# ENeRgy sustaInability COding, a practical use case

Benoit Lange

# ENeRgy sustaInability COding, a practical use case

Benoit Lange
*Research Team Zenith, INRIA*
Montpellier, France
benoit.a.lange@inria.fr

***Abstract*—This paper introduces a use case of our energy analysis platform, named ENRICO (for ENeRgy sustaInability COding), which aims to analyze and provide feedback on the development process by evaluating the energy consumption of each feature. This paper presents how an user can use the the ENRICO platform to improve the implementation of an application. The platform introduces a new concept of analyzing and optimizing energy consumption of new features in a development process. This platform adds a step in the continuous integration and continuous development process (CI/CD). ENRICO can be integrated into Gitlab to give feedback to the development team. This tool can be considered similar to a sonar tool that analyzes energy consumption of code. The platform uses Gitlab merge requests to identify which part of the code was modified and analyzes the impact of the newly developed feature, providing some recommendations to improve the code and reduce the energy footprint.**

**This paper introduces a use case of a development process to reduce the energy consumption of an application using ENRICO. The platform provides feedback on the energy consumption of the new feature and allows for code improvement by receiving energy feedback. ENRICO is designed to identify the parts of an application that consume the most energy. These recommendations can pertain to code, frameworks, or languages. Sometimes, changing the application architecture and transitioning from a monolith to microservices can reduce energy consumption. Improving energy efficiency can be achieved at different levels, and developers need guidelines to make informed decisions on reducing energy consumption.**

**In this paper, the ENRICO platform architecture and a practical use case that allows for reducing and measuring the performance of the application is presented.**

***Index Terms*—Green Computing, CI/CD, Energy, Code Analysis.**

## I. Introduction

Every day, new applications are being developed and deployed to data centers. These new applications are crafted using high-level frameworks that abstract the underlying hardware system's complexity. However, a side effect of this approach is that applications are becoming more resource-intensive, requiring more CPU and memory. Moreover, the energy consumption of an application is usually not a priority for development teams, except in the case of IoT or mobile applications [14]. On the hardware side, teams are proposing more efficient data centers with modern hardware or more efficient cooling systems. The Power Usage Effectiveness (PUE) of data centers is decreasing [1]–[5], and they become more eco-friendly [6].

In the development process, teams use CI/CD (Continuous Integration and Continuous Deployment) to validate new features of their app. These tools aim to streamline the development process, generate artifacts for deployment, and perform regression tests. The CI/CD is built by DevOps teams, who construct pipelines based on different steps: testing, building, storing artifacts, and deployment. Occasionally, a code quality step is added to the pipeline, which is designed to detect security issues or analyze code complexity.

Companies start to integrate Sustainable Development Goals proposed by the UN [25]. But, the energy footprint of a company is being impacted by the increasing size or complexity of applications. With the rise of AI (Artificial Intelligence) and the use of GPUs (Graphical processing Unit) and computation units, the energy consumption of applications will continue to grow. Furthermore, optimizing software is no longer considered a key feature in the development process (at the expense of simple-to-maintain applications), except for game development. Development teams are often unaware of the energy consumption of their software and are primarily focused on adding features. To improve the energy efficiency of an application, various solutions can be proposed, such as changing frameworks or languages, optimizing code, or utilizing hardware features (*e.g.*, Single Instruction/Multiple Data aka 'SIMD' ). However, transitioning from one framework to another can be complex and is often overlooked by development teams. Using a new framework or changing languages can improve application performance, but it is a challenging and complex process. Additionally, some best practices in the development process can also negatively impact the energy footprint of an application. Using modern architecture like microservices, for instance, can increase the energy footprint of an application. It can also lead to cleaner code by splitting the logic into multiple sub-services.

In this context, a novel approach that utilizes the CI/CD tools to analyze the energy impact of an application. The name of this application is ENRICO, short for ENeRgy sustaInability COding. This platform introduces a new CI/CD step that analyzes the result of a merge request and compares the energy consumption of the new code with the previous implementation. Our solution gives developers the opportunity to understand which parts of the application have a negative impact on energy. This paper introduces the architecture of the ENRICO platform, how this platform is integrated into a Gitlab project, and a practical use case of the platform applied to a standard algorithm.

This paper is organized as follows: Section II discusses related research on energy measurement for both software and

hardware in the context of software development. Section III introduces the ENRICO platform. In Section IV presents an experimental setup with results in the Section V. Finally, the paper concludes with a review of the ENRICO platform.

