Appears in: Proceedings of the X Brazilian Symposium on Computing Systems Engineering (SBESC 2020)
virtual, November 2020

EnergyBudgets: Integrating Physical Energy Measurement Devices into Systems Software

Luis Gerhorst[1], Stefan Reif[1], Benedict Herzog[1], Timo Hönig[2]

[1] Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany
[2] Ruhr University Bochum (RUB), Germany
`{gerhorst,reif,herzog}@cs.fau.de, timo.hoenig@rub.de`

***Abstract*—Excessive energy consumption is a critical problem for mobile computing systems due to their limited battery capacity. Software developers aim to improve energy efficiency by monitoring and profiling the energy consumption of their systems in order to discover and resolve energy hot-spots. However, energy measurement is often tedious since it involves a hardware setup as well as software integration. To support accurate but also convenient energy measurements, we propose the inclusion of external energy measurement devices into existing performance profiling subsystems. This approach allows the energy-consumption analysis of applications that run on the system under test (SUT) using the same tools as used for other performance metrics.**

To enable low-overhead self-monitoring, we propose a modular analysis approach, *EnergyBudgets*, which bridges external energy measurement hardware to the Linux `perf` subsystem. The evaluation of our implementation shows that energy budgets accurately measure the energy consumed by different workloads and allow for an overhead-reduction on the SUT by $20\,\%$ to $51\,\%$ in comparison to regular timers, while still guaranteeing the same level of precision.

***Index Terms*—energy aware systems, tools, performance evaluation, resource management, design methodologies**

I. Introduction

Modern operating systems (OSs) offer a variety of interfaces dedicated to measuring and quantifying resource demand related to time. For example, processors include components usable to identify the current point in time, and dedicated hardware devices (i.e., timers) that are programmed to notify the processor—usually based on interrupts—when a specified amount of time has elapsed. Since time measurement is provided by the hardware, most operating systems can offer corresponding standardised measurement subsystems. The Portable Operating System Interface (POSIX), for example, defines a variety of interfaces that are related to time, but so far no interface related to energy [1].

However, energy is an equally critical resource to many embedded computer systems. This applies for battery-powered devices [2], [3] as well as energy-harvesting [4], [5] systems. Nevertheless, only few hardware platforms offer a built-in way to monitor the energy consumed by the system. In consequence, no portable and generic built-in facilities are provided by OSs to manage this resource.

As an exceptional case of built-in energy measurements, recent Intel processors offer the the Running Average Power Limit (RAPL) interface, which allows convenient monitoring of the system's energy consumption with the help of hardware counters [6]. Tools such as the Linux kernel's `perf` [7] read the RAPL counters to enable userspace code to determine how much energy an application consumes. The RAPL interface, however, is only available on recent x86 processors [8]. On most embedded platforms, in contrast, energy measurements are usually not integrated into the processor, and therefore not available in `perf`. Furthermore, even on Intel processors that do support RAPL, external devices are required to monitor the whole-system energy consumption, including peripherals. As a consequence of this need for external energy measurement devices, it is common for developers and researchers to apply custom hardware and software efforts as dominant measurement setup [9], [10], [11], [12], [13], [14], [15] in order to achieve accurate physical energy analysis. This approach of measuring energy separately from the co-existing performance profiling infrastructure, however, leads to results that are hard to reproduce and vulnerable to bugs, causing measurement errors and, consequently, misguided system optimisations.

To reduce the efforts related to the use of external energy measurement devices, we propose to access them indirectly, via system-level tools for performance monitoring and profiling. This approach allows for easy-to-use, reproducible, accurate, and low-overhead physical energy measurement. In this paper, we introduce EnergyBudgets, a generic protocol which enables OSs to implement a variety of services employing external energy measurement devices. We have integrated an implementation of our protocol into the Linux kernel's `perf_event` subsystem. Our work allows measuring an application's physical energy consumption with an external device, and accessing results via the standard `perf` utility.

The contributions of this paper are three-fold. First, we present the concept and design of EnergyBudgets which solves the overhead, scalability, and synchronisation challenges of industry-standard physical measurement devices. Second, we discuss the current prototype implementation of EnergyBudgets and describe its integration into the Linux operating system kernel's performance analysis framework, making energy consumption *just another performance metric*. Third, we evaluate EnergyBudgets on the v4.19 Linux kernel and validate its functional and non-functional characteristics (i.e., overhead) with an ARM platform.

The paper is structured as follows, Section II reviews related work on systems power management and energy measurements. The design of EnergyBudgets is introduced in Section III and our `perf_event`-based implementation on an embedded system is documented in Section IV. Section V evaluates the accuracy and precision of EnergyBudgets, also comparing them to regular timer-based approaches. Section VI discusses implementation challenges, followed by Section VII which explores opportunities for future work. Finally, Section VIII concludes this paper.

II. Related Work

In this section, we present and discuss research and background knowledge related to the approach of EnergyBudgets.

A. Operating System Power Management

Le Sueur and Heiser demonstrated that generic processor features like dynamic voltage and frequency scaling (DVFS) as well as C states (sleep modes) can save power [16], but also that the power savings drawn from DVFS are limited [17]. This pushes more of the responsibility to make systems energy efficient towards application developers, who have to profile their applications to identify bottlenecks. This however, is only possible if the OS provides them with the appropriate tooling for this task.

