PowTrAn: an R Package for Power Trace Analysis

PowTrAn: an R Package for Power Trace Analysis

Luca Ardito, Marco Torchiano, Riccardo Coppola[a], Giulio Antoniol[b]

[a] *Control and Computer Engineering Dept., Politecnico di Torino, Italy*
[b] *Soccer Lab, Polytechnique Montreal, Canada*

Abstract

Energy efficiency is an increasingly important non-functional property of software, especially when it runs on mobile or IoT devices. An engineering approach demands a reliable measurement of energy consumption of software while performing computational tasks. In this paper, we describe PowTrAn, an R package supporting the analysis of the power traces of a device executing software tasks. The tool analyzes traces with embedded markers, a non-invasive technique that enables gauging software efficiency based on the energy consumed by the whole device. The package effectively handles large power traces, detects work units, and computes correct energy measures, even in noisy conditions, such as those caused by multiple processes working simultaneously. PowTrAn was validated on applications in realistic conditions and multiple hardware configurations. PowTrAn also provides data visualization that helps the user to assess the measurement consistency, and it also helps to highlight possible energy outliers.

Keywords: Energy Consumption; Power Trace Analysis; R language.

1. Motivation and Significance

A software program consists of a sequence of instructions that are run on an underlying hardware [1]. A device consumes energy due to the software it executes. Energy consumption can be considered as a non-functional requirement during software inception phase or as a property to be measured and monitored in production phase. For portable devices, such as laptops, tablets, and smartphones, energy consumption impacts battery life, resulting in a possible degradation of user experience [2], thus some users may prefer energy frugal application over a power-hungry one. In other domains, such as data centers or computing-intensive devices (e.g., those implemented by Bitcoin miners [3]), energy consumption increases electricity costs, which leads to a negative environmental impact. Challenges with measuring and reducing energy consumption are often addressed in an ad-hoc manner, as exemplified in Mochocki et al. [4].

While energy consumption can be estimated, through a battery discharge or CPU load data, an accurate evaluation must be based on physical measurements that can be linked to the software in real-time or offline. We developed a software package called PowTrAn (i.e., *POWer TRace ANalyzer*) that utilizes an *offline approach* for the collection of task-related data in power traces registered by a power meter. The data collected is used by different measurement devices, such as the HOBO UX120-018 Plug Load Data Logger[1] or RAPL[2].

When performing a physical power measurement on a device, discriminating the consumption due to the software under examination from other processes simultaneously running on that device is crucial. In practice, to gauge the energy consumption of an application while performing a specific task, it is necessary to identify the proportion of the power attributable to the task, which entails the following approach:

1. collecting energy data (i.e., energy traces),
2. identifying the relevant regions in the trace, (i.e., when the application or task was running),
3. estimate the application or task consumption, by separating it from the background contributions from the operating system and other applications.

This procedure requires a precise methodology to reconcile the physical power measures with the task execution timing. The approach supported by the software described in this paper consists of generating distinctive features in the power traces to markup the task execution. Although other approaches are possible, such as time synchronization, the use of markups is straightforward, precise, and does not require additional instrumentation.

This paper has four main goals: (i) describe the PowTrAn software and how it leverages offline power trace analysis, (ii) compare PowTrAn to other existing frameworks for

Email addresses: `first.last@polito.it` (Luca Ardito, Marco Torchiano, Riccardo Coppola), `giulio.antoniol@gmail.com` (Giulio Antoniol)

[1] https://www.powermeterstore.com/product/hobo-data-loggers-ux120-018-plug-load-data-logger Last Visited: 14/04/2020
[2] https://01.org/rapl-power-meter Last Visited: 14/04/2020

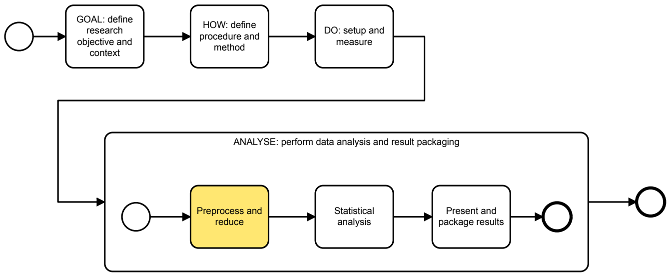

Figure 1: The energy study workflow as adapted from [5].

power analysis and how they solve several known problems in power trace analysis, (iii) describe how the software integrates into an analysis workflow within the R ecosystem, and (iv) provide examples of utilization of the software with real-world algorithms.

2. Background and Related Work

To better illustrate the role of PowTrAn, we first provide context in terms of a power assessment reference workflow, adapted from [5]. As shown in Figure 1, it encompasses four phases: (i) *Goal*, a definition of the research questions and context, (ii) *How*, a definition of the procedure, measurement method, and analysis method, (iii) *Do*, the setup of the devices and execution of the measurement, and (iv) *Analyze*, the analysis of the data. The latter phase includes three main activities:

- Pre-processing and data reduction: the power traces need to be pre-processed and reduced in size before being analyzed.
- Statistical analysis: the software uses reduced and pre-processed data to perform conventional statistical analysis.
- Present and package the results: after the results from the statistical analysis are available, they must be presented as diagrams and tables and packaged into a technical report.

PowTrAn was designed to fit in the energy assessment workflow and support the pre-processing activities. In particular, it takes care of several tasks:

- Reconciliation: the power trace must be combined with the information about the task timings,
- Task identification: the portion of the reconciled power trace that corresponds to the task executions must be identified;
- Reference identification: a reference value for the background tasks must be identified to offset the task consumption,
- Reduction: the size of the collected data is reduced for subsequent analyses because a single energy assessment experiment can obtain millions of samples.

