Evaluating Techniques for Method-Exact Energy Measurements

Towards a Framework for Platform-Independent Code-Level Energy Measurements

Felix Rieger
Philipps-Universität Marburg
riegerf@mathematik.uni-marburg.de

Christoph Bockisch
Philipps-Universität Marburg
bockisch@acm.org

ABSTRACT

Reducing the energy consumption of executing programs decreases the operational cost and the environmental impact. For some well-known tasks, domain-specific guidelines for energy-efficient code exist. However, we want to enable profiling the energy consumption of code and selectively optimize hot spots, *previously unknown*. On our way to providing method-exact information about energy consumption in different hardware components, we currently focus on measuring the CPU's energy-consumption due to running a program itself. We identify three main challenges: measuring energy consumption at a fine granularity, constructing repeatable measurements, and mapping measurements to code. By actually building and using a platform-independent measurement setup, we show that it is feasible to measure the energy consumption at the granularity of method executions. The measurement time series must be aligned with execution events such as relevant method executions. We demonstrate how to evaluate alignment techniques by the examples of *linear scaling* and *Dynamic Time Warping* for mapping the measurements to code. While both are not yet sufficient to reliably map measurements to fine-grained code locations, our framework demonstrates the general feasibility of our goal and enables further research in this direction.

CCS CONCEPTS

• **Software and its engineering → Power management**; *Software performance*; • **Hardware → Energy metering**; • **General and reference → Empirical studies**; *Measurement*; *Performance*;

KEYWORDS

CPU energy consumption, energy profile, time series scaling, energy benchmark

ACM Reference Format:
Felix Rieger and Christoph Bockisch. 2020. Evaluating Techniques for Method-Exact Energy Measurements: Towards a Framework for Platform-Independent Code-Level Energy Measurements. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3341105.3374105

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03.
https://doi.org/10.1145/3341105.3374105

1 INTRODUCTION

Reducing the amount of energy required for running programs decreases the economic as well as ecologic costs of operation. Thus, we should start to treat the energy used by a program as a resource, just like CPU load or memory consumption, which can be analyzed with are a large number of available tools, such as profilers and debuggers. If we want to enable this kind of fine-grained insight into the energy usage of programs in order to reduce their energy consumption, we need to know the contribution of each computation toward the total energy consumption. This knowledge can come from energy consumption models or from measurements. Whereby models can only approximate average consumptions for time windows of a given size, while measurements provide the real-world energy consumption of the actual system running a program at any time. As this is desirable, *we need a framework to measure the energy consumption of program code elements.*

Storing measurements on the computer that executes the program would influence the energy consumption to be observed. Therefore, measurements must be stored externally. To be useful, we impose several requirements on our framework. Since computations are executed at high speed, also the measurement frequency must be high and a sufficient bandwidth is needed to transfer and store the measurements. We need a method for synchronizing the program's events to the measurement. Because different executions of the same program typically vary in length, we also need to enable aligning repeated measurements for the same program entity. For this purpose, we investigate two extrapolation techniques, namely linear scaling and Dynamic Time Warping (DTW) [6].

2 CHALLENGES

2.1 Measurement

We want to measure energy consumption of the relevant components of a computer with high temporal resolution. Methods are executed quickly and different kinds of workloads stress different subsystems of a computer. Virtual sensors, such as Intel's RAPL [3], would be an easy to use solution. They are already included in the user's system and thus allow low-cost measurement with no special expertise required. However, since they are executed by the device that is supposed to be observed, measurements would be distorted. Besides, their temporal resolution is currently insufficient. We therefore aim at physical measurements.

Power distribution in a computer is hierarchical; the main power supply is connected to the power grid and converts mains voltage at relatively small currents into intermediate voltages at larger currents. The local power supply of a processor consists of a voltage regulator that is able to supply the average current needed, and local energy storage that is able to quickly supply energy for

short peaks in energy consumption. Local energy storage capacitors form low-pass filters, concealing the processor's short-term energy consumption from the power supply.

Thus, since we want to measure short events changing the energy consumption, we need to measure as close to the processor as possible. With modern circuit boards, it is often impossible to physically interrupt the circuit and insert a current measurement shunt resistor at close proximity to the processor. Since current through a conductor produces a magnetic field dependent on the current and the geometry, we can also measure this magnetic field instead. Since this is a non-contact measurement approach, we do not need to modify the circuit board, allowing us to measure closer to the processor. We need to identify affordable measurement hardware, which is still able to carry out measurements at sufficiently high-speed. Furthermore, the hurdle of setting up and operating this equipment must be as low as possible.