Weissel and Bellosa [18] use performance counters to estimate the optimal clock frequency for an application, trading in minor performance losses for substantial power savings. Although RAPL was not available in 2002, they already acknowledge that performance counters designed for energy profiling would benefit their approach. In 2020, these are still not ubiquitous, suggesting that external energy measurement devices will continue to be used in the future.

B. Energy Measurement and Accounting

Flinn and Satyanarayanan developed PowerScope, a tool which allows consumed energy to be attributed to individual processes and procedures [10] using time-based statistical sampling of the power consumption. EnergyBudgets in comparison, allow for energy-based statistical sampling [9].

To date, various energy monitoring devices and tools have been presented in the literature [14], [13], [19]. Jiang et al. designed sensor nodes that use a separate energy counter to monitor their own energy consumption [12]. All of the referenced monitoring devices use microcontrollers in between the analog-to-digital converter (ADC) and the system that stores the measurements. This shows that the hardware required for EnergyBudgets is already widespread and a potential user only has to flash the adapted EnergyBudgets firmware onto their microcontroller.

Pathak et al. develop an energy accounting policy to map the system energy consumption back to program entities [20]. EnergyBudgets can replace their energy model and be combined with their accounting technique to allow for fine-grained energy profiling of application code. In their implementation, they model the component energy consumption based on finite state machines (FSMs) whose power state transitions are system-call driven. However, modeling the energy consumption based on the device power state only works if the events that trigger transitions are infrequent and traceable. This is not possible on all modern hardware components. On recent multi-core processors, DVFS is no longer performed using discrete power states (P-States) which are selected by the OS, but instead by choosing the desired performance level on an abstract continuous scale (defined by the ACPI Collaborative Processor Performance Control specification [21] and implemented, for example, by AMD processors with the Zen 2 microarchitecture). The selection of the performance level can also be performed automatically by the processor without being triggered by the OS. This makes FSMs unsuitable to model processor power consumption because of a) the large number of possible power states and b) the untraceability of the frequent automatic transitions performed by the processor.

Power Sandbox, a more recent approach to accounting that isolates applications in their vertical hardware and software stack, assumes that the operating system has an interface to monitor per-component power consumption with low-overhead [22]. Our work provides this interface.

C. Energy Management Abstractions

Zeng et al. proposed the *Currentcy* model which allows an OS to manage energy as a first-class resource [23], [24]. For their implementation however, the absence of a generic interface for energy consumption forces them to rely on approximations which are sufficiently accurate but very system specific. A generic interface would allow their approach to be applied in a broad variety of systems.

Roy et al. have identified energy *reserves* and *taps* as useful low-level abstractions that allow control of application energy usage by the OS [25]. Our tool complements their work by giving the application a convenient method to determine when the energy budget of a time frame, being determined by the respective *tap* that feeds it and the *reserve* that was collected previously, is about to expire. Before the budget has been used up, the application may adapt its behavior when notified, or, after the budget has been exhausted, the OS may preempt the application on the EnergyBudget interrupt.

III. Design

This section discusses challenges in integrating external measurement devices into computing systems and concludes our design of EnergyBudgets which solves these problems. In order to allow software developers to easily monitor the energy usage their applications cause in systems, multiple requirements have to be met:

§A The user interface to access the energy consumption must be independent of a specific measurement device and its physical measurement method (e.g., shunts, hall effect sensors, battery charge monitoring). Such an interface is *generic* in the sense that various measurement methodologies with, for example, different temporal resolutions, can be applied and transparently accessed.

§B The user interface must allow an easy correlation of energy consumption with other performance metrics. This property is crucial to support energy profiling, such as the detection of energy hot-spots.

§C The overheads and interferences caused on the system under test (SUT) must be low. Especially on low-power embedded devices, the processor has little performance available, and most communication buses have a relatively slow data rate.

§D Events regarding energy must be synchronized with other performance-related events to enable correlation analysis. In consequence, communication between the energy-measurement infrastructure and the SUT must occur frequent, but with little overhead and noise. A typical approach based on signals via general-purpose input/output (GPIO) [11] is tedious in practice as information is transmitted bit-wise.

§E The interface must be integrated into the OS to support accounting of energy usage to individual applications [26], [27]. Besides, the OS integration promises to improve accuracy as it minimises overheads related to communication and event handling [28].

§F To improve the accuracy of accounting solutions implemented on top of the interface, the energy consumption from individual hardware components should be reported separately when the measurement hardware supports it.

Existing solutions to energy measurements do not meet these requirements. On battery-powered devices, the system can poll the remaining battery capacity, for example using ACPI [21] and derive the energy and power consumption from it. However, this approach is not generic (c.f., §A), and the battery monitoring operates at a low temporal resolution only reporting the accumulated energy consumption by all system components (c.f., §F). In consequence, this approach is usually not accurate enough for fine-grained power measurements [26]. In consequence, even devices that have built-in battery-capacity estimation are usually monitored using external measurement devices during development, which are attached while prototyping and removed in production [29].