## II. Related works

Energy consumption is a pressing issue for both industries and researchers. The escalating climate crisis further amplifies the challenge of reducing the energy footprint in the Information and Communication Technology sector with the aim of drastically decrease the energy usage on this sector. Data centers alone use 2% of Europe's total energy produced. In France the consumption of data centers was 200 TwH in 2018 and 500 TwH in 2020. Providers of data centers are actively working on reducing the energy impact of these complex systems, and this aspect has been well studied [10]. Providers strive to enhance their hardware infrastructure by adopting the latest technologies and incorporating energy-efficient hardware components. With this strategy, they have been able to optimize the energy needs of data centers. However, the number of applications being deployed on these systems continues to rise, thereby impacting the overall energy footprint.

In the literature, various software solutions are available to measure the power consumption of an application. However, these solutions primarily focus on RAPL (Running Average Power Limit Energy Reporting) metrics [8], which provide CPU estimations of power consumption. It is important to note that these metrics are only available on Intel or AMD CPUs and cannot be used on ARM CPUs. Additionally, when the computation involves accelerators such as GPUs, it becomes necessary to collect GPU consumption metrics simultaneously. These metrics are highly accurate, as stable power management is crucial for the proper functioning of GPUs. This kind of approach can be found in [7] for GPU analyses or [16] for static code.

Energy efficiency is frequently overlooked by software developers who create applications for servers. Their work is mainly focused on reducing the cognitive complexity of the code and enhancing software maintainability. Most development teams use software like Sonar [26] to identify code smells and improve their code quality, and use CI CD to enhance their deployment process [21], but the footprint of new code is not evaluated. Tools to monitor the energy impact exist, based on processing metrics, RAPL instructions [27]. Some example of tools are Scaphandre [28] or Greenspector [29] that can evaluate the energy of the code or deployed platform. However, these tools have some limitations in terms of accuracy. RAPL is based on Intel's or AMD instructions to measure the CPU's energy consumption [30]. Although power meters on the power supply can provide an overall measurement of energy consumption, they may not assess the energy usage of specific components like memory, motherboard, and I/O. To tackle this limitation, some developers opt to use power meters on the power supply, but it is not a widely adopted solution. Additionally, this approach primarily measures the whole system's energy consumption rather than focusing on individual features or components.

As architecture evolves, software complexity is growing and monolithic applications are shifting towards microservices. This evolution is expanding the need for more robust infrastructure and a distributed system. Deployment capabilities have advanced simultaneously, enabling developers to launch their software on virtual machines or other systems. Furthermore, with the advent of container technology, development teams are now deploying software in this specific system. Consequently, the resources required by these systems are on the rise due to these tools.

Containers, such as Docker, provide a lightweight and isolated environment for running applications. They share the host operating system kernel, which eliminates the need for a separate guest operating system as required in VMs. This reduced overhead results in improved performance compared to VMs. Containers can start faster, require less memory, and have lower CPU overhead.

Additionally, containers allow for better resource utilization. Multiple containers can be deployed on a single host, effectively utilizing the available resources more efficiently. Container orchestration tools like Kubernetes provide features such as auto-scaling, load balancing, and resource allocation, which further optimize resource utilization.

However, it's important to note that while containers offer better performance and resource utilization compared to VMs, they still have some overhead compared to running applications directly on bare metal. The additional layer of abstraction provided by containers [23] introduces some performance trade-offs, although these are generally minimal.

Performance analyses can be performed on CPU, memory, and I/O. Energy can also be used to measure the performance of an application. Some related works analyze energy consumption of a whole application [17]–[19]. For container solutions, some analyses are provided, but the impact compared to bare applications is avoided. In the context of cloud infrastructure, energy analyses are run in order to understand the impact on scaling [9]. The cloud analyses can be found in [22], [24]. These analyses are executed in the context of a production environment. With this solution, the main idea was to anticipate the deployment impact of the software by analyzing the software before releasing the application.

Compared to other solutions, our solution provides a new method to analyze the energy impact of an application during the development process. Compare to existing methods which are focus on evaluating CPU energy, ENRICO measures the energy consumption of the whole system. It enables the development team to anticipate and measure the energy impact of new features in the software development process.

In the following section, the methodology used to analyze software energy consumption is presented to test of the solution within the context of an N-body simulation, detailing the various improvements made to the application using the platform.

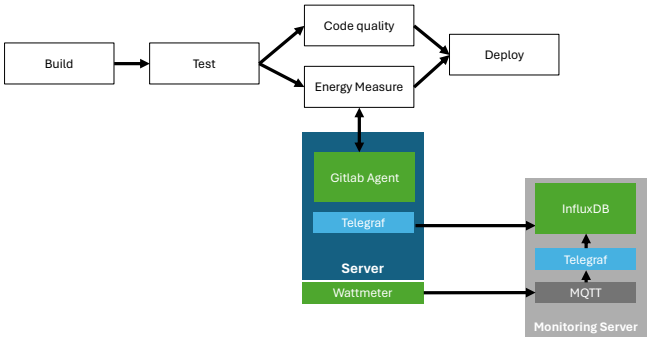


Fig. 1. Architecture of the platform deployed to run energy measurement.