2.2 Aggregation/Repeatability

We need to aggregate multiple measurements of the program execution to increase data quality, as this allows us to achieve greater robustness of the measurements. Energy consumption that is not caused by the execution of the program under test will be averaged out when the measurement is repeated. Each execution of the a program will take a varying amount of time on all but the most basic of systems, due to e.g. interrupts, I/O, scheduling, execution-time program optimization, and data-dependent timing from slight input variations. Thus, our measurement time series of the same program will have different lengths. This requires us to come up with a method for normalizing them to the same length, so we can then construct an aggregated measurement.

We make a few assumptions about programs we want to be able to measure. First, repeated executions of the program do not reorder the events in the program. Second, some events of the program may be left out.

For normalizing or scaling the measurements to the same length, an obvious solution candidate is to use linear scaling in the time domain (also known as linear resampling). But this method requires that there is some constant timing error that explains why some program executions take longer than others, which is not normally the case. Waiting for I/O is a counterexample: the program execution may process at normal speed before and after the I/O with a pause while waiting for the resource. This behavior cannot be modeled by a constant factor explaining the slowdown of a program. Thus, we need to use a method for non-linear resampling of our measurement time series. This method should identify common events in multiple time series and warp the time series in such a way that these common events coincide. We should get a distortion function that maps each point of a time series to one or more points of another time series. Since each time series has its own unique spurious events and changes in energy consumption, this method needs to be robust enough to cope with these events.

2.3 Mapping

We need to map the time-series energy consumption trace of the program to (parts of) the program's source code. To reason about the energy consumption of a method, we need to be able to reliably map the measured energy consumption to the method in the program's source code. For practical reasons, it is not advisable to use the computer that executes the program to measure the program's energy consumption. Thus, we choose to use separate devices for program execution and energy consumption measurement. Since one device executes the program and one device measures the energy, we need to find a way to synchronize these devices. Ideally, this is a hardware signal on the computer that can be asserted by the software being measured that also triggers data acquisition.

This could be achieved by fully instrumenting to the program under observation by adding method calls to acquire and persist the value of the high-speed timekeeper. However, this will likely have a sizable effect on the energy consumption of the program under test. Furthermore, it will have an impact on the shape of the energy consumption, complicating repeating the measurement to increase data quality.

For these reasons, we want to follow another approach: Multiple measurements should be applied with and without code instrumentation, while assuming the spurious energy consumption generated by code instrumentation to be noise. Then we can consider all measurements as repetitions for the same code, albeit the ones for the instrumented code include more noise than the others. Since measurement is automated, we can repeat them very often and instrument only few method calls each time. This way, we hope that the impact of the measurement error due to the instrumentation stays small. It requires us, nevertheless, to research techniques for dealing with noisy measurements including spurious timing errors.

3 STATE OF THE ART

In a previous survey on approaches to measuring the energy consumption of software [4] we discuss development tools supporting energy-aware development, specifically for energy optimization and energy debugging. We found many approaches for assessing the energy consumption of software systems, although none of them is platform-independent.

Running Average Power Level (RAPL) is the most prominent energy consumption model in the literature, as it is available on all Intel x86 platforms since the second Core 2 generation. It is a mainly model-based method for estimating energy consumption [3]. It was also used to measure energy consumption of one of the benchmarks we use to evaluate our measurement approach [1].

The goal of *worst-case energy consumption* analysis is to estimate the maximum amount of energy used for executing software to be able to give guarantees over meeting resource constraints. For an evaluation, they can use simple measurement setups, which measure the total energy consumption, comparing the total estimate with the measured total consumption. This estimate requires energy consumption models, which are typically only available for very specific embedded devices—they are thus not portable. Georgiou et al. [2] use their WCEC analysis to estimate the energy consumption for basic blocks. They also combine profiling with their consumption estimation, however not for dividing the energy consumption between code regions (as we will target in our approach) but for combining estimates to a total. Roth et al. [5] present a compiler framework with energy-oriented optimizations. The data on energy consumption used in the framework can come either from a model

or be measured. For this measurement, they use tailored hardware. They, however, do not discuss how the data items in the measured time series can be attributed to specific code regions.

4 EVALUATION

We evaluate our general methodology by carrying out multiple experiments. For these experiments, we acquire CPU energy consumption measurements of five benchmark programs. We then evaluate the quality of using Dynamic Time Warping (DTW) as a method for non-linear resampling before time series aggregation in order to carry out measurement repetitions, using linear resampling as the baseline approach. Then, we evaluate the adequacy of DTW as a method for mapping energy consumption measurement time series to program code based on measurements of an uninstrumented and an instrumented version of a program.

4.1 Experiment Setup

4.1.1 Measurement Setup. Since the CPU is responsible for a large part of the system's power consumption, we start with measuring the CPU. We use the AIM TTI I-Prober 520 current probe designed for non-invasive current measurement on PCB tracks. It has an analog bandwidth of 10 MHz, limiting the time granularity of our measurements. Using this probe, we do not need to modify the computer at all. We use the DTR line of the RS-232 port provided by the chipset of the benchmark computer as the synchronization signal. This signal and the current are acquired using a Rigol DS4014 digital storage oscilloscope. Our experiments were run on an x86-64 Fujitsu Esprimo E900 PC with an Intel Core i3-2100 processor at 3.1 GHz. The power supply and main board are proprietary parts.