A physical measurement device typically determines the power consumption by intercepting the power supply of the host system. It digitizes the current power draw measurement. Some measurement devices support the integration of power samples over time to yield the energy demand. A naïve approach is to connect the measurement device directly to the host system. By using a direct communication channel, the digital channel from the measurement device to the host system may, for example, continuously stream power samples to the host system, so that both power and timing-related information is available there. This approach, however, leads to continuous overhead (c.f., §C, §D, and §E) on the host system as it either has to integrate over power samples in real-time, or store all power values, but memory is a scarce resource in embedded systems. As the rate at which power samples are generated is determined by the measurement device, the host system has very limited control over the amount of data processing required. To avoid the processing overhead on the host system, a dedicated system can be used to store the samples temporarily [10]. However, this approach requires complicated synchronization of the two systems to correlate software events with power profiles [11] (c.f., §B).

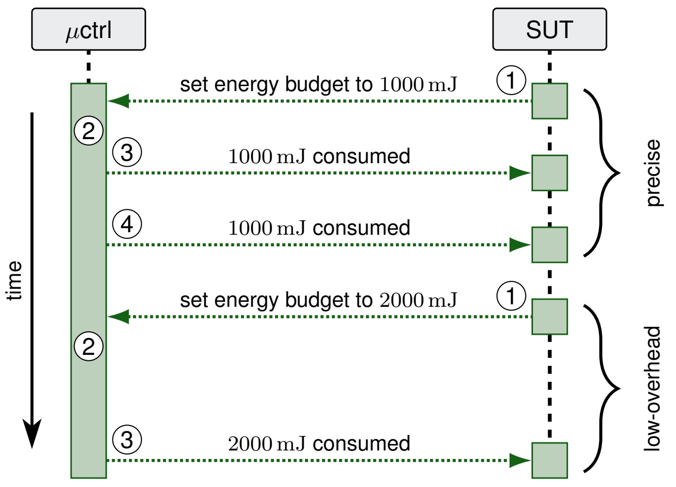

Fig. 1. Sequence diagram showing the packets sent between the measurement device and the host system with EnergyBudgets.

To summarize, energy measurement hardware is very specific to a particular use case, considering that the maximum power draw, the measurement resolution, as well as the sampling rate have to be dimensioned appropriately for the host system. In consequence, monitoring interfaces are tailored to specific use-cases. They are not standardized and their integration therefore creates additional friction when monitoring energy consumption.

To enable low-overhead energy monitoring, we propose the EnergyBudgets protocol. A high-level summary is as follows:

① The host system instructs the measurement device to notify it whenever a certain amount of energy has been consumed. It sets an *energy budget*. The energy budgets are configurable to allow for a trade-off between measurement precision and overhead.

② The measurement device integrates over the power samples internally, aggregating the energy demand.

③ The host system is only notified when the previously set energy budget has been consumed [12], [9], [30].

④ The measurement device proceeds accumulating the energy demand internally, and every time a multiple of the configured energy budget is exceeded, it sends another notification.

A sample communication sequence between the host system and the measurement device is illustrated in Figure 1. The protocol allows for a trade-off between precision (small energy budget and frequent interrupts) and accuracy (higher energy budget and less frequent interrupts). This trade-off allows it to be applied in a variety of domains each with

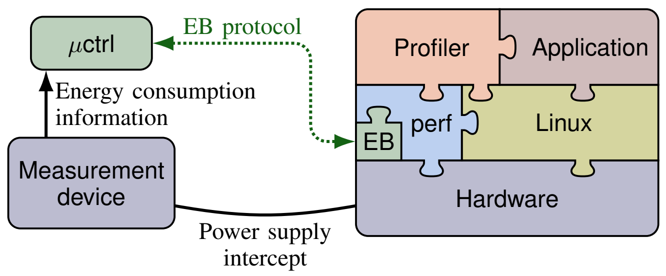

Fig. 2. The measurement device monitors the power consumption of the host system by intercepting its power supply. The energy consumed is transmitted using discrete packets while the current power draw is monitored continuously.

different power/performance profiles. It can both be used for performance analysis, and also for energy monitoring in production—if an EnergyBudgets-compatible measurement device is attached during operation, embedded devices can use small energy budgets to control the resource usage of the running application. Besides being broadly applicable for energy monitoring, EnergyBudgets also greatly simplify the limiting of energy usage with regard to naïve timer-based approaches, where the user has to frequently poll whether the current application's energy limit has been exceeded already.

Figure 2 summarizes the system architecture. It includes a microcontroller as indirection that translates between the physical energy measurement device and the EnergyBudgets protocol. On the host side, the protocol is tightly integrated into `perf`, the performance monitoring subsystem of Linux. It provides energy measurement values as *virtual performance counters* for individual system components. Behind the scenes, EnergyBudgets transparently and continuously updates the energy values, by communicating with the microcontroller.

IV. Implementation

In this section we present our modular open-source implementation of EnergyBudgets[1] that integrates into the existing performance profiling subsystem of the Linux kernel, allowing straight-forward analysis and correlation of energy with other performance metrics.

Our implementation of the EnergyBudgets protocol includes a kernel module for the SUT that communicates with the external measurement device, and a firmware for an AVR microcontroller which translates between the EB protocol and a physical energy measurement device. For current and voltage measurements, the microcontroller is attached to a LTC2991 current and voltage ADC [31]. The two devices and their connections are displayed in Figure 3 and Figure 4. The microcontroller and the host system communicate using a serial line. This has the advantage that the overhead is kept to a minimum in comparison to, for example, a network connection, while the interface is still available on a variety of devices. The communication protocol executed over the serial line employs tested and well-documented methods whenever

[1]The EnergyBudgets source code is available at https://gitlab.cs.fau.de/i4/pub/energybudgets under an open-source license.