For a non-invasive power measurement, the power consumption trace must be reconciled to the intervals when the tasks under consideration are performed. The reconciliation process can utilize two approaches:

1. synchronize the system clocks of the device running the measured software with the measurement device that collects the trace samples, and
2. instrument the code to add distinctive patterns to mark each task execution.

The clock synchronization requires accurate time synchronization between the device under test and the measurement device so that only the consumption related to the relevant tasks is recorded. This synchronization can be achieved using NTP (network time protocol) [6], and while this solution can be simple, it requires both devices to be connected at least to a LAN to reach the NTP server. Moreover, the precision of the synchronization might not be enough for power measurement purposes, especially for short-running tasks, as NTP has been observed to allows errors of up to 100ms, mainly due to network congestion [7].

The second approach enables the association of the consumption to a Software Under Test (SWUT) without clock synchronization, but simply adding markers in the SWUT as described, in Section 3.1.

We developed PowTrAn to address this specific use case by following these guidelines:

- Open-source: the software must be made available to the research community and researchers,
- Non-invasive: the software must require neither heavy instrumentation of the software under measurement nor presence of additional processes on the hardware device executing the software,
- Real measurement: the software must analyze actual physical measures of power consumption instead of estimates,
- Integration: the software must be part of statistical or computing environment and easily integrated into a robust statistical environment to enable researchers to perform further analysis and produce suitable visualizations.

The development intention is for PowTrAn to be the first step in an integrated analysis workflow.

PowTrAn is developed in R, a software environment for data analysis, manipulation, and visualization. R provides many packages for handling data of varied characteristics and sources [8]. To the best of our knowledge, PowTrAn constitutes the first effort in developing a power trace analyzer that leverages the R language and addresses non-invasive marker-based pre-processing. The choice of R is due to its popularity as an environment among scientists for performing data analysis. R is also widely used for big data, as it is easy to parallelize and interacts well with many other languages. Moreover, R provides excellent graphical capabilities that can be harnessed to produce control charts and assess the overall quality of the collected measures.

Many techniques to estimate and optimize the power consumption of applications and devices are described in the literature, and cover multiple levels of abstraction, from the electrical to functional levels. Lower-level techniques, even if more precise, require specific equipment and knowledge.

While the related software packages do present some of the detailed characteristics, none featured them all. Table 1 compares the available software packages with PowTrAn.

Pycoolr [10] is a monitoring and controlling software capable of sampling per-CPU core temperatures and CPU/DRAM consumption. Based on the Intel RAPL interface to take measurements, it outputs results in the JSON format for later analysis. The integration of Pycoolr in Python allows the usage of statistical libraries, like Panda or Mlpy to review the results. MuMMi [11] is an infrastructure for systematic measurements, built upon three existing frameworks of Prophesy (for performance modeling and prediction), PAPI (for hardware performance monitoring), and PowerPack (for power measurement and profiling). Eprof [12] is one of the first fine-grained off-device energy profiling software packages for Windows and Android mobile applications. Banerjee et al. [13] described a software that profiles the energy footprint of Android apps for finding energy anomalies. Atitallah et al. [9] provided a power trace analyzer to estimate power consumption and aid embedded software design, built on IP-XACT hardware descriptions. Naumann et al. [20] described a conceptual reference model for sustainable software, named GREENSOFT, that supports stakeholders involved in software development (e.g., developers, administrators, and users) in creating, maintaining, and using the software from a green perspective. The model covers, for each stakeholder, a model of the life cycle, power metrics, procedure models, recommendations, and software.

The "self-metering" approach presented in [21], [22], and [23] builds individualized online power models of smartphones. This action is possible if the device can read the online voltage and current values from its built-in battery interface. The primary limitation of the approach is the impossibility of incorporating current sensing to many smartphones.

Joulemeter [14], [15] models the energy consumption of memory, CPU, disk, and other components of a device, based on resource utilization. SES [16] is an energy monitoring software that collects energy consumption data with a cycle-by-cycle resolution, mapping each to the program structure. SES requires an extra module composed of measurement circuits, a profile controller, and an acquisition memory. Therefore, only certain embedded systems can use SES.

An example of a dynamic power management technique is Power-Sleuth [17] that fully describes the behavior of a software. In this work, the authors, instead of correlating power with events, developed a model that investigates the source of power consumption directly. Power-Sleuth locates program phases by using the ScarPhase library [24] to detect and classify each software phase.

Finally, DOME [19] is an evolution of PSAT [18], an open source Matlab and GNU/Octave-based software package for analysis and design of small- to medium-sized electric power systems. DOME is written in Python, and can parse data files to perform power flow analysis. The software is not open source.

All these related software collect and analyze power consumption data at various levels. PowTrAn is an open source library that addresses a specific use case (marker-based reconciliation); it can be included in any software-chain that collects and analyzes energy data.

3. Software Description

The PowTrAn R package[3] consists of roughly 800 lines of R code and can be installed through the commands shown in Listing 1.

Listing 1: The code to install the PowTrAn package.

```
install.packages("devtools")
library(devtools)
install_github("SoftengPoliTo/powtran")
```

Through the PowTrAn package, the procedure to analyze a power trace consists of the following steps:

- process the power trace with the `extract.power` function,
- perform a visual assessment using the control chart,
- analyze the energy values to assess the task under observation.

3.1. Trace markers

The technique adopted for identifying the task trace consists of generating one marker before and after the task.

