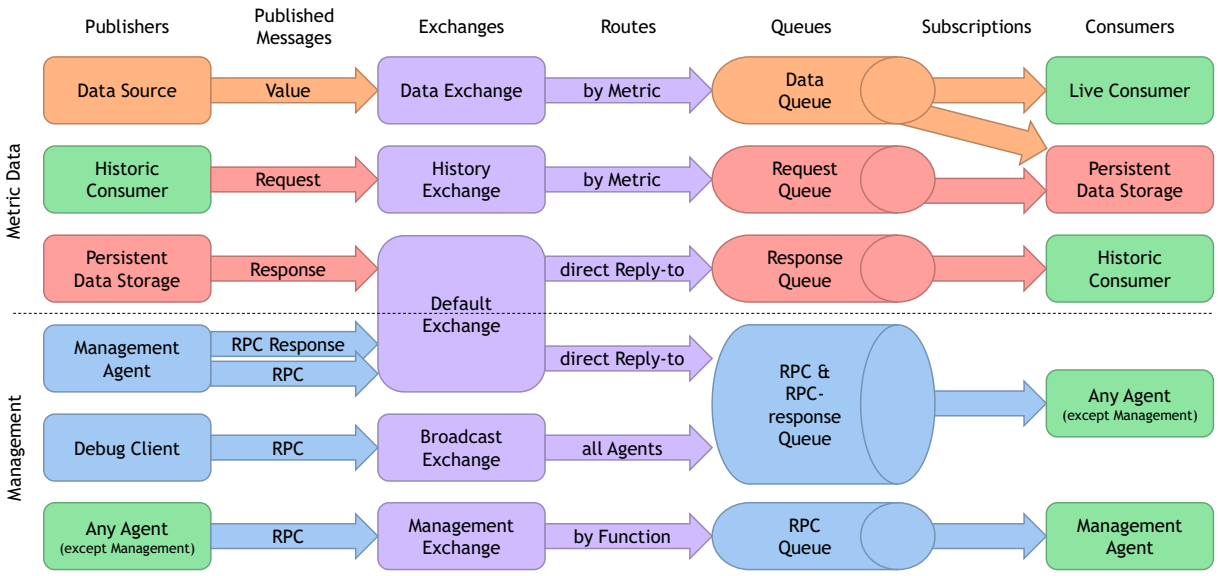


Figure 4.2: The exchanges and message paths within the RabbitMQ configuration of MetricQ.

MetricQ agents are consumers, at least of RPC-related messages. The routing of messages in AMQP depends on the exchange type. MetricQ uses three exchange types:

- A *topic exchange* delivers messages to any number of queues based on the *routing key* of a message and patterns used for *binding* queues to an exchange.
- The *default exchanges* uses a special implicit routing. All messages are routed to the queue that has the same name as the routing key of the message.
- *Fanout exchanges* send messages to all queues bound to it, ignoring the routing key of incoming messages.

For MetricQ, I use RabbitMQ [Piv19], an open-source implementation of AMQP, which is written in Erlang. AMQP was formalized as an ISO standard in version 1.0 [Int14b]. However, AMQP 1.0 offers fewer semantics and there are less client implementations available [Piv19, Supported Protocols]

Exchanges

The following describes how MetricQ leverages the existing RabbitMQ functionality with its particular configuration. An overview over the five exchanges used by MetricQ as well as the respective publishers, messages, routes, queues, subscriptions, and consumers is shown in Figure 4.2. Sources send the measurement results to the *data* exchange. This is a topic exchange that forwards the messages to any number of queues based on the metric name as the routing key and queue bindings (subscriptions to a set of metrics). The message broker discards data messages if there is no current subscription for a specific metric or duplicates it if there are multiple subscriptions.
Requests to receive a trace of power measurements from persistent storage are sent to the *history* exchange. Similarly to the data exchange, this is a topic exchange with the metric names used as routing keys. This allows separate agents for persistent data storage to be responsible for different disjunct sets of metrics.

The *management* exchange is used whenever another agent wants to communicate with the management agent. This is also a topic exchange, but uses the requested remote procedure call (RPC) function as a routing key, allowing to functionally distribute management messages across multiple agents.
While most published messages are not addressed to a specific receiver, but rather to a management function or just the data or history exchange in general, some messages must be sent to a specific agent. Such messages are published to the AMQP *default* exchange and are therefore addressed to specific queues that are exclusive to individual agents[2]. This scheme is used for response messages of RPCs and history requests. Moreover, this exchange allows to invoke RPCs in any non-management agent, e.g., for triggering a configuration update on a data source.
An additional *broadcast* (fanout) exchange allows to send RPCs to all agents in the infrastructure. While this is not necessary for the primary functionality, it can be used to discover all active agents or for debugging purposes. Exchanges are initially setup by the management agent, but they are also configured to be durable so that they are still available after a restart of the message broker.

Queues

On the consumer side, queues and routing configuration control the delivery of messages. All consumers of measurement data receive data messages through queues that are connected to the data exchange with the metric name as routing key. The use cases for live consumers are typically transient, e.g., a user selects a set of metrics for a live visualization. Queues for transient consumers carry unique names and are created by the management agent on request (*subscribe*) of consumers. They are configured to be automatically deleted by the message broker once the consumer disconnects. For the case of application power traces, this queue acts as temporary storage. Transformers and persistent data storage agents use similar queues, except that they are not transient. This allows a persistent storage or transformer agent to restart without losing any measurement data.
Queries for historic metric series use two more types of queues: Each persistent data storage agent uses a queue to receive queries for its set of metrics. Answers containing the query result are addressed via the default exchange to exclusive response queues of the requesting historic consumers.
The *management* queue receives messages through the management exchange. In the current implementation, the system uses a single management queue, which is configured to receive messages with any routing key (all RPC functions). However, it is possible to split this into multiple queues to distribute different management functions do different management agent implementations. Typically, there is only one consuming management agent instance to the management queue, but it is possible to replicate the management agent for load-balancing. The management queue is configured to be non-exclusive, which means that management request messages are also buffered when the management agent is temporarily unavailable. This enables a restart of the management agent without losing RPCs, as long as it completes within the timeout duration of the client.
Each agent, except the management agent, has an exclusive queue to receive RPCs and RPC responses. This RPC queue is the only queue that is declared by agents themselves, all other queues are declared by the management agent.

[2]All agents are identified by a unique name, its *token*. The token is also used to build the agent's RPC queue name.

4.2.2 Protocols

Two different protocols are used for communication between agents. Communication between the management agent and other agents is done in the form of remote procedure calls (RPCs). Hence, each call involves two messages, a function call and a response. An empty response is sent if the RPC has no return value. Both calls and responses use JSON to leverage a flexible definition of arguments and return values. All calls include a *function* attribute in both the JSON payload and as routing key, which enables flexible dispatching of RPCs. The other attributes depend on the function and can have different types and default values for convenience.

Contrary to the RPCs, messages containing *metric data* require a more efficient encoding than the verbose JSON. Therefore, MetricQ uses Protocol Buffers (protobuf)[3] for all messages that contain measurement data. Protobuf uses a binary encoding that is efficient both in terms of processing time and message size, while retaining portability and extensible definitions. The main structure for measurement data is the `DataChunk`, which contains a list of timestamps and a list of values. Both lists are of equal length and the nth timestamp corresponds to the nth value. Timestamps are differentially encoded, i.e., the first timestamp is an actual timestamp whereas all following timestamp values are computed as the difference of successive timestamps. The use of a separate timestamp list in combination with the differential encoding enables a very space-efficient encoding. Timestamps refer to the number of nanoseconds since 00:00:00, 1 January 1970 UTC minus any leap seconds, i.e., POSIX time or Unix time. While the timestamps are 64-bit signed integers in memory, protobuf uses base-128 variants for encoding. This encoding means that smaller values require less space, e.g., values less than 16 384 use 2 B in the buffer. Metric values are always represented as double-precision floating point, both in memory and as encoding. Request messages for historic data primarily include timestamps, while their respective responses have similar lists of timestamps and values as the live measurement data messages.

The ability to include more than one timestamp / value pair (chunking) and the efficient encoding stems from the requirement of supporting measurement with high update rates for individual measurement points in the order of 1 MSa/s. By having multiple measured values in one message, the relative cost of headers (space) and message handling (time) is reduced significantly. Enabled by the differential timestamps, the variable encoding is particularly efficient for high update rates — where it is also most important. For example at update rates of ≥ 62 kSa/s, only 10 B per value need to be transmitted (instead of 16 B per value). Isolated benchmarks with such metric time series data show that the fast generic compression algorithm LZ4[4] provides a similar data reduction ratio of ≈ 0.6. However, LZ4 compression/decompression is slower than protobuf encoding/decoding. In the context of general monitoring data, it has been demonstrated that floating point values can be efficiently encoded when applying bitwise XOR to successive values. It is, however, not clear how effective this approach would be for power measurement data in particular (e.g., [Pel+15]). A detailed comparison of the current approach using protobuf with different compression methods remains as future work.

[3] `https://developers.google.com/protocol-buffers`
[4] LZ4 — Extremely fast compression: `https://lz4.org`

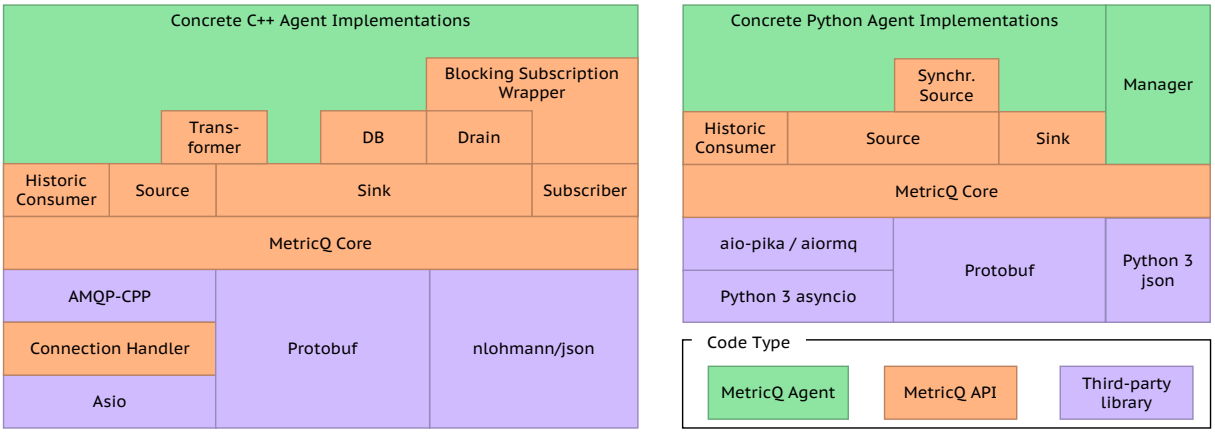


Figure 4.3: The components of the MetricQ C++ API, the Python API, and used third-party libraries.

4.2.3 Application Programming Interfaces

MetricQ provides two application programming interfaces (APIs) in order to support the efficient development of agents for the system. These interfaces hide all details of the implementation, e.g., messages and exchanges, while offering a high-level notion centered around metrics and data points. A C++ library is used for applications that require the best possible performance, while a Python module increases development productivity. Both interfaces are fully asynchronous, which enables efficient message handling without the necessity for additional threads. These interfaces also provide abstractions for executing RPCs and handling their responses. Both libraries are built on top of existing open source libraries to process AMQP, as well as protobuf and JSON encoding and decoding. Figure 4.3 gives an overview over the APIs and leveraged libraries.

The C++ library utilizes `AMQP-CPP`[5] for the message layer as well as `asio`[6] for TLS, networking, and the asynchronous event loop. A hierarchy of abstract base classes represent the different kinds of agents, i.e., consumer (also referred to as sink), source, persistent storage (db) and transformer. Those specific agents are implemented by deriving from them and implementing virtual methods. Collecting data for application power traces uses two different classes: One control agent handles the initial subscription. Another special sink, the *drain*, performs the eventual retrieval of the buffered measurement values. Additionally, blocking wrapper functions are available for the subscription and drain. These functions still use an asynchronous event loop internally, but hide it to simplify certain use cases that do not benefit from an asynchronous interface.

The Python library is built on top of `aio-pika`[7]. Using special decorators enables concise dispatching of RPCs, a functionality mainly used by the implementation of the management agent. While Python imposes a performance overhead, the use of native coroutines greatly simplifies the asynchronous control flow as opposed to the callback-based control flow in C++. In addition to the asynchronous polymorphic approach, there is a procedural blocking interface for sources, which allows easy adoption of existing code . This blocking interface internally runs a thread with the asynchronous event loop and dispatches all new measurement data as tasks to this event loop.

[5] `https://github.com/CopernicaMarketingSoftware/AMQP-CPP`
[6] `https://think-async.com/`
[7] `https://github.com/mosquito/aio-pika`

4.2.4 Efficient Metric Time Series Storage and Retrieval

In addition to live analysis, storage of metric data is crucial for many power and energy analysis use cases. This applies to both temporary and persistent storage.

Temporary Storage in Message Queues

Temporary storage for limited-time experiments is provided by the message broker. When starting the recording of an application power trace, a set of metrics is *subscribed*, i.e., the management agent creates a queue and bindings for the given metrics in response to an RPC. Over the course of the experiment, incoming messages containing data for these metrics are delivered to this queue, but not consumed. Once the experiment is completed, *unsubscribing* this queue will delete the bindings and generate a special end-message to mark that there will be no further messages in the queue. Now the consumer can *drain* all stored metric data from the queue for any kind of post-mortem analysis. After processing the end-message, the queue is deleted with a *release* RPC. To avoid resource leaks with non-conforming clients, e.g., a client that crashes before calling *release*, this queue uses a configurable timeout after which it is deleted with all its buffered messages.

This approach allows clients to retain no active connection in-between subscription and unsubscription — only remembering the name of the subscribed queue. Consequently, this allows a system under test to control the recording for an experiment with no ongoing perturbation. Particularly when measuring idle systems, it is critical to not even have an open connection that may cause activity from processing heartbeats or keep-alive packets.

RabbitMQ is well suited for temporarily storing measurement data without requiring additional services. It transparently pages out data from queued messages to disk and, if necessary, temporarily throttles sources if memory or disk consumption reaches a critical watermark. All of theses thresholds are configurable. While RabbitMQ provides the building blocks for temporal storage, MetricQ, in particular the management agent, controls the dynamic configuration, i.e., the creation and deletion of queues and their bindings.

Flat File-Based Persistent Storage

For efficient persistent storage, it is important to exploit the intrinsic properties of typical time series data and power measurement data in particular. The primary way to access data is the timestamp associated with a value. All timestamps for incoming samples of one metric are monotonous and approximately equidistant. New data is always appended to existing data, i.e., there are no changes to existing data. For some use cases, truncation of existing data may be desirable. However, in the scope of this work, it is not considered as a primary feature. In special scenarios, the monotonicity of timestamps may not be guaranteed, e.g., due to wireless data transmission from measurement devices [AC16]. Generally, this can be rectified by an appropriate protocol or buffering. Within RabbitMQ, the order of messages is always preserved[8]. In MetricQ, each metric corresponds to one independent univariate time series. This arises from the general design which uses metric identifiers

[8] The ordering guarantee is limited to one chain with single channels for both publisher and consumer, single exchange, and single queue. This condition is satisfied for the configuration used by MetricQ [Piv19, Broker Semantics].

as routing keys for messages used for both insertion and retrieval. Thus, there are always exactly two columns (timestamp and value), and never any operations covering multiple metrics.

In Section 2.3.1, I discussed the limitations of existing time series databases. The aforementioned properties (append-only, time-indexed, monotonous, approximately equidistant, and univariate) can be leveraged for a focused design and implementation. I propose a storage scheme that appends each incoming measurement point, consisting of an 8 B integer timestamp and a 8 B floating point value, directly to a file. This scheme implies that time monotonicity is enforced. Insertion with this scheme is very efficient as it entails only linear writes to one file for each metric. There is no overhead for managing an index tree or irregular writes as are typically necessary with generic database storage schemes [Pac07, Ch. 10]. It is also not necessary to manage an additional write ahead log.

Retrieving the position for a given timestamp uses binary search in logarithmic time — exploiting the ordering of timestamps within the flat file. This can be further improved by exploiting the regular metric readout rates. Similarly to insertion, reading the measurements maps to efficient linear reads — only the binary search for timestamps requires random reads.

Due to the mapping of one file per metric, multiple metrics cannot share timestamps. This is consistent with message protocol of MetricQ, which also uses separate messages for separate metrics. Since read accesses to a file occur at the 16 B boundaries of each timestamp-value-pair, it is not directly possible to use a more space-efficient encoding for timestamps or values. Nevertheless, it is possible to use a transparent underlying block-wise compression, e.g., on the file-system layer, as long as reading at a specific logical position can be performed in constant time.

4.2.5 Hierarchical Timeline Aggregation

While the flat file-based direct storage offers efficient insertion of data as well as logarithmic-time seeking to timestamps, the cost of retrieving measurement data still scales linearly with the number of values in the requested time interval. Thus, it is not easily feasible to support requests over longer time intervals even if not all raw values are required, e.g., for charts of the average power or an energy consumption value.

To that end, I propose the concept of Hierarchical Timeline Aggregation (HTA). This concept exploits the nature of measurement series. On top of the raw timestamp-value data, aggregated measurement results for fixed time intervals (*aggregation intervals*) are stored. The lowest aggregation *level* $k = 0$ uses a configured *minimum aggregation interval duration* i_0. Higher aggregation levels k use larger intervals with a duration of $i_k = i_0 * f^k$ and an *interval factor* $f \in \mathbb{N}, f \geq 2$. An upper limit $i_{k_{\max}}$ ensures that no impractical aggregation levels with very few intervals are used.

There are different ways to span the aggregation intervals: Either they have a fixed duration or they use a fixed number of values per interval (and thus a dynamic duration). Using a fixed number of values allows to have well-balanced intervals without requiring a known regular update rate of raw values $\overline{r}$ (in Sa/s). However, a fixed and aligned duration allows direct addressing of intervals for a given timestamp. Therefore, I use aligned intervals of fixed length, i.e., they span from $n * i_k$ to $(n+1) * i_k$ (half-open, $n \in \mathbb{N}$).

Aggregation Types

Conceptually, many different aggregation functions a are possible, as long as they are associative, i.e., they can be applied independently to partitions of the time series. In other words, the series of x_j (aggregate values with their respective time intervals) satisfies the following for every increasing sequence of j_k:

$$a(x_1, \ldots, x_m) = a\left(a\left(x_1, \ldots, x_{j_1}\right), a\left(x_{j_1+1}, \ldots, x_{j_2}\right), \ldots, a\left(x_{j_k+1}, \ldots, x_m\right)\right) \tag{4.1}$$

This condition ensures that applying the aggregation operation to a subset of measurement data for the intermediate intervals produces the same result as applying the operation to the full set of measurement data in one step. The current implementation stores the following data in each aggregation interval:

- *minimum* measured value during the interval,
- *maximum* measured value during the interval,
- *count* of all measurements corresponding to the interval,
- *sum* of all measured values corresponding to the interval,
- *active time* for which a measured value was known, typically this is equal to i_k,
- *integral* of the metric value over the active time within the interval,
- *timestamp* at the beginning of the interval (redundant, used for consistency checks).

Since the generic aggregation functions use aggregates as input and output, raw values first need to be converted to intervals in order to generate the input for aggregation level 0. For aggregation functions that only refer to values, i.e., *minimum*, *maximum*, *count*, and *sum*, this is trivial. However, aggregation functions that consider time, i.e., *active time*, *integral*, and *timestamp* require further consideration. They require a continuous signal rather than discrete samples. To reconstruct this signal, I consider that many power measurement devices provide average power readout values. This implies the assumption, that a particular measured value is valid from the previous timestamp to its associated timestamp. Consequently, the continuous representation of the time series is a left-continuous step function. Formally, this representation of a discrete time series is defined as zero-order hold everted (ZOHE) in [LVE16, Def. 7]. Under this model, continuous-time-based aggregations can be computed from raw values. In general, different temporal models can be used for different aggregations.
Based on the different aggregates, more statistics can be computed: While *count* and *sum* allow to compute the arithmetic mean over the values themselves, *integral* and *active time* provide the weighted mean that takes non-equidistant timestamps into consideration. A possible extension to this is to also include the cumulative sum of squares which then allows to compute the standard deviation.

Insertion into Aggregation Levels

While inserting data into an HTA store, the aggregation levels are computed on the fly. For each new measurement point, the lowest aggregation level is updated. Whenever a measurement point after the end time of the lowest current aggregation interval becomes known, this aggregation interval is completed. A completed aggregation interval is added to the current aggregation interval of the next higher level. With this approach, the worst-case cost for modifying aggregations upon insertion scales with the number of aggregation levels, which is limited by the maximum interval. Amortized over time, however, the average amount of modified aggregations per insertion is limited as follows. While the lowest aggregation level is always updated, the chance to update higher levels depend on the regular insertion rate $\overline{r}$ and the minimum aggregation interval i_0 as well as the interval factor f.

$$O_{\text{aggregation}} = 1 + \frac{1}{\overline{r} i_0} \cdot \sum_{k=0}^{k_{\max}} \frac{1}{f^k} \tag{4.2}$$

$$< 1 + \frac{1}{\overline{r} i_0} \cdot \frac{1}{1 - 1/f} \tag{4.3}$$

$$= 1 + \frac{f}{\overline{r} i_0 (f-1)} \tag{4.4}$$

$$\leq 1 + \frac{2}{\overline{r} i_0}, \text{ given that } f \geq 2 \tag{4.5}$$

Thus, the aggregation cost for insertion has an amortized constant time complexity and can be tuned by choosing $i_0 \gg 1/\overline{r}$. Further, since only one set of aggregated values per aggregation level needs to be tracked, the active data set to track all open aggregation intervals is small. Overall, the practical performance cost of adding an HTA on top of raw data storage is minimal.

Querying Aggregate Data

Queries for a timeline of measurement data can specify the minimal required resolution, e.g., the number of pixels for a plot. Based on this constraint, an appropriate aggregation level is chosen for the result. If the minimum aggregation interval does not provide sufficient resolution, the raw measurement data is used transparently instead. This principle of aggregate timeline retrieval is displayed in Figure 4.4. The choice of i_0 and f as well as the required resolution effectively limit the amount of data that needs to be processed for a timeline request with an arbitrary duration.

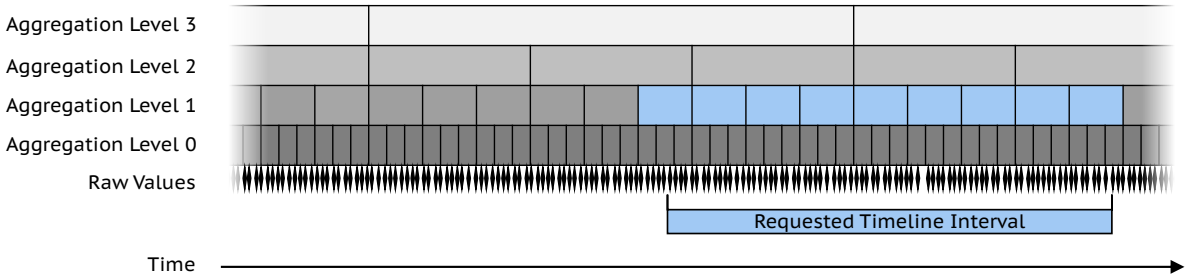


Figure 4.4: Retrieving an aggregate timeline for an arbitrary request interval and a given resolution. The utilized data is highlighted in blue ($f = 3$).

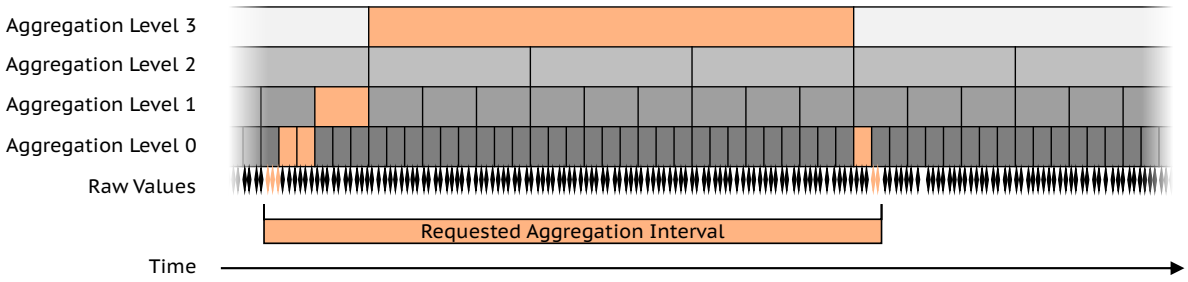


Figure 4.5: Computing an efficient aggregate over an arbitrary request interval. The utilized data is highlighted in orange ($f = 3$).

In addition to the timeline, the HTA concept allows efficient and exact computation of energy, average power, or any aggregated individual value for arbitrary request intervals. The approach to compute this is illustrated in Figure 4.5. First, the largest possible aggregation intervals that are fully within the requested interval are considered, i.e., *aggregation level 2* in the example. This leaves out gaps at the requested interval borders, which are added from lower aggregation levels. Eventually, the original raw data layer completes the aggregation up to the interval boundaries.

Storage Concept for Aggregation Levels

The data from aggregation intervals is stored in append-only binary files similar to original measurement data. All intervals are aligned and use and equidistant on one level. While writing, no interval is ever omitted: If there was no metric value for an interval, an appropriate one is written as soon as the first value after the missing interval becomes known. This storage scheme allows direct indexing based on the queried timestamp and the known timestamp of the first recorded measured value (*epoch*). Similarly to storage of raw measurement points, no explicit compression is used for storing aggregation data. However, compression can be added on the file-system layer.
In addition to enabling efficient retrieval of aggregated timelines and aggregate values for arbitrary request intervals, the HTA can speed up indexing in the raw measurement data file. The position of a timestamp-value pair within the original data can be computed from the number of values with smaller timestamps. This number can be bounded using the *count* within the aggregation intervals, similarly to computing a single aggregate value. Technically, this approach has the same time complexity as binary search ($O(log(N))$). Practically, however, it requires less file accesses and can exploit caching at higher aggregation intervals, which contain small data sets that are often used.

Comparison with Similar Concepts

The HTA concept offers specific advantages over existing time series database concepts and implementations (compare Section 2.3.1). The use of multiple levels containing summarized data is similar to RRDtool [Oet17] as well as the model for multiresolution time series data described in [LVE16]. However, the existing approaches use summarization as lossy compression with the goal of only requiring a fixed amount of storage. In contrast, the HTA is designed for efficient *and* accurate queries over arbitrary durations. Further, the hierarchical scheme ensures that most of the time, only one aggregation needs to be updated on an insert. Schemes with arbitrary aggregation levels need to

update all aggregation levels for each insertion. Moreover, RRDtool is designed for low-granularity data, which is also indicated by one-second-granularity of timestamps. BTrDB [AC16] places aggregates in a tree-structure for allowing efficient queries. The data in BTrDB is stored in a versioned copy-on-write tree, which allows out-of-order insertion. While this approach provides similar benefits as HTA, it does use in a more complex storage format.
The downsampling / retention functionality of popular time series databases could be used to create aggregation hierarchies similar to HTA. In InfluxDB [Inf19a, Downsampling and data retention], aggregations / downsampling can be manually defined using *retention policies*. Similarly, TimescaleDB [Tim19] uses special precomputed views named *continuous aggregates* to speed up aggregate queries. However, this approach requires a manual configuration of multiple levels. Moreover, queries do not transparently select an appropriate retention level. Instead, queries need to address a specified retention policy or aggregate view. On the client side, this selection is not practical, as the specific configuration of queried metrics is not known. A transparent implementation would require a sophisticated middleware for query rewriting. For InfluxDB, there have been discussions[9] about improving the access to databases with multiple retention policies, but there is no consistent concept or implementation. In contrast to the existing approaches for approximate query processing that use sampling or signal processing [Cha+01; BCD03; Cha+11; Per+15], the performance benefit from the HTA does not sacrifice correctness but only granularity for aggregate timelines.
The simple storage concept of using flat append-only files exposes beneficial I/O patterns, similar to write ahead logs. Due to the inherent locality, the underlying I/O layers can transparently improve performance through coalescing, caching, and prefetching without the need for explicit consideration in the implementation. While compression is also possible, a transparent generic compression does not offer the same level of data reduction as time series data specific encoding (compare [Pel+15]).

Implementation as a Persistent Data Storage Agent in MetricQ

I implemented the concepts for flat file-based storage and Hierarchical Timeline Aggregation in a persistent data storage agent for MetricQ. The actual HTA functionality is encapsulated in a C++ library, which is used by the storage agent implementation in addition to the C++ MetricQ API and two asynchronous event loops. One of the event loops handles the AMQP connections and processes messages to the level of protobuf structures. The AMQP event loop uses a single thread for all C++ agents because access to a stream needs to be serialized in any case. The protobuf structures, either for inserting new values or retrieving historic requests, are then dispatched to the other event loop, which uses multiple worker threads for processing. Race conditions are avoided by using per-metric strands as execution policies. That means, that requests concerning a specific metric are never executed simultaneously. The HTA I/O operations themselves are blocking because Linux does not offer non-blocking file I/O[10]. Dispatching the requests to a separate event loop allows the connection handling to still process messages (especially heartbeats), while blocking I/O operations are running on the other event loop. Using multiple data worker threads allows parallel processing on a per-metric level within the persistent data storage implementation.

