VS Code Power Measurement: A VS Code Extension for Measuring Power Consumption

Ole Peder Brandtzæg Aaron van Diepen
Rolf Piepenbrink Jasper Teunissen

April 14, 2023

GitHub repository

1 Introduction

In recent years, more research has been dedicated to energy-aware software. A crucial aspect of the development of such software is measuring energy consumption, as a means to assess the degree of energy efficiency. Though, this is not a trivial task. From a practical standpoint, measuring energy consumption is inherently flaky. There are numerous variables, some of which are difficult to control, that may influence the results. For instance, tasks running in the background can affect the measured energy consumption.

Additionally, human factors play an important role in energy efficiency as well. Firstly, the programmers who develop the software can impact the energy efficiency depending on the code they write; inefficient code could result in higher energy consumption. Secondly, the programmers have to be aware that their actions affect the energy consumption of their software.

Without awareness, energy efficiency is unlikely to be actively taken into account during the development process. It is simply not yet always part of the thought process behind software development. Bad coding habits, for instance, could contribute to lower efficiency.

To bridge this gap, we propose a solution that aims to achieve two main goals: increase awareness of energy-aware software development, and increase the accessibility of tools that enable programmers to incorporate energy efficiency in the development process. More specifically, we are targeting software developers who are interested in considering energy consumption in their development cycle from within the IDE they use. The three concepts of our core design enable us to create a tool that achieves this, namely accessibility, usability and integration. This glues the perspective of our intended users and the goals we aim to fulfil.

Given that software developers generally use an integrated development environment (IDE) to write code, and given that IDEs are extensible to meet a

developer's needs, we argue that integrating an energy measurement tool in an IDE is a suitable approach to achieve the goals of our envisioned solution.

2 Related Work

For our project, we are mainly interested in software-based power measuring tools, or power meters, since we intend to publish it as an extension available from an IDE. Measuring power consumption can be done in a variety of ways. For instance, Intel's RAPL (Running Average Power Limit) technology and Nvidia GPU NVML are both capable of measuring the power consumption of a machine. Though, while powerful, these tools miss some of the wanted functionalities; solely using these tools does not allow one to measure the energy consumption per process.

A few other tools have built upon these and improved their usability and reliability. Prime examples of such tools are Scaphandre [3], PowerAPI [7] and perf [6]. These are all tools that allow for real-time power measuring with a software-based approach and can make use of the RAPL interface. In short, Scaphandre runs as a daemon on a given system and allows for obtaining the approximate power consumption per individual process in real time and makes this data available through various exporters, among them printing to standard output, emitting JSON and exposing a Prometheus endpoint. PowerAPI works similarly and allows for the data to be stored in a variety of databases and file formats. perf, on the other hand, is a Linux command offering fine-grained access to the kernel's perf_events interface. They are all open source under lenient licenses.

Conveniently, Jay et al. [5] have performed an extensive comparison between these three. They found that the tools are fairly similar in terms of functionalities and capabilities. We conclude that the most significant differences lie in user-friendliness. Since we want to be able to create a plug-in for an IDE, we are interested in the extensibility and ease of use. Of the three Scaphandre seems to be the best fit; it's actively developed and offers a per-process breakdown of the system's power consumption, which is beneficial for our usecase, i.e. measuring a specific program.

3 Solution

We have created a tool that enables software developers to measure and analyse the energy consumption of the software they write, all integrated in a single IDE. Our solution is intended to be integrated into the standard software development cycle. Since the developers themselves are the ones who are targeted with our tool, we have to create a tool that fits their needs. We derive a set of requirements our solution has to adhere to in order to achieve our main goals.

Requirements

- Measure the power consumption of the code run from an IDE.
- Allow for real-time energy consumption analysis.
- Communicate the obtained measurements to the user visually.
- Be accessible from and/or integrated within an IDE.
- Be free and open source.
- Have a low barrier of entry.