[3] Code available on GitHub: https://github.com/SoftengPoliTo/powtran. So far, the package is not available on CRAN.

Software	Open source	Non-invasive	Physical meas.	Integrated
Atitallah et al. [9]	No	Yes	No	No
Pycoolr [10]	Yes	No	Yes	Yes
MuMMi [11]	No	No	No	No
Eprof [12]	No	No	No	No
Banerjee et al. [13]	No	Yes	Yes	No
Joulemeter [14][15]	Yes	Yes	No	No
SES [16]	No	No	Yes	No
Power-Sleuth [17]	Yes	Yes	No	No
PSAT [18]	Yes	Yes	Yes	No
DOME [19]	No	Yes	Yes	No
PowTrAn	Yes	Yes	Yes	Yes

Table 1: A Comparison of power consumption analysis approaches

This marker is a square impulse generated through a sequence of sleep, busy, and sleep. The busy phase is produced by generating a 100% utilization of the core. The two sleep phases are obtained by injecting a sleep period to keep the core idle, thus causing a minimum power consumption. The tailing energy can substantially impact the measurement, and, as suggested in [25], the final sleep, before running the task, can be long, such as a couple of minutes. For this reason, the sleep time could be longer than the busy time. However, in our examples, we assume that 1 second is sufficient for allowing the tail energy to disperse.

The marker is generated using the fragment of Java code shown in Listing 2, which is designed to work on multi-core architectures. The code generates one `busy` thread for each CPU and lets each CPU work for the given marker duration.

As mentioned above, markers are placed before and after each execution of the observed task, so in practice, a marker separates two tasks.

3.2. Extract.power function

The starting point of the analysis process is a power trace (e.g., a vector `data` comprised of numeric values). The primary function of the package, `extract.power` processes the power trace, and produces the results with its prototype shown in Listing 3), .

This function requires the following arguments:

- `data`: the power trace collected using any power monitor,
- `t.sampling`: the sampling period used to collect the trace,
- `N`: the number of task repetitions in the trace,
- `marker.length`: the expected width of the marker pulse,
- `baseline`: the method used to compute the baseline power, i.e. the background power not linked to the software under test.

The output of the function includes a table with the energy consumed by each task repetition, that can be plotted to produce a control chart or visualized via other PowTrAn functionalities.

Specifically, the output contains the work units that have been identified within the power trace. The *work unit* is defined as an atomic time window during which the execution of the analyzed software is subdivided. For each work unit, the following information is reported:

- `start` and `end` sample index of the work unit,
- `duration` in seconds,
- real power levels: for the work unit (`P.real`) and for the two idle phases preceding and following the work unit (`P.idle.left` and `P.idle.right`),
- effective power (`P`) and energy (`E`).

A control chart can be generated starting from the analysis result to visually assess the results of the analysis using the standard `plot()` function provided by the package.

The function performs four steps of pre-processing, including reconciliation through marker detection (Section 3.5), task identification of task data (Section 3.4), reference identification, and size reduction (Section 3.5).

3.3. Marker detection

The first step to enable processing of the power traces requires reconciling them to the timings of software tasks by detecting the markers inserted into the power trace.

Two factors are affect the detection of the markers:

- noise makes the detection of the markers edges difficult, and the measurement of the power level imprecise,
- size increases the complexity of the processing phase,[4] and the appropriate algorithms must be selected carefully. Also, graphical representations must use a downsampled version to make the trace discernible and

[4] for an experiment that is lasting 1 minute, at a sampling rate of 10kHz, we get 600k samples.

Listing 2: The code excerpt for marker generation written in the Java language.

```java
final static int N_THREADS = Runtime.getRuntime().availableProcessors();
private static void generateMarker(long markerLength)
                                    throws InterruptedException {
  //SLEEP
  Thread.sleep(markerLength);
  // BUSY
  final long endBusy = System.currentTimeMillis() + markerLength;
  final Thread[] ts = new Thread[N_THREADS];
  Runnable busy = ()->{ // Busy code
        while(endBusy>System.currentTimeMillis()){
          for(int i=0; i<markerLength;++i){ }
        }};
  Arrays.setAll(ts, t -> new Thread(busy,"PowTrAn"+t));
  for(Thread t : ts) t.start(); // start busy threads
  for(Thread t : ts) t.join(); // wait for all busy threads
  // SLEEP
  Thread.sleep(markerLength);
}
```

Listing 3: The `extract.power` function prototype.

```r
library(powtran)
res <- extract.power(data,          # samples
                     t.sampling,    # sampling period
                     N,             # num.
                         repetitions (30)
                     marker.length, # marker step
                         duration
                     baseline       # method for
                         baseline computation
)
```

avoid severe performance issues when using vector formats like PDF.

The procedure to analyze the data is comprised of five steps, detailed in the following subsections.

3.3.1. Step detection

A preliminary phase of the marker detection consists of identifying the rising edges of the marker pulses. Any noise present in the signal produces spurious edges that must be discarded to detect the markers correctly.

These spurious edges can be removed with a low-pass filter that eliminates high-frequency noise. However, the typical implementation of a low-pass filter uses an FFT, that provides poor performance on large-signals, and marker steps can also result in the Gibbs phenomena [26]. A similar result can be achieved by considering a moving average that is computationally faster.

The power signal with the embedded markers (see Figure 2) can be considered similar to a piecewise constant (PWC) signal [27], which can be analyzed by piecewise constant smoothing, or as a level-set recovery. The power trace during the experimental task is not guaranteed to be constant, so the signal is not precisely PWC.