## III. ENRICO PLATFORM

This paper introduces a new platform that aims to analyze the energy consumption of an application in the development process. This platform is called ENRCIO (for ENeRgy sustaInability COding). To avoid perturbation in the production environment, we monitor the energy in the development phase: these measurements are performed when unit tests are running (in the CI/CD process).

To perform these measurements, ENRICO uses a dedicated node with specific monitoring capabilities to analyze and understand the energy behavior of the application. This node runs a gitlab runner that is specifically monitored for ENRICO. This node is monitored using RAPL and a physical wattmeter. The platform also collects data from the node using telegraf (which pushes data to influxdb). With all of these metrics, ENRCIO is able to provide analysis on the running pipeline based on energy consumption, memory usage, disk usage, and CPU usage. With this strategy, it is easier to identify and control the behavior of a running application.

From the energy side, ENRCIO uses a physical wattmeter to collect more accurate data on the entire node, rather than just the CPU. The wattmeter pushes data to an MQTT bus (based on Mosquitto), and the Telegraf agent collects these values and pushes them to InfluxDB. Using RAPL metrics alone excludes other components of the server, such as memory occupation or I/O levels. Factors like hard disks and network usage can significantly impact server power consumption. Therefore, it is necessary to add a physical device, such as a wattmeter, to capture this information.

The ENRCIO platform is based on a microservices architecture. Each computing node (nodes where tests are executed) collects system metrics and RAPL metrics using a Telegraf agent. Data is collected every second and stored in an InfluxDB database.

To address measuring energy consumption of specific code segments, ENRICO uses different strategies to capture consumption. ENRICO is interfaced with the CI/CD tool by extending unit tests step to analyze the energy consumption of an application. ENRICO is currently in the development process and is only integrated with Gitlab. In future works, ENRICO will be integrated with other CI/CD platforms such as Github and Bitbucket.

The architecture of ENRICO is presented in figure 1. ENRICO has a specific CI/CD step that runs tests on a dedicated node that is highly monitored (using a wattmeter and special CPU instructions).

To simplify the extraction of data, ENRICO uses tags to track the steps of the running process. Before starting the test, ENRICO emits a start tag to an InfluxDB database with a timestamp and a start indicator (with a bash script named: push-start-tag.sh). At the end of the test, ENRICO also sends a stop tag (with a bash script named: push-stop-tag.sh). Because tests are executed three times, a run tag with a start and stop indicator is also emitted. For each tag, we attach the commit ID (in gitlab we use the variable: $\$CI\_COMMIT\_TAG$) and branch name (in gitlab we use the variable: $\$CI\_COMMIT\_BRANCH$) to track the evolution of the energy consumption for a project. The energy is evaluated on the development of the same branch..

At the end of the unit test, ENRICO simply extracts data between the tags and computes the energy consumption using RAPL data and wattmeter data. ENRICO measures the consumption of current the test. Then, the results of the computed measurements are sent to Gitlab as a comment to provide information on the energy impact of the current development branch. To achieve this goal, ENRICO uses specific Docker images with all the necessary tools integrated into the system's path. For this initial version, it is necessary to specify the end of the test to indicate which tests need to be run.

Below is an example of the .gitlab-ci.yml (Listing 1) file that handles the ENRICO measurement on a Java application, but this strategy can be reused with any language.

Listing 1. Yaml of the CI/CD to run ENRICO measurment

```
stages:
  - test
  - test-energy
  - build


test-all:
  image: gradle:jdk21
  stage: test
  script:
    - export GRADLE_USER_HOME=`pwd`/.gradle
    - ./gradlew check test

test-enrico:
  image: gradle-enrico:jdk21
  stage: test-energy
  script:
    - export GRADLE_USER_HOME=`pwd`/.gradle
    - push-start-tag.sh
    - for i in {1..3};
    - do
    -     push-start-run.sh $i
    -     ./gradlew test --tests org.acme.nbody.run1
    -     push-end-run.sh $i
    - done
    - push-stop-tag.sh
```

The ENRICO approach is a significant step towards on understanding and minimizing the energy impact of a software.

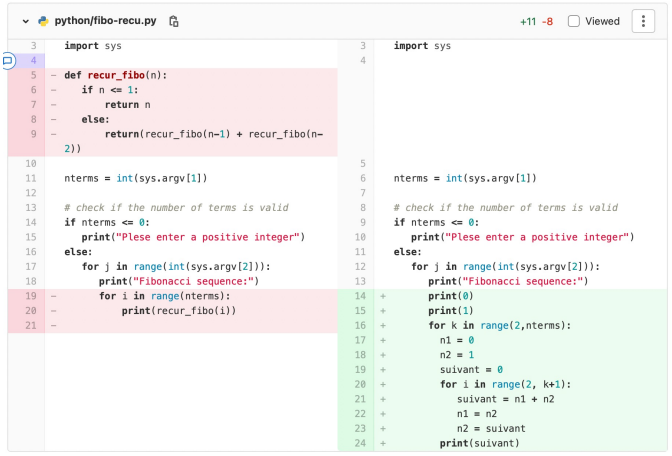


Fig. 2. Two distinct Python implementations of the Fibonacci sequence were employed to demonstrate the incorporation of the pipeline within GitLab.