We are immediately presented with a set of practical questions. How do we measure energy consumption? And, which IDE should we choose? The answer to the first question is Scaphandre. This covers one of the main limitations we expected to face, namely that energy measurements often only provide information of the entire system and is strongly influenced by numerous variables, such as background tasks running during the measurement process.

The question regarding the choice of IDE can be answered in a quite straightforward manner. Since we want our tool to be free and open source, we deem this to be important qualities of the IDE we choose as well. Though, since we also want this tool to be widely available, we have to consider the popularity of the IDE as well. Users are unlikely to switch IDEs solely for the existence of our tool. Therefore, our eyes fell on Visual Studio Code (VS Code). VS Code is open source and free to use. Besides that, it has a sufficiently large user base and allows for easy extensibility.

With these two answers in mind, a closer look can be taken at the requirements. It becomes clear that our philosophy should revolve around three concepts: accessibility, usability and integration. Accessibility refers to the concept of ease of access. That is, it should not require much set and the barrier of entry should thus be low. Furthermore, the tool should work in several use cases and should not require the user to meet very specific conditions. Though, we do set a couple of boundaries and optimise our tool within said boundaries, to narrow the scope and make this project more manageable. The scope will be explained later on in this section.

Usability describes the way we mean to empower users with relative ease. The tool's functionalities and usage should be familiar to the user. We interpret this as the tool having a vast set of features that allow the user to actively consider and analyse the energy consumption of the software they write. For example, through the use of various graphs and figures, we communicate the results of the measurements to the user and enable them to act upon it.

Finally, integration is also one of the three main pillars of our tool. While related to the other two, special attention has been given to this component. Concretely, since the tool is integrated in VS Code, users only have to include the extension in their workspace. Furthermore, we want our tool to seamlessly blend in with the environment, so that the controls and feel are familiar to the user, thereby lowering the barrier of entry.

3.1 Scope

Given the context in which our tool is developed, we have to set certain parameters to define the scope. More specifically, the scope has been narrowed such that a development time of three weeks is feasible, without forsaking the core functionalities of the tool. Firstly, while it would be interesting to have a more holistic comparison of energy consumption measurements, there are certain limitations that are difficult to circumvent. The same program run on different machines, for example, will likely result in different readings. Rather than having a wide scope with not fully fledged functionalities, we argue that a narrower scope with more sophisticated features is more desirable, both from the user's perspective as well as ours. With this methodology, the tool can be kept relatively simple and allows for maintaining a low level of entry. We thus define the scope as *local*. The tool is run locally by the user from within their IDE. This still enables the user to inspect the energy consumption and gives an indication of where potential energy outliers may occur. Analysing the results is therefor done locally as well.

Due to the limitations of power meters, comparing results from different machines is not trivial. Implementing this would broaden the scope too much in our opinion and make the tool too complex. Instead, the focus of our tool lies with the progress of a local machine. Analysing the progress made with regards to energy consumption reduction, we are convinced that providing a local history of measurements suffices.

3.2 Implementation

As touched upon briefly, there are two main components in terms of implementation: measuring energy consumption and the VS Code plugin. These two have to be interfaced, though, which was one of the challenges encountered in the development of the tool. Scaphandre's functionalities are ample. However, porting them to an extension requires quite some work. This mostly relates to the requirements of creating a VS Code plugin: the extension has to written in TypeScript. Scaphandre is somewhat unstable in terms of creating bindings, so we instead opted for calling commands from within the extension code.

The measurements are stored in a file in global storage, which is a folder specifically dedicated to our extension and accessible to the extension at all times. The path to said folder is specified in the VS Code context.

Regardless, obtaining the measurement data is only one part of the problem. The second part is selecting the data corresponding to the task of running the program of interest to the user. That is, software is run from within VS Code. The energy measurements contain data of all tasks running on a machine, though separated per tasks. If all data of all tasks were shown, the result would be a convoluted and cluttered overview. To solve this issue, we make use of the process ID (pid). Determining which pid belongs to the task of interest to the user is not as straightforward, however. VS Code's debug API does not expose the pid of the process of interest. The pid is simply a numeric identifier with

no contextual information. Thus, since the tool itself cannot identify which process to look for, due to this lack of context, the selection is left to the user. To make this work with Scaphandre, we have extended its functionalities by implementing a function that enables selecting a process' energy consumption based on the pid.