[9] `https://github.com/influxdata/influxdb/issues/7198`
[10] While Linux and POSIX offer some support for asynchronous I/O, it is not possible to open a file asynchronously, which would also be necessary.

4.3 Performance Evaluation

As discussed in Section 4.1, sophisticated energy measurements present demanding requirements to the performance of a measurement data processing. In this section, I evaluate the performance of MetricQ and provide a comparison with a state-of-the-art time series database system. I used several benchmarks to determine whether the MetricQ implementation meets the defined requirements. The benchmark results also reveal the performance boundaries of MetricQ with a particular focus on metric data rate for live processing, application power tracing, and persistent storage.

I used a synthetic data source for the benchmarking to allow controllable data rates and utilize a reproducible environment. A recording of actual measured values from the *artemis* system provides power values for the synthetic data source to allow a realistic value compression for the comparative time series database. The synthetic data source generates measurement points in batches (chunks) at regular time intervals whereas the timestamps within each chunk are interpolated on a linear scale. This generation scheme emulates a timestamping similar to a measurement with a device that provides a block of measurement results at a constant rate without individual timestamps, e.g., National Instruments DAQ or ZES LMG in scope mode (see Chapter 3).

A set of benchmarks covers measurement data collection with respect to different use cases: For determining the maximum ingestion rate, all metric data points were *discarded*. In this configuration, measurement data was published to the message broker, but no consumer was subscribed. Therefore, there were no queues to which the respective messages were delivered. This benchmark primarily serves as a baseline and to better identify the cause of performance bottlenecks. However, it can be relevant in practice to collect more measurement data than can be processed. Keeping measurement data sources running continuously, but only recording or analyzing them selectively, simplifies control and management of the measurement infrastructure.

The end-to-end performance was determined with a *dummy* consumer. This consumer subscribes to a set of metrics and unpacks all messages to the point of individual timestamps and values. At this point, the dummy consumer does not process the data any further.

In the third test configuration (*hta*), persistent data collection was included using the HTA implementation. The aggregation interval configuration used $i_0 = 30/\overline{r}$, $k_{\max} = 7$, and $f = 10$.

In all of the three configurations, it is not trivial to determine the maximum end-to-end throughput. Simply publishing as many data points as possible is neither a viable nor representative continuous mode of operation. There are several buffers within components of the system that allow short-time bursts while maintaining asynchronous operation. Within the C++ source library, write operations are buffered as asynchronous tasks in the asio layer. If the messages cannot be sent fast enough, the buffer will accumulate pending tasks and eventually run out of memory. Moreover, RabbitMQ uses the *credit flow* mechanism[11] to throttle publishers to prevent resources exhaustion. Therefore, the maximum throughput is determined by gradually increasing the data rates or cardinality of the synthetic sources until the operation is no longer stable. To achieve the maximum possible message throughput, I disabled credit flow throttling. On the subscriber side, a prefetch count[12] of 400 was configured for the data channel. This limited the number of messages that are delivered to a consumer before receiving acknowledgments. The limit must be large enough to effectively hide the latency

[11] `https://www.rabbitmq.com/blog/2015/10/06/new-credit-flow-settings-on-rabbitmq-3-5-5/`
[12] [Piv19, Consumer Acknowledgements and Publisher Confirms], `https://www.rabbitmq.com/confirms.html`

by overlapping transmission and processing, as long as there are enough messages in a queue. An unlimited prefetch count, however, could overflow the receive buffer. The HTA implementation also dispatches incoming data chunks to a separate asynchronous asio event loop (see also Section 4.2.3). Nevertheless, messages are only acknowledged after processing is complete so that there are never more than 400 messages buffered in the two event loops. If the consumers cannot process data at the rate of producers, the messages are buffered in queues within RabbitMQ. While this mechanism is essential for temporary situations in practice, a permanent operation at such an imbalance would not be sustainable.

I designed the benchmarks to measure end-to-end times, i.e., from right before the first source sends the first value to after the last consumer receives its last value. To that end, the synthetic benchmark source sends a special *end message* that causes the benchmark consumers (*dummy* and *hta*) to shut down and record the current timestamps. For the *discard* use case, only the times from the benchmark sources were used. All agents print their respective timestamps to the console as a basis to compute the total end-to-end performance.

In addition, I tested the temporary storage used for application power tracing using a simple benchmark *drain*. It subscribes to incoming metrics for a given duration and then measures the time it takes to retrieve all measurements from the queue.

The above mentioned benchmarks were all implemented with the high-performance MetricQ C++ library. Moreover, I used Python and MPI to implements a controller that orchestrates the distributed services needed for the benchmarks. This controller spawns the broker, management, source, and consumer processes on the different involved nodes. It also collects the output from all of the involved processes and computes the total end-to-end time for the parallel execution.

Finally, I determined the end-to-end HTA query latency with another benchmark. This benchmark uses the MetricQ Python module to generate and measure queries to the HTA implementation.

4.3.1 Benchmark Hardware Specifications

All of the following benchmarks were performed on a special set of NVMe nodes from the *taurus* HPC system at TU Dresden[13]. Each node is equipped with two Intel Xeon E5-2620 v4 processors with a nominal frequency of 2.1 GHz and up to 3.0 GHz with Turbo mode as well as 64 GB of main memory. All nodes are interconnected with two EDR InfiniBand connections using Mellanox MCX555A-ECAT adapters providing dual port 100 Gbit connectivity. The connections for data payloads of different clients alternated between the two InfiniBand networks. Scenarios with only single clients of each type use only one of the InfiniBand ports.

For high-performance persistent storage, each node contains eight Intel DC P4610 SSDs at a formatted capacity of 2.9 TB. In normal operation, the nodes are not accessed directly but export a parallel BeeGFS file system. For the following benchmarks, however, each SSD was separately formatted with a local ext4 file system. If not noted otherwise, each HTA storage instance used a single SSD exclusively. AMQP/HTTP connections without TLS were used for all benchmarks. Experiments were performed with the following software versions: GCC 8.2.0, RabbitMQ 3.7.13, Erlang 21.3, Python 3.6.6.

[13] `https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/NvmeStorage`

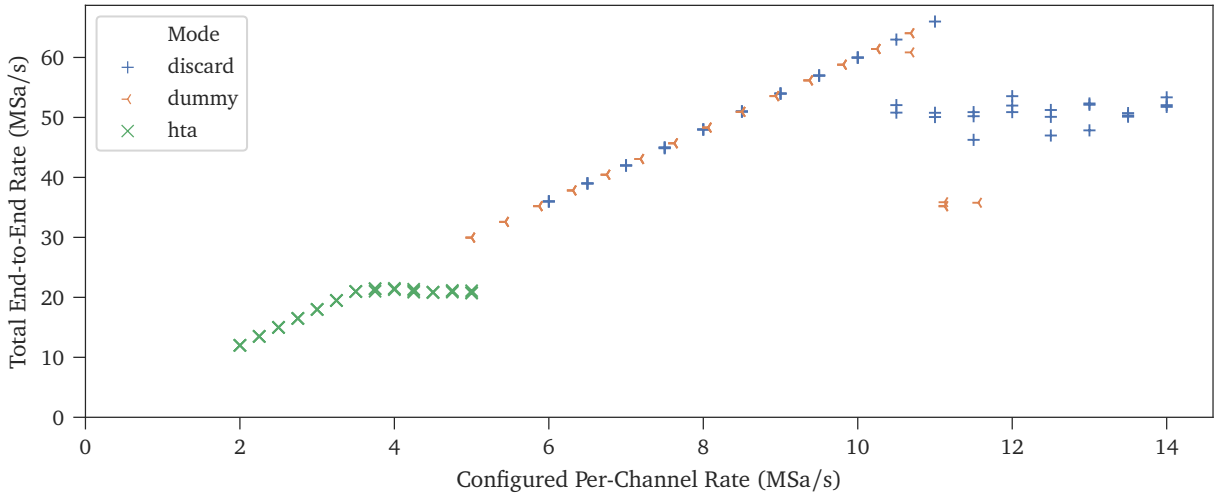


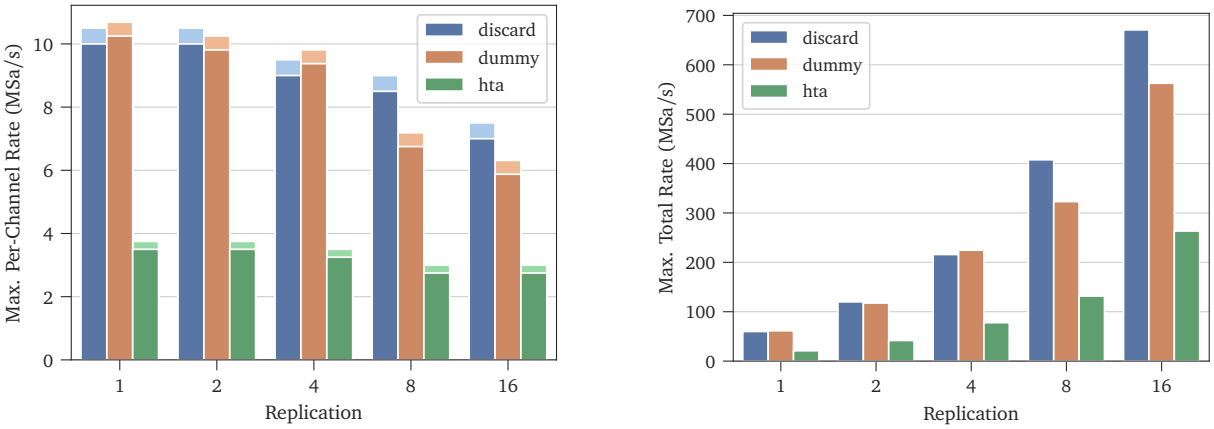
Figure 4.6: The total effective end-to-end metric throughput for a single data source and consumer at different requested per-channel metric rates.

4.3.2 Throughput in Symmetric Configuration with Replication

One of the main motivations for developing MetricQ was to enable processing metrics at the data rates of high-resolution power measurements. The first benchmark used a single data source with six measurement channels and was executed for each of the different consumer modes *discard*, *dummy*, and *hta*. For all modes, the metric data rate rate per channel was increased until performance degradation caused a timeout. In order to limit the overall experiment time, I narrowed down the tested range of data rates individually for each mode by coarse grained benchmarks beforehand. All experiments were set to produce 40 s of measurement data and were repeated three times.

The result from each experiment is the total end-to-end metric rate, i.e., the total number of values transmitted for all metrics, divided by the time between the first send and last consume (last send for *discard*). Given the power measurement use case, I use the unit Sa/s for reporting metric data rates. Figure 4.6 shows that the *discard* and *dummy* scenarios exhibit similar patterns with no performance degradation up to ≈ 10 MSa/s per-channel (≈ 60 MSa/s total). For higher per-channel rates, the end-to-end performance decreases while variation increases. The bottleneck in both cases is in the source implementation: Once the requested metric rate exceeds the possible send rate, the source generates more asynchronous tasks than can be processed. The contention on the event loop decreases performance even further and the end-to-end time exceeds the intended 40 s since the source publishes the locally buffered messages. Such an over-utilized scenario would eventually lead to memory exhaustion. The *dummy* workload is affected by over-utilization even more than *discard*. Observation shows that a over-utilized *dummy* source publishes messages in bursts rather than at a constant rate, possibly due to flow-control originating from the consumer queue.

The performance for the *hta* benchmark saturates at 21.0 MSa/s with no variations. An over-utilized HTA agent /buffering in the RabbitMQ queue and an increased time to complete processing all measured values, thus saturating the end-to-end rate. Thanks to the limited prefetch-count, the HTA agent is not flooded by messages and continues to process at a constant rate. RabbitMQ handles the buffering gracefully. While this benchmark is not affected, in practice the message broker performance



(a) Maximum per-channel sampling rate configurations without performance degradation. The lowest setting with performance degradation is shown in lighter colors.

(b) Total (6 channels × replication) end-to-end metric rates for the highest tested configuration without performance degradation.

Figure 4.7: Throughput characteristics for different replication levels and consumer modes.

may degrade when paging out messages to disk. If more data is posted than can be processed, the disk space would be exhausted if more data is posted than processed.
The results show, that MetricQ fulfills the stated ingestion performance requirement (a) of 1 MSa/s per channel for six channels by a factor of 10. Requirement (d), stating a total persistent storage insertion performance of 1 MSa/s, is surpassed by a factor of 21 (compare Section 4.1).

Symmetric Replication

All agents within the system can be parallelized on a per-metric granularity. Therefore, I further conducted a symmetric replication benchmark case with p sources and p consumers for $p \in \{1, 2, 4, 8, 16\}$. Up to eight sources or consumers were executed on one node. The HTA agents used three worker threads such that each worker thread was running on one hardware thread. The sources and consumers were linked pairwise, i.e., one consumer exclusively consumed data from one source. For presenting the results, I consider the maximum rate without performance degradation, i.e., the maximum configured per-channel sampling rate at which none of the three experiment repetitions exhibited an end-to-end runtime that is more than 2 % longer than the configured duration. This indirect nature of determining the maximum sustainable throughput introduces some uncertainty due to the granularity of tested metric rates. The true maximum rate lies within the determined rate and the next higher configuration, i.e., the lowest per-channel rate at which performance degradation was observed. The confidence in this interval can be increased by using more repetitions and the width of this interval can be decreased by a more fine-grained configuration of tested rates. However, the current results confirm that MetricQ exceeds the stated requirements with a significant headroom.
Figure 4.7a shows the maximum per-channel metric data rate and Figure 4.7b displays the corresponding total end-to-end rate for the different replication levels and consumer modes. For up to four parallel source/consumer pairs, the total throughput scales almost linearly. While *discard* and *dummy* modes perform similarly, both drop slightly at a replication level of four. This effect could be caused by reduced Turbo frequencies at increased core utilization, but was not investigated further. At higher

replication levels, the *dummy* throughput was lower than in *discard* mode, indicating that the bottleneck shifted from the source to the message broker or network. The highest total end-to-end rates reached at $p = 16$ are 671 MSa/s and 563 MSa/s for *discard* and *dummy* configurations, respectively. The maximum per-channel rate for the *hta* case dropped only from 3.75 MSa/s to 3.00 MSa/s across all replication levels. The total metric rate scaled up to 263 MSa/s when using sixteen HTA instances on the same number of SSDs across two nodes.

Underlying Limitations of Achievable Performance

The experimental conditions with high-performance networks and SSDs were deliberately chosen to reveal non-obvious limitations. Practical experience shows that even an environment with a regular gigabit network and a NFS partition supports demanding requirements (see Section 4.3.7).
To put the results from the high-performance benchmark environment in context, I estimated the underlying I/O bandwidth and data rates: At the given per-channel rates, ignoring minimal relative overhead at the high chunk sizes, each timestamp-value pair requires 11 B in protobuf. Thus the 563 MSa/s passing through the message broker at $p = 16$, *dummy* correspond to 49.5 Gbit/s of the theoretically available 200 Gbit/s. However, higher bandwidth is typically achieved using low-level RDMA access rather than generic TCP sockets (see [Mel18]). In the HTA implementation, each timestamp-value pair uses 16 B and aggregates 56 B each. For the given configuration, each measurement sample requires 18.07 B on average — ignoring the diminishing cost of headers. Consequently, the achieved 21.0 MSa/s with one HTA client / SSD correspond to 381 MiB/s. Intel specifies up to 3200 MB/s for sequential writes for the used SSDs [Int18b].

4.3.3 Throughput with Many Data Sources and Single Consumers

To understand the performance ceiling for an instrumented HPC system, the metric cardinality was increased for the following benchmark. I used a total of 45 *taurus* NVMe nodes for this benchmark. 42 nodes were used for sources, with 32 sources each (one per hardware thread). This resulted in 1344 distinct source processes with a separate RPC and data connections to the message broker. Each source published six metrics at an increasing rate. The remaining three nodes executed the message broker, 32 management agent processes, and the consumer (*dummy* or *hta*). This test demonstrates the use of parallel management agents to speed up the connection phase of many simultaneous clients (each type on one dedicated node).
An HPC use case does not necessarily imply a large number of distinct source agents. The HDEEM measurement data collection, for example, uses a single process running on an administrative node. This data source process collects all of the data from the 1456 BMCs to forward it as messages. A future solution, however, could possibly run an agent directly on each BMC (compare Section 2.2.2, DiG).
To facilitate the large amount of active clients and metric data files, the number of file handles for the RabbitMQ and *hta* processes was increased to 120 000. Increasing the number of sources further, by oversubscribing the nodes, resulted in sporadic connection issues, which manifested as timeouts during the initial AMQP connect. It has been documented that RabbitMQ can successfully handle between 20 000 to above 500 000 connections, but that takes “effort and experimentation with

multiple RabbitMQ and OS settings"[14]. Considering that the system under test is part of a production HPC system, no such attempt at tuning for even more clients was made.
For the given configuration of sources, I executed benchmarks with a single *dummy* or *hta* instance, as well as the *discard* scenario with no consumers. The *hta* was configured with 15 worker threads to ensure a dedicated core was available for the message handling and each HTA worker. Further, the *hta* was configured to distribute metrics across all eight SSD partitions. Experiment runtime was increased to 300 s to mitigate the influence of the increased connection setup time — the result should represent sustained operation.
After a preliminary experiment to determine the approximate maximum data rate, the source per-metric rate was increased with at least five repetitions for each rate setting. I configured the chunk size such that there is always one chunk per second for each metric in all configurations. The highest rate setting that allowed stable operation without degraded end-to-end performance is considered to be the maximum metric data rate:

- *discard*: 75 kSa/s per channel, 601 MSa/s total end-to-end. This value is slightly lower than the achieved 671 MSa/s for *discard* with 16 sources and sinks (see Section 4.3.2), which can be explained by the reduced chunk size and the additional effort required to handle a large amount of clients.
- *dummy*: 8.6 kSa/s per channel, 68.9 MSa/s total end-to-end. Here, the single consumer is clearly the bottleneck and throughput is only slightly higher than in the symmetric *dummy* configuration, where the single source is the bottleneck (61.4 MSa/s, see Section 4.3.2).
- *hta*: 3.4 kSa/s per channel, 27.3 MSa/s total end-to-end. This performance is higher than the previous test with one source and one *hta* agent (21.0 MSa/s, see Section 4.3.2) indicating that this scenario benefits from using multiple SSD partitions and more HTA worker threads. The difference, however, is not proportional to the additional resources suggesting that the bottleneck is not HTA write performance but rather message ingestion or other shared resources.

The performance requirement (b) from Section 4.1 was surpassed in all configurations, even with a single consumer. This benchmark used 1.3 times the required sources and achieves 75 × of the required per-metric data rate. If a consumer bottleneck was introduced, the required metric rate is exceeded by a factor of 8 (factor of 10 when factoring in the number of sources).
The above scenario still exploited chunking within each metric. To understand the limitations for even larger cardinalities, where chunking is not applicable, the per-metric rate was fixed to 1 Sa/s at a chunk size of 1 and the number of channels was increased instead. This experiment was only performed for the *discard* mode. Up to 300 metrics per source (403 200 metrics total) showed a consistent performance at a total rate of 0.40 MSa/s. Higher configurations with 350 and 400 metrics per source ran stable, but with flow-controlled source channels, at 0.43 MSa/s. Even higher cardinalities resulted in unstable operation. This shows that while MetricQ benefits from per-metric chunking to enable very high total metric data rates, it also allows a high cardinality at lower update rates. Consequently, the performance requirement (c) (25 000 metrics at 1 Sa/s) was exceeded by a factor of 16.

[14] [Piv19, Networking], `https://www.rabbitmq.com/networking.html`, `https://groups.google.com/forum/#!topic/rabbitmq-users/LXqvS1MVA20`

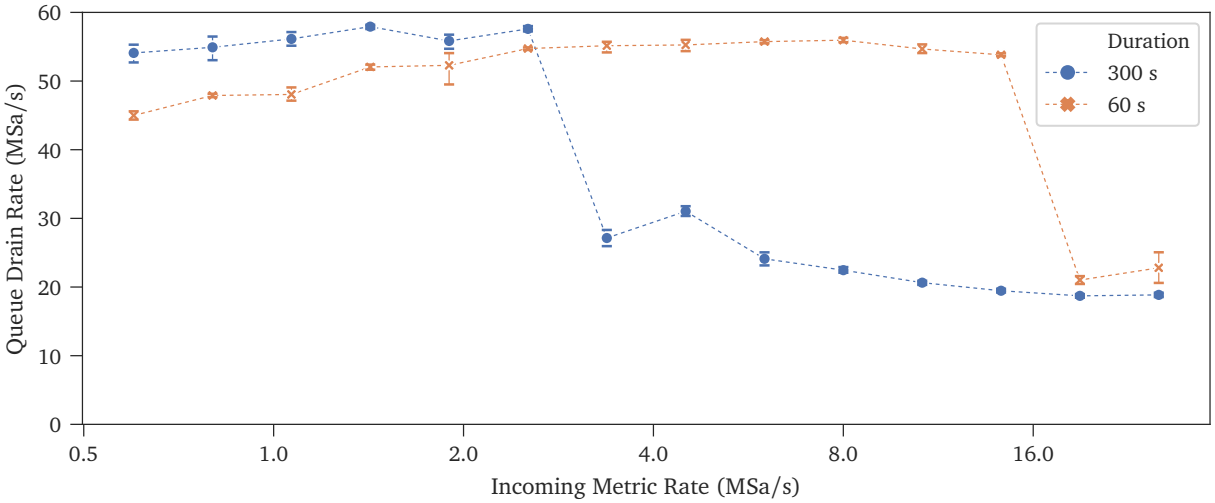


Figure 4.8: *Drain* performance for retrieving temporarily stored metric data at a given incoming metric rate and duration of recording.

4.3.4 Temporary Storage in Message Queues

The *drain* benchmark evaluated the use of RabbitMQ for temporarily storing metric data in queues. For this benchmark, I used two nodes — the message broker and consumer ran on different nodes, communicating over one InfiniBand link. The disk location for RabbitMQ was placed on one of the SSDs. Over a duration of 60 s or 300 s, the *drain* subscribed to a set of six metrics generated by one source at a given target rate. I increased the per-channel metric generation rate, beginning with 0.1 MSa/s with five repetitions for each configuration.

The highest achieved total metric rate to store messages in the queue was 25 MSa/s (4.2 MSa/s per channel). At higher rates, i.e., 34 MSa/s, the queue, and therefore the source, is placed into flow state, throttling incoming messages. Figure 4.8 shows the rate at which the stored data was retrieved from the queue. The result reveals two distinct performance levels at ≈ 55 MSa/s and ≈ 20 MSa/s. The performance levels correspond to cases where RabbitMQ retains queued messages in memory, and cases where it pages messages out to disk. Since this effect correlates with the total size of messages in a queue, the experiments with a shorter duration of recording remained at high performance levels for larger incoming metric rates. Further, performance is slightly lower for smaller queues, i.e., short duration and low incoming metric rate, influenced by a fixed overhead. In this benchmark, the performance requirement (e) stated in Section 4.1 was exceeded by a factor of 11.

4.3.5 Persistent Metric Time Series Request Performance

An important goal of MetricQ is the interactive visualization of large volumes of persistently stored data. To evaluate the infrastructure and the HTA implementation in this regard, a data set with six metrics recorded at 1 kSa/s for one year was generated synthetically. This amounts to a total of 189 billion values or 3.1 TiB as stored in the HTA. For this benchmark, each of the six metrics were stored on a dedicated SSD partition and the HTA was configured with six worker threads. The message broker and management agent were running on one node, while the HTA implementation

and benchmarking agent were running on another node. Since all persistent request messages and answers pass through the message broker, each message passes the InfiniBand network twice.
In order to facilitate interactive charts, the benchmark focuses on the time to answer persistent data requests (query response latency). I implemented a benchmark driver based on the MetricQ Python module to generate those requests and to measure the end-to-end latency. In addition, the HTA implementation was instrumented to expose certain internal latencies:

1. *End-to-end latency* was measured in the client between issuing the first query and receiving all responses. It therefore includes the round-trip message transfer through the message broker as well as client-side message processing in Python.

2. *Server request latency* denotes the time elapsed from receiving to sending the respective protobuf messages, not including AMQP parsing.

3. *HTA worker latency* denotes the time elapsed for the HTA operation within the worker thread.

All query configurations were repeated 20 times and each of the repetitions was performed on a randomly selected time interval to avoid overly specific caching. In normal interactive visualization, some requests may overlap so that there may be caching, but I focused on the worst-case. The latency was measured for the full factorial combinations of each of the following factors:

1. The queries were either requests for a *timeline* or a single minimum, maximum, and average *aggregate* for the entire duration. Requested timelines were configured to a granularity of least 1000 aggregate intervals and include minimum, maximum, and average for each interval. A single requested aggregate value represents an analysis use case, e.g., job energy consumption.

2. Either one randomly selected metric, or all six stored metrics were requested. Since the infrastructure only allows one metric per request, the latter case issues six requests in parallel.

3. The duration of the queried time interval was increased from one second to four months on a logarithmic scale.

The resulting latency of all configurations is shown in Figure 4.9. All latencies were below 30 ms, with only minimal variations for each configuration. This is well below the response time of 100 ms that is perceived as instantaneous [Nie94] and gives headroom, e.g., for rendering or including additional metrics. The performance requirement (f) of a maximum latency of 1 s stated in Section 4.1 was exceeded by a factor of 33.
For *timeline* queries, a pattern shows the impact of queried time interval: Up to 30 s, the required granularity of 1000 data points implies the use of non-aggregated measured values — the result set size increases with the query duration, as does the latency. Larger query durations allow the use of aggregation levels in the HTA: After every factor 10 of the queried duration, a higher aggregation level is chosen, reducing the data size that needs to be read and transmitted again. This pattern shows how the concept of the HTA provides an effective bound for the query latency. The remaining peaks could be further limited by reducing the minimum interval and the interval factor. However, this would come at the cost of increased storage size overhead, which is 13 % for the given configuration.

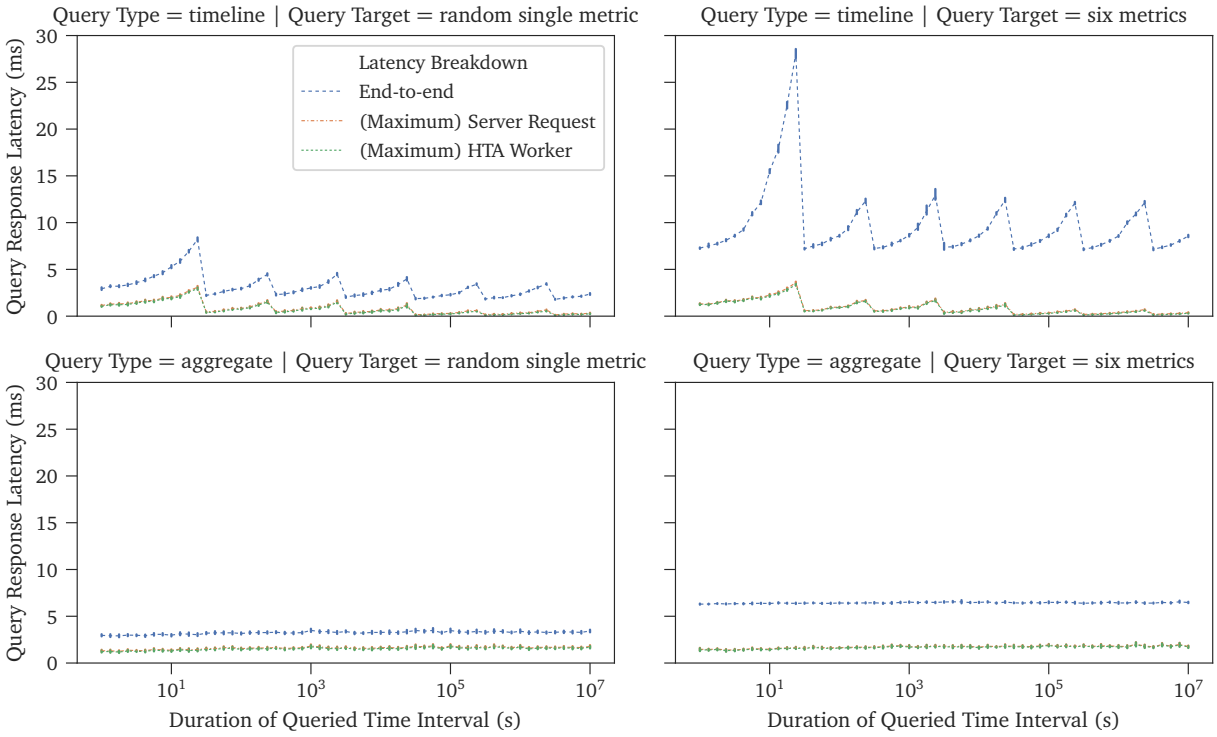


Figure 4.9: Query latencies for different query types, targets, and query time intervals. Error bars show the 95 % confidence interval around the mean for each measurement configuration.