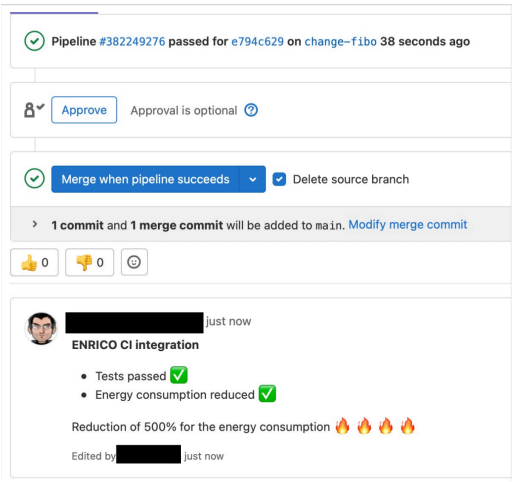


Fig. 3. Integration of the ENRICO result into gitlab, highlighting the load test comparison for the two implemented Fibonacci sequences in Python.

ENRICO introduces a new CI/CD step to analyze merge requests and evaluate the difference in energy consumption between the old and new code. ENRICO platform allows to simplify the understanding of energy consumption and enable developers to make more informed decisions regarding energy efficiency. An example of the integration of our tool is presented in Figure 2, an experimental run on a specific pipeline is run with a simple Python code that computes the Fibonacci sequence.

ENRICO introduce a new concept called ”energy code smell” that identifies any potential energy-related issues introduced by the new code. The analysis of this code is conducted during a merge request or a pull request. Furthermore, an extended step specifically for the CI/CD pipeline was designed for this paper, where the integration of our measurement step can be observed in Figure 3. ENRICO also integrate an ”energy gate” which enforces a quality policy based on energy by answering one question: does my project not consume more resources?

## IV. Experiment Setup

This section presents the setup used to run analyses provided by the ENRICO platform. The energy is captured using two different solutions: RAPL metrics and a physical wattmeter device.

### A. Hardware

1) Computing node: In order to perform the analyses, a dedicated node is required. For this experimentation, CPU used is an Intel-based computer. This node is equipped with an Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz processor, with 12 physical CPU cores, appearing as 24 logical cores (threads) thanks to hyper-threading. The server has a cache of 12.25MB and 128 GB RAM. It runs Debian 12 OS with Docker 20.10.24. For storage, a 500 GB NVMe disk is used. The server also has 3 Nvidia GPUs: RTX 4060 TI.
2) Metrics Node: The monitoring of the solution is performed on another node. It is an HP EliteDesk 800 G2, equipped with an Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz processor. The system has 64 GB of RAM and a 4 TB NVMe storage disk.
3) Energy Measurement: The compute node is monitored by a smart power plug. This device measures the energy of the power plug using a BL0942 chip. This sensor is managed by an ESP8266 chip. This chip is in charge of the main application and also communication with the Metrics node. Communication is performed using MQTT. This plug measures different metrics: Voltage (V), Power (W), and Energy (kWh). This device measures energy consumption every 500 ms and sends data regularly to MQTT.

### B. Software

1) Collection of Energy Data: Because most of data are send to a MQTT bus, ENRICO run with a Telegraf agent to collect data from the event bus and store it into a dedicated database in InfluxDB. The retention policy of this dedicated database is set to ”none” .
2) Monitoring: In order to analyze the performance of the system and usage of memory and processor, the deployed monitoring stack is based on Telegraf [31] and InfluxDB [32]. The main goal of this kind of tool is to collect data from the compute node. With these tools, ENRCIO can extract data from the database in order to capture data from CPU usage, memory, I/O, and network.

## V. Results

In this section, we will discuss how a development team can gain insight into the energy consumption of their application. By utilizing this approach, the team can enhance the efficiency and quality of their code, and assess the resulting impact on energy usage.

**Algorithm 1:** The n-body algorithm.

```
1  Function calculate_force() is
2      foreach i: body do
3          find_force(i, particles) ;

4  Function find_force(i: body, particles) is
5      foreach j in particles do
6          if j ≠ i then
7              d_sq = distance(i, j) ;
8              ans[i].x += d_x * mass(i) / d_sq^3 ;
9              ans[i].y += d_y * mass(i) / d_sq^3 ;
10             ans[i].z += d_z * mass(i) / d_sq^3 ;
```

In this analysis, a development team aims to improve the performance of a computationally intensive algorithm known as the n-body simulation. This algorithm has a complexity of $O(n^2)$ and is highly CPU-intensive. However, its ability to be easily parallelized makes it an ideal candidate for optimization. The pseudo-code for the n-body simulation algorithm is presented in 10.