Instead, we adopt a level-set recovery approach based on kernel density estimation using the following procedure:

- estimate the kernel density,
- identify the primary peaks in the density function,
- determine the thresholds between the power level clusters,
- represent the signal as a sequence of level runs.

3.3.2. Identification of markers

Markers can be identified based on three key characteristics:

- any individual marker pulse begins with a rising edge,
- markers must match a repeating pattern, with a set number of cycles,
- an individual marker pulse has a predefined width that should be recognizable within a specified level of tolerance.

The period of the repeating pattern is identified by finding the maximum of the auto-correlation function [28]. The offset of the first marker pulse with respect to the beginning of the power trace is identified by finding the maximum of the cross-correlation function applied to the trace and an ideal pulse train with the previously determined period.

Once the periodicity and phase of the trace are determined, the edges that most likely initiate the marker pulses are identified by means of a cross-correlation of a periodical function with the edges, as shown with the relative plot in Figure 3, defined as:

$$\left(1 + cos\left((x - first) \cdot \frac{n \cdot 2\pi}{last - first}\right)\right)^2 \tag{1}$$

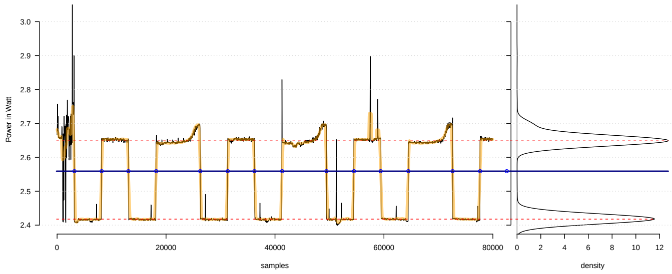

Figure 2: The power signal with embedded markers.

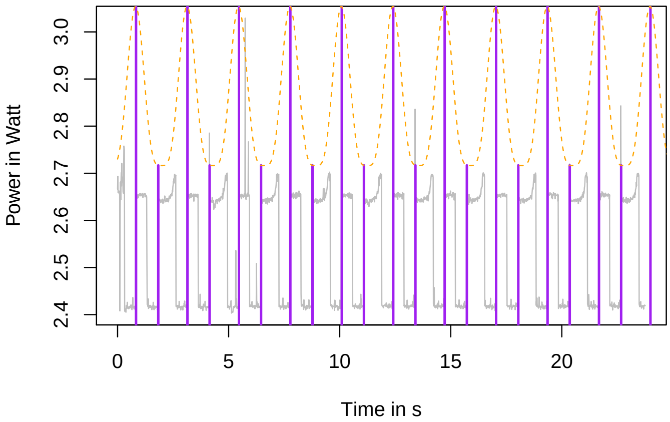

Figure 3: A plot of the adopted periodic function.

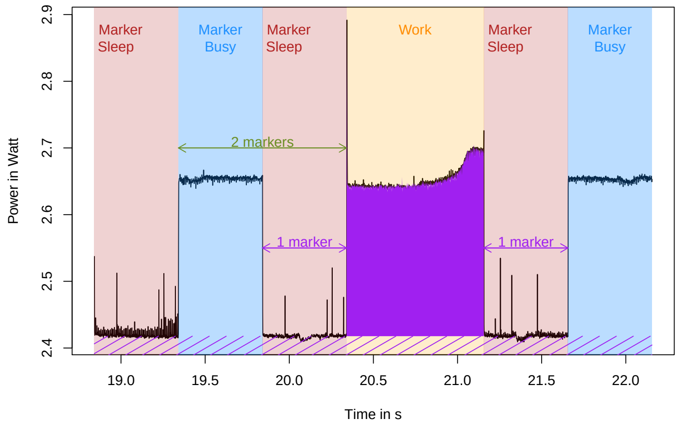

Figure 4: A work attributable to the task under consideration.

3.4. Identification of work units

This task consists of detecting the beginning and end of the work units within the power trace, by observing the rising edges of the marker pulses as a reference:

- the beginning of the work unit is estimated to be k marker pulse widths after the previous edge, where $k = 1 + \frac{sleeptime}{busytime}$,
- the end of the work unit is estimated to be one width before the next edge.

This design decision offers the double advantage of being easy to implement and avoiding the issue of spurious edges that would have otherwise hampered solutions based only on edge detection. A work unit attributable to the task under consideration is illustrated in Figure 4.

3.5. Effective power and baseline estimation

After identifying the work units, the power consumed by the system to conduct the task can be computed and is subject to two main decisions described in the following.

(1) *What is the amount of power ascribed to the program under test?* A first approximation might be that the program consumes the power recorded during the work unit (or its average). However, such a value also includes the power consumed by the idle system. A difference exists between real and effective power, where the former is a measured value, and the latter is the portion specifically used for performing a computational task.

The measured power must be compared to a baseline value that is not directly used for the computational tasks under consideration. Such a baseline power is typically a result of the idle system or other processes executed concurrently.

As shown in Figure 4, the baseline power is estimated based on the power measured during the sleep phases of the markers, and this can be performed by following several strategies. In general, local and global estimations can be distinguished by the following:

- Local: only the sleep phases immediately before and after the task under consideration are considered, which offers the advantage of offsetting possibly non-constant background processes,
- Global: all sleep phases enclosing the tasks are considered, which offers the advantage of filtering local noises by averaging the levels.

The selection of the specific sleep period to consider depends on the behavior of the system. For example, an energy-demanding task could trigger a frequency scaling [29] that alters the baseline on the local scale.