The results also show that most of the latency resides in message transfer and client-side processing, while the HTA operations require less time. On the server side, there is no noticeable difference between the total *server request* time and *HTA worker* time. This means that the worker threads start processing incoming queries with no negligible delay and the result is immediately dispatched. For queries regarding very long time intervals, the HTA time further decreases. This effect can be caused by caching of the highest aggregation levels that contain relatively few entries overall.
All requests for aggregate were answered in less than 8 ms, independently of the queried time interval. Including all six metrics in the query does increase the end-to-end latency by less than a factor of six. This shows that the parallel requests were in effect, but do not scale perfectly. Note that the internal *server request* and *HTA worker* latencies in Figure 4.9 of multiple concurrent queries refer to the maximum of each individual one. It is possible that the increased latency for multiple metric requests is in part due to the processing of increased data volume within the Python benchmark client.

4.3.6 Performance Comparison with Contemporary Time Series Storage Solutions

To put the MetricQ performance in context, I performed similar benchmarks with the contemporary time series database InfluxDB (see Section 2.3.1). I used InfluxDB 1.7.4-1 in the default configuration except for compaction[15] as suggested by official benchmark documentation[16]. In the basic configuration, InfluxDB uses one path for storage, located on a single SSD partition. The InfluxDB daemon was running on one *taurus* NVMe node, while all benchmark drivers were running on a second node. Communication between clients and the database used one InfiniBand link and HTTP.

[15] `compact-full-write-cold-duration="0"`
[16] `https://github.com/influxdata/influxdb-comparisons`

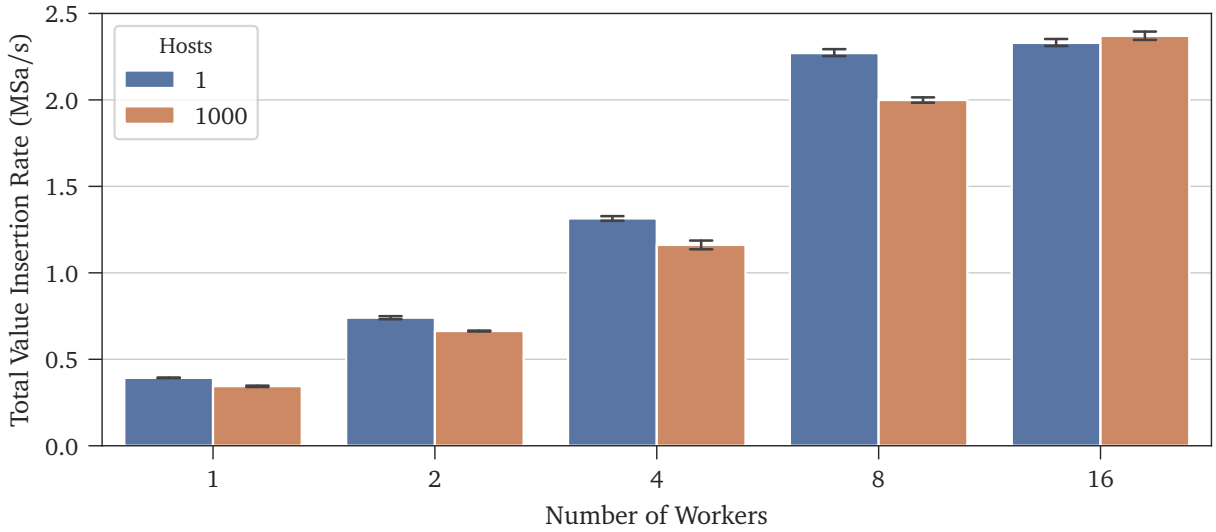


Figure 4.10: Value insertion rates for InfluxDB using bulk load workers. Rates are given for a configuration with 1 host, 6 channels, 0.4 MSa/s and 1000 hosts, 6 channels, each 0.4 kSa/s.

Insertion Performance of InfluxDB

To determine the maximum insertion rates of InfluxDB, I extended the benchmarking code provided by the vendor[16] (see also [HKK17]). Contrary to the MetricQ benchmarks, the InfluxDB benchmark is not a data source with a real-time sampling rate, but generates data as fast as can be inserted. The rate of artificial timestamps for this data has diminishing impact on the performance — nevertheless, I determined and used the approximate sampling rate that InfluxDB can handle in real time. While the original implementation focuses on DevOps and IoT use cases, I implemented a simulated power monitoring device similar to the previous benchmarks. This simulated device has six channels at a rate of 0.4 MSa/s per channel. Data from all six channels (metrics) is inserted in measurement points with shared timestamps, thus exploiting the multivariate capabilities of InfluxDB. For a measurement duration of 40 s, 96 million metric values or 16 million measurement points are generated. A second configuration with increased cardinality uses 1000 simulated hosts, each with six metrics reporting at 0.4 kSa/s per channel. The benchmark driver used up to sixteen worker threads on one node and bulk inserts with the default size of 5000 points (30 000 values). Each configuration was repeated four times. Contrary to the benchmarks in Section 4.3.2 that use replication, the InfluxDB workers all process the same metric stream from a single simulated measurement device.

Figure 4.10 shows that InfluxDB reached a maximum insertion rate of ≈ 2.3 MSa/s, which is significantly lower than the 21.0 MSa/s of the *hta* benchmark without replication[17] described in Section 4.3.2. The increased cardinality reduces performance slightly for smaller worker counts, but performance with 16 workers is the same for both 1 and 1000 hosts. These numbers are consistent with the results from [HKK17], which reports a write throughput of 1.4 MSa/s for 4 writers, even though the use case and hardware is different. It is worth noting that the InfluxDB benchmark requires eight or more workers to saturate a single `influxd` instance. While the insertion rate of the HTA implementation is ≈ 9× higher than InfluxDB, a single-threaded data source in MetricQ can publish data at rates even three times higher than the HTA storage can consume (compare Section 4.3.2).

[17] The referred configuration without replication also utilized only a single SSD.

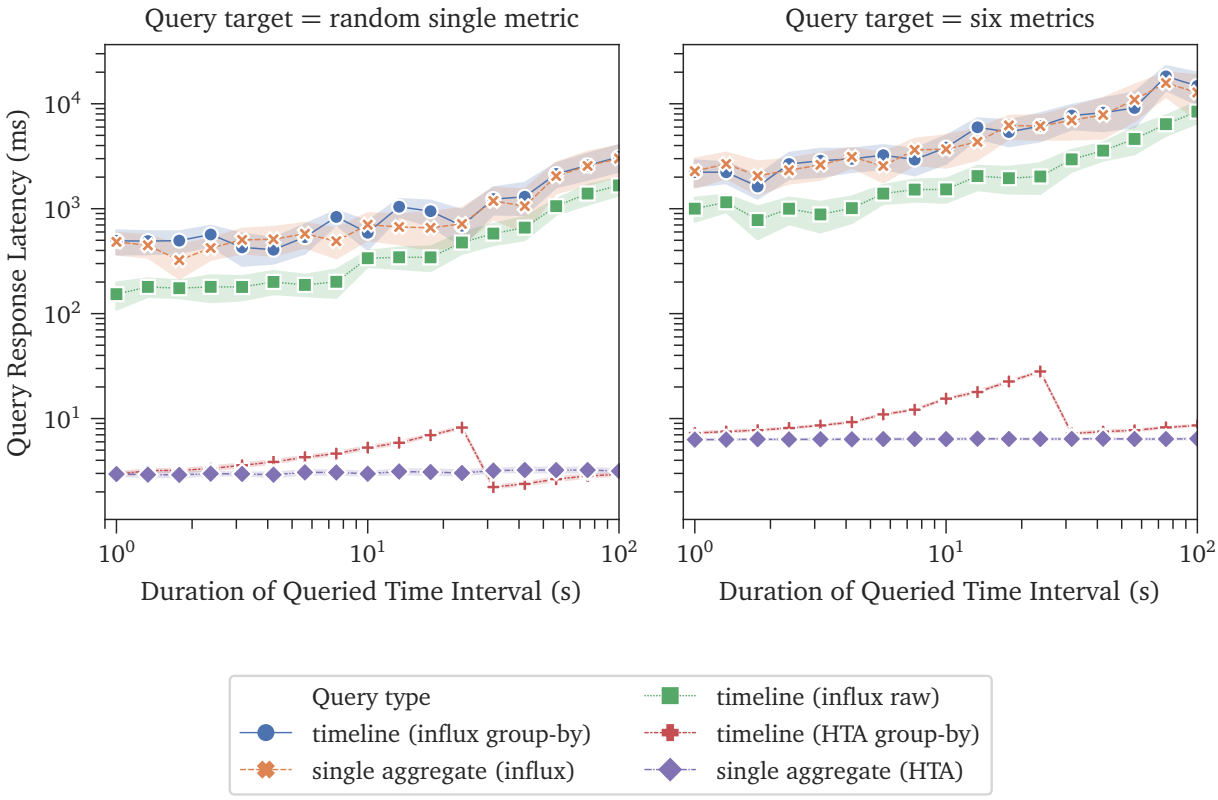


Figure 4.11: End-to-end query response latencies for InfluxDB and the HTA implementation. Both axis use a logarithmic scale. All lines shows the 95 % confidence interval around the mean as colored bands.

Query Performance of InfluxDB

To compare query latency, I used a similar setup as described in Section 4.3.5 with a benchmark client built on the InfluxDB Python module. One advantage of InfluxDB is the compression of data. The data set with 189 billion values required only 0.5 TiB disk space (2.9 B/Sa) as opposed to 3.1 TiB (18.1 B/Sa) in the HTA storage. This high compression may be favored by the limited number of unique values (4096) and perfectly equidistant and aligned timestamps of in the simulated measurement. However, the six metrics are spread across 2794 files and directories, which required increasing the number of allowed file handles for the database generation, even with only six data series. In order to limit the overall experiment time, the queried time intervals are only measured up to 100 s. For benchmarking InfluxDB, three query types were used:

1. A *group-by timeline* similar to the HTA timeline requests, which transparently return aggregated intervals. The query requests `min`, `max`, and `mean` of each metric and groups the result such that there are exactly 1000 aggregates (as opposed to *at least* 1000 aggregates with HTA).

2. A *single aggregate* requesting `min`, `max`, and `mean` of each metric over the entire queried interval. This query is identical to the single aggregate for HTA requests.

3. A *raw timeline* that simply gets all measurement points for the requested interval.

For comparison, the HTA results presented in Figure 4.9 are used. Figure 4.11 shows the end-to-end query response latencies for different query types, query targets (random single metric and all six metrics), and queried time intervals. The observed query latencies of InfluxDB range from 10 ms to 40 s, with large variation across the repetition of each configuration. In the fastest configuration, for example, latencies range from 12.9 ms to 368 ms with a mean of 153 ms and a standard deviation of 103 ms. Queries to InfluxDB that use any kind of aggregation are consistently and significantly slower than requesting the full timeline. Requesting measurement data from all metrics is significantly slower than querying a random single metric showing no clear benefit from the grouping of multiple metrics in measurement points with shared timestamps.

Dataheap Metric Ingestion Rate

The published results in [KHN12] allow a basic approximate comparison with *Dataheap* (see also Section 2.3.2), which also offers a scalable implementation supporting "millions of performance counter updates each second". The authors used dual-socket 6-core Intel Westmere-EP servers with 10 Gbit Ethernet. Their performance evaluation focuses on publishing metric values, similar to the MetricQ *discard* benchmarks. Dataheap uses bulk transfers that contain values of multiple metrics at one timestamp, similar to measurement points in InfluxDB. One single-threaded Dataheap source can publish 0.5 MSa/s — compared to 60 MSa/s with MetricQ. The centralized Dataheap system is saturated at 6 MSa/s with up to 120 client threads — two orders of magnitude less than 671 MSa/s in the corresponding replicated MetricQ *discard* benchmark. Even accounting for the hardware differences, there is a significant performance benefit of MetricQ. Persistent insertion performance of Dataheap depends on the MySQL backend and has not been tested exhaustively.

BTrDB Insertion Performance

With its aggregated tree structure and the use case of micro-synchrophasors data, BTrDB [AC16] has similarities to the HTA concept. For its Go implementation, an insertion rate of 53.35 MSa/s was reported when using four primary nodes and 16 Ceph Object Store Daemons [AC16, Sec. 7, Tab. 1]. Insertion for the BTrDB benchmark uses chunks containing 10 000 records. The comparison to MetricQ *hta* is not straightforward as functionality is distributed differently. Based on the used hardware, it is closest to the configuration with 16 HTA instances using 16 SSDs on two nodes that reached 263 MSa/s. There are too many differences in the hardware configuration to make a accurate comparison, it can only be said that the performance is within the same order of magnitude. The published BTrDB query latencies ([AC16, Sec. 6.3, Fig. 6]) show a similar pattern as the HTA results from Figure 4.9. *Statistical queries* in BTrDB are similar to timeline queries to the HTA. The published performance of queries to a large real-world data set ranges from 100 ms to 250 ms. While this is slower and exhibits much more variance than the HTA benchmarks at <30 ms, the data set is larger, and the storage back-end more layered. Thanks to the tree-based aggregation, BTrDB also answers statistical queries over years worth of data efficiently.

4.3.7 Practical Usage of MetricQ for High-Resolution Measurements and HPC Monitoring

MetricQ is currently being used for a variety of measurement data at TU Dresden. The main focus is on the energy measurement test systems (see Chapter 3), the HPC power measurements of *taurus*, and data center monitoring. One virtual machine with 16 Intel Xeon Gold 6136 hardware threads, 32 GiB memory, and 500 GB disk space is exclusively running the RabbitMQ message broker. Most agents are running on another virtual machine with the same hardware specification. This includes the management agent, HTA storage agents, a Grafana endpoint, a WebSocket server, monitoring consumers, and various transformers. Three NFS-mounted volumes with a total capacity total of 30 TB are used for the persistent storage. These NFS volumes use a transparent compression, which achieves a data reduction of 22 %. Data sources are distributed among various measurement-specific systems. A total of 38 agents are continuously connected to the message broker, i.e., 22 data sources, 4 transformers, 4 live consumers, 6 HTA storage agents, one historic consumer (Grafana), and the management agent. Overall, the system manages ≈ 27 000 metrics, most of which are recorded for long-term storage. For metrics that were present in the previous measurement infrastructure Dataheap, collected data from up to ten years was imported into the HTA storage.

The energy measurement test systems publish measurements of four ZES LMG power analyzers. In the current configuration, one LMG670 device provides 151 kSa/s for each of the 7 channels[18]. For sustainable persistent storage, these high-resolution metrics are aggregated to 100 Sa/s. Further, two LMG450 devices and one LMG95 device produce a total of 9 AC measurements at 20 Sa/s. The HDEEM HPC power measurements on *taurus* publish 4464 metrics (three per node) at 1 Sa/s each[19]. The remaining metrics include additional power measurements from PDUs, temperature measurements for each *taurus* node, and a variety of measurements from building automation devices. These metrics use sampling rates ranging from 1 Sa/s to daily updates.

The heterogeneous composition of this MetricQ instance, as well as the use of shared resources (virtual machines, shared network, and shared storage) make it difficult to report repeatable and reproducible performance numbers. Therefore, I used dedicated systems and a controllable workload for the performance evaluation in the previous subsections. So far, there was no insurmountable performance blockade when using MetricQ in this broad practical scenario. The Python consumer implementation required performance tuning[20] to enable the live-visualization use case presented in Section 5.1.1. After the tuning, this consumer instance can sustainably process the incoming measurements — further significant increases of metric data rates might require switching the implementation to C++ or parallelizing it. The HDEEM data source has the highest message publishing rate (4464 messages per second) and is therefore split into three instances to avoid channel rate limiting. While the LMG670 data source has the highest aggregated metric rate at ≈ 1 MSa/s and the highest network traffic of ≈ 10 MB/s, it operates without any throttling or limitations due to the efficient chunking. Overall MetricQ meets the requirements even in this complex and demanding scenario. I use this MetricQ instance for the examples presented in Chapter 5 as well as some experiments in Chapter 3. More details about that data-center aspects of this infrastructure are describe in [Ils+19].

[18] The seventh measurement channel was added after the tests described in Chapter 3.

[19] The high-resolution measurements of HDEEM at 1000 Sa/s and 100 Sa/s are not available for continuous collection.

[20] The `uvloop` library improves performance compared to the default Python event loop. Moreover, I contributed performance improvements to the `aiormq` library (`https://github.com/mosquito/aiormq/issues/37`).

4.4 Conclusion

In this chapter, I described MetricQ — a scalable infrastructure for processing high-resolution energy measurement data. The concept behind MetricQ is based on a scalable message broker, which flexibly connects measurement data sources and consumers. Further, I presented Hierarchical Timeline Aggregation (HTA), a concept for persistent measurement data organization that exploits the properties of time series metric data to enable fast aggregate queries over large data sets. The HTA concept is backed by a file-based storage implementation which provides both efficient insertion and data retrieval.

In an extensive performance evaluation, I demonstrated, that the MetricQ implementation overfulfills the demanding performance requirements of both high-resolution power measurements and large-scale HPC power monitoring. For both insertion and queries, my approach significantly out-performs a prevalent state-of-the art time series database. The solution is practically used in a deployment with a broad range of measurement data sources. Combined with the enhanced power measurements of Chapter 3, this software infrastructure lays the foundation for the extensive analysis scenarios described in the next chapter.

5 Energy Efficiency Analysis

In the previous chapters, I discussed advances to power measurement as well as a complementing software infrastructure for collection, distribution, storage, and retrieval. To gain an actual scientific insight for energy efficiency research, or to discover anomalies in operation, the measurement data needs to be made accessible to a user. In the following, I will discuss specific ways to utilize power measurement data, with a focus of data managed by MetricQ.
First, I discuss general approaches to leverage the data from the measurement infrastructure for various analyses and interactive visualizations in Section 5.1. A particularly challenging aspect of creating application power traces, the synchronization of fine-grained events with timestamps from different clocks. Therefore, in Section 5.2, I present and evaluate a novel approach for time synchronization of application power traces based on correlation sequences. Finally, in Section 5.3, I showcase several practical use cases for the discussed measurements and the infrastructure implementation.

5.1 General Energy Efficiency Analysis Scenarios

The following discussed analysis scenarios are aligned with the requirements from Section 4.1. This includes visualization of live data in Section 5.1.1 and interactive visualization of historic measurements in Section 5.1.2. Both visualizations are especially important for the purpose of energy-efficient operation. Energy efficiency research is particularly focused on application power tracing in order to leverage many diverse sources of information. In Section 5.1.3, I cover the main concepts and challenges for integrating high-resolution power measurements in application performance traces. Further, I describe the technical interfaces that are used to leverage different data sources. The graphical visualization of application power traces is discussed in Section 5.1.4.

5.1.1 Live Visualization of Power Measurements

A typical method for visualizing power measurements in the context of system monitoring is a dashboard that gives an overview over the current state of the system. This can include live power consumption measurement data. Such a dashboard is depicted in Figure 5.1. It shows the current power consumption of the HDEEM partitions in *taurus*. The visualization uses the per-processor measurements and is arranged by the physical topology (nodes, chassis, racks). Each of the 2592 measured values is color-coded in a heat map and updated once every second. This structured overview allows to quickly locate patterns in the system utilization or anomalies in operation.
The visualization can further be enhanced with additional data. Power data from different sources (e.g., PDU) can complement the more fine-grained values, providing a confirmation at different levels in the power conversion. Other metric data, such as from the cooling infrastructure, has a tight correlation with power consumption and can also be handled since MetricQ can be used for generic

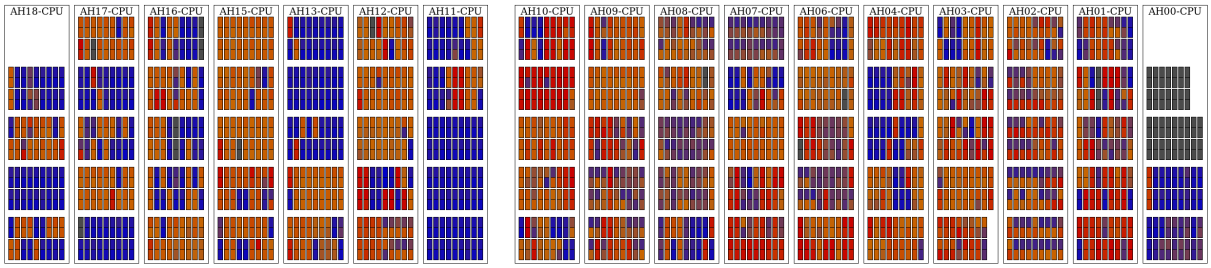


Figure 5.1: A heat map of the live power consumption of each processor in the racks (AH*NN*-CPU) of the *taurus* HDEEM partitions updated at 1 Sa/s.

metric data. Further non-metric data, such as information about scheduled jobs, can provide even more insight but needs to be collected from different interfaces.
Realizing this particular visualization with MetricQ involves additional software components: The visualization itself is implemented in JavaScript and runs in a browser. The code for this website was originally developed for Dataheap [KHN12], and adapted to MetricQ. A specific JavaScript library takes care of subscribing to metrics and invoking the drawing functions. This library, however, does not directly communicate with the infrastructure through AMQP. Communicating over a protocol such as AMQP is not well supported for JavaScript running in browsers. Instead, a WebSocket server acts as a bridge between the browser and the measurement data infrastructure. Within the infrastructure, the WebSocket server implements a *live consumer*.

5.1.2 Visualization of Long-Term Measurements

The second important use case is the visualization of persistently recorded data, i.e., historic charts of the power measurements. This can range from displaying measurements of a few seconds to several years. In Section 4.3.5, I demonstrated, that the HTA storage concept and its implementation in MetricQ is capable of answering aggregate queries over large data sets in real-time.
In order to achieve a versatile visualization without implementing a custom front-end from scratch, I leverage Grafana (see also Section 2.3.2). At the client side, a Grafana data source implementation, derived from the simple JSON data source[1], makes historic data from MetricQ accessible to Grafana. It uses a minimal protocol over HTTP with requests and answers encoded in JSON. On the other end, a Python program implements an endpoint for this Grafana data source. It derives from the Python MetricQ implementation for a historic consumer.
Figure 5.2 shows a chart created with Grafana containing four measurements from components of a single system (*ariel*). The screenshot shows one month of measurements, but the time range of the chart can be freely chosen and interactively zoomed into. Grafana allows to freely configure charts from metrics recorded in MetricQ and also other data sources. Users can further select from the possible aggregations (minimum, maximum, mean, and count) as well as a moving average function. On one hand, the Grafana frontend enables powerful dashboards, giving an overview of commonly needed metrics for both short term and long term trends. On the other hand, a user can quickly build a custom combination of charts from all possible metrics and their aggregations for arbitrary time intervals.

[1] `https://github.com/grafana/simple-json-datasource`

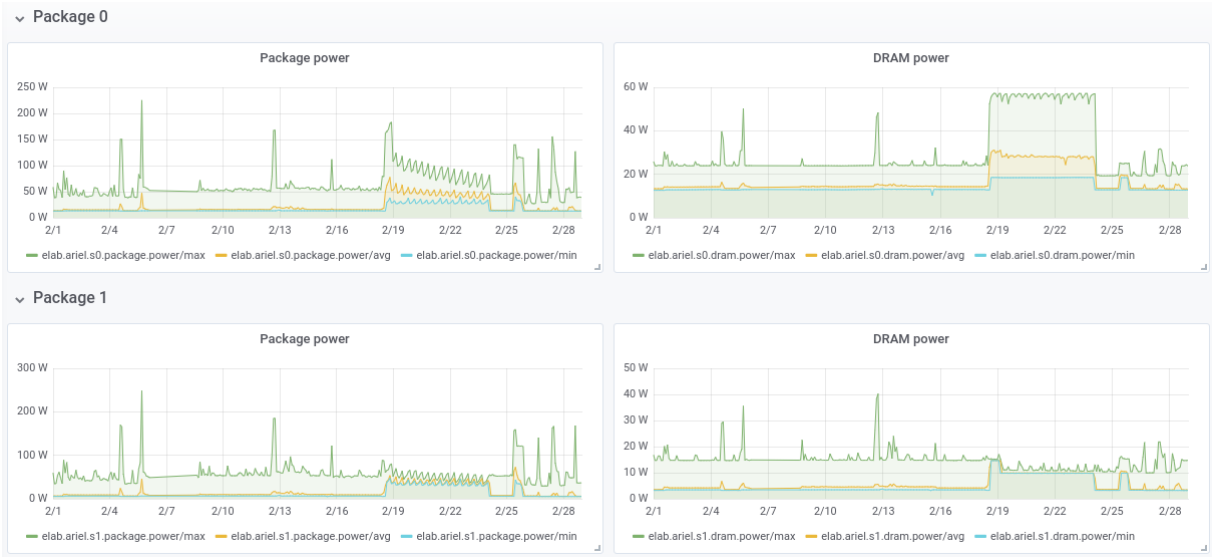


Figure 5.2: A dashboard with timeline charts of one month of power measurements on *ariel* in Grafana. Each chart shows the minimum, average, and maximum values of an aggregated 100 Sa/s measurement.

5.1.3 Integration in Application Performance Traces

The discussed visualizations for live and historic measurements focus exclusively on displaying the power values along a shared time axis. For additional insight, it is important to combine the information from power measurements with application and system performance measurements.
In particular, I focus on traces, which retain the full temporal information of both power measurements and application/system events. Alternatively, the measurements and events can be combined on a reduced level, e.g., attach the energy consumption to an application profile. Similarly, such information can also be combined for a live visualization, e.g., displaying current executed job, CPU usage, and power consumption (see Section 2.5.3).
In terms of application power tracing, I consider explicit experiments with a clear scope of time and involved resources. In the following, I discuss measurements originating around application events, but experiments involving system events can be handled likewise. During the experiment — the execution of an application — events about the application are generated either through instrumentation or sampling (see Section 2.5.1). These events are collected and recorded within a measuring system that runs on the same system under test, possibly even in the same process.
Figure 5.3 shows how the general *out-of-band post-mortem* approach can be realized with MetricQ (see also Section 2.5.3, Section 4.2). The queue is of central importance, as it buffers the measured values during the experiment within RabbitMQ. This functionality is completely decoupled from the interfaces, i.e., the measurement source implementation is agnostic to how the data it publishes is used. During the experiment, the application monitoring framework only collects events from the application and other information from the system under test. The correlation between these events and the power measurements is done with respect to the shared time axis.
While monitoring tools strive to collect a broad range of metric data, it can be difficult to implement the data collection using many different interfaces within the core of the monitoring software. A plugin interface can leverage many different interfaces for metric data without creating strong dependencies

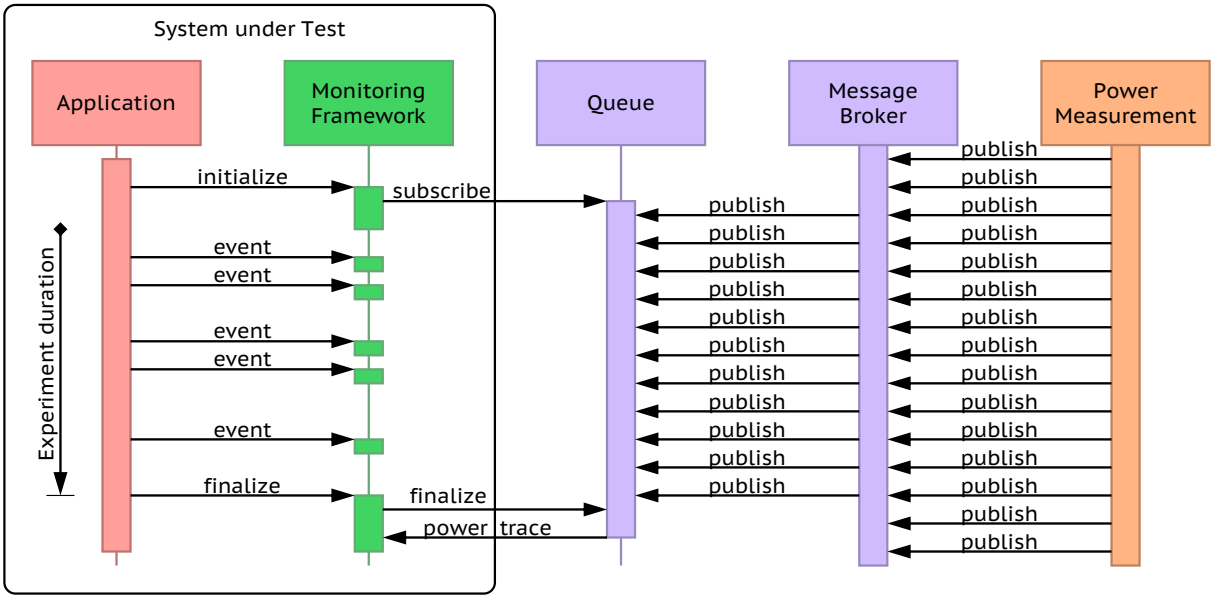


Figure 5.3: A workflow for the *post-mortem* combination of application event traces and power measurements within an experiment.

that can be difficult to maintain. Two publications that I co-authored [Sch+11; Sch+17a] described a flexible plugin interface for VampirTrace and Score-P respectively. These interfaces allow users to leverage various kinds of additional metrics by loading a shared library at runtime. The monitoring tool `lo2s`, which offers performance measurements for both application and system events, supports the same interface so that plugins can be used by multiple tools (see also Section 2.5.1).