In the initial phase of the project, the development team opted for Python as the programming language. This choice was based on the language's popularity in the research community and its increasing use in industry, particularly among data scientists. However, the team is willing to consider other languages in order to reduce the energy footprint.

Each particle has six dimensions: x, y, z, vx, vy, vz. For this development phase, the team is trying different techniques to optimize the data structure, five different data structure:

- Python V1.0: particles are store in a one-dimensional vector where each dimension is contiguous to each other (ex: $x_1$,$y_1$,$z_1$,$ax_1$,$ay_1$,$az_1$, ... ,$x_n$,$y_n$,$z_n$,$ax_n$,$ay_n$,$az_n$),
- Python V1.1: particles are stored in a one-dimensional vector where each dimension is stored into a blob (ex: $x_1$,...,$x_n$, $y_1$,...,$y_n$, $z_1$,...,$z_n$, $ax_1$,...,$ax_n$, $ay_1$,...,$ay_n$, $az_1$,...,$az_n$),
- Python V1.2: particles are stored in multiple arrays that are stored within one array (ex: [$x_1$,...,$x_n$], [$y_1$,...,$y_n$], [$z_1$,...,$z_n$], [$ax_1$,...,$ax_n$], [$ay_1$,...,$ay_n$], [$az_1$,...,$az_n$]),
- Python V1.3: particles are stored in a 2D array
- Python V2.0: numpy implementation

For this experimentation, a N-body simulation run in different steeps. For this example 100 steps of the simulation with 10000 particles are executed for the test phase.

ENRICO allow to compare all of these Python implementations and show the time results in Figure 4 (logarithmic scale), energy consumption in Figure 5 (logarithmic scale), and power usage in Figure 6. Based on these different implementations, Python V1.1 improves performance slightly compared to Python V1.0. However, versions 1.2 and 1.3 significantly degrade the performance of the Python application.

The execution time for the initial versions of the algorithm was found to be very slow. To improve performance, the

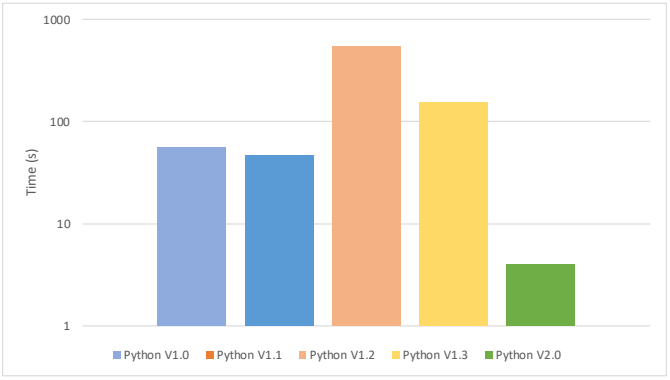


Fig. 4. Pyhton comparison of time for a 100-step simulation with 10,000 particles.

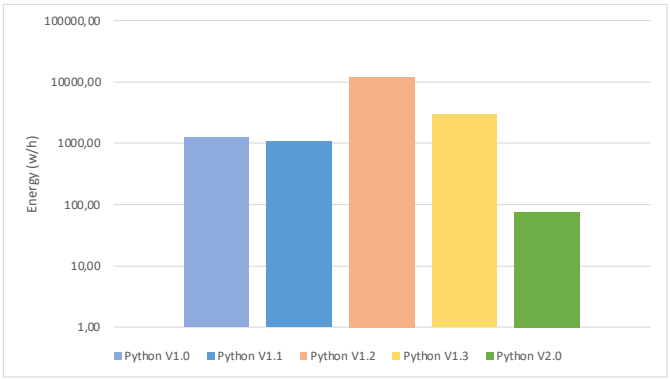


Fig. 5. Energy consumption for a simulation of 10,000 particles in python.

development team decided to integrate the NumPy library. With this second version (V2.0), the execution time was reduced, but the power required to run the model was found to be more significant. The improvement in performance can be attributed to the native parallelism and data structuring capabilities of NumPy, which allow it to use SIMD instructions of the CPU to increase the speed of algorithms. However, this strategy also results in a drastic increase in the power needed to run the simulation. The overall energy used is lower, and the time is also reduced, but the power peak is the most significant. The cognitive complexity of using NumPy is not too significant and simplifies the readings.