Next, once the data has been reduced to information relevant to the user, the third step is to present the data. A graph can be generated and shown in the IDE interface, where time and energy consumption are plotted. Note that this is in real-time, with the option of regressive comparison.

Finally, another part of the solution is related to publishing the tool as an extension, so it is available to others using VS Code. This process, while relatively simple, is not to be overlooked, since our tool would be ineffective at achieving our goals if it were not publicly usable. In VS Code, our tool can be found as usual, or by searching with by its ID: *vscode-power-measurement.power-measurement*.

4 Design Choices

In terms of design, we decided to focus on seamless integration. We argue that the barrier of entry can be kept low by simulating, or mimicking, a style that is familiar to the user. Since they use VS Code, the controls for our tool are similar to those for debugging or running a piece of code, a basic procedure assumed to be known by the user.

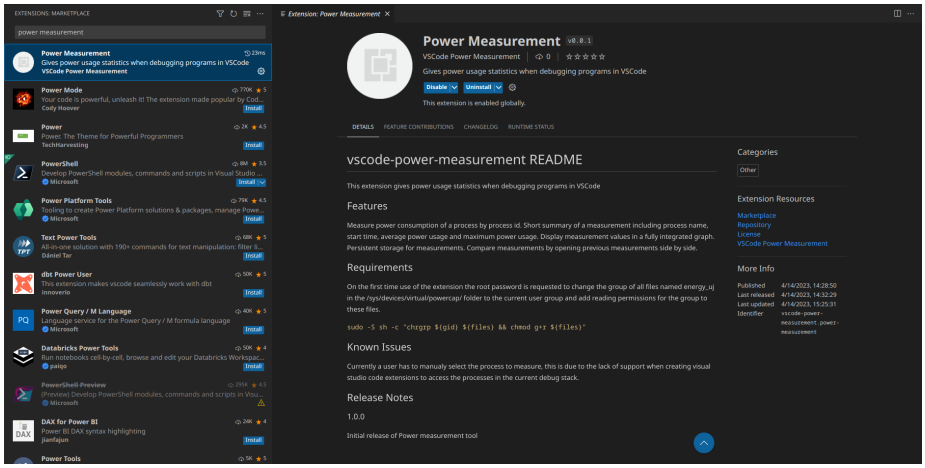
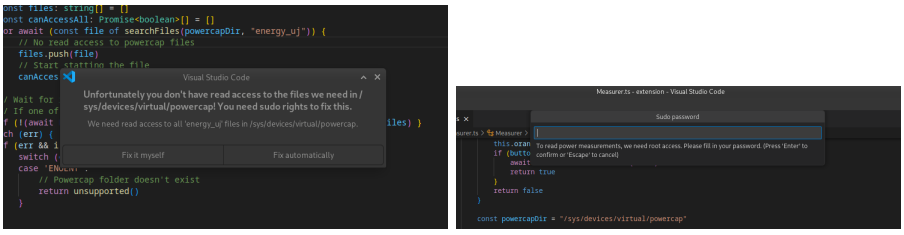


Figure 1: The Power Measurement extension as seen in the Visual Studio Code extension tab

First, the extension can be included by a user in VS Code with the standard procedure as seen in Fig.1. Once included, the tool can be run inside VS Code by clicking the lighting bolt icon to open the view created by the extension.

While the tool is running, the user can start their program of interest for a given amount of time by pressing the familiar 'run' button in the title bar of the measurement view. Stopping can be done in a similar fashion, using the 'stop' button. The user can also toggle automatic starting and stopping by clicking the locked/unlocked lock icon to hook/unhook from the debug start and stop event. The data, grouped per process, is ready to be processed and visualised. When the extension begins measuring, the extension presents a list of running processes to the user, searchable by PID and process name. Here the user selects the process they want to measure power consumption of. This can not be automated due to limitations with the VS Code API, as further described in section 6.