For the HDEEM measurements, two ways of integration are available. The continuous monitoring of *average* power at 1 Sa/s is published to MetricQ and is therefore available for application power tracing in Score-P. The high-resolution measurements at 100 Sa/s to 1000 Sa/s, however, are not available for continuous readouts with the current API. Instead, HDEEM provides a recording of full-resolution measurements within the BMC for experiments up to eight hours. This interface enables full-scale high-resolution application power traces with an additional plugin. Section 5.3.3 showcases an example use case for this approach.

For different power measurement interfaces, other approaches are still preferable. RAPL, for example, is typically read locally on the system under test (see Section 2.2.4 and Section 3.6). Thus, an external buffering for *post-mortem* integration is not applicable. Instead, it can be read either in regular intervals or at application events. In all cases, no explicit measurement timestamp is available and the local timestamp has to be used. This results in inaccuracies when converting energy to power as discussed in Section 3.6.1.

5.1.4 Graphical Analysis of Application Power Traces

Score-P and `lo2s`use the same output format, OTF2, and can therefore use the same means for further analysis and visualization. Vampir [Knü+08] can be used to interactively display OTF2 files. While the main focus of Vampir is to display events of a parallel application trace in an interactive timeline, it can also be used for system events. OTF2 and Vampir support arbitrary metric data on the same time scale as the application events (see also Section 2.5.1).

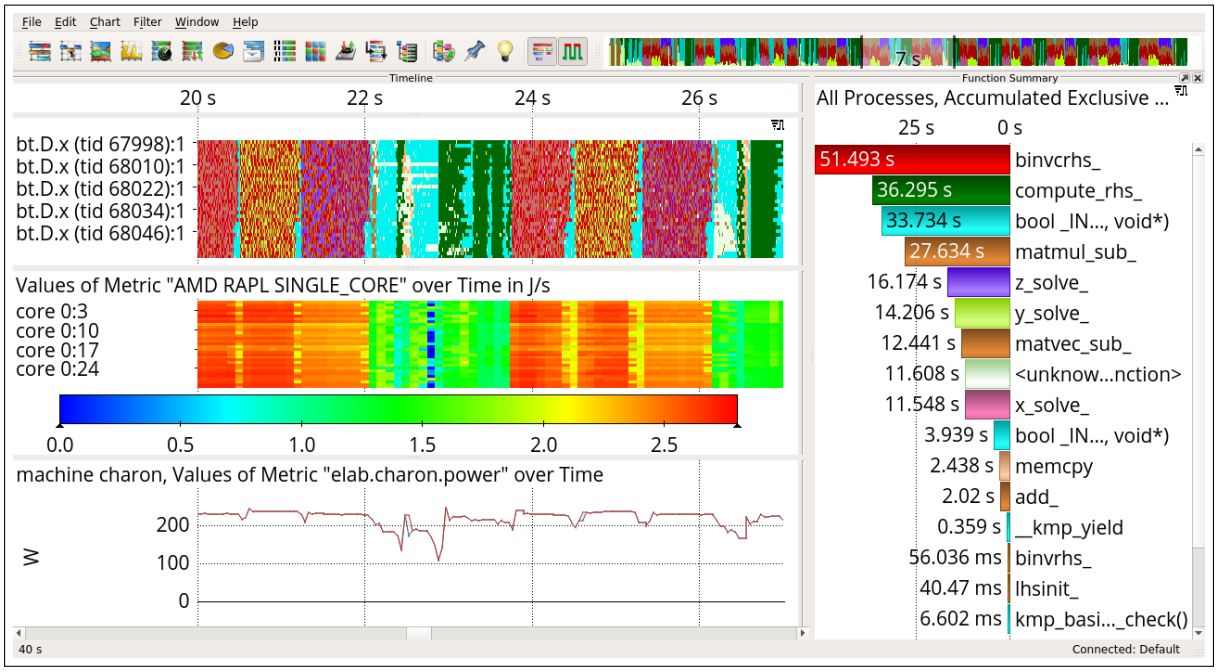


Figure 5.4: An example visualization of an application power trace with Vampir. The top shows the menu, available charts, and full trace preview. The left side shows the *master timeline* (top) with application events, *performance radar* (middle) heat map with per-core power values, and *counter data timeline* (bottom) with system power measurements for a seven-second time interval. The right side displays the *function summary*, a statistical profile for the selected time interval.

Figure 5.4 shows an exemplary visualization of an application power trace from the NPB BT benchmark [VH03]. The three timeline charts on the left present application events and power measurements in the selected time interval. Events from the parallel application, in this case color-coded function samples, are shown in the *master timeline* (top left). The master timeline gives an overview of the activity of the application both over time and threads. The *counter data timeline* (bottom left) was originally used to display rates of performance monitoring counters, but it can be used to display power measurements. For a concise visualization of separate measurements of multiple hardware components, e.g., per-core or per-node in a multi-node application, the *performance radar* (middle left) is used. The performance radar shows the multi-dimensional values as a color-coded heat map. Note that the displayed per-core measurements are a specific feature of `lo2s`. The plugin interface discussed in Section 5.1.3 only supports per thread, per process, per host, or unique metrics. Score-P does not allow metric plugins to associate values with the hardware components within a compute node.

With the support for arbitrary metrics and display of versatile events in various timeline displays, Vampir facilitates a comprehensive visual analysis of application traces and system event traces containing power measurements. Moreover, the scalable implementation can efficiently handle high-resolution measurements.

5.2 Correlating Power Measurements with Application Events

When analyzing power measurement traces in the context of application or system events, they are typically correlated by a shared time axis. However, there are several factors that impede an accurate correlation of the two, particularly for power measurements at high sampling rates. In the analog domain, any filters, including low-pass filters inherent to the measurement domain or necessary anti-aliasing filters, result in a delay between the power draw at the consumer and the measurement signal. Digital filters can add even further delays. Additionally, there is often a delay between the analog-to-digital conversion and taking the measurement timestamp that adds uncertainty to the timestamp associated with a measurement sample. For example, the National Instruments data acquisition API for reading a buffer of analog samples does not offer any timestamps[2]. In such a case, the timestamps have to be reconstructed from additional clocks of the controlling systems.
The data acquisition system and system under test are typically separate and thus use separate clocks, introducing another source of uncertainty in the temporal correlation between application / system events and measured values. Those clocks are commonly synchronized using the network time protocol (NTP). In local networks, NTP can offer a better accuracy than 100 µs [Mil06] — whether that is sufficient or not depends on the sampling rate. For some network configurations, more precise network time synchronization can be achieved, e.g., by using switches as NTP masters. GPS based clocks are another option that require additional hardware.
In [Lib+16] and [Lib+18b], Libri et al. showed a detailed evaluation of synchronization mechanisms in HPC systems with a focus on being able to correctly align monitoring data. For two synchronized embedded monitoring nodes, the authors report an NTP accuracy of 17.5 µs and a precision of 8.4 µs. Leveraging the precise time protocol (PTP), they achieve an accuracy and precision of 16.1 ns and 513.7 ns respectively (see [Lib+16]). On the compute nodes of the HPC cluster, they report further improvement towards an accuracy of 2.6 µs and a precision below 2.7 µs with NTP and "sub-microsecond" using PTP (see [Lib+18b]). The authors further validate the solution with a synthetic benchmark that shows a good correlation between the recorded timestamp of an application state transition and a power measurement recording. These results are very important for measurements in large distributed systems. However, this approach cannot universally be applied, for instance when using integrated power measurement devices that lack an option for network-based time synchronization — like the LMG670 power analyzer. It also does not account for possible filter delays and delayed timestamping. The high clock synchronization accuracy comes at a certain cost: With NTP, the fastest possible polling interval of 8 s is used. PTP requires two daemons: One performs the required network communication at 1 Hz, the other synchronizes the system clock with the PTP Hardware Clock at 12 Hz. It was shown that the network traffic is negligible in terms of total bandwidth, even for a large-scale system. The perturbation of regularly scheduled tasks, however, may have an undesired impact, particularly for use cases that include idle phases.
Another consideration for synchronization is to use electrical signals that can be controlled from a CPU. For instance, the serial port of a computing system could be used to carry a signal to a data acquisition device. While the signal itself would be sufficiently fast for synchronization, preliminary tests on the systems under test showed that writing a single byte to a serial port takes as much as 8 ms.

[2] As documented by the API function `DAQmxReadAnalogF64`
`http://zone.ni.com/reference/en-XX/help/370471AE-01/daqmxcfunc/daqmxreadanalogf64/`

This is likely caused by buffering in the kernel. A high precision direct time synchronization would require general-purpose input/output (GPIO) pins with minimal latency in a sub-microsecond range. On the application level, that means the output must be triggered without going through the Linux kernel. In [Lib+16], the authors demonstrated how this can be used to *evaluate* time synchronization accuracy.

5.2.1 Challenges for Time Synchronization of Power Measurements

Measurements operating in the order of 1 Sa/s require little consideration in terms of synchronizing timestamps as long as NTP is enabled in general. For the LMG450 measurements with readouts at 20 Sa/s, the delay over serial connection is still insignificant, but a careful NTP configuration is required. The HDEEM measurements operate at up to 1 kSa/s. In this configuration, the analog and digital filters introduce a significant delay. This delay is corrected with static offsets of −5.6 ms and −35 ms for the blade and VR measurements, respectively. These offsets are internally applied to the timestamps within the HDEEM library. Therefore, different timestamps are provided for blade and VR measurements in the API. Clock synchronization is ensured by local NTP servers running on the admin nodes of *taurus*, which are closely connected to both the compute nodes (system under test) and BMC hosts (measuring system). In-band measurements further use an HDEEM-specific kernel module that allows to send a low-latency GPIO signal from the compute node to the BMC upon starting and stopping measurements for more accurate clock synchronization (see [Ils+18c]). With sampling rates up to $1.\overline{21}$ MSa/s, the measuring systems *apollo*, *artemis*, *diana*, and *ariel* pose more stringent requirements for time synchronization. With the LMG670-based measurements on *ariel*, the measurement timestamps are generated by the measurement device. A remote command is used to set the system time on the device to a given timestamp [ZES16, Sec. 8.9.265]. In the implementation of the measurement control software (MetricQ data source), this coarse-grained synchronization is performed when establishing the connection to the measurement device. To limit the drift of the clock on the measurement device, the data source is automatically restarted daily.
Many traces contain distinct patterns that can be recognized in both the system / application event trace and the power trace. This allows a manual alignment, e.g., using a visual comparison session in Vampir. For the measurements in *apollo*, *artemis*, and *diana*, which are all based on National Instruments data acquisition, I designed an automatic time synchronization. The basic idea is to artificially create a pattern in power consumption that can be automatically detected and used for synchronization. The metric plugin generates a transition from idle to a high-power kernel at both the begin and end of a measurement. A heuristic is then used to locate these transitions in the power consumption signal and align the timestamps with those recorded during generation of the transitions. Over the course of the measurements, timestamps are corrected using a linear interpolation between the two synchronization points. The heuristic is very sensitive to noise in the power consumption signal by overlapping activity. Thus, the synchronization requires exclusive access to all cores of the system to limit the impact of noise. A transition from idle to active on all cores is used due to a particularly high change in amplitude (power). In practice, this method is typically accurate in the order of 50 µs when compared with manual alignment [Ils+15a]. However, this pattern can be easily distorted by background tasks activating the processor during the idle phase, making the heuristic less reliable.

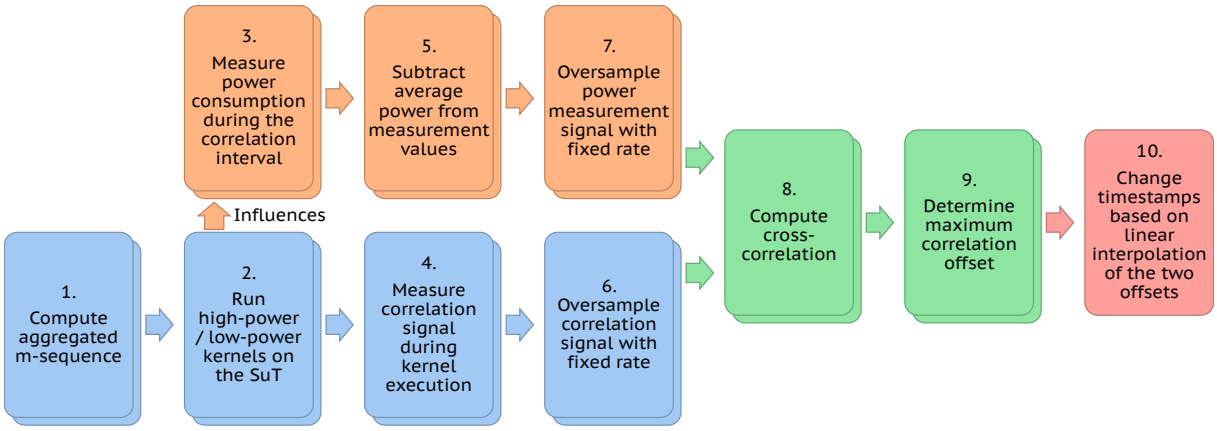


Figure 5.5: A workflow for automatic time synchronization of power measurements. The Steps 1 to 4 are performed both before and after the experiment. The Steps 5 to 8 are executed twice for post-processing. Step 10 uses the two results of each invocation of Step 9.

5.2.2 Reliable Automatic Time Synchronization with Correlation Sequences

In the following, I present a novel approach for synchronizing power measurements and application events based on correlation sequences and crosscorrelation. Such statistical techniques for synchronizing signals are used for instance in wireless communication and radar applications. In general, the approach involves the following steps: First, a signal is generated from a correlation sequence. This sequence is characterized by a low auto-correlation for non-zero shifts. The correlation signal is transmitted over a channel and then received again. I leverage the power consumption signal itself as a unconventional channel from the system under test to the measuring system. A crosscorrelation is then used to determine the time-shift between the transmitted and received power signal. The mathematical principles of signal correlation and its application to established fields are described in literature, e.g., [Sol05; Lük92]. Applying the principles to time synchronization of power measurements, however, has certain specific aspects, which I discuss in the following. An overview of the proposed workflow is shown in Figure 5.5.

5.2.3 Creating a Correlation Signal on a Power Measurement Channel

First of all, leveraging signal correlation for synchronizing power measurements requires a way to turn a correlation sequence into a signal that can eventually be measured as the power consumption. Special workloads with known distinct power consumption can be used to affect the power consumption signal in a controlled matter. This technique to convert a binary sequence into a correlating power consumption signal establishes an indirect and unusual communication *channel*.

Short pulses provide the sharpest autocorrelation — increasing the resolution at which the peak can be determined under noise. However, as discussed in Section 3.5, the bandwidth of the power measurement channel is limited. Thus, arbitrarily short pulses are not suitable as basic block for the correlation signal. This limitation can be overcome by using a longer coded signal pattern. The increased duration increases the transmitted signal energy and results in a sharp correlation spike (see [Sol05, Sec. 12.1]). In addition, longer pulses are detectable with the limited bandwidth of the power measurement channel.

A simple way to generate longer sequences with good auto-correlation properties is to use maximum length sequences (m-sequences) [Lük92, Ch. 10]. In particular, I use binary m-sequences that are generated using an irreducible polynomial in $GF(2)$. A binary m-sequence generated using a polynomial with the degree r, has a length of $2^r - 1$. The exponents of the polynomial define a linear-feedback shift register that can efficiently compute the binary sequence [Lük92, Sec. 3.4.3]. In general, there exist sequences with better autocorrelation properties. For instance Barker sequences are defined to have un-normalized autocorrelation, which is bounded by 1 for all non-zero offsets. However, such sequences are only known for lengths ≤ 13 [Sol05, Sec. 12.3], and are thus not suitable for the described use case. Long m-sequences on the other hand, can be generated efficiently.

When creating the pattern by running the different kernels, it is difficult to precisely control the rate at which the workload varies. On any non-real-time computer system it cannot be guaranteed to run a kernel for an exact amount of time. In general, any workload with an influence on power consumption could be used — e.g., disk accesses — in conjunction with a suitable measurement channel. However, the low-power and high-power kernels presented in Section 3.4.1 achieve particularly sharp transitions of CPU power consumption at a fine-grained time scale. Running these workloads only on a single core provides the sharpest possible transitions with limited amplitude changes. Multiple threads would increase the amplitude at the cost of ambiguity of timestamps and possibly longer transitions. Therefore, I execute the low-power and high-power kernels in a single thread, which is pinned to one logical CPU. The lower significance due to low amplitude changes is compensated by increasing the length of the correlation sequence.

Ideally, there should be no measurable change of the measurement signal within the time intervals corresponding to one value of the correlation sequence. Running control code between repetitions of the same kernel could cause such changes. Thus, I aggregate the binary sequence by grouping successive values to (value, count) pairs, e.g.:

$$1,1,1,0,1,0,0 \rightarrow (1,3),(0,1),(1,1),(0,2) \tag{5.1}$$

Rather than using the given time for each kernel invocation, a deadline timer is incremented for each kernel invocation. This approach enforces an amortized constant rate, even if each individual kernel does not achieve its expected duration perfectly. To further take the imperfect kernel execution durations at an individual level into account, all transition times are recorded. This provides an accurate actual transmitted signal for correlation in which -1 represents low-power and 1 the high-power kernels.

As shown in Figure 5.6, a padding phase, in which the low-power kernel is executed, is added both before and after the executing the actual correlation signal. The padding reduces the impact of noise from the experiment itself, that could otherwise affect the power measurement signal within the correlation interval. As long as the clock-offset is smaller than both the correlation interval and padding duration, the power measurement signal within the correlation interval will overlap only with the correlation and padding intervals. This concludes the Steps 1, 2, and 4 of the workflow depicted in Figure 5.5.

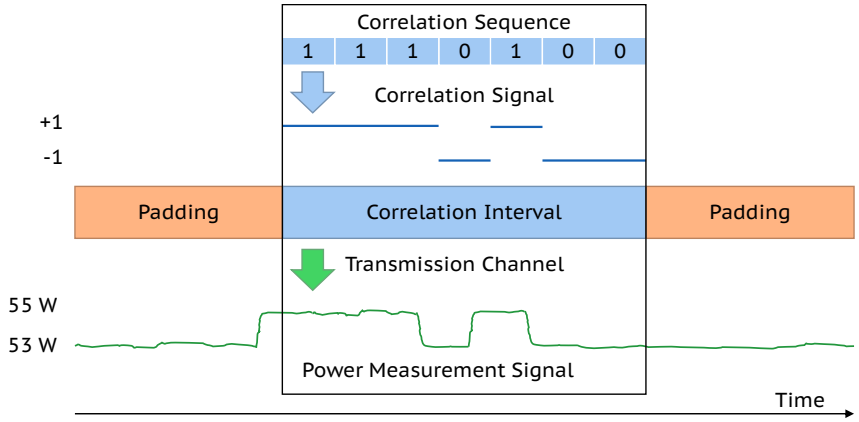


Figure 5.6: The relationship between the correlation sequence, correlation signal, and power measurement signal during the correlation interval. Power values are given as an example.

5.2.4 Processing the Correlation Signal and Measured Power Values

On the *receiver* side of the channel, the power measurement is influenced by the invoked kernels during the correlation interval as depicted in Figure 5.6. In the transmission channel, many effects are applied to the correlation signal, some of which need special consideration. The theoretical *correlation signal* is defined as:

$$s(n) \in \{-1, 1\}, n \in \{0, 1, ..., N-1\} \tag{5.2}$$

Technically, this is a *sequence* since I discuss the signal in the digital domain. However, this sequence is different than the initial m-sequence in that it is based on actual measured transition timestamps and has a different sampling rate. For now, I assume common and uniform sampling rates and defer the consideration of different sampling rates.
To this sequence, the time offset δ (in units of the sampling period) is applied — the value of δ is to be determined[3]:

$$s'(n) = s(n-\delta) \tag{5.3}$$

Going to the power domain, the range of the signal changes by an offset and factor:

$$p(n) = as'(n) + b \tag{5.4}$$

$$= as(n-\delta) + b \tag{5.5}$$

An inherent low-pass filter and noise further affect this signal (see also Section 3.5.1). The delay of the filter is included in δ, but I make no further assumptions about filter and noise. Considering the noise, it is important to retaining a high peak of the correlation such that it can be detected correctly. In Section 5.2.6, I evaluate the characteristics of the peak for practical examples. Given that the correlation sequence s was based on an m-sequence, it can be assumed that its auto-correlation $\varphi_{ss}(\tau)$ has a sharp peak at $\tau = 0$.

[3] δ is assumed to be constant during the relatively short correlation interval.

Using the transformation rules for crosscorrelation from [Lük92, Tab. 2.3], the crosscorrelation of the two simplified real-valued signals (correlation signal s and power measurement signal p) is the following[4]:

$$\varphi_{sp}(\tau) = a\varphi_{ss'}(\tau) + b\sum_{n=0}^{N-1-\tau} s(n) \tag{5.6}$$

$$\varphi_{ss'}(\tau) = \varphi_{ss}(\tau-\delta) \tag{5.7}$$

$$\varphi_{sp}(\tau) = a\varphi_{ss}(\tau-\delta) + b\sum_{n=0}^{N-1-\tau} s(n) \tag{5.8}$$

While the first summand in (5.8) contains the shift δ — which is to be determined — and is only scaled by a, the second term introduces noise in the correlation signal. The partial sum over $s(n)$ is essentially a random walk. The average of p can be removed by applying an additional transformation:

$$\overline{p} = \frac{1}{N}\sum_{m=0}^{N-1} p(m) \tag{5.9}$$

$$p_0(n) = p(n) - \overline{p} \tag{5.10}$$

Further, the mean of the correlation sequence, and consequently the correlation signal s is negligible ($1/N$ [Lük92, Sec. 10.7.1]). Thus, in the idealized case with $\overline{s} = 0$, from (5.5) follows:

$$\overline{p} = a\overline{s} + b \tag{5.11}$$

$$b = \overline{p} \tag{5.12}$$

Based on (5.8), the correlation using the transformed p_0 is:

$$\varphi_{sp_0}(\tau) = \varphi_{sp}(\tau) - \overline{p}\sum_{n=0}^{N-1-\tau} s(n) \tag{5.13}$$

$$= a\varphi_{ss}(\tau-\delta) \tag{5.14}$$

The additional noise is canceled out, there remains only the shifted and scaled initial autocorrelation. Therefore, in Step 5 of Figure 5.5, the average is removed from the measured power consumption trace. Utilizing the known autocorrelation peak of φ_{ss} at 0, δ can be determined:

$$\delta = \arg\max_{\tau} \varphi_{sp_0}(\tau) \tag{5.15}$$

In traditional applications for crosscorrelation, such a transformation is typically not necessary due to the more symmetric sending and receiving of the signal. Practical results confirmed that removing the mean from the power measurement signal increases the statistical significance of the correlation.

[4]The arbitrary impact from noise and filter is omitted from the formula, as I focus on the impact non-zero mean in the power consumption signal.

5.2.5 Common Oversampling of the Correlation Signals at Different Rates

Before applying the crosscorrelation, the sampling rates of both signals need to be equalized. Initially, the recorded correlation signal and power measurement signal have different sampling intervals. Since I use a generic interface to leverage various kinds of measurement devices, I cannot even assume that the sampling rates of power measurements are constant over time.

First, the semantics of the time-discrete measurement points on a continuous time scale have to be considered. The synthetic correlation signal has a well-defined continuous definition: The state (-1: low-power, $+1$: high-power) is constant between the recorded transitions i.e., a rectangular waveform. For the measured power values, a temporal validity from each measurement point to the previous one is assumed. This is true for averaged power measurements that are timestamped with their last measurement sample going into the average. For instantaneous samples, it would be more accurate to apply a low-pass filter to the pulsed signal for re-sampling. However, since the time synchronization does not attempt to achieve a precision of less than the sampling period for instantaneous power samples, the given assumption is sufficiently accurate. Based on these assumptions the re-sampling can be performed with simple generic code rather than using filters for sample rate conversion.

In Steps 6 and 7 of Figure 5.5, both the correlation signal and the transformed power measurement signal are oversampled with the same fixed rate. Due to the lengthy time intervals required for the kernels to influence the measured power consumption at significant amplitude, the low original rate of the correlation sequence would present a significant restriction on the precision of the resulting synchronization. In this use case of synchronizing power measurements, the configurable oversampling rate allows a trade-off between possible precision of the synchronization and cost for computing the crosscorrelation. For constant measurement sampling rates, it is also possible to re-sample only the correlation signal at the timestamps of power measurement samples and keep the power measurement signal as is. While this is true in the discussed example, the implementation attempts to be more general by avoiding this assumption and thus oversampling both signals.

Determining the crosscorrelation from the given signals can be very computationally demanding. The actual computation is more efficient in the frequency domain [Lük92, Sec. 2.2.1], and a fast Fourier transform (FFT) can be used to transform the signals to and from the frequency domain.

Algorithm 1 describes the efficient approach for determining the offset using FFTs. It includes Steps 5 through 9 of Figure 5.5. When computing the correlation arrays, negative offsets are wrapping around in a circular fashion. Thus, to avoid overlap, the sequences are extended with zeros to at least twice the size minus one. The sizes of both arrays are equal due to the identical correlation interval and oversampling rate. For the implementation, I use the FFTW (Fastest Fourier Transform in the West) library[5].

The experiment can be long enough, that the clock drift may significantly impact the time offset during its duration. Therefore, the correlation signal is repeated before and after the experiment, and a linear interpolation is used for timestamp correction in the final Step 10 of Figure 5.5.

[5] Available as open-source at `http://www.fftw.org/`

Algorithm 1 Determining the time offset between the correlation signal (s) and the measured power (p) given a common oversampling period (t) and the correlation interval (i)

function FINDOFFSET(s, p, t, i)
  $s_s \leftarrow \text{SAMPLE}(s, t, i)$
  $p_s \leftarrow \text{SAMPLE}(p - \overline{p}, t, i)$ ▷ remove average
  $N \leftarrow |s_s|$ ▷ also equal to $|p_s|$
  $N_e \leftarrow 2^{\lceil ld(2N-1) \rceil}$ ▷ power of 2 size improves FFTW performance
  $s_e \leftarrow \text{EXTEND0}(s_s, N_e)$ ▷ pad by at least factor of 2 to avoid circular overlap
  $p_e \leftarrow \text{EXTEND0}(p_s, N_e)$
  $S \leftarrow \text{FFT}(s_e)$
  $P \leftarrow \text{FFT}(p_e)$
  $\varphi \leftarrow \text{IFFT}(S^* \cdot P)$ ▷ where S^* is the element-wise complex conjugate of S
  $\gamma \leftarrow \arg\max_\tau \varphi(\tau)$ ▷ offset with maximum autocorrelation
  if $\delta < N$ **then**
    return $\delta \cdot t$
  else
    return $(\delta - N_e) \cdot t$ ▷ circular index corresponds to negative offset
  end if
end function

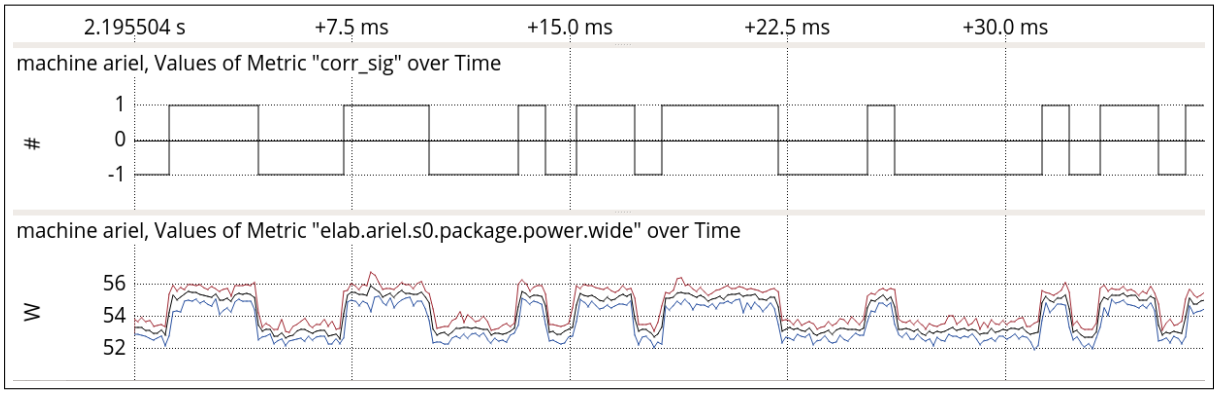
5.2.6 Evaluation of Correlation and Time Synchronization

This novel approach for time synchronization is used with the high-resolution measurements of *ariel*. The synchronization kernels are pinned to core 0, and the CPU power consumption of the processor package 0 is used. For the evaluation, a measurement sampling rate of 152 kSa/s is used. All parameters of the time synchronization are configurable, the following settings have been used for the evaluation: The correlation uses an m-sequence with $N = 8192$ and a time quantum of 1 ms (a correlation interval of 8.2 s) as well as a padding of 2 s. The common oversampling period is 5 µs.