In addition to these two strategies, PowTrAn allows using a zero baseline, i.e., all power consumption is attributed to the software under test. This option can be applied when a ranking among the alternatives is the objective of the

Table 2: Alternate strategies for energy computation.

Scope	Pros/Cons
Local	Discards background processes that are not uniform during the experiment's execution time, especially erratic processes that occur unevenly.
Global	Filters measurement noise occurring during the experiment.
Zero	Applies the total system power without discerning between the process under consideration and other background processes, but is not a precise measurement.

measurement: as the precise amount of power consumed by a software to perform a task is not relevant, and the goal is to understand which software is consuming more.

(2) *What level of detail must be considered?* One option is to consider all the individual power values recorded in the trace, while the other is to calculate an average. Because the goal is to compute the energy (i.e., the integral of power over time), the basic average is equivalent in terms of the final results and more efficient in terms of memory resources.

To perform a size reduction on the data, each work unit has the energy consumed by the task under evaluation computed by:

$$E = t \cdot (\overline{P} - P_{baseline}). \quad (2)$$

where t is the task time, $\overline{P}$ is the average power measured during the task execution, and $P_{baseline}$ is the baseline power corresponding to the power consumption not directly attributable to the task execution.

4. Illustrative Examples and Validation

Validation of power analysis software should address the following aspects:

- ability to synthesize the power trace to reduce the data size,
- processing performance,
- potential to assess the quality of the collected data.

To illustrate the issues regarding the analysis of power traces, we consider two case studies on the two platforms of a Raspberry Pi 1A and an LG Nexus 4. Both devices use a CPU-based on ARM architecture. The Raspberry Pi 1A device adopts a single-core 32-bit CPU running at 700MHz, and the Nexus 4 utilizes a quad-core 64-bit CPU running at 1.5 GHz.

Table 3 lists the complete details about these case studies, which are distinct in many respects, so the resulting energy data cannot be directly compared. However, these two examples allow for assessment of how the software behaves in different conditions.

For both case studies, the task consisted of sorting an array of `integer` type elements. Each case applies different algorithms to perform this computation, specifically a quick sort for the Nexus 4 and bubble sort for the Raspberry Pi. In each experiment, we repeated the task 30 times, as several repetitions were required to average measurement errors.

4.1. Synthesis

The results from the analysis of the first case study are reported in Table 4.

Starting from $7.1 \cdot 10^5$ samples, the PowTrAn analysis produced a table with the information concerning each of the 30 repetitions of the measured task, with the first ten are sampled in Table 4.

Every line in the table reports the data synthesized from a repetition, and includes the following information:

- the start and end index of the specific sample in the sequence,
- the task duration, and based on this case with 8146 samples (from 18136 to 26282) and a frequency of 10 kHz, resulting in a value of 0.816 s,
- the real power, i.e., is the average power consumption measured during the execution of the task,
- the baseline power computed for this case has been computed using a local scope, so a slight difference is observed in each record,
- the effective power computed as the difference between the above two values,
- the energy consumed to perform the task.

4.2. Performance

PowTrAn demonstrated the processing of one million samples per second, producing the aggregate data described above. In practice through our tests, we processed 2.5 minutes of power traces per second.

4.3. Quality assessment

Figures 5 and 6 present the control charts generated by the package for assessment of the quality of the collected power trace. Each control chart is divided into two areas:

- the top portion reports a miniature view of the analyzed trace, where the work units and markers are identified;
- the bottom portion includes four diagrams that report the results of the analysis, including:

Table 3: The details about the case studies.

Device	Algorithm	Array Size	Time [ms]	Samples
Raspberry Pi 1A	Bubble Sort	10k	817	712698
LG Nexus 4	Quick Sort	50k	86	3703

Table 4: The results from the analysis (an excerpt of the complete data)

Sample index				Power		
start	end	t	P real	P baseline	P effective	E
18136	26282	0.815	2.653	2.417	0.236	0.192
41276	49454	0.818	2.653	2.416	0.236	0.193
64446	72604	0.816	2.654	2.418	0.237	0.194
87596	95759	0.816	2.654	2.418	0.237	0.194
110756	118931	0.818	2.656	2.418	0.239	0.196
133926	142092	0.817	2.654	2.418	0.238	0.194
157086	165255	0.817	2.655	2.418	0.239	0.195
180246	188410	0.816	2.654	2.418	0.238	0.194
203406	211563	0.816	2.654	2.418	0.237	0.194
226556	234721	0.817	2.654	2.418	0.237	0.194
			...			

  – the top right chart shows the distribution of the average power detected in the work units, represented in details with a strip chart and summarized with a box plot;
  – the bottom right chart shows the distribution of the work units durations, using the same visualization as the previous,
  – the bottom left chart shows the distribution of the energy consumed by each work unit,
  – the top right diagram shows power vs. duration, and also reports the iso-energy curves, which provides an opportunity to diagnose possible outliers in the results.

This last chart described is also useful consider possible trade-offs between speed and power. As modern processors scale the operating frequency automatically to adapt to varying workloads, the same task executed at a low frequency could last longer and consume lower power, while the opposite occurs at higher frequencies. We expect two such runs to consume a similar amount of energy, i.e., to appear approximately on the same iso-energy line. Thus, these reference lines enable a diagnosis of executions that consume similar energy for alternate duration vs. power configurations.