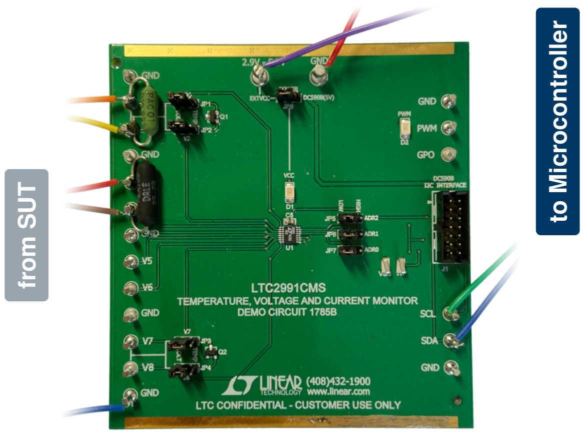

Fig. 3. The LTC2991 current and voltage ADC is employed for accurate physical energy measurements.

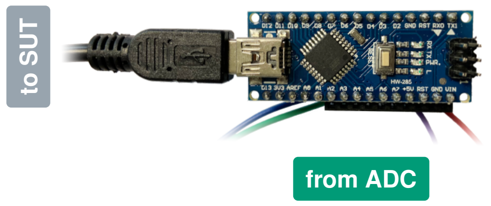

Fig. 4. An AVR microcontroller accumulates the measurement samples and communicates them to the SUT over a serial port (e.g., `/dev/ttyUSB0`).

possible, for example, the Serial Line Internet Protocol (SLIP) is used for framing [32], and XMODEM's CRC-16 is used to ensure data integrity [33], [34]. To minimize the amount of data, and therefore overhead, data is encoded in binary form. Figure 5 displays an overview over the interacting subsystems when our kernel module is used for energy measurements. On the SUT, the EnergyBudgets module registers both a line

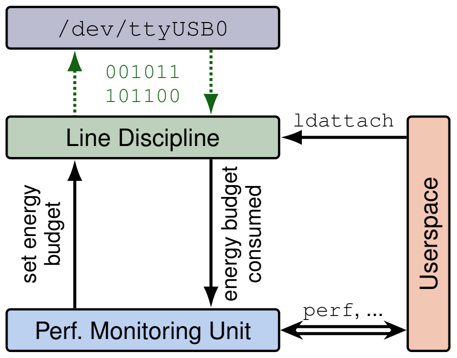

Fig. 5. Illustration displaying the interacting subsystems when EnergyBudgets are used for measurements. Userspace attaches a custom line discipline to the serial port on which the measurement device is connected. Thereafter, the line discipline interprets the packets and feeds them into the performance monitoring unit from which userspace can request the information, for example using the `perf(1)` tool.

discipline and a `perf` performance monitoring unit (PMU). When a new measurement device is connected to a serial port on the host system, the user attaches the custom line discipline to that serial port, using for example, `ldattach` [35]. This causes data received on this port to be interpreted as EnergyBudget packets, which constitute the notifications from the measurement device about consumed energy chunks. In response to these, the `perf` counter is updated internally. Note that the bytes received on this particular serial line are not copied to userspace to minimize latencies and context-switch related interferences. The module exports `sysfs` files that allow userspace to control how often the system is notified by the measurement device. Notifications can either be energy or time-driven, that is, the system is either notified whenever a set timer elapses or whenever a certain amount of micro joules has been consumed. To ensure portability, the code implementing the protocol is encapsulated in a library which at this point is already used by both the kernel module and the AVR firmware. Using our work, an application developer can measure the energy consumed by their application simply by entering the command shown in Listing 1.

Listing 1. Using `perf stat`, application developers can easily determine the amount of energy consumed during a task using the EnergyBudgets (`ebudgets`) PMU.

```
$ perf stat -e ebudgets/energy/ sleep 1
Perf. counter stats for 'system wide':
       66121 uJ ebudgets/energy/
 1.015791273 seconds time elapsed
```

Energy consumption is reported in the same way, and recording is started and stopped in synchronisation with any other `perf_event` metric which include various hardware and software events. This enables more advanced analysis tasks like correlation of energy consumption with software and hardware events, the creation of energy models, and hotspot detection. Thus, EnergyBudgets make energy *just another performance metric* for application developers.

V. Evaluation

In this section, we first evaluate whether the measurements performed with our tool show the expected properties for known workloads. We then compare them to measurements performed without our tool to determine the measurement trueness. Finally, we demonstrate that energy budget interrupts allow for higher-precision energy measurement than timer interrupts at the same interrupt rate.