4.1.2 Data Processing. We have the intuition that a program's energy consumption depends on certain events in the program. These events could, for example, include computing prime factors of a number or following pointers in a data structure. We posit that, since these events use different parts of the processor, we should see different energy consumption behavior. Also, energy consumption should change for sufficiently different events. Thus, our program will exhibit changes in energy consumption over time, caused by the composition of events. We find that this is very similar to speech: a word consists of changes in sound over time, caused by the composition of distinctive events (phonemes). Thus, we try to use methods from the domain of speech processing. Constructing a repeated measurement is thus akin to normalizing the length of two different recordings of the same word in a way that the corresponding phonemes are aligned. For constructing repeated measurements, we use Dynamic Time Warping. It was developed for time normalization in the context of speech processing [6]. Thus, we consider it a reasonable candidate as a method for non-linearly resampling our measurement time series in order to enable measurement repetition. DTW requires that the (partial) order of events in the repeated program executions stays the same.

For our evaluation, we compare the quality of matching events in repeated executions (runs) of a program. We use the synchronization signal to mark events in a program execution. Then, we use DTW to construct a mapping function that maps the measurement time series of the first run to the time series of the second run, essentially resampling the measurement of the first run to the length of the second run. We use this mapping function on the synchronization signals of the first run to predict where they should be in the second run. The absolute difference between the time of the predicted and the actual synchronization signals gives us the prediction error. We normalize this prediction error by dividing it by the absolute length of our measurement, giving the relative prediction error. We evaluate the relative prediction error of DTW against the relative prediction error of linear resampling.

To evaluate our approach to mapping energy consumption to code, we first need to ascertain that we can carry out repeatable measurements. If this prerequisite is fulfilled, we should see that a method without data-dependent behaviour, which is called multiple times at different times during the program execution, uses the same amount of energy each time. We should also be able to observe a change in the energy consumption of a method when its implementation is exchanged, while the energy consumption of the rest of the (unmodified) program is unaffected.

4.1.3 Benchmarks. For our experiments, we use a number of benchmarks written in Java. They are designed to capture broad range of common program behaviours.

The *braach-orig* [1] benchmark from the thesis of Fabian Braach features AES computations and exercises the ALU and FPU. It sleeps for 10 ms between different tasks, resulting in a very distinctive energy usage profile. The *braach-mod* benchmark is our modified version that does not sleep between different tasks. The *dtwtest* benchmark was specifically designed to perform badly when linear resampling is used, and should perform well using DTW. The *felix* benchmark features runtime interruptions of the program and data-dependent algorithms. The *image-io* benchmark is designed to include I/O operations.

4.2 Results

We carried out an experiment running the 5 benchmarks, acquiring 50 current measurements each, consisting of 3 runs, giving 150 runs per benchmark. We computed the relative prediction error of DTW and linear resampling on each unique pair of runs. Our dataset has 9873 (felix) to 11175 (all others) combinations per benchmark.

4.2.1 Is the relative prediction error of DTW lower than the relative prediction error of linear resampling? As discussed above, linear resampling performs poorly when the program pauses execution to wait for some resource, resuming execution afterwards. DTW should perform better in this scenario. Thus, our first hypothesis is that *the relative prediction error of DTW is smaller than the relative prediction error of linear resampling*. In order to test this hypothesis, we use a t-test to determine if the relative prediction errors of DTW and linear resampling are different, and if so, which one has a lower relative prediction error. For the *braach-orig*, *braach-mod* and *imgio* benchmarks, linear resampling gives a smaller relative prediction. For the *dtwtest* and *felix* benchmarks, DTW gives a smaller relative prediction error. Thus, we can not say that the relative prediction error of DTW is lower than the relative prediction error of linear resampling. We suspect that our statistical methods might be unsuitable for reasoning about the data, as the relative prediction error does not follow a normal distribution. We provide an example of the distributions for two benchmarks in Figure 1.

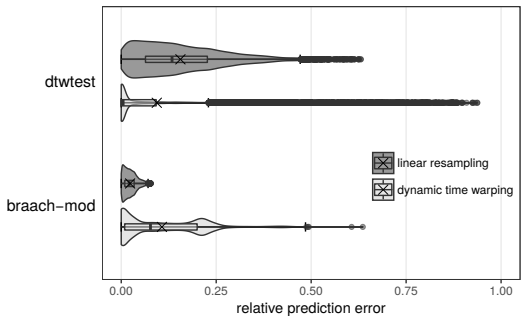

Figure 1: Relative prediction errors of typical benchmarks

4.2.2 Evaluating mapping: Do multiple calls of the same method at different times during program execution use the same amount of energy? Can we correctly attribute local changes in energy consumption to local code modifications? To evaluate these questions, we require a sufficiently reliable approach for making measurement repetitions possible. As we can see by now, our chosen approach does not fulfill the quality criteria for such an approach. Thus, we just summarize the results of our experiments.