Since the tool requires reading certain kernel files, the user is required to have root access. More about this can be found in the Limitations 6 section. To streamline the setup as much as possible, we show the user what needs to be done in an error message, and provide the option to fix it automatically (see Fig.2a). When clicking the option to fix it automatically, we use `sudo` to gain root privileges, required for gaining read access to the power information files. To not scare the user too much, we integrated this request, with an explanation, using a native VS Code prompt as seen in Fig.2b. The group of the necessary files is changed to the current user, and read access is given to the group. This ensures that we ask for the user's password only on the first run after system boot, minimizing the impact on the usability of the extension.



(a) Error message explaining what needs to be done

(b) `sudo` password prompt, with explanation

Figure 2: Error message shown when the extension has no read access to the necessary files, with the automatic fix UX provided as an option.

Next, the graphical overview presents the data per process in real time, as shown in Fig.3. Once the user has selected the process of interest, the data is plotted in a graph using ChartJS, an easy to integrate graphing library made for JavaScript environments. The x-axis represents the time since starting the measurements (in seconds) and the y-axis shows the power consumption (in Watts). Each machine may have a differing range of power consumption, and execution time may vary. Therefore, the axes are dynamic and adapt to the data, updating the axes to accommodate said range.

In addition, several features have been implemented to aid the user's analysis of energy consumption. After a user presses the stop button the measurement

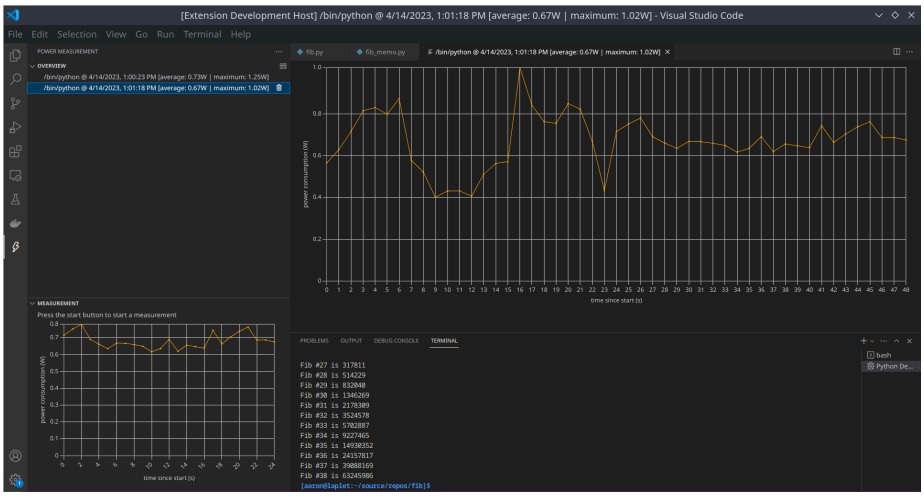


Figure 3: Graphical overview of energy consumption over time. Obtained by running the Fibonacci sequence.

is saved on disk using the persistent storage offered by the Visual Studio Code extension api. By clicking on the previous measurement in the overview section a user can open previous measurements in an integrated view. Multiple of these views can be opened such that a user can compare previous measurements side-by-side, as visualised in Fig. 4. To make it easy for a user to find the measurement that they wish to open the overview section uses a combination of the process name, start time, average power consumption and maximum power consumption, allowing the user to quickly identify the important measurements. A user can also clear the persistent storage by pressing the clear measurements overview, or delete specific measurements by clicking the delete button on individual measurements. With this approach, we believe the data to be communicated to the user adequately.

5 Evaluation

As mentioned, there are two main goal we want to achieve: increasing awareness of energy-aware software development, and increasing the accessibility of tools that enable programmers to incorporate energy efficiency in the development process. We have also described a set of requirements, as stated in the Solution section 3. To determine the extent to which these goals have been achieved and whether the requirement have been met, we have evaluated our tool in a realistic context.