To confirm the correctness of our implementation, we have measured the power consumption of known workloads on the Microchip SAMA5D3 Xplained board which hosts a single-core 528 MHz ARM Cortex-A5-based microprocessor unit. The system runs the Yocto Project Reference Distribution (Poky) v3.0 *Zeus* with a v4.19 Linux kernel. All resources required to build the used applications and systems software are published under an open-source license. Figure 6 shows the energy consumed by the processor, the embedded memories

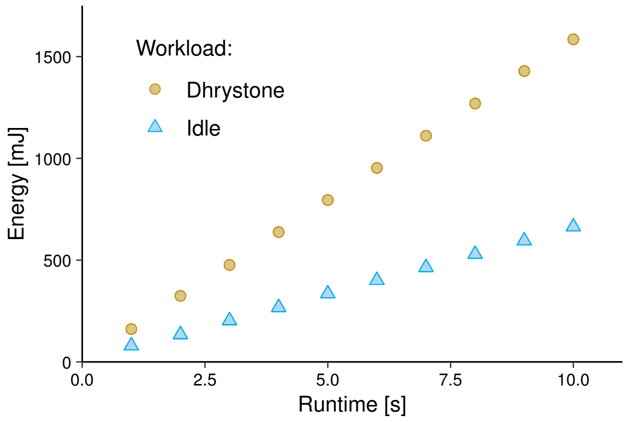

Fig. 6. Energy consumption of the SAMA5D3 Xplained board's core components while idle and under load measured using the EnergyBudgets perf PMU. As expected the amount of energy consumed increases linearly with the runtime because of the constant power draw. The power draw during Dhrystone is 2.4 $\times$ higher than the power draw while the system is idle.

and the peripherals[2] while idle and during the Dhrystone benchmark, measured using an energy budget of 8.0 mJ. Each measurement presented in this section represents the mean value calculated from 10 samples. For graphs, the variation among the samples was always below the displayed resolution (i.e., point size). For numerical values, the maximum distance to the respective minimum and maximum is indicated using a plus-minus sign ($\pm$) behind the value. The displayed distance also accounts for the measurement precision (i.e., 8.0 mJ for the following measurement). While the system is idle, allowing the CPU to frequently enter low power standby mode, our tool reports a mean power consumption of 66.4 ± 0.5 mW. During the Dhrystone 2 [37] benchmark a mean power draw of 158.4 ± 1.0 mW is reported. This closely matches the power consumption during Dhrystone reported in the board's datasheet, which is 162.5 mW [38]. The difference in 4.1 ± 1.0 mW between our value and the consumption reported by Microchip may be caused by a difference in

- measurement hardware and even in hardware of the same type, for example, due to aging and manufacturing variances. The datasheet does not indicate which measurement hardware was used by Microchip [38].
- system software, compiler version, and specific implementation of the Dhrystone benchmark.
- measurement software. While we used EnergyBudgets we do not know which measurement system was used by Microchip and whether it imposes overhead on the SUT.

To determine to which degree the difference is caused by EnergyBudgets, we have also measured the power consumption during Dhrystone 2 and while idle without EnergyBudgets but still using the same physical measurement device and the same systems software. To process and store the measurement data, we use a separate system therefore generating no overhead

[2]Measured by intercepting VDDCORE using JP1. The incorrect JP1 routing described in the board's user guide [36] was fixed.

on the SUT. In this setup, our measurement hardware reports a mean power draw of 66.8 mW while idle, and 154.7 mW during Dhrystone 2 when measured over the course of 10 s. The standard deviation among the 16 ms power samples was 0.7 mW and 1.5 mW, respectively. We conclude that *Energy-Budgets* can measure the idle power draw of the device with high accuracy and has no significant impact on the overall system performance.

The reduction of the mean Dhrystone power draw by 3.7 mW when the EnergyBudgets tool is not running on the SUT can be explained by an increase in the cache-pressure that is caused by the regular processing of the notifications from the measurement device. We conclude that the difference between our measurement and the measurement by Microchip is caused by differing measurement hardware and systems software.

In the following, we compare our energy-centric monitoring approach with the common time-based methodology. EnergyBudgets guarantee to the processor, that since the last notification, at most the set energy limit has been consumed. When a regular timer is used instead, the processor can not know whether the pending interval is one with large or small energy consumption. Therefore, to *guarantee* that at most a certain amount of energy was consumed since the last interrupt, the user has to assume the maximum power draw possible when deciding on the timer interval. This causes the timer interval derived to be unnecessary small as the device rarely consumes that much power during a typical workload. The small interval causes more frequent interruptions to the SUT and therefore hurts measurement trueness.

To demonstrate this advantage of EnergyBudgets, we have measured the energy consumption of a fixed workload using both energy budgets between 0.8 mJ and 40.0 mJ, as well as regular timers between 4 ms and 200 ms. Microchip does not list any information on the maximum power draw of the core components in the datasheet or user guide of the SAMA5D3 Xplained board [39], [36], [38]. Still, this information is required to determine the guaranteed level of precision for energy measurements when regular timers are used. Therefore, the user has to estimate the value based on additional measurements. EnergyBudgets in turn, do not require this information in the first place which is another advantage. Our measurements show a maximum power consumption for the board of *at least* 178.0 mW. Given some headroom we therefore estimate 200.0 mW to be the upper limit.[3] Assuming this maximum power draw, Figure 7 displays how energy budgets compare to regular timers regarding the overhead required for a given guaranteed level of measurement precision. Overhead constitutes itself in the number of notifications to the host system that update the energy counter maintained by `perf`. The set energy budget, or if regular timers are used, the timer interval multiplied with the maximum power draw, gives an upper limit on the amount of energy consumed after the last notification. Because computing systems rarely consume the

[3]EnergyBudgets outperform regular timers even when a lower maximum power draw is assumed. Choosing any value above 150.0 mW leads to the conclusion that EnergyBudgets give better precision.