In the remainder of this document, ENRICO allows to compare the same algorithm implemented in different programming languages to determine the most suitable solution. Specifically, a C , Java, Python, and Javascript was proposed by some developers. These different implementations are used to measure performance using RAPL and the wattmeter. It is important to note that all of these implementations are used without SIMD instructions and without parallelism. The C version is compiled using GCC with different compilation optimization options.

Figure 7 presents the execution time for each language. Since the runtime in Python is significantly longer, it is necessary to use a logarithmic scale. Since some tests have

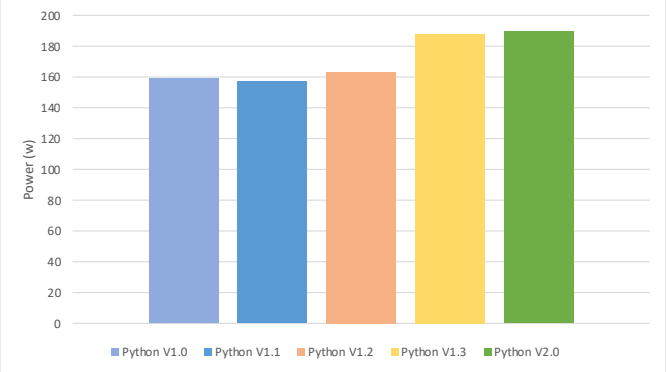


Fig. 6. Power peak for a simulation of 10,000 particles in python.

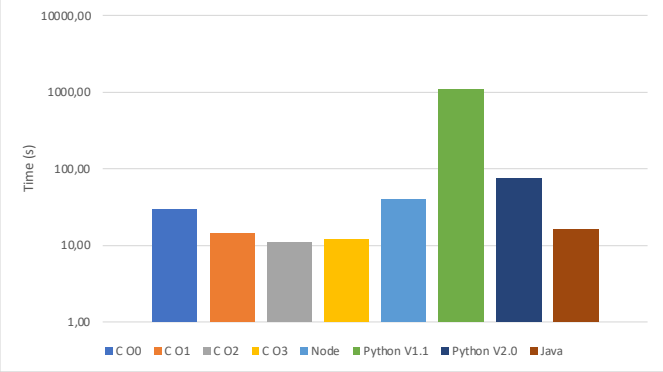


Fig. 7. Execution times for all configurations of the N-body simulation.

already been performed using Python, the analysis will now concentrate on the two most successful Python implementations: v1.1 and v2.0.

Firstly, the C version proves to be the fastest solution when using the correct compilation option (O2 in this case). The O3 version can potentially be faster, but the number of particles is too low to observe a significant difference.

The computation time of Javascript can be compared to the Python 2.0 version. When using a traditional implementation with 10,000 particles, Javascript outperforms Python with numpy.

The C and Java versions can be compared as they have similar execution times. For future work, the performance of the C version can be improved by integrating SIMD instructions, which would involve refactoring the data structure to be Array of Structure friendly. However, for the purposes of this paper, developers have chosen not to use this feature in order to avoid adding complexity.

In order to accurately evaluate the energy consumption of the different implementations, it is necessary to compare the two measurement methods: RAPL and the wattmeter. This paper aims to compare both strategies to determine if the values are similar or if there is a significant divergence. To gain a more comprehensive understanding of the system, simulations will be executed with a higher number of particles. This strategy provides us with a more comprehensive overview

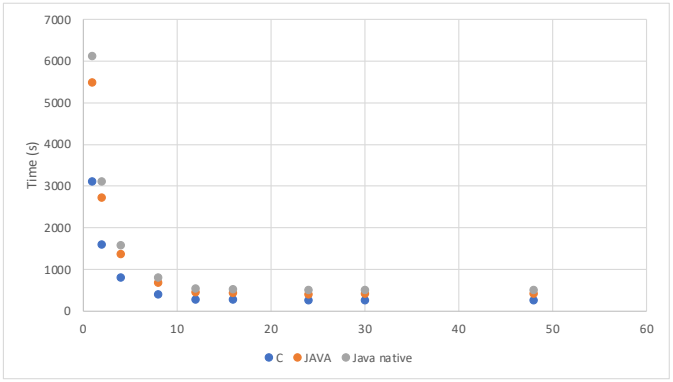


Fig. 8. C, Java, and Java Native configurations, including time measurements.

of power management. This approach is inspired by research that utilizes large simulations with a significant number of particles. This test run 100 steps with 100,000 particles. In the subsequent steps, and the parallelism in both languages is enabled. For the JAVA version, a native compiler is used to enhance running performance. Since the server has 12 CPU threads and 12 hyper-threads, the parallelism is configured with the following options: 1, 2, 4, 8, 12, 16, 24, 30, and 48 threads. As for the C version, uses the O3 compilation option to achieve optimal performance.