By comparing the two control charts, we observe the following:

- the trace for the Raspberry Pi is more regular compared to the one recorded with the Nexus,
- the distribution of power is narrow and symmetrical for the Raspberry Pi while it is more dispersed and skewed for the Nexus,
- the two duration distributions appear similar,
- reviewing the power vs. duration chart, two behaviors are observed. For the Raspberry Pi, a cloud of data points that follows the iso-energy lines where, in most cases, a longer duration corresponds to lower power, thus resulting in approximately similar energy. For the Nexus 4, a different pattern is observed with a tight cluster of data points and a set of points scattered around with varying levels of duration and energy,
- the Raspberry shows a clean symmetric shape in the energy, while the Nexus energy is highly skewed.

The analysis of the summary control plot represents a crucial step for evaluating the quality of the power trace and guiding the following additional analysis.

For example, based on the two plots described above, the energy consumption values for the program running on the Raspberry Pi are accurate. On the other hand, the values collected on the Android device are less accurate, so before proceeding with the analysis of the data, an outlier removal phase must be considered. While this process of removing outliers is not included in PowTrAn, the software provides sufficient information about which data might be reviewed as potential outliers.

5. Impact and Conclusions

We presented PowTrAn, an R-based power trace analyzer that constitutes the first step of an analysis workflow integrated into the R ecosystem.

PowTrAn represents a novel software package for processing physical power consumption measurements with offline

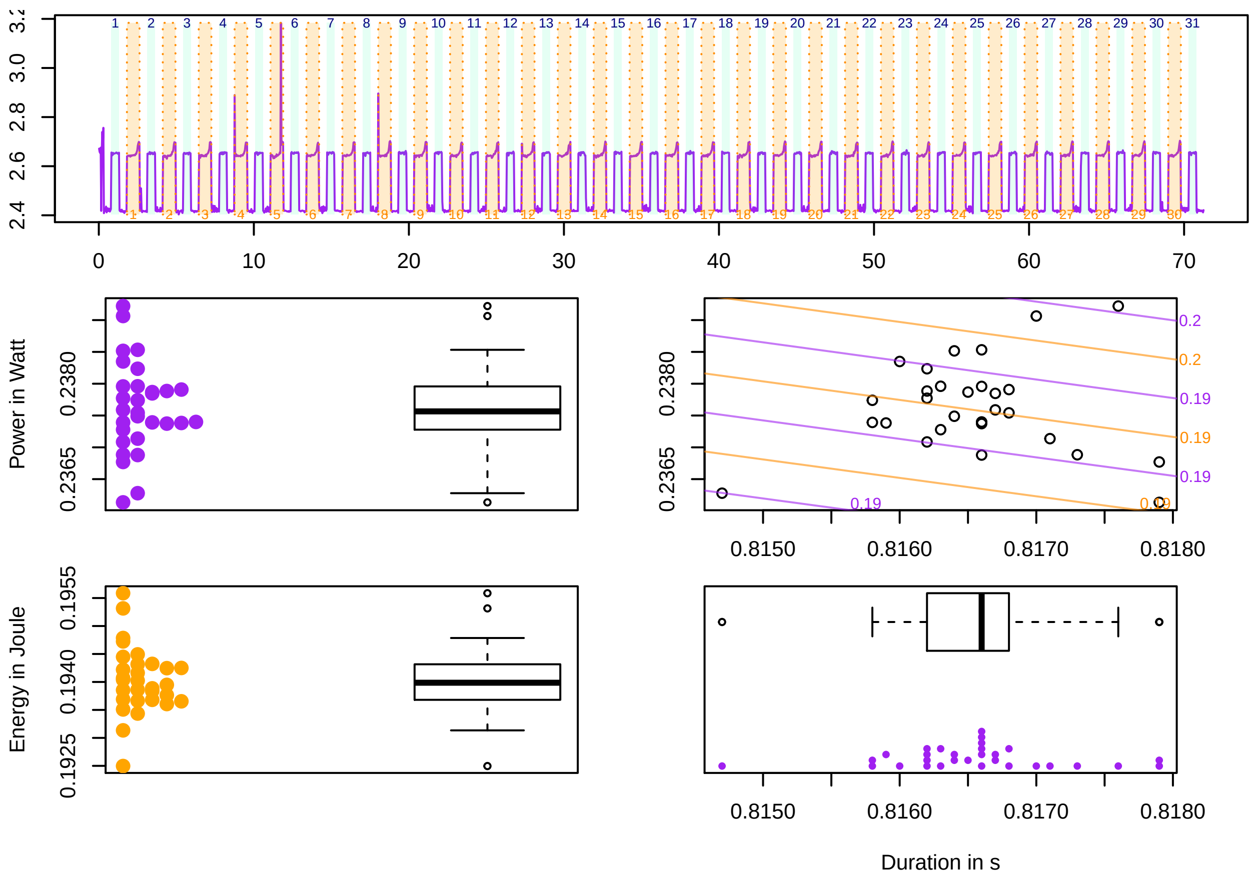

Figure 5: A summary control plot for the Raspberry Pi.

reconciliation that utilize markups. This paper provided a comprehensive description of the R package, and the software has already been applied in previous research, including:

- an analysis of various sorting algorithms, including bubble, counting, merge and quick sort, that were implemented in three programming languages (Java, ARM, and C) [30],
- a comparison of different image encoding and decoding algorithms run on mobile devices [31],
- the creation of a CPU power model for a Single Board Computer [32].

These works demonstrate the applicability of the PowTrAn package to a variety of application domains. We previously refined the initial ideas concerning the insertion of the markers as well as the analysis approach during earlier studies [30] [31].

We also tested PowTrAn in multiple conditions spanning operating systems, environments, and applications, and we demonstrated it could produce accurate results even in noisy systems.