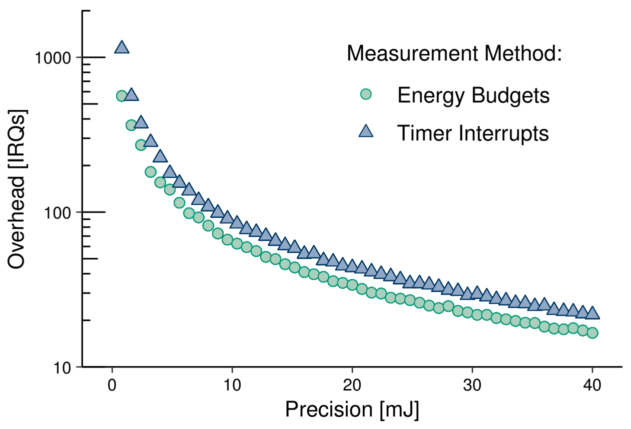

Fig. 7. Measurement displaying the tradeoff between overhead and precision enabled by EnergyBudgets. To guarantee the same level of precision, energy budgets require fewer packets to the host because regular timers force the user to assume the worst-case power draw, that is the maximum power draw possible, occurred since the last notification.

maximum amount of power possible, energy budgets allow for lower overhead while guaranteeing the same upper limit of unaccounted energy consumption.

VI. Discussion

EnergyBudgets provides a generic concept and a practical solution for a broad range of different application scenarios that require energy measurements at the software level. In contrast, approaches that pursue similar concepts [12], [9], [30] to EnergyBudgets are designed to address specific individual problems, only. They particularly provide one-shot solution for specific problems at hand. We believe that the modular open-source design that we provide with EnergyBudgets jointly with the well-integrated protocol supports and encourages future work. Both, our host systems' software and the protocol work off-the-shelf and do not require any further modification to be used by others. As part of our research work on EnergyBudgets we provide a firmware for the LTC2991 measurement device and the addition of new ADCs is aided by our portable communications library, therefore only requiring basic AVR programming skills.

In the previous section we have shown that the interrupts updating the energy counter on the SUT do not significantly impact the measurement accuracy of EnergyBudgets. Still, to allow for more rapid evaluation with fewer measurements, greater accuracy and even less overhead is always desirable. For plain energy measurements without any profiling, the SUT would ideally only request an update to the energy counter from the measurement device when it actually needs the value after the application has terminated. This would allow for measuring the energy consumption without any overhead on the SUT while the process is running. However, we do not believe this is possible without significantly changing the design of the Linux kernel's `perf_event` subsystem which was created under the assumption that performance counters are maintained by the CPU and not by an external device

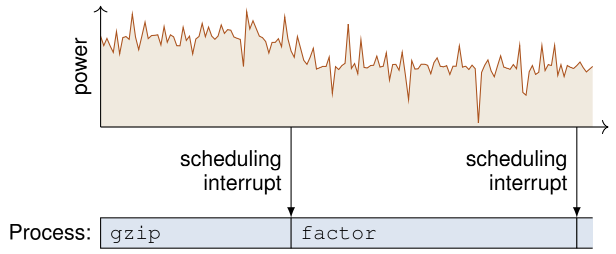

Fig. 8. Illustration displaying how energy budgets can be used for energy-driven scheduling. Applications with increased power draw due to IO (`gzip`) get preempted earlier than applications that only exercise the processor (`factor`).

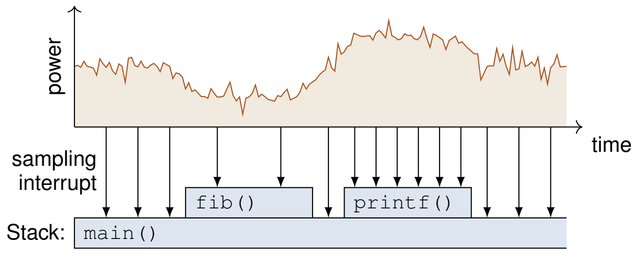

Fig. 9. Our tool allows for energy-driven statistical sampling. The more energy a subroutine consumes, the higher is the number of samples for that routine, therefore indicating where an application consumes the most energy.

connected over a serial port. Updating the kernel's energy counter on demand would induce a significant polling delay which we do not expect to be tolerable in the perf code path. A possible solution to this problem may be the use of an interface capable of direct memory access (DMA) between the SUT and the measurement device. This would allow the measurement device to continuously update the kernel's energy counter without creating overhead on the CPU of the SUT.

EnergyBudgets accurately measures the energy consumption of individual system *components* (e.g., as in the evaluation: the processor core, the DDR2 memory, the peripherals, and the NAND flash), this must not necessarily reflect the energy consumed by a specific application. On multi-core systems, for example, multiple processes trigger energy consumption on the processor simultaneously, and even on single-core systems applications are preempted frequently to serve interrupts and schedule background tasks. Still, while profiling, measuring the energy consumption of individual hardware components allows for the analysis of the consumption a specific implementation choice triggers in individual system components. This is very valuable to developers while profiling, especially because embedded systems become increasingly specialized into running dedicated homogeneous applications. More fine-grained analysis is achieved on the basis of combining EnergyBudgets with an solution for energy accounting [20], [27], [26], [25]. Because EnergyBudgets already reports the energy consumption of arbitrary individual system components separately, the accuracy of the chosen accounting mechanism is improved, too. Also, as illustrated in the following section, the complexity associated with fine-grained accounting is not even required when an individual CPU core is monitored using small energy budgets. This is the case, because the application code interrupted on it is directly responsible for the associated energy consumption.