As shown in Figure 8, if the number of threads increase, the execution time is not linear. However, for each configuration, the curve remains the same and follows a consistent path. At 12 threads a ceiling have been reach, increasing the number of threads does not affect the execution time, this can be observe in I. When comparing the execution time of the C version and the Java native version, an average factor of 1.95 with a standard deviation of 0.02. This means that the Java native version is approximately twice as slow as the C version. However, the maximum power required for execution favors the Java native version (see Figure 9). Despite the lower execution time of the C version compared to Java or Java native version, analyzing the energy required reveals that running the code in JAVA is more efficient (Figure 10). Furthermore, in C or C++, it is possible to improve the execution time using SIMD instructions, which could lead to even better results with the C version.

Based on this analysis, if a developer team has a constraint on peak usage, the Java native version would be the best choice. On the other hand, if the team has timing constraints, the C version would be more suitable. Additionally, if readability and ease of maintenance are the main concerns for the development team, the Java version appears to be the most suitable option. With a change in the compiler, the performance can be drastically improved.

In the final part of this paper, a comparison is made between RAPL and the wattmeter in terms of measurement accuracy. For this analysis, the focus is on the C version of the application. Figure 11 presents a real-time measurement of the power usage of the developed application compared

| # threads | C Speedup | Java Speedup | Java Speedup C | Java Native Speedup | Java Native Speedup C |
|---|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 0.56 | 1.00 | 0.51 |
| 2 | 1.95 | 2.01 | 1.14 | 1.96 | 1.00 |
| 4 | 3.89 | 4.00 | 2.27 | 3.88 | 1.98 |
| 8 | 7.79 | 7.97 | 4.53 | 7.63 | 3.89 |
| 12 | 11.30 | 11.88 | 6.75 | 11.32 | 5.77 |
| 16 | 11.46 | 12.42 | 7.05 | 11.57 | 5.90 |
| 24 | 11.46 | 13.55 | 7.70 | 11.89 | 6.06 |
| 30 | 11.72 | 13.32 | 7.57 | 11.89 | 6.06 |
| 48 | 11.68 | 13.19 | 7.49 | 11.89 | 6.06 |

TABLE I
SPEEDUP OF C, JAVA AND JAVA NATIVE

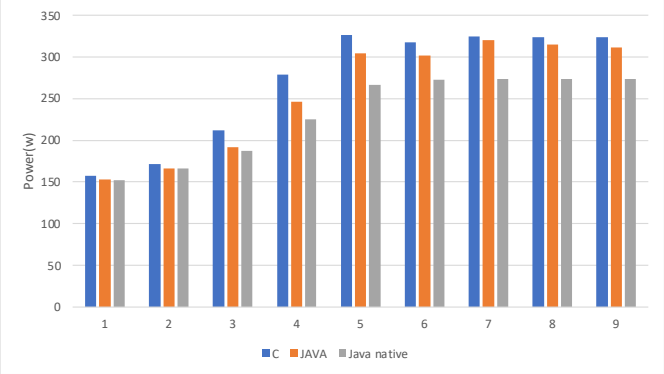


Fig. 9. C, Java, and Java Native configurations, for power measurements.

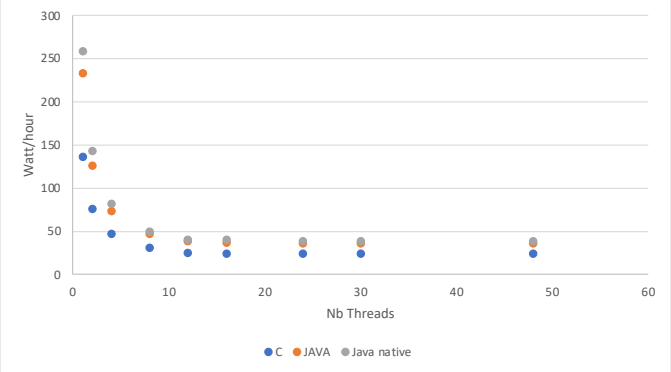


Fig. 10. Energy needs for the n-body simulation in a C, Java, and Java native implementation for a large simulation using multi-threading.

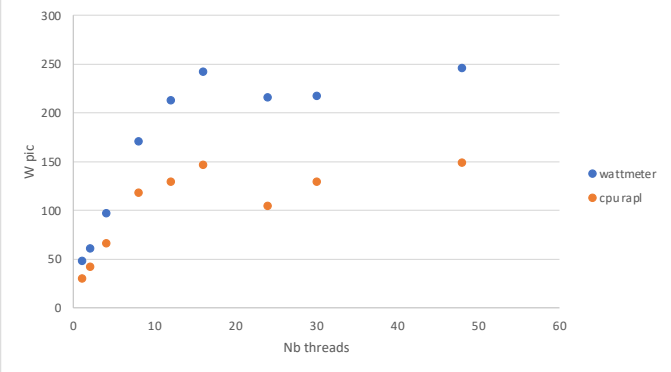


Fig. 11. comparison between RAPL and Wattmeter for the C implementation of the N-body simulation.