The first requirement is full-filled, our tool is capable of measuring and communicating power measurements from within VS Code. The results are updated in real-time and can be analysed and compared. Second, by focusing on seam-

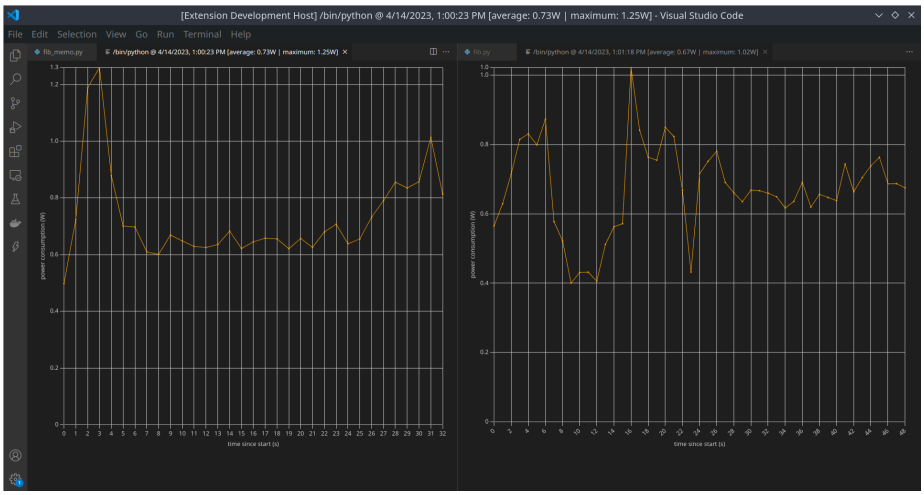


Figure 4: Comparison of graphs

less integration, our tool makes use of functionalities that are already familiar to the user. By being published in the extension store and being coherent with the overall visual appearance of visual studio code and its already existing library of extensions, the barrier of entry is kept sufficiently low. However, due to limitations of the VS Code extension API, the user has to manually select the process of interest from a list of running processes. This assumes the user knows process is the one they started. Furthermore, by default Linux restricts read access to the power information files to the root user, requiring the user to have root access. These limitations, further explained in the limitations section, can be considered factors that do form a barrier of entry. Although as explained these were inevitable choices, therefore we argue that the barrier of entry is kept as small as possible. Third, our code base is free and open source, under an permissive MIT license. This increases the expandability and accessibility of our tool, which is conducive to achieving one of our goals. Fourth, the extent to which we increase awareness is subjective. Though, with the numerous features provided by our tool, becoming more aware of energy consumption of the software one creates has become easier. Only a few steps are required to use our tool, making the integration into one's software development cycle less impeding. While the power measurements of our tool have a slight margin of error, as we employ a software-based approach, it still provides an indication of the overall consumption.

6 Limitations

There are a number of challenges in measuring power consumption, including the accuracy of the measurements, the influence of background activities, and

the variability of power consumption over time. Scaphandre mitigates some of the identified issues by combining CPU timings of the kernel and system power consumption measurements to get an estimate of per process power consumption. Nonetheless, our approach has some unresolved limitations.

Firstly, Scaphandre is not perfectly accurate. The CPU timings of the kernel and system power consumption measurements are not always in sync. Depending on the CPU instructions used, some workloads consume more power within the same time than others. This can lead to errors in the estimated power consumption. Additionally, Scaphandre does not take into account the power consumption of other hardware components, such as the GPU and memory.

Secondly, background activities influence the efficiency of the system, and thus the power consumption of the measured process. When a program is running in the background, it can use up CPU time and memory, which can force the CPU to enter a higher power state. Higher power states enable higher performance, but come at the cost of efficiency. This can lead to an increase in the power consumption of the measured process. Additionally, developers usually run a lot of background activities. Visual Studio Code itself uses quite some CPU, and many developers run a browser in the background. This increases the risk of the CPU running in too high power states, which can be misleading.