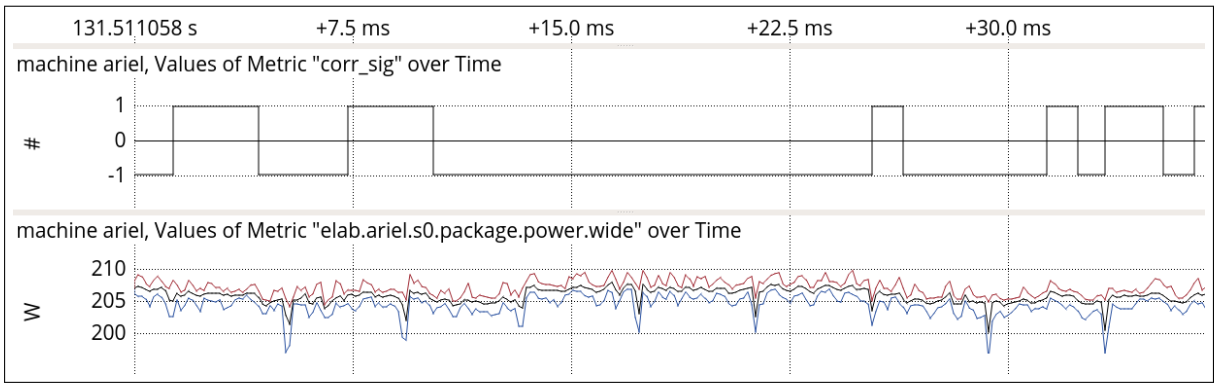
In the following, I discuss two synchronization scenarios:

1. The remainder of the system is in normal *idle*, i.e., there is only the normal background activity during synchronization. This is the ideal case for synchronization.

2. A *FIRESTARTER* process is running in the background during the synchronization interval on *all* cores (see also Section 3.4.1). This leads to a high background power consumption, masking the impact of the correlation signal. Further, the operating system will switch between executing the correlation kernels and FIRESTARTER on core 0. This noise has a strong impact on correlation and would make the classical time synchronization with a heuristic impossible and a manual synchronization very difficult. In addition, the scheduling makes it very difficult to execute the correlation kernels precisely at the expected time intervals. Thus, this scenario reveals whether the deadline timer approach is effective in maintaining good correlation properties. It also shows whether the recording of actual transition timestamps produces a correct correlation signal.



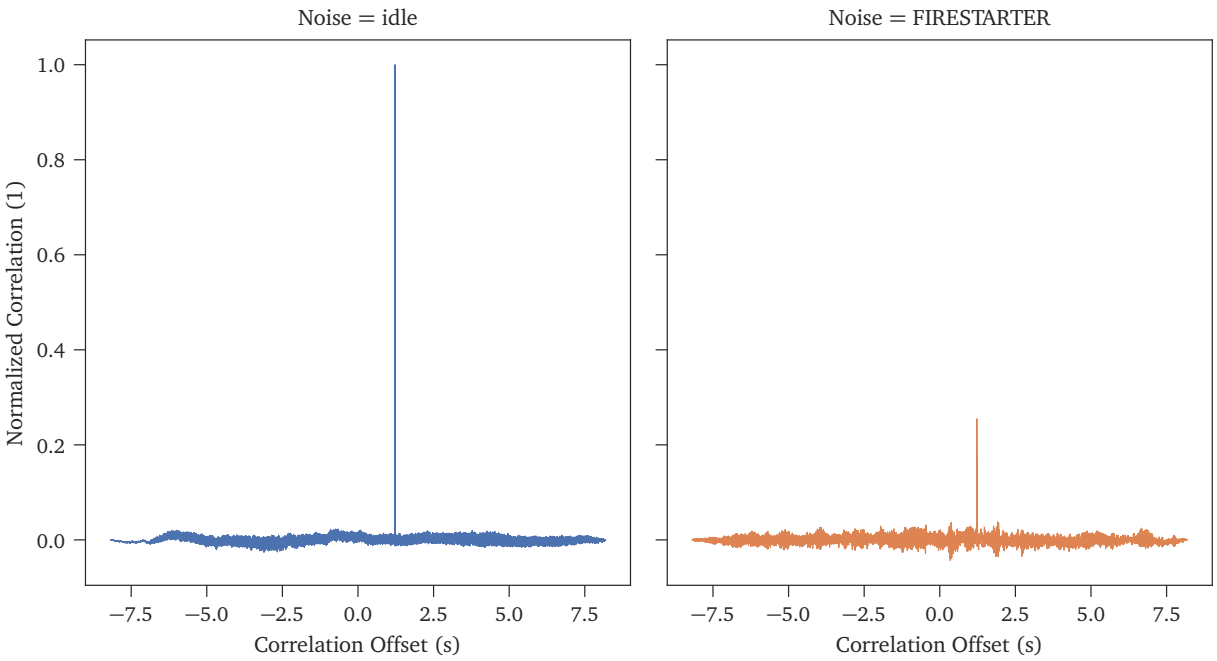
(a) *Idle* - no background activity.



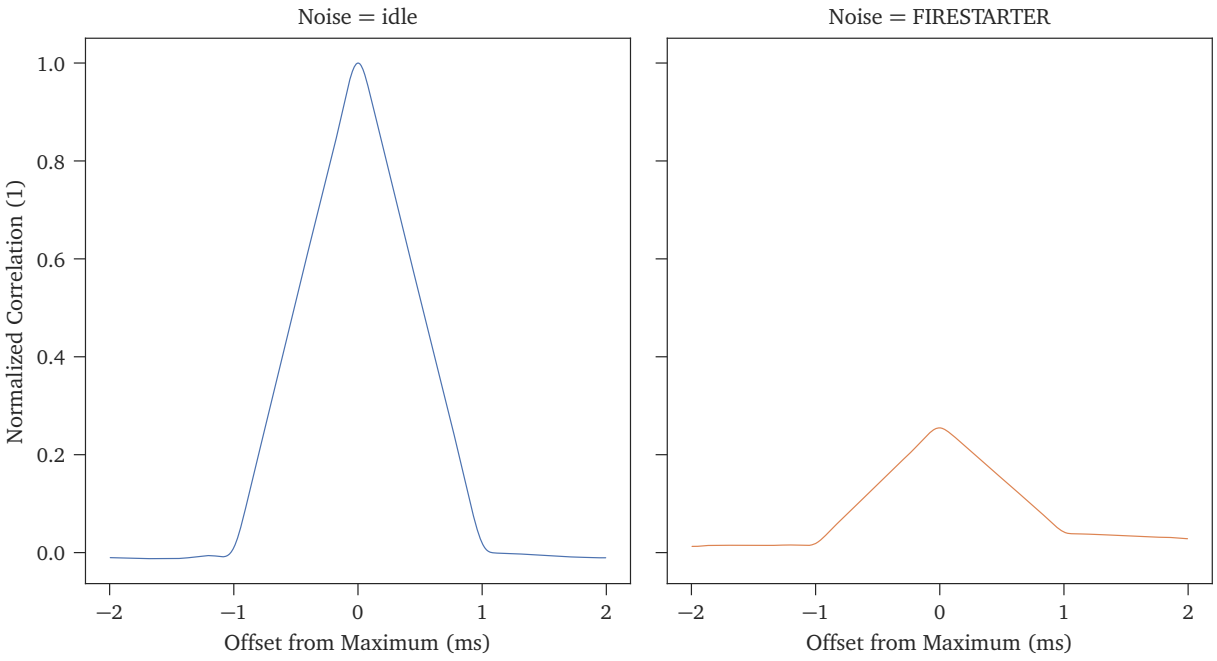
(b) *FIRESTARTER* running in background.

Figure 5.7: A comparison of the synthetic correlation signal (top) and the measured power consumption (bottom) after applying the time correction. The displayed time intervals are short parts of the total correlation intervals, each at the same point within the underlying m-sequence. The different colors (red, black, blue) show the maximum, mean, and minimum power respectively.

The two different scenarios are setup to affect synchronization before and after an experiment respectively. Figure 5.7 shows the correlation signal and measured power for both scenarios with applied time synchronization. While the correlation is clearly visible in the *idle* case (Figure 5.7a), it is hardly possible to visually identify the correlation when *FIRESTARTER* is running (Figure 5.7b). Further, the correlation signal in Figure 5.7b itself is faulty - it shows the same part of the underlying m-sequence as Figure 5.7a but is missing some impulses. This is due to contention for CPU time between the synchronization kernels and *FIRESTARTER*. The deadline timer ensures that the overall sequence is not distorted even though locally some impulses are omitted and skewed.
The results of the crosscorrelation are shown in Figure 5.8. In both scenarios, the correlation spike is well defined and significant — but it is clearly stronger in the *idle* case. When looking at the spike in detail (Figure 5.8b), the linear slope around ±1 ms of the maximum is clearly visible. This corresponds to the 1 ms time quantum used to generate the correlation signal. Reducing the time quantum would result in a narrower spike [Sol05, Sec. 12.1], but at some point the amplitude would vanish due to low-pass of the channel. Nevertheless, the well-defined peak — rather than a flat plateau — allows to determine the offset at a more fine-grained resolution than the 1 ms time quantum. This shows that the oversampling of the coarse-grained correlation signal does effectively improve precision.



(a) Full range of offsets.



(b) Limited range of offsets around the detected peak.

Figure 5.8: A crosscorrelation between the recorded correlation signal and the measured power consumption. The correlation is shown for no background activity (idle) and continuous background activity (FIRESTARTER). The correlation amplitude is normalized with the same factor for both signals.

Quantifying the significance of the correlation is challenging due to the width of the peak. One metric that is commonly used to evaluate the correlation quality is the factor F between main maximum at γ and side maximum at $\tau \neq \gamma$ defined as:

$$\gamma = \arg\max_{\tau} \varphi(\tau) \tag{5.16}$$

$$F = \max_{\tau \neq \gamma} \frac{\varphi(\gamma)}{|\varphi(\tau)|} \tag{5.17}$$

However, this factor is meaningless in this use case, because there are similar values very close to the maximum due to the oversampling. The merit factor suffers from the same limitation (see also [Lük92, Sec. 4.1]). Therefore, I define a modified factor that excludes all the correlations φ within the time quantum from the denominator. Let ϵ be the number of samples within the correlation time quantum, i.e., $\epsilon = 200$ for a time quantum of 1 ms and an oversampling period of 5 µs.

$$F' = \max_{\|\tau-\gamma\| \geq \epsilon} \frac{\varphi(\gamma)}{|\varphi(\tau)|} \tag{5.18}$$

The practical implementation of the condition $\|\tau - \gamma\|$ considers the circular properties within the correlation array. For the discussed tests, the modified factor F' is 44.03 with *idle* and 6.16 with *FIRESTARTER*, confirming the observation of Figure 5.8. Additional tests indicate that this factor decreases when reducing the length of the m-sequences. For the *FIRESTARTER* scenario this eventually becomes problematic and results are unreliable for $N = 1024$, while for *idle* the correlation remains reliable. The implementation in the application power tracing plugin uses F' as an indicate as to whether the synchronization was successful.

In the mentioned example, each crosscorrelation was performed for 1.6 million data points and the execution took 1.3 s. This computational overhead adds to the 24 s overhead from two synchronization intervals with padding. This overhead does not impose practical limitations for post-processing-base application power tracing. However, executing the extensive synchronization patterns makes this approach unsuitable for real-time analysis.

Figure 5.9 shows the temporal correlation between scheduled tasks and the power consumption trace with corrected timestamps. The change in power consumption matches the transition of scheduled tasks closely. However, the change in power consumption exhibits a slope of ≈ 150 µs. While the transition of tasks lies within the transition of power measurements, it could be argued, that the power measurement slope should *begin* at the task transition. By that argument, the synchronization could be considered to have an error of ≈ 70 µs. Ultimately, the automatic offset detection using correlation sequences provides a time synchronization within the precision of a power transition slope. I demonstrated a correct synchronization with significant correlation even under the adverse influence of *FIRESTARTER* — yielding a high confidence in the synchronization under imperfect conditions.

The proposed synchronization approach works pairwise between one measurement channel and a single-node system under test. It can be combined with traditional network-based synchronization techniques (e.g., [Lib+18b]) or message-ordering (e.g., [Bec+10]) to achieve a global correlation between events and measurements in a distributed system.

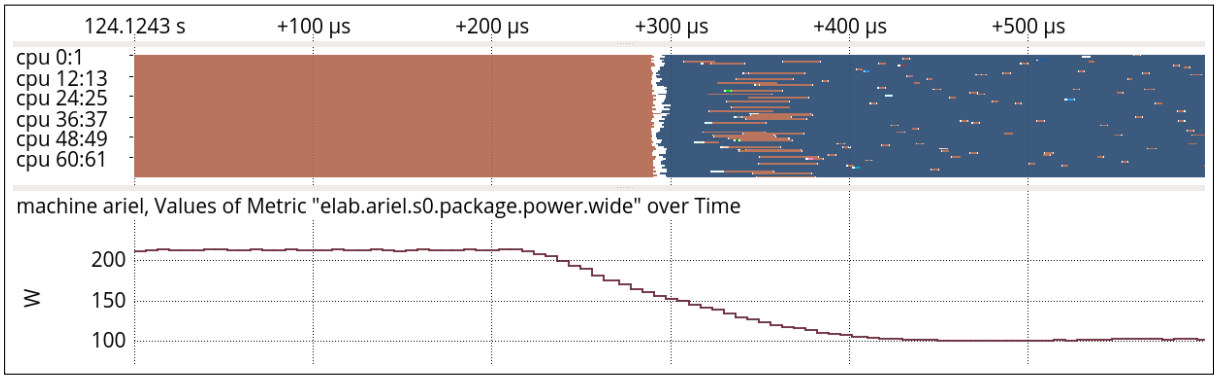


Figure 5.9: A correlation of system events using the automatic time synchronization. The top panel shows the scheduled tasks (brown is FIRESTARTER, blue is the monitoring process), the bottom panel shows the power consumption.

5.3 Use Cases for Application Power Traces

The previous chapters and sections lay the foundations for collecting and processing high-resolution power measurements and correlating them with application and system events. In the following, I present three specific examples of how the resulting application power traces can be leveraged to gain valuable insights. Each example has particularly challenging constraints that connect back to the requirements for power measurements and their processing. Section 5.3.1 covers the analysis of a complex power anomaly that depends on sophisticated system event monitoring. This particular anomaly affects idle systems and thus requires a passive monitoring without any perturbation from regular power measurements. In Section 5.3.2, I show how measurements can be used to understand and model C-state transitions. Specifically the transition from an active to an idle state cannot be observed directly due to the lack of an observable event. The power consumption trace, however, reveals properties of these transitions — on the presumption of a measurement with a very high temporal resolution. Finally, in Section 5.3.3, I demonstrate the scalable measurement capabilities of HDEEM. To that end, I discuss an application power trace of a benchmark running on more than a thousand nodes of a production HPC system.

5.3.1 Analyzing Complex Power Anomalies

One of the most important techniques for saving energy on computing systems are C-states (see also Section 2.4.2). Contemporary processors offer a range of C-states that allow a trade-off between wake-up latency and power consumption (see also [SMW14]). Whenever there are no active tasks assigned to a logical CPU, the operating system may decide to enter a sleep state, e.g., using the `mwait` instruction. The operating system selects the C-state that is to be entered and passes the choice to the processor. On a system with multiple hardware threads per core, each core uses the lowest (shallowest) C-state of all its hardware threads. In [Ils+18a], I described an anomaly called *Powernightmares*. This anomaly was discovered on recent workstation and server systems running the Linux kernel and caused excessive energy consumption of $\sim 10\,\%$ on affected systems [WI18].

High-resolution energy measurements and their combination with sophisticated system monitoring played a crucial role in discovering the issue, identifying its cause, and evaluating solutions. Particularly, the energy measurements on *diana*, *taurus*, and *ariel* were used in the process.

Idle Power Management in Linux

Like most contemporary operating systems, Linux uses preemptive multitasking, i.e., a scheduling-clock interrupts each logical CPU in regular intervals to run the task scheduler. Since these scheduling interrupts cause wake-ups from C-states and thus increase the power consumption in idle, they can be disabled when entering a C-state. This feature is called *dyntick-idle*, also referred to as *nohz* or *tickless* and is the default mode of operation for Linux (see [McK]). Therefore, a logical CPU can remain in idle mode continuously until the next interrupt from a timer or external source, reducing the idle power consumption.

When going to idle, the targeted C-state is selected by the *cpuidle governor* in Linux. The choice of a C-state presents a trade-off between a large reduction of power consumption (deep C-state) and fast wake-up latency (shallow C-state). The actual way to enter idle is implemented in an architecture-specific *cpuidle driver*. For the discussed systems, this is the microarchitecture-aware `intel_idle` driver. It uses the `mwait` instruction, which passes the selected C-state as a hint to the processor.

During the initial discovery of Powernightmares, around Linux version 4.10, the following two cpuidle governors were available: The *ladder* governor increases or decreases the chosen C-state from the consecutively numbered available states depending on whether the previous choice was deemed correct. While this behavior is appropriate for tick-based kernels, it is problematic with systems running in dyntick-idle [PLB07, Sec. 4.1]. In dyntick-idle, the governor needs to be able to select the most appropriate C-state directly, skipping any intermediate C-states, because it may be in one selected state for a long time.

Therefore, tickless Linux systems use the *menu governor* by default. The menu governor combines several inputs to a heuristic that predicts the duration of an upcoming idle phase. In order to choose the most efficient C-state, this prediction is compared to the *target_residency* of all available C-states. This *target_residency* defines a break-even-point at which the additional energy for the transition becomes less than the energy saved from reduced power in the idle state.

The prediction heuristic in the menu governor utilizes several sources of information. In principle, the governor knows the next pending timer that will likely cause a wake-up on the CPU. This information, however, may be incomplete or inaccurate. Thus, a correction factor based on the previous prediction accuracy is applied. In addition, not all triggers for wake-ups can be known in advance, e.g., hardware interrupts may occur unexpected. To account for that, the menu governor uses a simple recording of the eight previous sleep durations to detect repeating intervals. From this list, up to two high values can be filtered as outliers. If the variance among the remaining values is still too high, this predictor is ignored. The overall prediction is the minimum of the next timer event and a heuristically detected repeated interval. Further mechanisms in the menu governor attempt to limit the performance impact for I/O intensive workloads and enforce a maximum latency requested by device drivers or the user.

Idle Power Anomalies

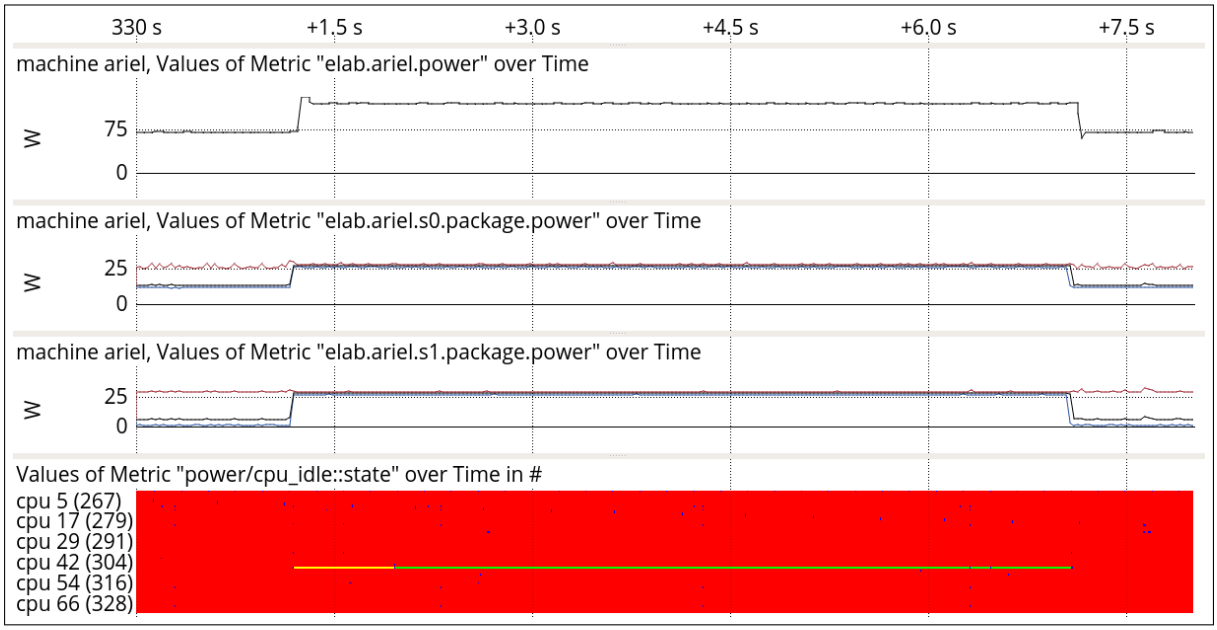
Powernightmares were first observed during energy efficiency research on the instrumented system *diana*. Sporadically, the system power consumption would increase from the normal idle power consumption of ≈ 73 W to above 100 W during phases with no discernible activity. This was confirmed by the fully external measurement to rule out a potential perturbation by the monitoring itself.

In [Ils+18a], I describe the initial investigation on *diana* as well as practical examples from *taurus*. For this thesis, I revisit Powernightmares on the newer *ariel* system using the LMG 670-based power measurements (see Section 3.1). Especially the automatic time synchronization discussed in Section 5.2 provides an accurate insight into the issue without the need for manual time alignment. Figure 5.10 shows the system power trace during a Powernightmare as measured on *ariel*. The trace combines power values from the high-resolution, per-socket power measurements (LMG 670), AC power measurements (LMG 450), as well as events from the operating system captured with `lo2s`(see Section 2.5.1). Scheduling events provide the information on active tasks, while the `power/cpu_idle` tracepoint events reveal which C-state was selected at what time and logical CPU by the operating system. The collection of this information has only minimal impact on the idle system, there are no additional regular wake-ups from measurement. This is achieved by measuring and recording power values externally. Further, the system events are recorded in buffers by the kernel only when the CPU is already active. For [Ils+18a], additional information from CPU residency counters was collected, which implied sporadic activity by the monitoring system. While this information confirmed, that the hardware actually uses the C-states selected by the operating system, it is not essential to illustrate the issue.

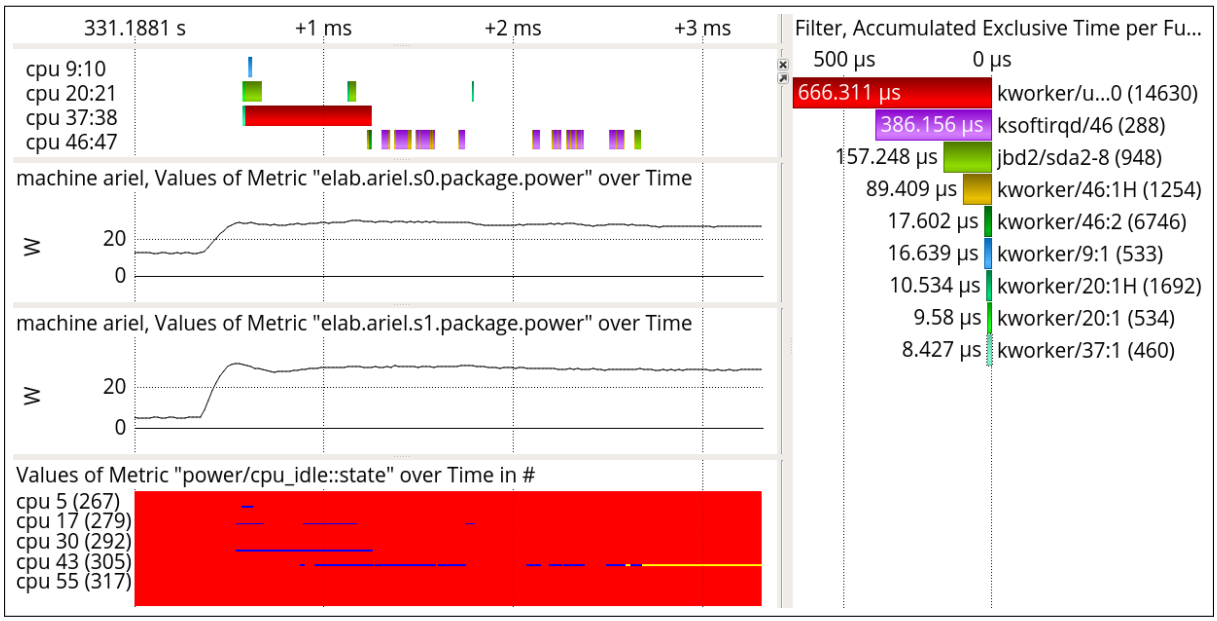
Figure 5.10a reveals that the average power consumption on both sockets and the full system is significantly increased for a duration of ≈ 6 s. The recorded C-state selections reveal the cause: A single hardware thread remains in the C1 and later C1E core C-states for this interval rather than using C6 like all other hardware threads. For the full experiment duration, there is no significant activity scheduled on any of the logical CPUs — there are only rare and short wake-ups, which are normal, even for an idle system. For the depicted long idle phases, anything other than the deepest C-state (C6) is a poor choice for energy efficiency.

Figure 5.10b shows the first few milliseconds of this long high-power phase and reveals the scheduling pattern that leads to this Powernightmare. During a time of only ≈ 2 ms, a set of tasks are repeatedly scheduled on four different logical CPUs. This activity includes multiple `kworker`, `ksoftirqd`, and the `jbd2/data2-8` tasks. The latter is related to the Journaling Block Device (JBD), which provides journaling for the ext4 root file system. It is conceivable that all shown activity is related to journaling, but fully understanding the relationship between all involved processes would require an additional investigation. In general, such a pattern of interaction between different tasks, including kernel threads, is common and was shown to trigger Powernightmares under normal idle conditions. For example, in [Ils+18a, Fig. 2], I analyze similar event sequences related to activity of the parallel file system Lustre.

The reason why this activity pattern ends in an inefficient idle state goes back to the repeated idle duration prediction heuristic of the menu governor. The governor, which runs separately on each logical CPU, observes the repeated scheduling of certain tasks with very short gaps of idle in-between.



(a) The full duration of the *Powernightmare* occurrence. The middle two processor package power charts show maximum/average/minimum in red/black/blue.



(b) The trigger pattern that causes a Powernightmare. Since the chart only shows a 3 ms-interval, only high-resolution per-socket power measurements are used. The top panel shows task scheduling on CPUs with the same color-coding as the right panel, which shows the amount of time these tasks were scheduled during the selected interval.

Figure 5.10: An observation of a Powernightmare with an external power measurement and non-intrusive event tracing. Both Vampir screenshots show the power measurements as well as the selected C-state (indicated by `power/cpu_idle::state`): C6 (red), C1E (green), C1 (yellow), active CPU (dark blue)

From this observation, the governor wrongly predicts the upcoming idle duration to be short again. Thus, a shallow C-state, i.e., C1 or C1E, that is suitable for the short predicted idle duration is chosen. The original discussion in [Ils+18a] was based on dual-socket systems (*diana* and *taurus*) with Intel Haswell processors. For these systems, the impact was clearly measurable, but negligible when considering sustainable average power. I used a synthetic trigger workload to quantify the impact of Powernightmares in a reproducible way. On newer systems, such as the dual-socket Intel Skylake system *diana*, the impact on idle power consumption when running an unmodified Ubuntu installation increased significantly by $\sim 10\,\%$. Due to the high variability over time and many factors that influence an idle system, this number should only serve as a rough classification of magnitude. Independent measurements in the Intel OTC Server Power Lab showed similar impact, and high variability, on multi-socket systems based on Broadwell, Skylake, and Knights Landing processors [WI18].
Paradoxically, the impact of a Powernightmare becomes worse on systems that are tuned for a low idle power consumption by reducing the rate of interrupts. The reason is that a governor is only able to correct the wrong decision at a scheduling point. So less frequent wake-up interrupts also imply longer phases of Powernightmares. However, even waking up after a longer period than expected is not always followed by efficient deep sleep phases. As can be seen in Figure 5.10a, the affected logical CPU actually only switches from C1 to C1E after a short activation — rather than going to the deepest C-state C6 after the fist wake-up. The reason for this delayed change is that the first long idle period can be removed as an outlier in the repeatable-interval-detection heuristic.
The impact of Powernightmares on sustained idle power consumption is characterized by two more factors that explain the increased average power consumption on newer generation systems. On the one hand, the increased number of hardware threads, and thus internal kernel tasks, make it statistical more likely for a Powernightmare to occur. This effect is very difficult to quantify due to the high variability. On the other hand, the impact of package C-states on overall system power consumption has grown significantly from Sandy Bridge, over Haswell, to Skylake (server systems). For example, the power consumption when all cores are in the deepest sleep state, allowing the system to go into a low-power package C-state, decreased from 81 W to 70 W across the three generations. Contrary, the power consumption with just a single core in the shallow C1E increased from 96 W to 122 W [WI18, Part 2, Slide 5].