6. References

[1] L. Ardito, G. Procaccianti, M. Torchiano, A. Vetrò, Understanding green software development: A conceptual framework, IT PROFESSIONAL 17 (1) (2015) 44–50. doi:10.1109/MITP.2015.16.

[2] J. Bornholt, T. Mytkowicz, K. S. McKinley, The model is not enough: Understanding energy consumption in mobile devices, in: Proceedings of 2012 IEEE Hot Chips 24 Symposium (HCS), 2012, pp. 1–3. doi:10.1109/HOTCHIPS.2012.7476509.

[3] P. Fairley, Blockchain world - feeding the blockchain beast if bitcoin ever does go mainstream, the electricity needed to sustain it will be enormous, IEEE Spectrum 54 (10) (2017) 36–59. doi:10.1109/MSPEC.2017.8048837.

[4] B. Mochocki, K. Lahiri, S. Cadambi, Power analysis of mobile 3d graphics, in: Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings, DATE '06, European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2006, pp. 502–507.

[5] L. Ardito, R. Coppola, M. Morisio, M. Torchiano, Methodological guidelines for measuring energy consumption of software applications, Scientific Programming 2019 (2019) 16. doi:https://doi.org/10.1155/2019/5284645.

[6] D. Mills, J. Martin, J. Burbank, W. Kasch, Network time protocol version 4: Protocol and algorithms specification, RFC 5905, RFC Editor, http://www.rfc-editor.org/rfc/rfc5905.txt (June 2010).
URL http://www.rfc-editor.org/rfc/rfc5905.txt

[7] N. Minar, A survey of the ntp network (December 1999).
URL http://www.media.mit.edu/~nelson/research/ntp-survey99/

[8] R Core Team, R: A Language and Environment for Statistical

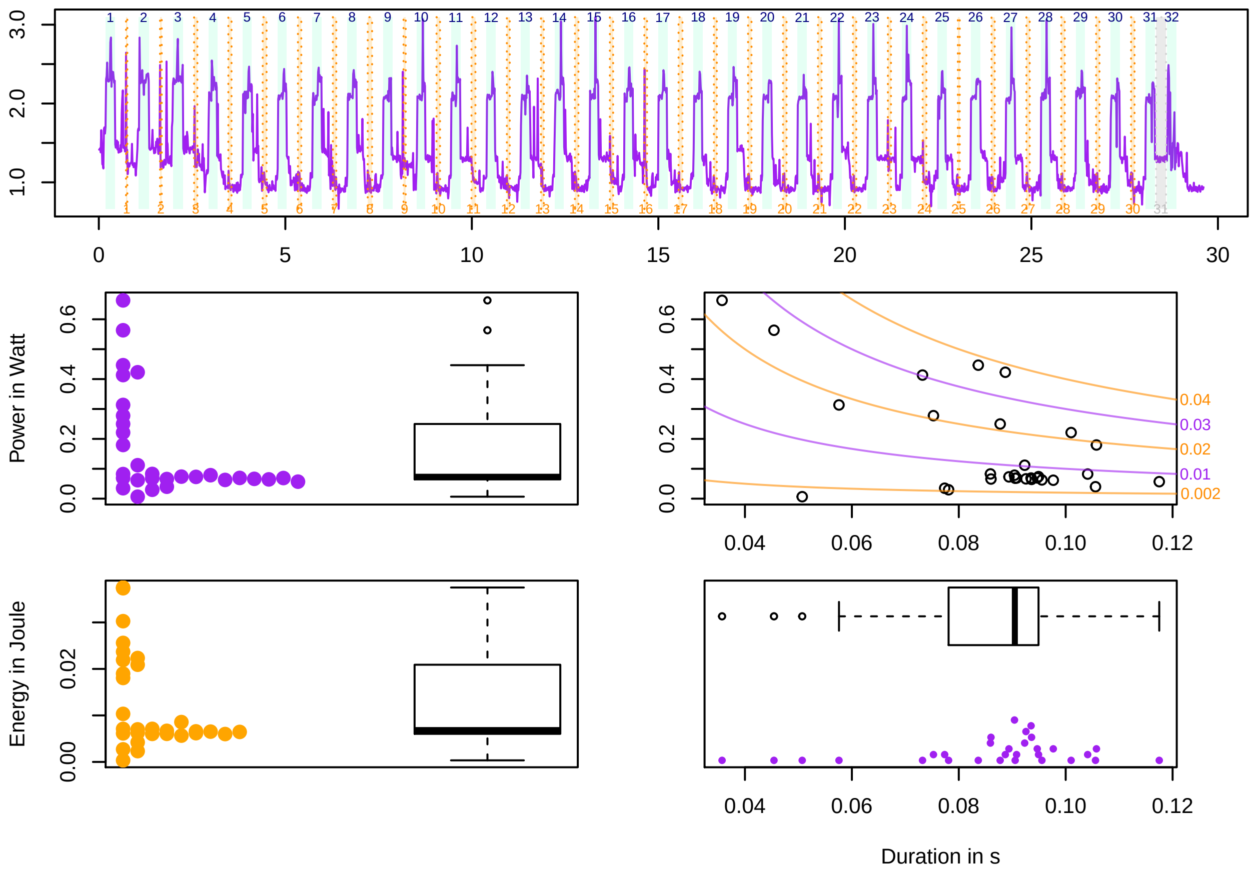

Figure 6: A summary control plot for the Android Nexus 4

Computing, R Foundation for Statistical Computing, Vienna, Austria (2018).