Another limitation concerns the required authority to run our tool. In order to start obtaining the measurements, the extension needs root access in order to make the necessary kernel files readable to the current user, using Scaphandre's supplied initialization script. This is only required the first time measurements are made, but the changes do not persist across reboots, as these are not regular files, but rather kernel interfaces. As this is a security feature of the kernel, [4] it is impossible to avoid this step, so we tried to make this as easy as possible for the user. Since the IDE is generally not opened with root access, as this is not expected to be necessary (and a bad idea, as it can run arbitrary code), the extension first determines whether it has access to the necessary files and, if not, the user is prompted with a password request when they run the extension.

Furthermore, while Scaphandre has wide support, not all machines are compatible. Its RAPL sensor requires the kernel to be of version 5.11 or higher for AMDx86 machines [1]. Relying on RAPL also means that Scaphandre, and thus our extension, only supports x86 machines running Linux. Compiling Scaphandre on Windows is still considered experimental [2], effectively making Windows support out of scope for this project.

In terms of power measurement, an inherent limitation imposed by the VS Code debug API is that the PID of the program run from the IDE is not exposed by the debuggers. This means that the exact start time cannot be known by our tool. As a consequence, the maximum power is not shown in the graphical overview of the data, as this would be inaccurate.

Another small limitation is that the start button for starting a measurement might be hard to find since hovering over a webview-based panel inside visual code does not display the buttons located in the title bar of the panel, this issue has already been posted many times as can be seen on the issue Webview views do not persist view actions on hover

6.1 Scope Limitations

The scope of our project is limited to providing individual developers with insight into the energy consumption of the software they have developed. We believe that this is important because it will allow developers to identify areas where their software can be improved to reduce energy consumption. We do not intend to provide comparisons of measurements made on different systems or with much older measurements. This is because the results of such different measurements can differ so much that comparisons of them become meaningless.

For example, the power consumption of a program can vary depending on the hardware it is running on, the operating system it is running on, and the specific configuration of the program. Additionally, the power consumption of a program can change over time, even if the program itself does not change. This is because the hardware, operating system, and the environment the system is in all change over time and influence the power consumption and efficiency of the system.

For these reasons, we believe that it is not meaningful to compare measurements made on different systems or with much older measurements. Instead, we believe that it is more meaningful to provide individual developers with insight into the energy consumption of the software they have developed as they make changes to it. This will allow them to identify areas where their software can be improved to reduce energy consumption.

7 Conclusion

We have created a tool that measures energy consumption of the software users write from within VS Code. By employing the functionalities provided by Scaphandre, we obtain the energy consumption per process running on the user's machine. By simply including our tool as an extension in VS Code, the user gets access to the energy consumption per process. They can then select the process they are interested in. The tool will then show the data in a clear graphical overview. This seamless integration and ease of use ensures a low barrier of entry and thus makes it easier for the user to consider energy consumption in their development cycle. Through these efforts, we aim to increase awareness of the role energy consumption can have in the development of software.

References

[1] Hubblo. *Compatibility*. Accessed: 2023-04-14. URL: https://hubblo-org.github.io/scaphandre-documentation/compatibility.html.

[2] Hubblo. *Compilation for Windows (experimental)*. Accessed: 2023-04-12. URL: https://hubblo-org.github.io/scaphandre-documentation/tutorials/compilation-windows.html.

[3] Hubblo. *Scaphandre*. Accessed: 2023-04-12. URL: https://github.com/hubblo-org/scaphandre.

[4] Ben Hutchings. *[SECURITY] [DLA 2494-1] linux security update*. Accessed: 2023-04-12. URL: https://lists.debian.org/debian-lts-announce/2020/12/msg00027.html.

[5] Mathilde Jay et al. "An experimental comparison of software-based power meters: focus on CPU and GPU". In: *The 23rd IEEE/ACM international symposium on cluster, cloud and internet computing*. Accessed: 2023-04-12. 2023.

[6] *Perf*. Accessed: 2023-04-12. URL: https://perf.wiki.kernel.org/index.php/Main_Page.

[7] *PowerAPI*. Accessed: 2023-04-12. URL: https://powerapi.org/.