VII. Future Work

In future work EnergyBudgets can be used to implement a variety of other OS services besides simple energy measurements. Using programmable EnergyBudgets, a scheduler can preempt applications based on energy limits not CPU time budgets as illustrated in Figure 8. Also, interrupts occurring every N joules can be used to determine *where* in an application most of the energy is consumed [40], [9]. Figure 9 illustrates how increased energy consumption in a subroutine causes a greater number of profiling samples be collected for the code section when the energy budget is sufficiently small. For this, the interrupt handler must record the current context, for example, the current value of the instruction pointer, for later analysis which may be implemented using `perf`'s existing sampling capabilities.

VIII. Conclusion

This paper presented EnergyBudgets, which leverage a powerful design that allows for the integration of external energy measurement devices into OS kernels. We have integrated energy budgets into the Linux `perf_event` subsystem, giving users a convenient and powerful interface to perform physical energy measurements. Our evaluation shows that energy budgets both allow for accurate measurements, but also that they outperform traditional timers regarding their guaranteed precision. In future work, energy budgets may be used for energy profiling and also energy-driven scheduling.

We therefore are the first to present a design that integrates physical energy measurement devices directly into the feature-rich performance profiling subsystem of a general-purpose OS, making energy *just another performance metric* to users.

Acknowledgment

This work was partially funded by the FAU Emerging Talents Initiative (ETI) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 146371743 – TRR 89 “Invasive Computing” and grant no. SCHR 603/15-2. Furthermore, we thank Heiko Janker for his help in implementing the firmware for the AVR microcontroller and in setting up the measurement hardware.

A preliminary work of EnergyBudgets was submitted to the 2020 ACM Student Research Competition (SRC) Grand Finals [41], we thank the anonymous reviewers for their thoughtful feedback.

References

[1] IEEE and The Open Group, “POSIX.1-2017,” 2017, accessed August 7, 2020. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/

[2] J. R. Lorch and A. J. Smith, “Software Strategies for Portable Computer Energy Management,” *IEEE Personal Communications*, vol. 5, no. 3, pp. 60–73, June 1998.

[3] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," in *Mobile Computing*, T. Imielinski and H. F. Korth, Eds. Boston, MA: Springer US, 1996, pp. 449–471.

[4] J. Hester and J. Sorber, "The Future of Sensing is Batteryless, Intermittent, and Awesome," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys, 2017, pp. 1–6.

[5] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent Computing: Challenges and Opportunities," in *Proceedings of the 2nd Summit on Advances in Programming Languages*, ser. SNAPL, 2017, pp. 8:1–8:14.

[6] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *Proceedings of the 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design*, ser. ISLPED, 2010, pp. 189–194.

[7] V. M. Weaver, "Self-monitoring Overhead of the Linux perf_event Performance Counter Interface," in *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software*, ser. ISPASS, 2015, pp. 102–111.

[8] J. Pan, "RAPL (Running Average Power Limit) Driver," 2013, accessed August 7, 2020. [Online]. Available: https://lwn.net/Articles/545745/

[9] F. Chang, K. I. Farkas, and P. Ranganathan, "Energy-Driven Statistical Sampling: Detecting Software Hotspots," in *Proceedings of the 2002 International Workshop on Power-Aware Computer Systems*, ser. PACS, 2003, pp. 110–129.

[10] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, ser. WMCSA, 1999, pp. 2–10.

[11] S. Reif, P. Raffeck, H. Janker, L. Gerhorst, T. Hönig, and W. Schröder-Preikschat, "Earl: Energy-Aware Reconfigurable Locks," in *Proceedings of the the 9th Embedded Operating Systems Workshop*, ser. EWiLi, 2019, pp. 1–6.

[12] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro Power Meter for Energy Monitoring of Wireless Sensor Networks at Scale," in *Proceedings of the 2007 6th International Symposium on Information Processing in Sensor Networks*, ser. IPSN, 2007, pp. 186–195.

[13] T. Stathopoulos, D. McIntire, and W. J. Kaiser, "The Energy Endoscope: Real-Time Detailed Energy Accounting for Wireless Sensor Nodes," in *Proceedings of the 2008 International Conference on Information Processing in Sensor Networks*, ser. IPSN, 2008, pp. 383–394.

[14] A. Schulman, T. Thapliyal, S. Katti, N. Spring, D. Levin, and P. Dutta, "BattOr: Plug-and-Debug Energy Debugging for Applications on Smartphones and Laptops," Stanford University, Tech. Rep., 2016, accessed June 2, 2020. [Online]. Available: http://cseweb.ucsd.edu/~schulman/docs/battor-tr.pdf

[15] I. Manousakis, F. S. Zakkak, P. Pratikakis, and D. S. Nikolopoulos, "TProf: An Energy Profiler for Task-Parallel Programs," *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 1–13, March 2015.