Preventing Inefficient Sleep States in Practice

In [Ils+18a], my co-authors and I proposed a fallback-timer mechanism to mitigate the impact of Powernightmares on average idle power consumption. Whenever the next known timer event is much farther in the future than the heuristically predicted idle time, a 10 ms-timer is set. If the heuristic was correct, the CPU wakes up before the timer and the timer is canceled leaving only the overhead from setting and canceling the timer. If the heuristic was wrong, the CPU wakes up from the timer and is allowed to enter a deep sleep state, ignoring the current residency history. This way, only a short time is spent in the shallow sleep state, reducing the energy impact from a potential Powernightmare.
To show the effectiveness of this mitigation, I used a synthetic trigger workload. This workload wakes up the processor eight times in 10 µs intervals before sleeping for 10 s. While this workload spends $> 99\,\%$ of time in idle, it increases the power consumption of *diana* from 73 W to 119 W with affected kernels. Figure 5.11 plots a density distribution of power consumption samples collected over 20 min.

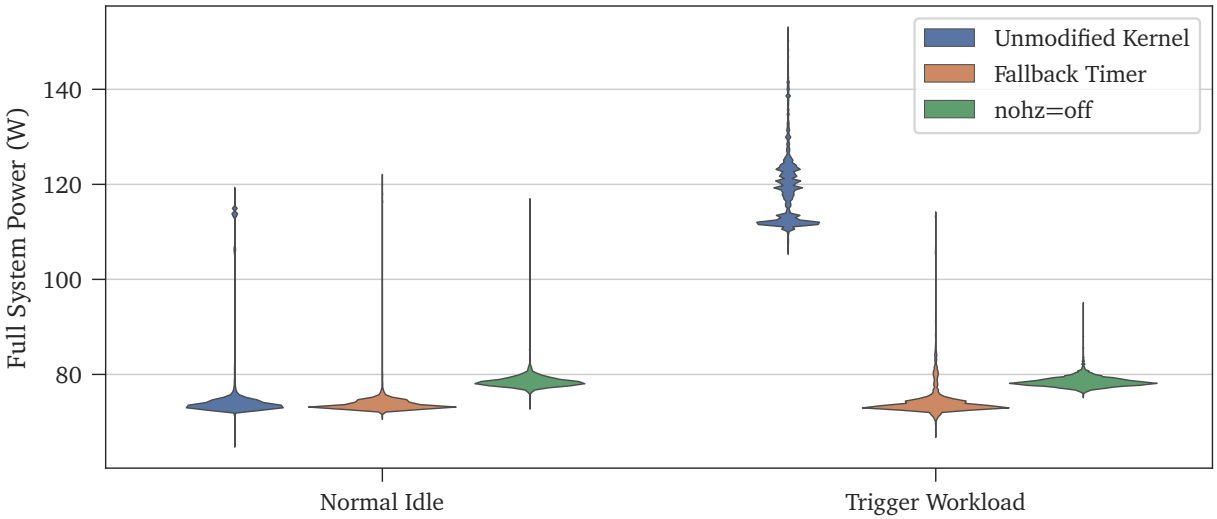


Figure 5.11: A violin-plot of the full-system power consumption of *diana* for different configurations of workloads and mitigation strategies. The unmodified kernel is 4.11.0-rc8 (8b5d11e) (adapted from [Ils+18a]).

In normal idle, with just the default daemons and tasks running, the impact of Powernightmares on the unmodified kernel can be seen as a small spike for power consumptions around 115 W. With the trigger workload, the entire distribution of power samples is shifted. The plot also shows that the fallback timer effectively mitigates the impact of Powernightmares in both normal idle and with the trigger workload.

An alternative mitigation could be the `nohz=off` kernel configuration. In this configuration, the scheduling tick timer is not deactivated during idle phases. Due to this regular timer at 4 ms intervals, Powernightmares have no significant impact. However, as shown in Figure 5.11, the idle power consumption is increased to 78.5 W — for both normal idle and the trigger workload. We discussed other approaches to address the problem in [Ils+18a, Sec. IV-A], but they have distinct disadvantages.

In the context of the original publication, I started a discussion with the Linux kernel developer community. With the increased impact on later hardware generations, the topic received more attention. Eventually, a fix was implemented that combines the ideas of a fallback timer and `nohz=off`: In an interaction with the scheduling subsystem, the existing scheduling tick timer functions as a fallback timer. Instead of always disabling the scheduling tick timer upon entering idle (dyntick-idle mode) or keeping it always enabled (`nohz=off`), it is now is kept enabled *conditionally*. This solution is more intrusive to the software structure than a dedicated fallback timer — it required significant changes across different kernel subsystems. However, the implementation does not only avoid additional timers, it also saves the latency from disabling the scheduling timer when it is not necessary to do so. This can even improve performance on certain I/O intensive workloads that exhibit regular short idle durations.

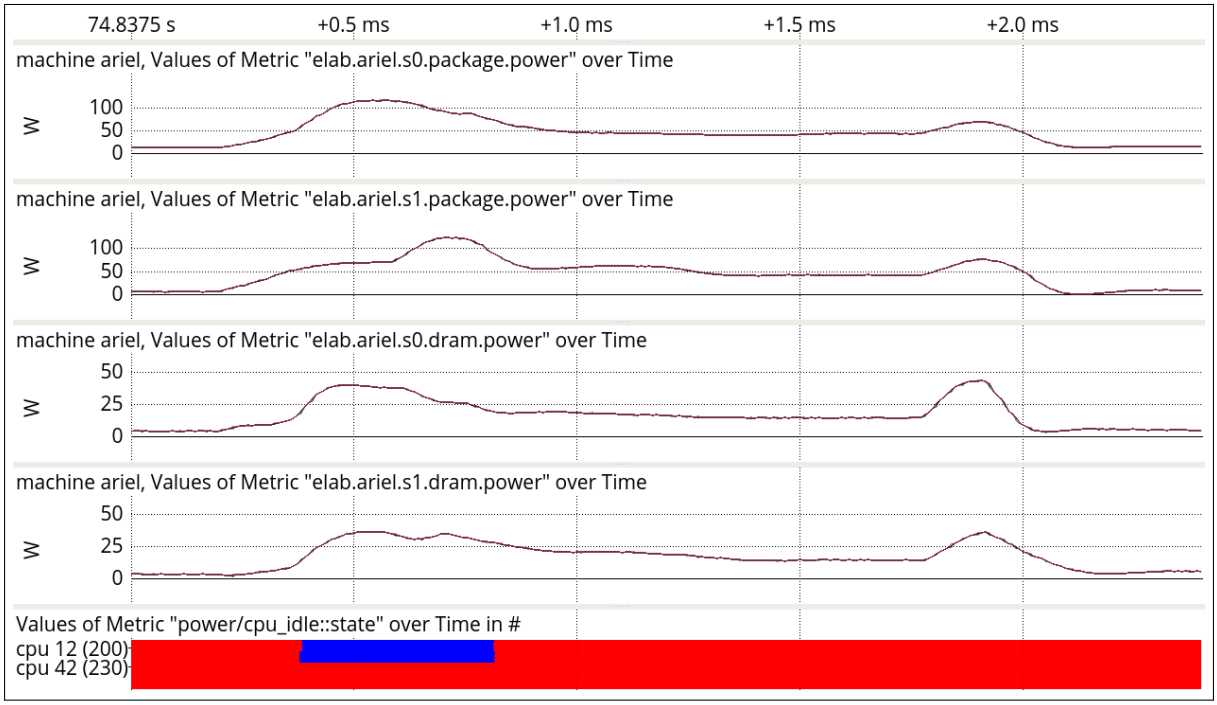
Using the sophisticated measurements on *ariel*, I evaluated several iterations of the patch-set developed by the Intel kernel developer Rafael J. Wysocki. The resulting system power traces repeatedly provided feedback for the development to ensure the patch fulfills its goal without negative side effects[6].

[6]For examples of feedback based on system power traces, see `https://lkml.org/lkml/2018/3/20/238`, `https://lkml.org/lkml/2018/3/21/641`, and `https://lkml.org/lkml/2018/4/10/604`.

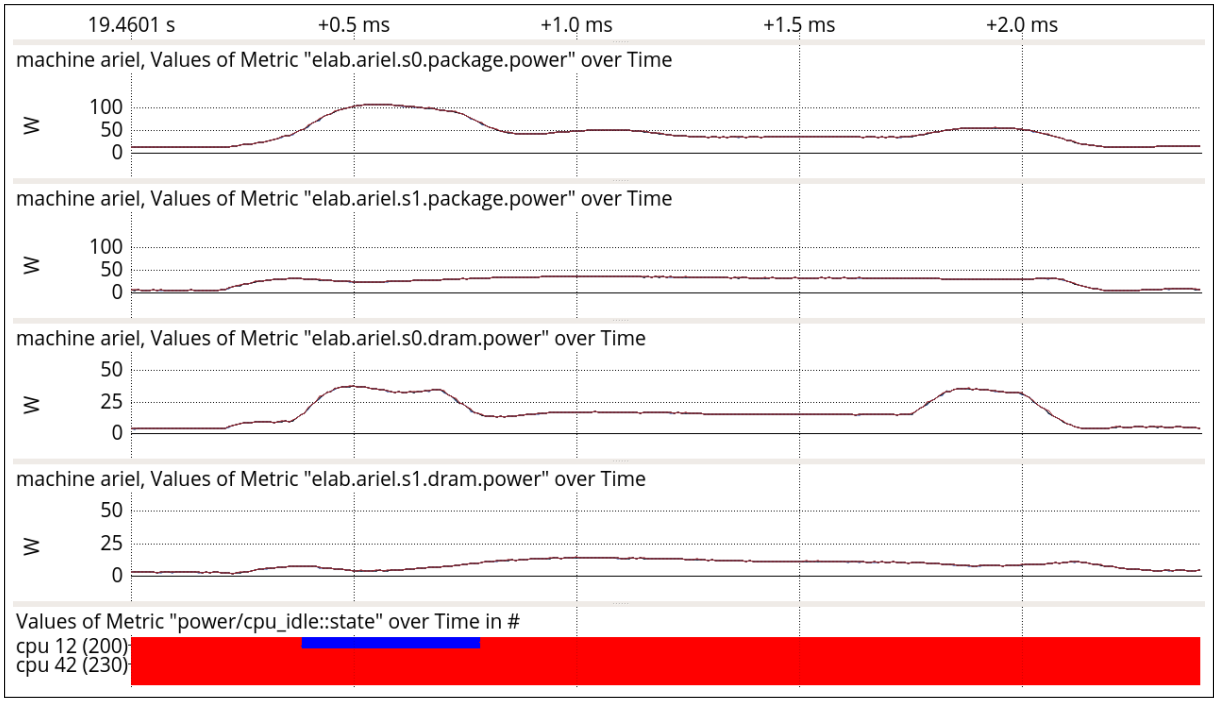
Eventually, patch set was included in the Linux release 4.17. More recently, Wysocki developed a new idle governor for tickless systems — the timer events oriented (TEO) governor [Ryb18]. As the name suggests, this governor focuses on leveraging timer events, but still uses a pattern detection.
The discovery of Powernightmares and the path to a solution highlight the significance of sophisticated energy measurements. Now, as a traceable result of this research, a large number of servers worldwide directly benefit by a reduced energy consumption. This use case also shows that power measurements provide the biggest insight when combined with specific software measurements — in this case specific fine-grained operating system events. This combination yields a unique perception of a complex and impactful interaction between software, the operating system, and hardware. The careful design of the measurement configuration allows an observation without significant perturbation, even of idle systems. Moreover, an accurate and comprehensive energy measurement enables a strong validation of improvements.

5.3.2 Quantifying C-State Transitions

As demonstrated in the previous section, C-states are one of the most important techniques for energy efficiency. In order to exploit them optimally, their characteristics must be well-understood. To model C-states, the power consumption in a state as well as the latency of a transition are of particular interest. A value for the worst-case wake-up latency is exposed to the operating system via ACPI table entries. However, it has been demonstrated, that the values documented in ACPI tables can differ significantly from practical measurements [SMW14; Hac+15; Sch+19]. These measurement results also show that wake-up latencies depend not only on the core C-state, but also package C-state, the location of the core initiating the wake-up, and sometimes also the core frequency. The power consumption within a state can be determined with an low-resolution accurate power measurement. However, the correct measurement domain needs to be considered and the influencing factors (e.g., package C-states) need to be identified and controlled.
While previous publications only evaluate the time that it takes to *leave* an idle state, I described a methodology to quantify the time and energy for *entering* an idle state in [Ils+18b]. Utilizing the high-resolution power measurements of *diana*, it became possible to closely observe the power consumption during a transition to a deep sleep state. Thus, a detailed model for C-states can be built that includes quantitative information of the time it takes before a processor package reaches a stable sleep power. This is a practical definition of *entering* a C-state — it does not necessarily coincide with all internal state changes of the processor.
For this thesis, I revisit the analysis of C-state transitions on the more recent *ariel* system (see Section 3.1). Figure 5.12 visualizes transitions from a deep sleep state (all cores in C6) to active and deep sleep again. This pattern is created by a synthetic workload that iterates over a small computational kernel, performs an OpenMP barrier synchronization, and a sleep phase of 250 ms. The workload is configured to modify a data set of 2 MiB in order to fill the caches (1 MiB L2 and 1.375 MiB L3 per core). During the experiments, the core frequency is set to the nominal value of 3 GHz by using userspace governor. Figure 5.12a shows two distinct peaks for all four attached power domains on processors and DRAM: The first peak correlates with the core activity with a duration of $\approx 440\,\mu s$ as reported by the operating system. The second peak occurs right before the power consumption returns to stable low-power idle for all power domains. Between the peaks is a phase of



(a) All cores of the two processor packages briefly wake up from continuous C6.



(b) All cores of the first processor package (socket 0) briefly wake up from continuous C6.

Figure 5.12: A trace of power consumption during C-state transitions. During the active phase, all active cores modify a 2 MiB data set. The top four panels of each figure show the power consumption per processor package and its associated memory. The bottom panel of each figure shows the active C-state on each hardware thread as requested by the operating system: C6 (red), active CPU (dark blue).

≈ 900 µs of medium power consumption. Notably, the power consumption of each processor package is mostly around 40 W, which is less than the power consumption if all cores are continuously in C1E state[7], the second lowest core C-state available on the system. A conceivable explanation is that that all cores quickly enter the target state C6, but the package enters its PC6 with a certain delay. The peak at the end may indicate a cache flush. Without modifying the memory during the wake-up phase, no such peak can be observed. The trace also contains a recording of C-state residency counters. However, as their collection requires explicit reads, it is done in much more coarse-grained intervals. According to these residency counters, the processor packages spend a noticeable amount of time in package-C2 (≈ 2.3 % of ref-cycles). This could explain a part of the low-power phase during the transition into the C-state, but the quantitative observation is not entirely conclusive.

It is noteworthy, that the Intel idle driver in Linux uses a target residency value of only 600 µs while the measurements show that it takes longer to reach a stable low-power consumption. However, there is no indication that the long time to settle the power consumption has a negative impact on wake-up latency. Due to the relatively low power plateau during the transition, it may still be beneficial for energy efficiency to chose C6 for sleep phases of < 1 ms.

Figure 5.12b shows a similar C-state transition, but with only one processor package being activated. Overall the sequence is similar for the activated package. The power consumption of the package that remains inactive is increased at a relatively stable intermediate level during both the activity and transition of the active package. This increase can be explained by a less efficient package C-state that is required to maintain cache coherence.

An accurate quantitative model would require a more sophisticated analysis with a significant number of samples rather than the visual evaluation of the two example transitions. It is also important to consider the specific conditions of transitions: e.g., which state is entered, what is the state of caches, how many cores are active, and what are the states of the non-participating cores. Further, the core and uncore frequencies do have an impact of the power consumption at the more shallow C-states. Therefore, further investigation should isolate their impact on power consumption during a C-state transition. It is also conceivable that the hardware uses its own prediction to delay when a C-state becomes effective. It is briefly documented that Intel processors may use a “delayed C state algorithm” that can “reject [...] deep sleep states” [Int14a, p. 121]. Thus, for building a quantitative model, the specific use case has to be considered when carefully choosing the factors and conditions in the experiment.

Similar measurements on the Haswell-EP system *diana* showed a transition pattern with a duration of ≈ 230 µs [Ils+18b]. However, there is a difference in the measurement configuration: For the measurements on *diana* one core was constantly active. The results from comparable measurements on the Skylake-SP system *ariel* were less consistent and are subject of future investigation. Thus, the two architectures cannot be directly compared using the presented results.

Investigating C-state transitions shows, how high-resolution power measurements can yield conceptual and quantitative information about processes that are otherwise impossible to observe. As with the previous use case, it is crucial, that the measurement does not impose any perturbation. The collection of operating system events and their accurate temporal correlation lays the foundation for a detailed visual analysis.

[7] In C1E at nominal frequency, the two packages consume ≈ 53 W and ≈58 W respectively

5.3.3 Measuring the Dynamic Power Consumption of HPC Applications

The previous two sections focused on particularly high-resolution node-level measurements in combination with fine-grained system event logging. To demonstrate power measurements in the context of large-scale HPC applications, I used Score-P with the HDEEM metric plugin. Note that HDEEM buffers its high-resolution power measurements within the BMC for the duration of an experiment. Therefore, contrary to the previous two sections, MetricQ was not utilized in the following use case. I first presented and discussed the following application power traces together with my co-authors in [Ils+18c].

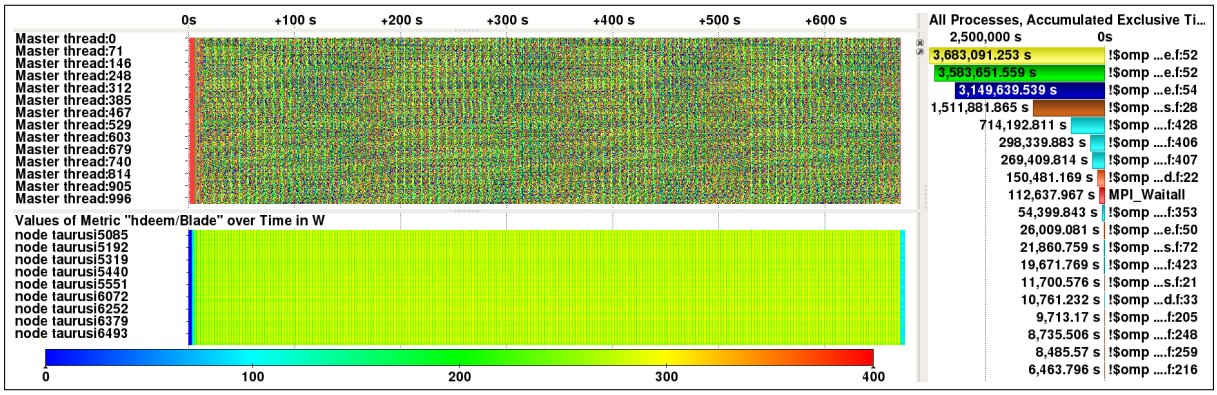
This use-case focuses on the Block tri-diagonal solver (BT) pseudo application from the NAS Parallel Benchmarks (NPB) [VH03]. Specifically, I used the hybrid (multi-zone) MPI / OpenMP implementation in Version 3.3. The traces were created with the problem class F in an execution on 1024 nodes with 24 threads each. Executing this configuration took 677 s and resulted in a trace with a total size of 360 GiB.

Figure 5.13 shows different views of this trace in Vampir. The overall picture of application events and node-level power measurements is shown in Figure 5.13a. Due to the large amount of data and fixed number of pixels, this view provides limited insight. While the function summary reveals where most time is spent overall, the timeline charts show some repeated patterns, but the patterns are too noisy and fine-granular for this presentation.

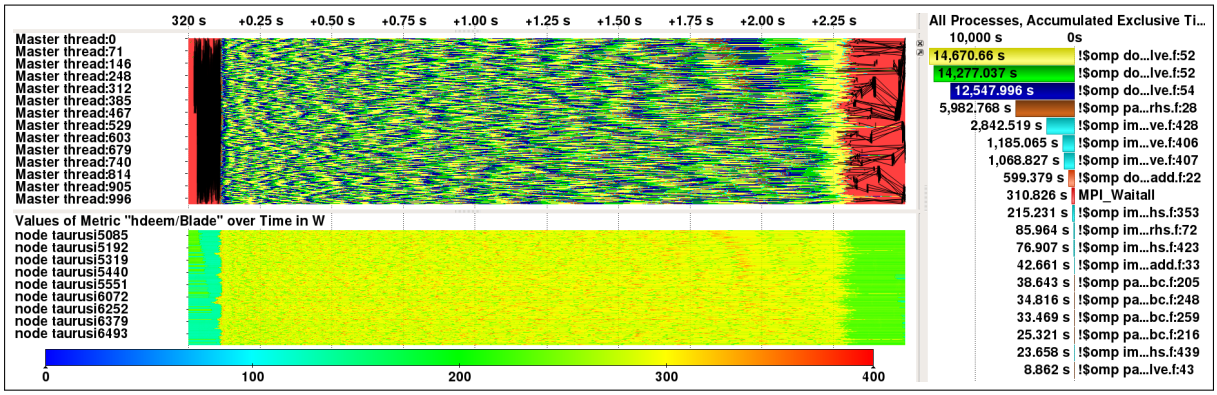
The utility of such large traces, which retain all detailed information, arises from being able to focus on smaller portions of the execution. For example, Figure 5.13b depicts the time interval of one MPI iteration in the application. In this magnification, more patterns within one iteration can be identified. It reveals a correlation between the executed function and the power consumption. Most significantly, the power consumption during the MPI communication phase (red) is lower than during the computation of the solver.

Figure 5.13c looks even closer — at one inner iteration on one node. Further, the power measurement of DRAM channels as well as a recording of L3 cache misses is included. In this view, the characteristics of each function in terms of power consumption can be identified. During the `rhs.f` parallel region (brown), the overall power consumption as well as the DRAM power is at its maximum. This region also exhibits the highest L3 cache misses. Moreover, the region is executed multiple times separated by short synchronizations. However, the specific characteristics of this region differ between the invocations. The three solve functions each exhibit a different power consumption that remains constant within the function execution. Only when some threads complete the computation and enter a synchronization, the power consumption of the compute node decreases.

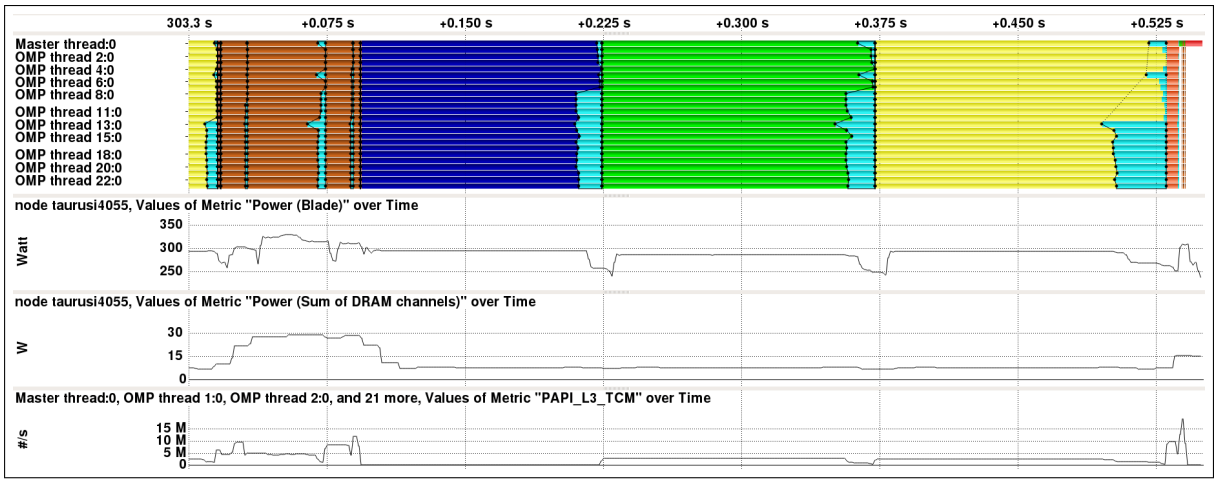
This exploratory visualization confirms that even complex and dynamic relationships between application regions and power consumption, as well as hardware performance counters, can be exposed with large-scale application power traces. Particularly Figure 5.13c shows function-specific details within a time interval of only ≈ 500 ms — emphasizing the need for high-resolution power measurements well beyond 1 Sa/s, even for large-scale applications.



(a) A full view of the full application execution. Top: executed functions, bottom: compute node power consumption heat map, right: accumulated exclusive time spent in different functions.



(b) A visualization of one MPI iteration. Top: executed functions, bottom: compute node power consumption heat map, right: accumulated exclusive time spent in different functions.



(c) A magnification of one inner iteration of one compute node. The charts show (top to bottom) the executed functions on each thread, total compute node power consumption, DRAM power consumption, and sum of L3 cache misses on all application threads.

Figure 5.13: An application power trace of NPB-BT-MZ with HDEEM measurements of 1024 *taurus* nodes. The top panel in each chart shows which thread executes which function at any given time. Dominating functions are `z_solve` (yellow), `y_solve` (green), `x_solve` (dark blue), a parallel region in `rhs.f` (brown), OpenMP synchronization (cyan), and MPI (red) (adapted from [Ils+18c]).

5.4 Conclusion

In this chapter, I described how the sophisticated measurements of Chapter 3 can be leveraged to gain elaborate insight into energy efficiency aspects of computing systems. The infrastructure portrayed in Chapter 4 plays a vital role in making the measurement data accessible for different analysis scenarios. Both live visualization of power measurements and interactive exploration of long-term measurement collections are invaluable for energy-efficient operation of HPC systems. The generation and visualization of application power traces, which combine information from the application execution, system events, and power measurements, are particularly important for energy efficiency research. To overcome the challenge of different clocks on the measuring system and system under test, I devised a novel approach for time-synchronization of power measurements. This approach utilizes the power consumption of the system under test as a hidden channel to transmit a carefully crafted correlation signal and apply crosscorrelation to determine the temporal offset. Its implementation provides automatic time synchronization for application power traces and is highly resilient to noise.

Equipped with those building blocks, I discussed challenging use cases for application power traces: Power anomalies on idle systems are particularly hard to observe, as they are invalidated by intrusive measuring systems. Notwithstanding this challenge, the low-perturbation post-mortem application power traces yielded important insight into the intricate interaction between software, operating system, and hardware. I discussed how this analysis eventually lead to a substantial improvement in the official Linux release — saving 10 % of idle energy on contemporary server systems. Further, I laid out how the high-resolution measurements magnify details of C-state transitions lasting only around a microsecond. Finally, I demonstrated the scalable HDEEM measurements on the basis of a parallel benchmark running on more than a thousand nodes of a petascale HPC system.

Beyond the scope of this thesis, the READEX project uses HDEEM to control and verify automatic energy efficiency tuning [Sch+17b]. Moreover, my co-authors and I used the measurement system of *artemis* to determine the most energy efficient configurations of data processing algorithms in [Göt+14a] and different database queries in [Göt+14b]. The results of this work confirm that the most energy-efficient configuration differs even among similar workloads. This finding emphasizes once more that detailed energy measurements are essential for optimizing energy efficiency.

6 Summary and Outlook

This thesis was motivated by the growing importance of energy consumption for the efficient operation of High Performance Computing systems. Contemporary energy measurements for such systems, however, do not provide the high temporal resolution necessary to fully analyze the dynamic impact of applications and the operating system on power consumption. Moreover, the high data rates of scalable high-resolution energy measurements exceed the capabilities of existing solutions for managing and storing measured values. I addressed these challenges by providing improvements to energy measurement, measurement data processing, and energy efficiency analysis.

High-Resolution Energy Measurement

First, I discussed various measurement techniques, with a focus on improving the temporal resolution beyond state of the art. I described several measurement setups that leverage custom-built sensors and sophisticated power analyzers with sampling rates up to 1 MSa/s. Equipped with such high sampling rates, I analyzed the shortest possible variations in power consumption at different instrumentation points. At the common 12 V power input, workload changes as short as $\approx 140\,\mu s$ can be observed in the power consumption signal with the full amplitude change. I confirmed this visual observation of timeline charts by analyzing the power spectral density of the measurement signal for a novel synthetic binary white noise generator. Measurements at the voltage regulators can resolve even shorter workload changes down to $\approx 12\,\mu s$.