We want to test if time series are correctly mapped to method calls. We measure the energy consumption of a program that has several calls of the same method interspersed throughout the execution. Our second hypothesis is thus: The energy consumption of the method should not change between subsequent calls. Hence, we need to test that the energy consumptions of the repeated method calls come from the same set, i.e. that they are not different. However, we compute a statistically significant difference in energy consumption. As a variation of this hypothesis, if we change such a method call locally, the energy consumption of the unaffected part of the program should not change. This leads us to our third hypothesis: We can determine that the energy consumption of unmodified and unaffected parts of the program does not change if we modify an independent part of the program. We need to ascertain that the energy consumption of the modified method is different from the energy consumption of the original method. The difference in energy consumption of the whole program can then be attributed to just this difference. In our evaluation, we could not identify that the energy consumption of a modified method is significantly different from the energy consumption of the original method.

4.3 Discussion

Our evaluation shows that our current approach of using DTW for creating repeatable measurements does not work sufficiently well. Nonetheless, we posit that our evaluation framework, consisting of experiment setup, benchmarks, and hypotheses has some merit and can be used for evaluating different approaches. Even in benchmarks that seem to disadvantage linear resampling, such as the dtwtest benchmark, DTW produces a large number of outliers with large prediction errors. The overall result of our evaluation is the need for further research. We were able to create a multitude of benchmark programs with different energy consumption patterns. These benchmarks clearly show differences between our baseline mapping function (linear resampling) and our proposed method (DTW). We also notice that linear resampling performs very badly for some programs, e.g. programs that execute their events at greatly varying times during subsequent runs. An approach for matching the shape of the energy consumption provides better results. DTW seems to be very sensitive to noise. However, filtering needs to be done carefully. It removes high-frequency information that might be critical for correctly matching the shape of the energy consumption.

5 CONCLUSION AND FUTURE WORK

In this paper, we outlined the challenges for fine-grained energy consumption measurement of program methods: measurement, repeatability, and mapping. We take a first step in solving these challenges. We outline measurement approaches, tools and caveats. We define a portable approach for repeatable energy consumption time series measurements by non-linearly resampling them so that the underlying events that cause changes in energy consumption coincide. We also define an approach for mapping energy consumption to code. This approach first takes a measurement time series of the program. Then, instrumentation is added to the program and another measurement is taken. We treat this instrumented measurement as a noisy measurement and compute the non-linear resampling function that maps it to the original measurement. When we apply this function to the timestamps, we should get the corresponding points in the original measurement, allowing us to compute the correct energy consumption.

In our evaluation, we use Dynamic Time Warping as an approach to non-linear resampling and compare it to linear resampling. However, we find that DTW does not perform as we hoped. Unfortunately, DTW is not robust with respect to noise and does not fulfill our quality criteria, leading to inconclusive results.

As speech processing algorithms also operate on time series data, we want to evaluate other methods from this domain as possible candidates for a robust non-linear resampling approach. Feature-based methods that use local reasoning are obvious candidates. We also want to explore ways to filter our measurement data to give better results without sacrificing timing accuracy. Furthermore, we plan to include more components in our measurements and adapt our benchmarks accordingly.

REFERENCES

[1] Fabian Braach. 2017. Ein statistisch rigoroser Energie-Benchmark. BSc. thesis.
[2] Kyriakos Georgiou, Steve Kerrison, Zbigniew Chamski, and Kerstin Eder. 2017. Energy Transparency for Deeply Embedded Programs. *ACM Trans. Archit. Code Optim.* 14, 1, Article 8 (March 2017), 26 pages. https://doi.org/10.1145/3046679
[3] Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. 2011. Power management architecture of the 2nd generation Intel® Core™ microarchitecture, formerly codenamed Sandy Bridge. In *Hot Chips*, Vol. 23. https://doi.org/10.1109/HOTCHIPS.2011.7477510
[4] Felix Rieger and Christoph Bockisch. 2017. Survey of Approaches for Assessing Software Energy Consumption. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems (CoCoS 2017)*. ACM, New York, NY, USA, 19–24. https://doi.org/10.1145/3141842.3141846
[5] Mikko Roth, Arno Luppold, and Heiko Falk. 2018. Measuring and Modeling Energy Consumption of Embedded Systems for Optimizing Compilers. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES '18)*. ACM, New York, NY, USA, 86–89. https://doi.org/10.1145/3207719.3207729
[6] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (1978), 43–49.