[16] E. Le Sueur and G. Heiser, "Slow Down or Sleep, that is the Question," in *Proceedings of the 2011 USENIX Annual Technical Conference*, ser. USENIX ATC, 2011, pp. 1–6.

[17] E. Le Sueur and G. Heiser, "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower, 2010, pp. 1–8.

[18] A. Weissel and F. Bellosa, "Process Cruise Control: Event-driven Clock Scaling for Dynamic Power Management," in *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES, 2002, pp. 238–246.

[19] S. Köhler, B. Herzog, T. Hönig, L. Wenzel, M. Plauth, J. Nolte, A. Polze, and W. Schröder-Preikschat, "Pinpoint the Joules: Unifying Runtime-Support for Energy Measurements on Heterogeneous Systems," in *Proceedings of the 2020 International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS, 2020, to appear.

[20] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys, 2012, pp. 29–42.

[21] UEFI Forum, Inc., "ACPI Specification, Version 6.3," 2019, accessed July 30, 2020. [Online]. Available: https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf

[22] L. Guo, T. Xu, M. Xu, X. Liu, and F. X. Lin, "Power Sandbox: Power Awareness Redefined," in *Proceedings of the 13th EuroSys Conference*, ser. EuroSys, 2018, pp. 1–15.

[23] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Currentcy: A Unifying Abstraction for Expressing Energy Management Policies," in *Proceedings of the 2003 USENIX Annual Technical Conference*, ser. USENIX ATC, 2003, pp. 43–56.

[24] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," in *Proceedings of the 10th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, 2002, pp. 123–132.

[25] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Energy Management in Mobile Devices with the Cinder Operating System," in *Proceedings of the 6th Conference on Computer Systems*, ser. EuroSys, 2011, pp. 139–152.

[26] R. Neugebauer and D. McAuley, "Energy is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS," in *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, ser. HotOS, 2001, pp. 67–72.

[27] F. 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*, ser. EW, 2000, pp. 37–42.

[28] B. Herzog, L. Gerhorst, B. Heinloth, S. Reif, T. Hönig, and W. Schröder-Preikschat, "INTspect: Interrupt Latencies in the Linux Kernel," in *Proceedings of the 2018 VIII Brazilian Symposium on Computing Systems Engineering*, ser. SBESC, 2018, pp. 83–90.

[29] P. Wägemann, F. Harbecke, B. Heinloth, H. Hofmeier, and W. Schröder-Preikschat, "An Energy-Neutral, WiFi-Connected Room Display with Hand-Crank-Based Energy Harvesting," 2019, accessed July 31, 2020. [Online]. Available: https://www.fau.de/2019/05/news/kluge-ideen-aus-dem-maschinenraum-der-fau/

[30] P. Wägemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh, "A Kernel for Energy-Neutral Real-Time Systems with Mixed Criticalities," in *Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS, 2016, pp. 1–12.

[31] Analog Devices, "LTC2991," 2019, accessed June 2, 2020. [Online]. Available: https://www.analog.com/en/products/ltc2991.html

[32] J. Romkey, "A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP," 1998, accessed June 2, 2020. [Online]. Available: https://tools.ietf.org/html/rfc1055

[33] C. Forsberg, "XMODEM/YMODEM Protocol Reference," 1986, accessed June 2, 2020. [Online]. Available: http://techheap.packetizer.com/communication/modems/xmodem-ymodem_reference.html

[34] AVR Libc Developers, "AVR Libc: CRC Computations (Function `_crc_xmodem_update`)," 2016, accessed June 2, 2020. [Online]. Available: http://www.nongnu.org/avr-libc/user-manual/group__util__crc.html

[35] T. Schmidt, "ldattach - Attach a Line Discipline to a Serial Line," 2008, accessed June 2, 2020. [Online]. Available: https://manpages.debian.org/buster/util-linux/ldattach.8.en.html

[36] Atmel, "SAMA5D3 Xplained User Guide," 2015, accessed June 2, 2020. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11269-32-bit-Cortex-A5-Microcontroller-SAMA5D3-Xplained_User-Guide.pdf

[37] R. P. Weicker and UnixBench Contributors, "The BYTE UNIX Benchmarks, DHRYSTONE Benchmark Program (`dhry2`)," 2018, accessed June 2, 2020. [Online]. Available: https://github.com/kdlucas/byte-unixbench/tree/070030e09f6effdf0c6721e8fcc3a5c6fb5bed1a

[38] Atmel, "SAMA5D3 Series Datasheet," 2016, accessed June 2, 2020. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11121-32-bit-Cortex-A5-Microcontroller-SAMA5D3_Datasheet_B.pdf

[39] Microchip Technology Inc., "SAMA5D3 Xplained," 2019, accessed August 7, 2020. [Online]. Available: https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/ATSAMA5D3-XPLD

[40] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin, "Profiling Software for Energy Consumption," in *Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*, ser. GreenCom, 2012, pp. 515–522.

[41] L. Gerhorst, "SOSP: U: EnergyTimers — Integrating Physical EnergyMeasurement Devices into Operating System Kernels," 2020, accessed October 8, 2020. [Online]. Available: https://src.acm.org/binaries/content/assets/src/2020/luis-gerhorst.pdf