Moreover, I evaluated the accuracy of the presented measurement approaches. Of the custom-build measurements, shunt-based 12 V DC instrumentation showed the best error of less than 1.7 % or 2.3 W absolute. In contrast, measurements using the voltage regulators themselves show significant workload-specific discrepancies of up to 6.3 % or 18 W and even higher in idle configurations.

I further performed a rigorous evaluation of the integrated HPC measuring system HDEEM, which revealed issues with calibration as well as aliasing due to insufficient interfaces. This actionable feedback resulted in improvements in the final version that is deployed in a 1456-node production HPC system.

Additionally, I discussed and evaluated energy counters provided by processors themselves as an alternative to dedicated measurements. While Intel's RAPL initially suffered from significant workload-specific errors, the contemporary implementations provide consistent results. However, it is not feasible to verify the absolute calibration due to differences in the measurement domain. Since RAPL does not provide total power consumption on server systems, it cannot be used for system-level energy efficiency analysis.

Scalable Processing of Measurement Data

The second main contribution of this thesis is the design of a scalable infrastructure for processing high-resolution energy measurement data. In this design, I leverage a message broker to connect decoupled agents that produce and consume measurement data. The design of the infrastructure enables the handling of measurement data at very high sampling rates from a large number of sensors. With a series of benchmarks, I demonstrated that the implementation reaches end-to-end processing rates of more than 500 MSa/s with a single message broker node and 16 instances of data sources and live metric consumers each. This result exceeds the demanding performance requirements arising from the aforementioned high-resolution measuring systems.

The measurement infrastructure also includes an efficient time series database that I explicitly designed for high-resolution measurements. A novel concept behind this database — Hierarchical Timeline Aggregation — facilitates a multiresolution data storage for efficient and accurate queries over arbitrary time intervals. The corresponding implementation leverages an efficient flat-file storage scheme and achieved an insertion rate of 21 MSa/s with a single database instance. I further used benchmarks to demonstrate that the database can answer aggregate timeline queries over 189 billion data points with a maximum end-to-end latency of 30 ms regardless of the requested time interval. Moreover, I described a productive deployment of the measurement infrastructure that is used for a petascale HPC data center

Energy Efficiency Analysis

Finally, I discussed a range of general scenarios for energy efficiency analysis: live dashboards, historic charts, as well as application power traces and their visualization. Moreover, I addressed the challenge of correlating power measurements with application and system traces. At the short timescales of the described high-resolution measuring systems, conventional network clock synchronization is not sufficient. Therefore I devised a novel workflow for timestamp synchronization between out-of-band power measurements and in-band application monitoring. This workflow uses a carefully crafted synthetic workload to send a synchronization signal through the power measurement as a hidden channel and then applies cross correlation to recover the timestamp offset between the measuring system and the system under test. I demonstrated that this approach to synchronization is especially resilient to adverse influences from background activity.

I then presented three practical use cases that leverage the correlation of energy measurements with application/system monitoring. In one of the use cases, I analyzed a power anomaly in the interaction between the operating system and hardware during idle phases. The high resolution of power measurements and precise attribution to operating system events provided the necessary insight to understand the triggers and underlying causes of this anomaly. Ultimately, this discovery led to an improvement in the Linux kernel, which saves $\sim 10\,\%$ of idle energy on server systems. This use case emphasizes how the presented combination of sophisticated energy measurements, a capable processing infrastructure, and analysis techniques can lead to valuable insights and advances in energy efficiency.

Outlook

Energy efficiency will remain an important topic for High Performance Computing systems — potentially with an even stronger link to performance — perpetuating the role of sophisticated energy measurements. While Section 3.5 revealed the limits of temporal granularity when measuring computing power, it also presents an opportunity to precisely tailor measurement solutions with the best possible temporal resolution at a reduced cost. This can help towards ubiquitously available high-quality energy measurements.

The discussed HDEEM solution for scalable energy measurement of HPC systems supports a post-mortem data collection at 1 kSa/s. Continuous data recording with HDEEM is only possible at lower resolutions. Since the results presented in Section 4.3 confirm that it is feasible to process measurement data at higher readout rates at scale, the next step is continuous high-resolution power monitoring for HPC systems. A similar trend can be observed with the DiG HPC power measurements, in which embedded measuring controllers publish measurement data at 1 kSa/s to a message broker, albeit not yet at the highest possible resolution [Bor+18; LBB18].

The described metric processing infrastructure MetricQ performs very well for high-resolution data as it allows efficient handling of multiple data points of one metric in a single message. To further increase the scalability regarding the cardinality, it is also possible to transfer measured values from multiple metrics in one message. This approach, however, breaks the strict one-to-one mapping of metrics to routing keys and therefore requires a formal metric grouping scheme. It is also possible to extend metric grouping to the HTA database to benefit queries referring to groups of metrics. By sharing the timestamps across all metrics of a group, the compression during transfer and storage can be improved, at the cost of more restrictive grouping. Moreover, the HTA implementation can be extended to support additional statistical measures, e.g., the standard deviation.

The use cases discussed in Section 5.3 present many starting points for further energy efficiency analysis. In particular, the C-state characteristics obtained in Section 5.3.2 could be statistically quantified and used to build detailed models. These models can then be used to select more efficient C-states in workloads that frequently alternate between idle and active phases.

By integrating energy measurements as an ingrained part of the performance analysis and optimization workflow, the full potential of scalable energy measurements of HPC applications will be unlocked. The elimination of harmful power anomalies discussed in Section 5.3.1 serves as an example of how improving energy measurements can advance energy efficiency research and ultimately lead to optimizations that reduce the energy consumption of globally deployed computing systems.

A Bibliography

[AA02] Stan W. Amos and Roger S. Amos. *Newnes Dictionary of Electronics*. Elsevier Science, 2002. ISBN: 9780080524054.

[AC16] Michael P. Andersen and David E. Culler. "BTrDB: Optimizing Storage System Design for Timeseries Processing." In: *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 39–52. ISBN: 978-1-931971-28-7. URL: https://www.usenix.org/conference/fast16/technical-sessions/presentation/andersen (visited on Feb. 4, 2020).

[Ach+99] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. "The Aqua Approximate Query Answering System." In: *Proceedings of the 1999 ACM SIGMOD international conference on Management of data - SIGMOD '99*. ACM Press, 1999. DOI: 10.1145/304182.304581.

[Adh+09] Laksono Adhianto, S. Banerjee, Mike W. Fagan, Mark W. Krentel, G. Marin, John Mellor-Crummey, and Nathan R. Tallent. "HPCToolkit: Tools for performance analysis of optimized parallel programs." In: *Concurrency and Computation: Practice and Experience* (2009). DOI: 10.1002/cpe.1553.

[Adv13] Advanced Micro Devices. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*. rev. 3.14. 2013. URL: https://support.amd.com/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf (visited on July 10, 2018).

[Age+14] Anthony Agelastos et al. "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications." In: *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2014. DOI: 10.1109/sc.2014.18.

[Alm+18] Francisco Almeida et al. "Energy Monitoring as an Essential Building Block Towards Sustainable Ultrascale Systems." In: *Sustainable Computing: Informatics and Systems* 17 (2018), pp. 27–42. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2017.10.013.

[Auw+14] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. "A Case Study of Energy Aware Scheduling on SuperMUC." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 394–409. DOI: 10.1007/978-3-319-07518-1_25.

[Bac+08] Klaus Backhaus, Bernd Erichson, Wulff Plinke, and Rolf Weiber. *Multivariate Analysemethoden. Eine anwendungsorientierte Einführung*. (Springer-Lehrbuch) (German Edition). Springer, 2008. ISBN: 978-3-540-85044-1.

[Bat+14] Natalie Bates, Girish Ghatikar, Ghaleb Abdulla, Gregory A. Koenig, Sridutt Bhalachandra, Mehdi Sheikhalishahi, Tapasya Patki, Barry Rountree, and Stephen Poole. "Electrical Grid and Supercomputing Centers: An Investigative Analysis of Emerging Opportunities and Challenges." In: *Informatik-Spektrum* 38.2 (Dec. 2014), pp. 111–127. DOI: 10.1007/s00287-014-0850-0.

[Bau+19] Elizabeth Bautista, Melissa Romanus, Thomas Davis, Cary Whitney, and Theodore Kubaska. *Collecting, Monitoring, and Analyzing Facility and Systems Data at the National Energy Research Scientific Computing Center*. 2019. DOI: 10.1145/3339186.3339213.

[BCD03] Brian Babcock, Surajit Chaudhuri, and Gautam Das. “Dynamic Sample Selection for Approximate Query Processing.” In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM Press, 2003. DOI: 10.1145/872757.872822.

[Bec+10] Daniel Becker, Markus Geimer, Rolf Rabenseifner, and Felix Wolf. “Synchronizing the Timestamps of Concurrent Events in Traces of Hybrid MPI/OpenMP Applications.” In: *2010 IEEE International Conference on Cluster Computing*. IEEE, Sept. 2010. DOI: 10.1109/cluster.2010.13.

[Bed+10] Daniel Bedard, Min Yeol Lim, Robert Fowler, and Allan Porterfield. “PowerMon: Fine-grained and Integrated Power Monitoring for Commodity Computer Systems.” In: *Proceedings of the 2010 IEEE SoutheastCon*. Mar. 2010, pp. 479–484. DOI: 10.1109/SECON.2010.5453824.

[Bel00] Frank Bellosa. “The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems.” In: *Proceedings of the 9th workshop on ACM SIGOPS European workshop beyond the PC: new challenges for the operating system - EW 9*. ACM Press, 2000. DOI: 10.1145/566726.566736.

[Ben+17] Francesco Beneventi, Andrea Bartolini, Carlo Cavazzoni, and Luca Benini. “Continuous Learning of HPC Infrastructure Models using Big Data Analytics and In-Memory processing Tools.” In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, Mar. 2017. DOI: 10.23919/date.2017.7927143.

[Ber+10] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. “Decomposable and Responsive Power Models for Multicore Processors using Performance Counters.” In: *Proceedings of the 24th ACM International Conference on Supercomputing - ICS ’10*. ACM Press, 2010. DOI: 10.1145/1810085.1810108.

[Ber+12] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. “Counter-Based Power Modeling Methods: Top-Down vs. Bottom-Up.” In: *The Computer Journal* 56.2 (Aug. 2012), pp. 198–213. DOI: 10.1093/comjnl/bxs116.

[Bie15] Mario Bielert. “Evaluating Power Estimation Techniques: A Methodological Approach.” Diploma thesis. TU Dresden, 2015.

[BJ07] W. Lloyd Bircher and Lizy K. John. “Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events.” In: *2007 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, Apr. 2007. DOI: 10.1109/ispass.2007.363746.

[Bor+18] Andrea Borghesi, Andrea Bartolini, Antonio Libri, Francesco Beneventi, Daniele Gregori, Simone Tinti, and Piero Altoè. “The D.A.V.I.D.E. Big-Data-Powered Fine-Grain Power and Performance Monitoring Support.” In: May 2018. DOI: 10.1145/3203217.3205863.

[Bra+09] Jim Brandt, Ann Gentile, Jackson Mayo, Philippe Pebay, Diana Roe, David Thompson, and Matthew Wong. “Resource Monitoring and Management with OVIS to Enable HPC in Cloud Computing Environments.” In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, May 2009. DOI: 10.1109/ipdps.2009.5161234.

[Bur+14] Edward A. Burton, Gerhard Schrom, Fabrice Paillet, Jonathan Douglas, William J. Lambert, Kaladhar Radhakrishnan, and Michael J. Hill. “FIVR — Fully integrated voltage regulators on 4th generation Intel Core SoCs.” In: *2014 IEEE Applied Power Electronics Conference and Exposition - APEC 2014*. IEEE, Mar. 2014. DOI: 10.1109/apec.2014.6803344.

[Cha+01] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. “Approximate Query Processing Using Wavelets.” In: *The VLDB Journal* 10.2-3 (Sept. 2001), pp. 199–223. ISSN: 1066-8888. URL: https://dl.acm.org/doi/10.5555/767141.767147.

[Cha+11] Pimwadee Chaovalit, Aryya Gangopadhyay, George Karabatis, and Zhiyuan Chen. “Discrete Wavelet Transform-Based Time Series Analysis and Mining.” In: *ACM Computing Surveys* 43.2 (Jan. 2011), pp. 1–37. DOI: 10.1145/1883612.1883613.

[Cha+17] Mohak Chadha, Thomas Ilsche, Mario Bielert, and Wolfgang E. Nagel. “A Statistical Approach to Power Estimation For x86 Processors.” In: *The 13th Workshop on High-Performance, Power-Aware Computing (HPPAC’17)*. Orlando, FL, USA, 2017. DOI: 10.1109/IPDPSW.2017.98.

[Dav+10] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. “RAPL: Memory Power Estimation and Capping.” In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[Del13] Dell Inc. *PowerEdge R720 and R720xd. Technical Guide*. Version 6.0. 2013. URL: https://i.dell.com/sites/doccontent/shared-content/data-sheets/ja/Documents/Dell-PowerEdge-R720Technical-Guide-2018Jun.pdf (visited on Feb. 4, 2020).

[Dol+10] Manuel F. Dolz, Juan C. Fernández, Rafael Mayo, and Enrique S. Quintana-Ortí. “EnergySaving Cluster Roll: Power Saving System for Clusters.” In: *Architecture of Computing Systems - ARCS 2010*, ed. by Christian Müller-Schloer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 162–173. ISBN: 978-3-642-11950-7. DOI: 10.1007/978-3-642-11950-7_15.

[Dol+15] Manuel F. Dolz, Mohammad Reza Heidari, Michael Kuhn, Thomas Ludwig, and Germán Fabregat. “ArduPower: A Low-cost Wattmeter to improve Energy Efficiency of HPC Applications.” In: *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*. Dec. 2015, pp. 1–8. DOI: 10.1109/IGCC.2015.7393692.

[DPW16] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. “A Validation of DRAM RAPL Power Measurements.” In: *Proceedings of the Second International Symposium on Memory Systems*. MEMSYS ’16. Alexandria, VA, USA: ACM, 2016, pp. 455–470. ISBN: 978-1-4503-4305-3. DOI: 10.1145/2989081.2989088.

[DTM18] DTMF. *Redfish Scalable Platforms Management API Specification*. 2018. URL: https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.6.0.pdf (visited on Jan. 9, 2019).

[Eas+17] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. “Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions.” In: *High Performance Computing*, ed. by Julian M. Kunkel et al. Cham: Springer International Publishing, 2017, pp. 394–412. ISBN: 978-3-319-58667-0. DOI: 10.1007/978-3-319-58667-0_21.

[Eco+06] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. “Full-System Power Analysis and Modeling for Server Environments.” In: *Workshop on Modeling, Benchmarking, and Simulation (MoBS)*. 2006.

[Fay+16] Eyal Fayneh, Marcelo Yuffe, Ernest Knoll, Michael Zelikson, Muhammad Abozaed, Yair Talker, Ziv Shmuely, and Saher Abu Rahme. “14nm 6th-Generation Core Processor SoC with Low Power Consumption and Improved Performance.” In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. Jan. 2016, pp. 72–73. DOI: 10.1109/ISSCC.2016.7417912.

[Fre17] Mike Freedman. *Time-series data: Why (and how) to use a relational database instead of NoSQL*. Apr. 20, 2017. URL: https://blog.timescale.com/blog/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c/ (visited on Nov. 8, 2019).

[Ge+10] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications." In: *IEEE Transactions on Parallel and Distributed Systems* 21.5 (May 2010), pp. 658–671. ISSN: 1045-9219. DOI: 10.1109/TPDS.2009.76.

[GM16] Bhavishya Goel and Sally A. McKee. "A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors." In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2016. DOI: 10.1109/ipdps.2016.118.

[Göt+14a] Sebastian Götz, Thomas Ilsche, Jorge Cardoso, Josef Spillner, Uwe Assmann, Wolfgang E. Nagel, and Alexander Schill. "Energy-Efficient Data Processing at Sweet Spot Frequencies." In: *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*. Springer Berlin Heidelberg, 2014, pp. 154–171. DOI: 10.1007/978-3-662-45550-0_18.

[Göt+14b] Sebastian Götz, Thomas Ilsche, Jorge Cardoso, Josef Spillner, Thomas Kissinger, Uwe Assmann, Wolfgang Lehner, Wolfgang E. Nagel, and Alexander Schill. "Energy-Efficient Databases Using Sweet Spot Frequencies." In: *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, Dec. 2014. DOI: 10.1109/ucc.2014.142.

[Gra+17] Ryan E. Grant, James H. Laros, Michael Levenhagen, Stephen L. Olivier, Kevin Pedretti, Lee Ward, and Andrew J. Younge. "Evaluating Energy and Power Profiling Techniques for HPC Workloads." In: *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. Oct. 2017, pp. 1–8. DOI: 10.1109/IGCC.2017.8323587.

[GSS15] Corey Gough, Ian Steiner, and Winston Saunders. *Energy Efficient Servers*. Apress, Apr. 7, 2015. ISBN: 1430266384.

[Hac+13a] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E. Nagel. "Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison." In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2013. DOI: 10.1109/ispass.2013.6557170.

[Hac+13b] Daniel Hackenberg, Roland Oldenburg, Daniel Molka, and Robert Schöne. "Introducing FIRESTARTER: A Processor Stress Test Utility." In: *2013 International Green Computing Conference Proceedings*. IEEE, June 2013. DOI: 10.1109/igcc.2013.6604507.

[Hac+14] Daniel Hackenberg, Thomas Ilsche, Joseph Schuchart, Robert Schöne, Wolfgang E. Nagel, Marc Simon, and Yiannis Georgiou. "HDEEM: High Definition Energy Efficiency Monitoring." In: *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*. E2SC '14. IEEE, Nov. 2014. DOI: 10.1109/e2sc.2014.13.

[Hac+15] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. "An Energy Efficiency Feature Survey of the Intel Haswell Processor." In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, May 2015. DOI: 10.1109/ipdpsw.2015.70.

[Häh+12] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. "Measuring Energy Consumption for Short Code Paths using RAPL." In: *SIGMETRICS Performance Evaluation Review* 40 (Jan. 2012), pp. 13–17. ISSN: 0163-5999. DOI: 10.1145/2425248.2425252.

[Haj+16] Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. "Fine-Grain Power Breakdown of Modern Out-of-Order Cores and Its Implications on Skylake-Based Systems." In: *ACM Transactions on Architecture and Code Optimization* 13.4 (Dec. 2016), pp. 1–25. DOI: 10.1145/3018112.

[Haj+18] Jawad Haj-Yahya, Avi Mendelson, Yosi Ben Asher, and Anupam Chattopadhyay. *Energy Efficient High Performance Processors*. Springer Singapore, 2018. DOI: 10.1007/978-981-10-8554-3.

[Haj+19] Jawad Haj-Yahya, Efraim Rotem, Avi Mendelson, and Anupam Chattopadhyay. “A Comprehensive Evaluation of Power Delivery Schemes for Modern Microprocessors.” In: *20th International Symposium on Quality Electronic Design, ISQED 2019, Santa Clara, CA, USA*. 2019, pp. 123–130. DOI: 10.1109/ISQED.2019.8697544.

[Har+14] Alistair Hart, Harvey Richardson, Jens Doleschal, Thomas Ilsche, Mario Bielert, and Matthew Kappel. “User-level Power Monitoring and Application Performance on Cray XC30 Supercomputers.” In: *Cray User Group 2014*. May 2014. URL: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap136.pdf (visited on Feb. 4, 2020).

[HEK09] Joachim Hartung, Bärbel Elpelt, and Karl-Heinz Klösener. *Statistik*. Gruyter, de Oldenbourg, June 3, 2009. ISBN: 3486590286.

[HKK17] Vlasta Hajek, Tomas Klapka, and Ivan Kudibal. *Benchmarking InfluxDB vs. Elasticsearch for Time Series Data, Metrics & Management*. 2017. URL: https://www.influxdata.com/blog/influxdb-markedly-elasticsearch-in-time-series-data-metrics-benchmark/ (visited on Feb. 23, 2020).

[HP11] Chung-Hsing Hsu and Stephen W. Poole. “Power Measurement for High Performance Computing: State of the Art.” In: *2011 International Green Computing Conference and Workshops*. July 2011, pp. 1–6. DOI: 10.1109/IGCC.2011.6008596.

[IBM18] IBM. *OCC Firmware Interface Specification for POWER9. Version 0.22*. Ed. by Martha Broyles. June 27, 2018. URL: https://github.com/open-power/docs/raw/d53d84c/occ/OCC_P9_FW_Interfaces.pdf (visited on Feb. 4, 2020).

[Ils+15a] Thomas Ilsche, Daniel Hackenberg, Stefan Graul, Joseph Schuchart, and Robert Schöne. “Power Measurements for Compute Nodes: Improving Sampling Rates, Granularity and Accuracy.” In: *2015 Sixth International Green Computing Conference and Sustainable Computing Conference (IGSC)*. Dec. 2015. DOI: 10.1109/IGCC.2015.7393710.

[Ils+15b] Thomas Ilsche, Joseph Schuchart, Robert Schöne, and Daniel Hackenberg. “Combining Instrumentation and Sampling for Trace-Based Application Performance Analysis.” In: *Tools for High Performance Computing 2014*. Springer International Publishing, 2015, pp. 123–136. DOI: 10.1007/978-3-319-16012-2_6.

[Ils+17] Thomas Ilsche, Robert Schöne, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. “lo2s — Multi-core System and Application Performance Analysis for Linux.” In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2017, pp. 801–804. DOI: 10.1109/CLUSTER.2017.116.

[Ils+18a] Thomas Ilsche, Marcus Hähnel, Robert Schöne, Mario Bielert, and Daniel Hackenberg. “Powernightmares: The Challenge of Efficiently Using Sleep States on Multi-core Systems.” In: *Euro-Par 2017: Parallel Processing Workshops*, ed. by Dora B. Heras et al. Cham: Springer International Publishing, 2018, pp. 623–635. ISBN: 978-3-319-75178-8. DOI: 10.1007/978-3-319-75178-8_50.

[Ils+18b] Thomas Ilsche, Robert Schöne, Philipp Joram, Mario Bielert, and Andreas Gocht. “System Monitoring with lo2s: Power and Runtime Impact of C-State Transitions.” In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2018, pp. 712–715. DOI: 10.1109/IPDPSW.2018.00114.

[Ils+18c] Thomas Ilsche, Robert Schöne, Joseph Schuchart, Daniel Hackenberg, Marc Simon, Yiannis Georgiou, and Wolfgang E. Nagel. “Power Measurement Techniques for Energy-Efficient Computing: Reconciling Scalability, Resolution, and Accuracy.” In: *SICS Software-Intensive Cyber-Physical Systems*. Apr. 2018. DOI: 10.1007/s00450-018-0392-9.

[Ils+19] Thomas Ilsche, Daniel Hackenberg, Robert Schöne, Mario Bielert, Franz Höpfner, and Wolfgang E. Nagel. “MetricQ: A Scalable Infrastructure for Processing High-Resolution Time Series Data.” In: *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*. 2019, pp. 7–12. DOI: 10.1109/DAAC49578.2019.00007.

[Ils09] Thomas Ilsche. “Analyse und Anwendung globaler und lokaler Datenquellen für Programmspuren.” Diploma thesis. TU Dresden, Dec. 2009.

[Inf19a] InfluxData Inc. *InfluxDB 1.7 documentation*. 2019. URL: https://docs.influxdata.com/influxdb/v1.7/ (visited on Nov. 7, 2019).

[Inf19b] InfluxData Inc. *InfluxDB 1.X: Open Source Time Series Platform*. 2019. URL: https://www.influxdata.com/time-series-platform/ (visited on Feb. 4, 2020).

[Int] International Rectifier. *IR3529 DATA SHEET, XPHASE3™ PHASE IC*. URL: https://www.infineon.com/dgdl/ir3529m.pdf?fileId=5546d462533600a4015355cd5b781758 (visited on Dec. 17, 2018).

[Int+13] Intel Corporation, Hewlett-Packard, NEC, and Dell. *IPMI Specification*. v. 2.0 rev. 1.1. Tech. rep. 2013. URL: https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html (visited on Feb. 11, 2020).

[Int06] International Bureau of Weights and Measures (BIPM). *The International System of Units (SI)*. 2006. URL: https://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf (visited on Feb. 11, 2020).

[Int13] Intel Corporation. *Power Supply. Design Guide for Desktop Platform Form Factors*. rev. 1.31. 2013. URL: https://www.intel.com/content/dam/www/public/us/en/documents/guides/power-supply-design-guide.pdf (visited on Jan. 11, 2019).

[Int14a] Intel Corporation. *Intel Xeon Processor E5 v2 and E7 v2 Product Families Uncore Performance Monitoring Reference Manual*. Reference Number: 329468-002. Intel. 2014. URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/xeon-e5-2600-v2-uncore-manual.pdf.

[Int14b] International Organization for Standardization. *Advanced Message Queuing Protocol (AMQP) v1.0 specification. ISO/IEC 19464:2014*. Standard. Geneva, CH, 2014. URL: https://www.iso.org/standard/64955.html (visited on Feb. 4, 2020).

[Int15] Intel Corporation. *Intel® Server Board S2600CP Family Intel® Server System P4000CP Family. Technical Product Specification*. rev. 1.9. May 2015. URL: https://www.intel.com/content/dam/support/us/en/documents/motherboards/server/s2600cp/sb/g26942005_s2600cp_p4000cp_tps_rev19.pdf (visited on Feb. 4, 2020).

[Int17a] Intel Corporation. *Intel Xeon Processor E5 v3 Product Family - Processor Specification Update*. Sept. 2017. URL: https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-v3-spec-update.pdf (visited on Feb. 11, 2020).

[Int17b] Intel Corporation. *Intel Xeon Processor Scalable Family Specification Update*. Aug. 2017. URL: https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-scalable-spec-update.pdf (visited on July 9, 2018).

[Int18a] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer’s Manual. Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4*. May 2018. URL: https://software.intel.com/en-us/articles/intel-sdm (visited on July 9, 2018).

[Int18b] Intel Corporation. *Product brief Intel® SSD DC P4610 Series*. 2018. URL: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/dc-p4610-series-brief.pdf (visited on Apr. 10, 2019).

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Apr. 30, 1991. 720 pp. ISBN: 0471503363.

[JCG08] JCGM - Joint Committee for Guides in Metrology. *Evaluation of measurement data — Guide to the expression of uncertainty in measurement*. 1st ed. JCGM 100:2008 (GUM 1995 with minor corrections). 2008. URL: https://www.bipm.org/en/publications/guides/gum.html (visited on Feb. 11, 2020).

[JCG09] JCGM - Joint Committee for Guides in Metrology. *Evaluation of measurement data — An introduction to the "Guide to the expression of uncertainty in measurement" and related documents*. 1st ed. JCGM 100:2008 (GUM 1995 with minor corrections). 2009. URL: https://www.bipm.org/en/publications/guides/gum.html (visited on Feb. 11, 2020).

[JCG12] JCGM - Joint Committee for Guides in Metrology. *International vocabulary of metrology - Basic and general concepts and associated terms (VIM)*. 3rd ed. JCGM 200:2012 (JCGM 200:2008 with minor corrections). 2012. URL: https://jcgm.bipm.org/vim/en/index.html (visited on Feb. 4, 2020).

[Jia+10] Y. Jiao, H. Lin, P. Balaji, and W. Feng. "Power and Performance Characterization of Computational Kernels on the GPU." In: *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. Dec. 2010, pp. 221–228. DOI: 10.1109/GreenCom-CPSCom.2010.143.

[JPT17] Soren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. "Time Series Management Systems: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 29.11 (Nov. 2017), pp. 2581–2600. DOI: 10.1109/tkde.2017.2740932.

[Juc12] Guido Juckeland. "Trace-based Performance Analysis for Hardware Accelerators." PhD thesis. TU Dresden, 2012.

[KHL18] Thomas Kissinger, Dirk Habich, and Wolfgang Lehner. "Adaptive Energy-Control for In-Memory Database Systems." In: *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*. ACM Press, 2018. DOI: 10.1145/3183713.3183756.

[KHN12] Michael Kluge, Daniel Hackenberg, and Wolfgang E. Nagel. "Collecting Distributed Performance Data with Dataheap: Generating and Exploiting a Holistic System View." In: *Procedia Computer Science* 9 (2012). Proceedings of the International Conference on Computational Science, ICCS 2012, pp. 1969–1978. ISSN: 1877-0509. DOI: 10.1016/j.procs.2012.04.215.

[Knü+08] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. "The Vampir Performance Analysis Tool-Set." In: *Tools for High Performance Computing*, ed. by Michael Resch et al. Springer Berlin Heidelberg, July 2008, pp. 139–155. ISBN: 978-3-540-68561-6. DOI: 10.1007/978-3-540-68564-7_9.