[9] Y. B. Atitallah, J. Mottin, N. Hili, T. Ducroux, G. Godet-Bar, A power consumption estimation approach for embedded software design using trace analysis, in: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, 2015, pp. 61–68. doi:10.1109/SEAA.2015.34.

[10] K. Ahmed, J. Liu, K. Yoshii, Enabling demand response for hpc systems through power capping and node scaling, in: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2018, pp. 789–796. doi:10.1109/HPCC/SmartCity/DSS.2018.00133.

[11] X. Wu, H.-C. Chang, S. Moore, V. Taylor, C.-Y. Su, D. Terpstra, C. Lively, K. Cameron, C. W. Lee, Mummi: multiple metrics modeling infrastructure for exploring performance and power modeling, in: Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, ACM, 2013, p. 36.

[12] A. Pathak, Y. C. Hu, 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, ACM, 2012, pp. 29–42.

[13] A. Banerjee, L. K. Chong, S. Chattopadhyay, A. Roychoudhury, Detecting energy bugs and hotspots in mobile apps, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 588–598.

[14] A. Sinha, A. P. Chandrakasan, Jouletrack-a web based tool for software energy profiling, in: Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), 2001, pp. 220–225.

[15] A. Kansal, F. Zhao, J. Liu, N. Kothari, A. A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 39–50. doi:10.1145/1807128.1807136.
URL https://doi.org/10.1145/1807128.1807136

[16] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, N. Chang, Energy-monitoring tool for low-power embedded programs, IEEE Des. Test 19 (4) (2002) 7–17. doi:10.1109/MDT.2002.1018129.
URL https://doi.org/10.1109/MDT.2002.1018129

[17] V. Spiliopoulos, A. Sembrant, S. Kaxiras, Power-sleuth: A tool for investigating your program's power behavior, in: Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS '12, IEEE Computer Society, USA, 2012, p. 241–250. doi:10.1109/MASCOTS.2012.36.
URL https://doi.org/10.1109/MASCOTS.2012.36

[18] F. Milano, An open source power system analysis toolbox, IEEE Transactions on Power Systems 20 (3) (2005) 1199–1206.

[19] F. Milano, A python-based software tool for power system analysis, in: 2013 IEEE Power Energy Society General Meeting, 2013, pp. 1–5.

[20] S. Naumann, M. Dick, E. Kern, T. Johann, The greensoft model: A reference model for green and sustainable software and its engineering, Sustainable Computing: Informatics and Systems 1 (4) (2011) 294 – 304. doi:https://doi.org/10.1016/j.suscom.2011.06.004.
URL http://www.sciencedirect.com/science/article/pii/S2210537911000473

[21] M. Dong, L. Zhong, Self-constructive high-rate system energy modeling for battery-powered mobile systems, in: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 335–348.

doi:10.1145/1999995.2000027.
URL https://doi.org/10.1145/1999995.2000027

[22] W. Jung, C. Kang, C. Yoon, D. Kim, H. Cha, Devscope: a nonintrusive and online power analysis tool for smartphone hardware components., in: A. Jerraya, L. P. Carloni, N. Chang, F. Fummi (Eds.), CODES+ISSS, ACM, 2012, pp. 353–362.
URL http://dblp.uni-trier.de/db/conf/codes/codes2012.html#JungKYKC12

[23] C. Krintz, S. Gurun, A run-time, feedback-based energy estimation model for embedded devices, in: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06), 2006, pp. 28–33.

[24] A. Sembrant, D. Eklov, E. Hagersten, Efficient software-based online phase classification, in: Proceedings of the 2011 IEEE International Symposium on Workload Characterization, IISWC '11, IEEE Computer Society, USA, 2011, p. 104–115. doi:10.1109/IISWC.2011.6114207.
URL https://doi.org/10.1109/IISWC.2011.6114207

[25] S. A. Chowdhury, V. Sapra, A. Hindle, Client-side energy efficiency of HTTP/2 for web and mobile app developers, in: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1, 2016, pp. 529–540. doi:10.1109/SANER.2016.77.
URL https://doi.org/10.1109/SANER.2016.77

[26] M. Stéphane, A Wavelet Tour of Signal Processing (Third Edition), third edition Edition, Academic Press, Boston, 2009. doi:https://doi.org/10.1016/B978-0-12-374370-1.00001-X.

[27] M. A. Little, N. S. Jones, Generalized methods and solvers for noise removal from piecewise constant signals, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 467 (2011) 3115–3140. doi:http://doi.org/10.1098/rspa.2010.0674.

[28] R. H. Shumway, D. S. Stoffer, Time Series Analysis and Its Applications (Springer Texts in Statistics), Springer-Verlag, Berlin, Heidelberg, 2005.

[29] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, 3rd Edition, Morgan Kaufmann, 2002.

[30] M. R. A. Rashid, L. Ardito, M. Torchiano, Energy consumption analysis of algorithms implementations, in: Proceedings of 9th International Symposium on Empirical Software Engineering and Measurement (ESEM 2015), IEEE CS, 2015, pp. 1–4. doi:10.1109/ESEM.2015.7321210.

[31] M. R. A. Rashid, L. Ardito, M. Torchiano, Energy consumption analysis of image encoding and decoding algorithms, in: Proceedings of 4th International Workshop on Green and Sustainable Software (GREENS), 2015, Vol. Green and Sustainable Software, IEEE, 2015, pp. 15–21. doi:10.1109/GREENS.2015.10.

[32] L. Ardito, M. Torchiano, Creating and evaluating a software power model for linux single board computers, in: 2018 ACM/IEEE 40th International Conference on Software Engineering Workshops, IEEE CS, 2018, pp. 1–8.