[Knü+12] Andreas Knüpfer et al. "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir." In: *Tools for High Performance Computing 2011*, ed. by Holger Brunst et al. Springer Berlin Heidelberg, 2012, pp. 79–91. ISBN: 978-3-642-31476-6. DOI: 10.1007/978-3-642-31476-6_7.

[Kog+08] Peter Kogge et al. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, ed. by Peter Kogge. 2008.

[Law06] Averill M. Law. *Simulation Modeling and Analysis*. McGraw Hill Higher Education, 2006. ISBN: 978-007-125519-6.

[LBB18] Antonio Libri, Andrea Bartolini, and Luca Benini. "Dwarf in a Giant: Enabling Scalable, High-Resolution HPC Energy Monitoring for Real-Time Profiling and Analytics." In: *ArXiv e-prints* (June 2018). arXiv: 1806.02698 [cs.DC].

[Ler12] Reinhard Lerch. *Elektrische Messtechnik. Analoge, digitale und computergestützte Verfahren*. (Springer-Lehrbuch) (German Edition). Springer, 2012. ISBN: 978-3-642-22608-3.

[Li+09] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures." In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*. ACM Press, 2009. DOI: 10.1145/1669112.1669172.

[Lib+16] Antonio Libri, Andrea Bartolini, Michele Magno, and Luca Benini. "Evaluation of Synchronization Protocols for fine-grain HPC sensor data time-stamping and collection." In: *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, July 2016. DOI: 10.1109/hpcsim.2016.7568419.

[Lib+18a] Antonio Libri, Andrea Bartolini, Francesco Beneventi, Andrea Borghesi, and Luca Benini. "MULTITHERMAN: Out-of-band High-Resolution HPC Power and Performance Monitoring Support for Big-Data Analysis." In: *Workshop on Energy Efficiency Tools for HPC (EETHPC)*. 2018. URL: http://eethpc.net/wp-content/uploads/2018/05/ISC2018_EETHPC_AL.pdf (visited on Feb. 4, 2020).

[Lib+18b] Antonio Libri, Andrea Bartolini, Daniele Cesarini, and Luca Benini. "Evaluation of NTP/PTP Fine-Grain Synchronization Performance in HPC Clusters." In: *Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems - ANDARE '18*. ACM Press, 2018. DOI: 10.1145/3295816.3295819.

[Lin] Linear Technology. *Single Resistor Gain Programmable, Precision Instrumentation Amplifier*. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/1167fc.pdf (visited on Feb. 4, 2020).

[LPD13] James H. Laros, Phil Pokorny, and David DeBonis. "PowerInsight — A Commodity Power Measurement Capability." In: *2013 International Green Computing Conference Proceedings*. IEEE, June 2013. DOI: 10.1109/igcc.2013.6604485.

[Lük92] Hans D. Lüke. *Korrelationssignale. Korrelationsfolgen und Korrelationsarrays in Nachrichten- und Informationstechnik, Meßtechnik und Optik*. (German Edition). Springer, 1992. ISBN: 3-540-54579-4.

[LVE16] Aleix Llusà Serra, Sebastià Vila-Marta, and Teresa Escobet Canal. "Formalism for a Multiresolution Time Series Database Model." In: *Inf. Syst.* 56.C (Mar. 2016), pp. 19–35. ISSN: 0306-4379. DOI: 10.1016/j.is.2015.08.006.

[Mai+18] Matthias Maiterth, Gregory Koenig, Kevin Pedretti, Siddhartha Jana, Natalie Bates, Andrea Borghesi, Dave Montoya, Andrea Bartolini, and Milos Puzovic. "Energy and Power Aware Job Scheduling and Resource Management: Global Survey — Initial Analysis." In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2018, pp. 685–693. DOI: 10.1109/IPDPSW.2018.00111.

[Mal11] Roman Malaric. *Instrumentation and Measurement in Electrical Engineering*. Brown Walker Press, Apr. 20, 2011. 244 pp. ISBN: 1612335004.

[Mar+17] Steven J. Martin, David Rush, Matthew Kappel, Michael Sandstedt, and Joshua Williams. "Cray XC40 Power Monitoring and Control for Knights Landing." In: *Cray User Group*. 2017. URL: https://cug.org/proceedings/cug2016_proceedings/includes/files/pap112s2-file1.pdf (visited on Feb. 4, 2020).

[McK] Paul E. McKenney. *NO_HZ: Reducing Scheduling-Clock Ticks*. URL: https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt (visited on Feb. 4, 2020).

[Meg] Megware. *ClustSafe Energiemanagement und Stromverteilung mit voller Kontrolle*. URL: https://www.megware.com/fileadmin/user_upload/PDFs/ClustSafe.pdf (visited on Jan. 10, 2019).

[Mel18] Mellanox Technologies. *ConnectX®-5 VPI Card. 100Gb/s InfiniBand & Ethernet Adapter Card*. 2018. URL: https://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_VPI_Card.pdf (visited on Feb. 11, 2020).

[Mil06] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, 2006. ISBN: 9781420006155.

[MiT17] MiTAC International Corporation. *Tyan S7106*. Version 1.0. 2017. URL: ftp://ftp.tyan.com/doc/S7106_UG_v1.0a.pdf (visited on Feb. 4, 2020).

[MK14] Steven J. Martin and Matthew Kappel. "Cray XC30 Power Monitoring and Management." In: *Cray User Group*. 2014. URL: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap130.pdf (visited on Feb. 4, 2020).

[Mol+10] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S. Müller. "Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86-64 Processors." In: *International Conference on Green Computing*. IEEE, Aug. 2010. DOI: 10.1109/greencomp.2010.5598316.

[Mor+10] Alan Morris, Allen D. Malony, Sameer Shende, and Kevin Huck. "Design and Implementation of a Hybrid Parallel Performance Measurement System." In: *2010 39th International Conference on Parallel Processing*. IEEE, Sept. 2010. DOI: 10.1109/icpp.2010.57.

[MRK15] Steven J. Martin, David Rush, and Matthew Kappel. "Cray Advanced Platform Monitoring and Control (CAPMC)." In: *Cray User Group*. 2015. URL: https://cug.org/proceedings/cug2015_proceedings/includes/files/pap132.pdf (visited on Feb. 4, 2020).

[Nat15] National Instruments. *NI 6122/6123 Specifications*. May 2015. URL: https://www.ni.com/pdf/manuals/371396b.pdf (visited on Feb. 4, 2020).

[Nat16a] National Instruments. *DAQ M Series. M Series User Manual*. NI 622x, NI 625x, and NI 628x Multifunction I/O Modules and Devices. July 2016. URL: https://www.ni.com/pdf/manuals/371022l.pdf (visited on Feb. 11, 2020).

[Nat16b] National Instruments. *Device Specifications NI 6255. M Series Data Acquisition: 80 AI, 1.25 MS/s, 24 DIO, 2 AO*. June 2016. URL: https://www.ni.com/pdf/manuals/375215c.pdf (visited on Feb. 11, 2020).

[Net+19] Alessio Netti, Micha Müller, Axel Auweter, Carla Guillen, Michael Ott, Daniele Tafani, and Martin Schulz. "From Facility to Application Sensor Data: Modular, Continuous and Holistic Monitoring with DCDB." In: SC '19 (2019). DOI: 10.1145/3295500.3356191.

[Nie94] Jakob Nielsen. *Usability Engineering*. Elsevier LTD, Oxford, Nov. 1, 1994. Chap. 5.5. 362 pp. ISBN: 0125184069. URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (visited on Apr. 4, 2019).

[Nor98] Harry Smith Norman R. Draper. *Applied Regression Analysis*. JOHN WILEY & SONS INC, May 13, 1998. 736 pp. ISBN: 0471170828.

[Oet17] Tobias Oetiker. *RRDtool logging & graphing*. 2017. URL: https://oss.oetiker.ch/rrdtool/ (visited on Apr. 9, 2019).

[ON 15] ON Semiconductor. *Single-Phase Voltage Regulator with SVID Interface for Computing Applications*. 2015. URL: https://www.onsemi.com/pub/Collateral/NCP81251-D.PDF (visited on Feb. 11, 2020).

[ONe+96] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. "The log-structured merge-tree (LSM-tree)." In: *Acta Informatica* 33.4 (June 1996), pp. 351–385. ISSN: 1432-0525. DOI: 10.1007/s002360050048.

[ONe14] Melissa E. O'Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College, Sept. 2014. URL: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf (visited on Feb. 4, 2020).

[Pac07] Sasha Pachev. *Understanding MySQL Internals: Discovering and Improving a Great Database*. OREILLY MEDIA, Apr. 11, 2007. 234 pp. ISBN: 0596009577.

[Par10] Rainer Parthier. *Messtechnik: Grundlagen und Anwendungen der elektrischen Messtechnik für alle technischen Fachrichtungen und Wirtschaftsingenieure*. Vieweg + Teubner, 2010. ISBN: 978-3-8348-0811-0.

[Pat+13] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. "Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing." In: *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS '13*. ACM Press, 2013. DOI: 10.1145/2464996.2465009.

[PC11] Vikas Ashok Patil and Vipin Chaudhary. "Rack Aware Scheduling in HPC Data Centers: An Energy Conservation Strategy." In: *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, May 2011. DOI: 10.1109/ipdps.2011.227.

[Pel+15] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. "Gorilla: A Fast, Scalable, In-Memory Time Series Database." In: vol. 8. 12. VLDB Endowment, Aug. 2015, pp. 1816–1827. DOI: 10.14778/2824032.2824078.

[Per+15] Kasun S. Perera, Martin Hahmann, Wolfgang Lehner, Torben Bach Pedersen, and Christian Thomsen. "Modeling Large Time Series for Efficient Approximate Query Processing." In: *Database Systems for Advanced Applications*, ed. by An Liu et al. Cham: Springer International Publishing, 2015, pp. 190–204. ISBN: 978-3-319-22324-7. DOI: 10.1007/978-3-319-22324-7_16.

[Pil+95] Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. "Paraver: A Tool to Visualize and Analyze Parallel Code." In: *Proceedings of WoTUG-18: Transputer and occam Developments*. Vol. 44. 1. 1995, pp. 17–31.

[Piv19] Pivotal Software, Inc. *RabbitMQ Documentation. for the current release, 3.7.14*. 2019. URL: https://www.rabbitmq.com/documentation.html (visited on Apr. 1, 2019).

[PLB07] Venkatesh Pallipadi, Shaohua Li, and Adam Belay. "cpuidle: Do nothing, efficiently." In: *Proceedings of the Ottawa Linux Symposium (OLS)*. 2007. URL: https://www.kernel.org/doc/ols/2007/ols2007v2-pages-119-126.pdf (visited on Feb. 4, 2020).

[Pur+18] Avi Purkayastha, Steven Hammond, Ramkumar Nagappan, and Max Alt. "Holistic Approaches to HPC Power and Workflow Management." In: *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*. IEEE. Sept. 2018, pp. 1–8. DOI: 10.1109/IGCC.2018.8752150.

[Rot+12] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge.” In: *IEEE Micro* 32.2 (Mar. 2012), pp. 20–27. ISSN: 0272-1732. DOI: 10.1109/MM.2012.12.

[RV18] John W. Romein and Bram Veenboer. “PowerSensor 2: a Fast Power Measurement Tool.” In: *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2018. DOI: 10.1109/ISPASS.2018.00020.

[RWT+19] RWTH Aachen University, Gesellschaft für numerische Simulation mbH, Technische Universität Dresden, University of Oregon, Forschungszentrum Jülich GmbH, German Research School for Simulation Sciences GmbH, Technische Universität München, and Technische Universität Darmstadt. *Score-P Scalable performance measurement infrastructure for parallel codes*. 6.0 (rev. 14673). July 31, 2019. URL: http://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-6.0/pdf/scorep.pdf (visited on Dec. 18, 2019).

[Rya08] Thomas P. Ryan. *Modern Regression Methods*. John Wiley & Sons, Inc, Nov. 25, 2008. 642 pp. ISBN: 0470081864.

[Ryb18] Marta Rybczyńska. “Improving idle behavior in tickless systems.” In: *LWN* (2018). URL: https://lwn.net/Articles/775618/ (visited on July 1, 2019).

[Sav+15] Pavel Saviankou, Michael Knobloch, Anke Visser, and Bernd Mohr. “Cube v4: From Performance Report Explorer to Performance Analysis Tool.” In: *Procedia Computer Science* 51 (2015), pp. 1343–1352. DOI: 10.1016/j.procs.2015.05.320.

[Sch+11] Robert Schöne, Ronny Tschüter, Thomas Ilsche, and Daniel Hackenberg. “The VampirTrace Plugin Counter Interface: Introduction and Examples.” In: *Euro-Par 2010 Parallel Processing Workshops*. Springer Berlin Heidelberg, 2011, pp. 501–511. DOI: 10.1007/978-3-642-21878-1_62.

[Sch+16a] Robert Schöne, Thomas Ilsche, Mario Bielert, Daniel Molka, and Daniel Hackenberg. “Software Controlled Clock Modulation for Energy Efficiency Optimization on Intel Processors.” In: *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*. E2SC ’16. Salt Lake City, Utah: IEEE, 2016, pp. 69–76. ISBN: 978-1-5090-3856-5. DOI: 10.1109/e2sc.2016.015.

[Sch+16b] Joseph Schuchart, Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Ramkumar Nagappan, and Michael K. Patterson. “The Shift From Processor Power Consumption to Performance Variations: Fundamental Implications at Scale.” In: *Computer Science - Research and Development* 31.4 (Aug. 2016), pp. 197–205. DOI: 10.1007/s00450-016-0327-2.

[Sch+17a] Robert Schöne, Ronny Tschüter, Thomas Ilsche, Joseph Schuchart, Daniel Hackenberg, and Wolfgang E. Nagel. “Extending the Functionality of Score-P Through Plugins: Interfaces and Use Cases.” In: *Proceedings of the 10th International Workshop on Parallel Tools for High Performance Computing, October 2016, Stuttgart, Germany*, ed. by Christoph Niethammer et al. Cham: Springer International Publishing, 2017, pp. 59–82. ISBN: 978-3-319-56702-0. DOI: 10.1007/978-3-319-56702-0_4.

[Sch+17b] Joseph Schuchart et al. “The READEX Formalism for Automatic Tuning for Energy Efficiency.” In: *Computing* 99.8 (Jan. 2017), pp. 727–745. DOI: 10.1007/s00607-016-0532-7.

[Sch+19] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. *Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance*. accepted for publication. 2019. arXiv: 1905.12468 [cs.DC].

[Sch19] SchedMD. *Slurm Power Saving Guide*. Nov. 11, 2019. URL: https://slurm.schedmd.com/power_save.html (visited on Dec. 9, 2019).

[Ser+12] Harald Servat, Germán Llort, Judit Giménez, Kevin Huck, and Jesús Labarta. "Folding: Detailed Analysis with Coarse Sampling." In: *Tools for High Performance Computing 2011*. Springer Berlin Heidelberg, 2012, pp. 105–118. DOI: 10.1007/978-3-642-31476-6_9.

[SHM12] Robert Schöne, Daniel Hackenberg, and Daniel Molka. "Memory Performance at Reduced CPU Clock Speeds: An Analysis of Current x86_64 Processors." In: *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*. HotPower'12. Hollywood, CA: USENIX Association, 2012, p. 9.

[SM06] Sameer S. Shende and Allen D. Malony. "The Tau Parallel Performance System." In: *The International Journal of High Performance Computing Applications* 20.2 (May 2006), pp. 287–311. DOI: 10.1177/1094342006064482.

[SMW14] Robert Schöne, Daniel Molka, and Michael Werner. "Wake-up Latencies for Processor Idle States on Current x86 Processors." In: *Computer Science - Research and Development* (2014). DOI: 10.1007/s00450-014-0270-z.

[Sol05] Guang Gong Solomon W. Golomb. *Signal Design for Good Correlation*. Cambridge University Press, 2005. 458 pp. ISBN: 0521821045.

[sol19] solid IT. *DB-Engines Ranking of Time Series DBMS*. Nov. 4, 2019. URL: https://web.archive.org/web/20191107090350/https://db-engines.com/en/ranking/time+series+dbms (visited on Nov. 7, 2019).

[Sys15] System Management Interface Forum, Inc. *PMBus Power System Management Protocol Specification. Part II – Command Language*. rev. 1.3.1. 2015. URL: http://pmbus.org/Specifications/CurrentSpecifications (visited on Oct. 1, 2019).

[Tal+11] Nathan R. Tallent, John Mellor-Crummey, Michael Franco, Reed Landrum, and Laksono Adhianto. "Scalable Fine-grained Call Path Tracing." In: *Proceedings of the international conference on Supercomputing - ICS '11*. ACM Press, 2011. DOI: 10.1145/1995896.1995908.

[Tim19] Timescale, Inc. *TimescaleDB Docs*. 2019. URL: https://docs.timescale.com (visited on Feb. 5, 2020).

[Uni16] United EFI, Inc. *Advanced Configuration and Power Interface (ACPI) specification*. rev. 6.1. 2016. URL: https://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf (visited on Feb. 11, 2020).

[Vaz+17] Sudharshan S. Vazhkudai, Ross Miller, Devesh Tiwari, Christopher Zimmer, Feiyi Wang, Sarp Oral, Raghul Gunasekaran, and Deryl Steinert. "GUIDE: A Scalable Information Directory Service to Collect, Federate, and Analyze Logs for Operational Insights into a Leadership HPC Facility." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17*. ACM Press, 2017. DOI: 10.1145/3126908.3126946.

[Ver15] Vernier. *Watts Up Pro*. 2015. URL: https://www.vernier.com/files/manuals/wu-pro.pdf (visited on June 7, 2018).

[VH03] Rob F. Van der Wijngaart and Jin Haopiang. *NAS Parallel Benchmarks, Multi-Zone Versions*. NAS-03-010. Tech. rep. NAS, July 2003. URL: https://www.nas.nasa.gov/assets/pdf/techreports/2003/nas-03-010.pdf (visited on Feb. 26, 2020).

[Vir+20] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* (2020). DOI: 10.1038/s41592-019-0686-2.

[Wal+17] Matthew J. Walker, Stephan Diestelhorst, Andreas Hansson, A. K. Das, S. Yang, Bashir M. Al-Hashimi, and Geoff V. Merrett. “Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.1 (Jan. 2017), pp. 106–119. ISSN: 0278-0070. DOI: 10.1109/TCAD.2016.2562920.

[Wea18] Vince Weaver. *Linux support for Power Measurement Interfaces*. 2018. URL: http://web.eece.maine.edu/~vweaver/projects/rapl/rapl_support.html (visited on July 9, 2018).

[Web04] John G. Webster. *Electrical Measurement, Signal Processing, and Display*. CRC Press, 2004. ISBN: 0-8493-1733-9.

[Wel67] Peter D. Welch. “The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms.” In: *IEEE Transactions on Audio and Electroacoustics* 15.2 (June 1967), pp. 70–73. ISSN: 0018-9278. DOI: 10.1109/TAU.1967.1161901.

[WH11] Neil H. E. Weste and David M. Harris. *CMOS VLSI Design — A Circuits and Systems Perspective*. 4th ed. Pearson, 2011.

[WI18] Rafael J. Wysocki and Thomas Ilsche. “CPU Idle Loop Ordering Problem.” In: *Power Management and Scheduling in the Linux Kernel (OSPM Summit) II edition*. https://retis.sssup.it/luca/ospm-summit/2018/Downloads/idle_loop_rework.pdf (part 1), https://retis.sssup.it/luca/ospm-summit/2018/Downloads/2018_OSPM_powernightmares.pdf (part 2) (visited on Feb. 11, 2020). 2018.

[Yok16] Yokogawa Electric Corporation. *WT1800E Series High performance Power Analyzers*. WT1800E-01EN. Yokogawa Electric Corporation. 2016. URL: https://cdn.tmi.yokogawa.com/BUWT1800E-01EN.pdf (visited on Feb. 5, 2020).

[Yok17] Yokogawa Electric Corporation. *WT1801E, WT1802E, WT1803E, WT1804E, WT1805E, WT1806E Precision Power Analyzer User’s Manual*. IM WT1801E-02EN. 3rd ed. Yokogawa Electric Corporation. 2017. URL: https://cdn.tmi.yokogawa.com/IMWT1801E-02EN.pdf (visited on Feb. 5, 2020).

[ZES] ZES ZIMMER Electronic Systems GmbH. *4-Channel Power Meter LMG450*. ZES ZIMMER Electronic Systems GmbH. URL: https://www.zes.com/en/content/download/286/2473/file/lmg450_prospekt_1002_e.pdf (visited on Feb. 5, 2020).

[ZES15] ZES ZIMMER Electronic Systems GmbH. *DualPath explained - Why one A/D converter per channel is not (always) sufficient*. ZES ZIMMER Electronic Systems GmbH. May 2015. URL: https://www.zes.com/en/content/download/1115/13063/file/DualPath%20explained.pdf (visited on Feb. 5, 2020).

[ZES16] ZES ZIMMER Electronic Systems GmbH. *Instrument Family LMG600 — 1 to 7 phase precision power analyzer - User Manual*. V1.030 R33655. ZES ZIMMER Electronic Systems GmbH. Feb. 2016.

B Abbreviations

AC alternating current.

ACPI Advanced Configuration and Power Interface.

ADC analog-to-digital-converter.

AMQP Advanced Message Queuing Protocol.

API application programming interface.

AQP approximate query processing.

BMC baseboard management controller.

CMOS complementary metal–oxide–semiconductor.

COW copy-on-write.

CSV comma separated value.

DC direct current.

DiG Dwarf in a Giant.

DIMM dual in-line memory module.

DRAM dynamic random access memory.

DVFS dynamic voltage and frequency scaling.

FFT fast Fourier transform.

FPGA field-programmable gate array.

GPIO general-purpose input/output.

GPU graphics processing unit.

HDEEM High Definition Energy Efficiency Monitoring.

HPC High Performance Computing.

HTA Hierarchical Timeline Aggregation.

IPMI Intelligent Platform Management Interface.

IVR fully integrated voltage regulator.

JSON JavaScript Object Notation.

LSM-tree Log-Structured Merge Tree.

MIC Many Integrated Core.

MPI message passing interface.

MQTT Message Queuing Telemetry Transport.

MSR model specific register.

NTP network time protocol.

NVMe non-volatile memory express.

OS operating system.

OTF2 Open Trace Format 2.

PAPI Performance application programming interface.

PCG permuted congruential generator.

PCI Peripheral Component Interconnect.

PCIe PCI Express.

PDU power distribution unit.

PHC PTP Hardware Clock.

PSD power spectral density.

PSU power supply unit.

PTP precise time protocol.

PUE power usage effectiveness.

PXI PCI eXtensions for Instrumentation.

RAPL Running Average Power Limiting.

RDBMS relational database management system.

REST representational state transfer.

RPC remote procedure call.

SMT simultaneous multithreading.

SUT system under test.

TDP thermal design power.

TLS Transport Layer Security.

TSI Time Series Index.

TSM Time-Structured Merge Tree.

UUID universally unique identifier.

VIF variance of inflation.

VR voltage regulator.

WAL write ahead log.

C Glossary

compute node The set of components that run one operating system (OS) instance and share a memory address space. A compunte node can be part of a larger parallel computing system or an individual smaller system.

core An independant processing unit with dedicated resources for exeuting instructions.

hardware thread A processing unit that shares some resources of a core with other hardware threads. On systems without simultaneous multithreading, one hardware thread corresponds to one core.

logical CPU An independent processing unit as seen by the operating system. One logical CPU corresponds to one hardware thread.

measurand The quantity that is intended to be measured [JCG12].

metric Identifies a specific measurand that is repeatedly measured.

processor package A single physical piece containing the processor cores and other related components. A number of DRAM memory DIMMs are associated with each processor package, but not contained on it.

readout rate The rate at which measuremt values for one measurand are available for further analysis.

readout value The measured value that is available to the user or analysis as opposed to measured values that are used internally in a layered measuring system.

Slurm The Slurm Workload Manager is a job scheduler used by HPC systems, including *taurus*.

socket The component on a mainboard hosting a processor package. Socket and processor package are often used synonymously.

uncertainty In general discussion: An estimation of the upper limit of the absolute value of the total error. In the context of a formal uncertainty evaluation, *uncertainty* refers to a characterization of the dispersion of a measured value [JCG12].

D List of Software Contributions

During the creation of this thesis, I developed several open source software packages. I am the lead developer for the following packages with support and contributions of my colleagues:

Name	License	Referenced in
MetricQ	BSD-3-Clause	Section 4.2.2, Section 4.2.3
C++ and Python library for MetricQ as well as protocol definitions and RPC abstractions.		`https://github.com/metricq/metricq`
MetricQ management agent	GPL-3.0	Section 4.2
Python management agent implementation to orchestrate the MetricQ instance.		`https://github.com/metricq/metricq-manager`
MetricQ aggregator	GPL-3.0	Section 4.3.7
Agent to aggregate high-resolution measurement data for storage in the database.		`https://github.com/metricq/metricq-aggregator`
MetricQ Grafana endpoint	GPL-3.0	Section 5.1.2
HTTP(S) endpoint for visualizing historic metric charts with Grafana.		`https://github.com/metricq/metricq-grafana`
MetricQ WebSocket endpoint	GPL-3.0	Section 5.1.1
WebSocket endpoint for live visualization; The included JavaScript client library was developed by Mario Bielert.		`https://github.com/metricq/metricq-sink-websocket`
HTA library	BSD-3-Clause	Section 4.2.5
C++ library to access data in a file-based Hierarchical Timeline Aggregation (HTA) time series database.		`https://github.com/metricq/hta`
MetricQ HTA DB agent	GPL-3.0	Section 4.2.5
C++ agent on top of the HTA library to provide persistent storage for MetricQ.		`https://github.com/metricq/metricq-db-hta`
`lo2s`	GPL-3.0	Section 2.5, Section 5.3.1, Section 5.3.2
Linux OTF2 Sampling — a lightweight node-level monitoring software.		`https://github.com/tud-zih-energy/lo2s`
Score-P MetricQ plugin	BSD-3-Clause	Section 5.1.3, Section 5.3.1, Section 5.3.2
Metric plugin for integrating measurement data from MetricQ in Score-P and `lo2s`*.* *Includes the robust automatic time synchronization based on signal processing.*		`https://github.com/score-p/scorep_plugin_metricq`
Score-P HDEEM plugin	BSD-3-Clause	Section 5.1.3, Section 5.3.1
Metric plugin for integrating measurement data from HDEEM in Score-P and `lo2s`*.*		`https://github.com/score-p/scorep_plugin_hdeem`
Linux fallback-timer	GPL-3.0	Section 5.3.1
Proof-of-concept implementation of a fallback timer in the Linux menu idle governor to prevent Powernightmares.		`https://github.com/tud-zih-energy/linux/tree/menu_idle_fallback_timer`
roco2	GPL-3.0	Section 3.6
Main development by Mario Bielert [Bie15]; Enhanced for the RAPL evaluation on ariel*.*		`https://github.com/tud-zih-energy/roco2/tree/tilsche-diss`

Acknowledgments

At the end of this journey, I want to express my deepest gratitude for the generous support that I received throughout the making of this thesis.
First and foremost, my thanks go to my doctoral advisor Prof. Dr. Wolfgang E. Nagel, who supported me throughout my academic career and always showed patience with this thesis. I thank Prof. Dr. Wolfgang Lehner for sharing his expertise as additional supervisor.
The invaluable critical feedback from Dr. Andreas Knüpfer, Mario Bielert, and Dr. Robert Schöne has been instrumental for this work — thank you. My colleagues from the energy efficiency research group and the ZIH are the reason for the productive and enjoyable working environment. In particular, I appreciate the prolific discussions with Andreas Gocht on electrical engineering. Moreover, I thank Daniel Hackenberg, who always has my back and Prof. Dr. Florina M. Ciorba for stimulating the beginnings of this thesis. I am very grateful for the collaboration with Robin Geyer and Joseph Schuchart on the HDEEM acceptance test measurements, particularly during my parental leave. I also thank my exceptional student assistants Franz Höpfner and Philipp Joram for contributing to the implementation of MetricQ as well as Christian von Elm for his support with the development of `lo2s`.
At TU Dresden, I have the pleasure of excellent interdisciplinary support. My thanks go to Stefan Graul for the implementation of the measuring systems of artemis and diana. I thank Dr. Christian Scheunert and Axel Schmidt, who shared their knowledge of electrical engineering and signal processing in particular. I also thank Dr. Christoph Lehmann and Dr. Taras Lazariv for their input on statistical evaluations and mathematical modeling. Moreover, I thank Dr. Claudio Hartmann for his insight into time series databases. Being part of the collaborative research team Highly Adaptive Energy-efficient Computing (HAEC) has been a unique and stimulating experience. I thank the German Research Foundation (DFG) for funding this CRC 912.
Finally, my deepest gratitude goes to my wonderful family — especially my mother, who nurtured my passion for science, my wife, who found the right balance of patience and motivation, and my daughter — stay curious.