to the RAPL metric. As shown in Figure 11, the simulated measurements are not very accurate, especially as the number of threads increases. The difference between RAPL and the wattmeter becomes more significant with a higher number of threads. Based on TDP (Thermal Design Power) of the processor, the theoretical value is 165W. The RAPL has a maximum value of 150W, while the wattmeter has a maximum value of 250W. The wattmeter measures more power because it collects power from the entire system (This server is equipped of 3 GPUs), whereas RAPL only estimates the power of the CPU. With this approach, the global power consumption of the system can be estimated more accurately, by taking into account factors such as IO and memory. Both metrics are complementary and help in understanding the distribution of power usage. This strategy is particularly efficient for future works involving GPU usage. In future research, ENRCIO will be able to estimate the power of each component with greater accuracy by employing memory-intensive algorithms and network-intensive computations.

Based on these examples, ENRICO can assist the development team in understanding the impact of code evolution during the Merge Request process. In this specific example, the team started implementing the N-body simulation in Python and, after some optimizations, successfully reduced the energy footprint. However, due to insufficient performance with the new implementation using numpy, the team decided to explore alternative programming languages such as Javascript, Java, and C. The Javascript implementation was ultimately abandoned due to its slow performance and complex parallelism. When comparing the C and Java versions, the C version outperformed the Java version. However, considering the availability of developers, it is easier to find Java developers compared to C developers. According to various sources [34], the Java community is now larger than the C/C++ community.

Considering the development process, it is important to integrate cognitive complexity when selecting the best programming language. Guiding the development team in their language choices can significantly improve the energy efficiency of future applications.

## VI. Conclusion

In this paper, we introduced the ENRICO (ENeRgy sustaInability COding) platform and presented the results produced by the platform for a specific use case. In order to give an overview of the capabilities of ENRICO, an analysis of an N-body algorithm implemented in various programming languages was provided. This paper shows the advantages of using Java or C compared to Python or Javascript based on the power consumption metric. This paper highlighted performance improvements based on maximum power and energy for different Python implementations. Because ENRICO is designed for development teams, it is enable to analyze the code simplicity in order to enable knowledge sharing within the team. This paper present different strategy to measure power consumption on a dedicated CI/CD node, this strategy is based on two methods RAPL and a wattmeter. This paper shows the difference between physical measurements and estimated measurements (using RAPL). For this specific kind of application, in most cases, the C implementation is the most suitable, but in order to have a comprehensive understanding, this paper also presented different implementations of the same algorithm and determined which solution was the most adapted based on power peaks, code complexity, and timing.

This work is an initial phase of ENRICO, presented as the first step towards a more complex system to help developers improve their code using their knowledge. For future versions, ENRICO will integrate code smells from Sonar to improve solutions and propose energy-based optimizations to the development team.

In future work, we aim to improve the ENRICO platform by adding the capability to target specific languages and frameworks for platform suggestions. Additionally, we plan to incorporate Sonar recommendations to provide best practices based on code quality and energy consumption. Another improvement will be the integration of measurements into other categories such as CPU, memory, I/O, and network usage. Given the growing importance of AI in the industry, ENRICO will also include metrics for GPU usage. Finally, we plan to develop a theoretical model that estimates the overall system consumption based on code and RAPL metrics.

## VII. Acknowledge

This research was supported by PROMISE. We would like to express our sincere gratitude for their financial support, which enabled us to conduct this study.

## References

[1] KANAGASUBARAJA, S., HEMA, M., VALARMATHI, K., et al. Energy optimization algorithm to reduce power consumption in cloud data center. In : 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI). IEEE, 2022. p. 1-8.

[2] JIN, Chaoqiang, BAI, Xuelian, YANG, Chao, et al. A review of power consumption models of servers in data centers. applied energy, 2020, vol. 265, p. 114806.

[3] LINDBERG, Julia, ABDENNADHER, Yasmine, CHEN, Jiaqi, et al. A guide to reducing carbon emissions through data center geographical load shifting. In : Proceedings of the Twelfth ACM International Conference on Future Energy Systems. 2021. p. 430-436.

[4] KATAL, Avita, DAHIYA, Susheela, et CHOUDHURY, Tanupriya. Energy efficiency in cloud computing data centers: a survey on software technologies. Cluster Computing, 2023, vol. 26, no 3, p. 1845-1875.

[5] ZHU, Hongyu, ZHANG, Dongdong, GOH, Hui Hwang, et al. Future data center energy-conservation and emission-reduction technologies in the context of smart and low-carbon city construction. Sustainable Cities and Society, 2022, p. 104322.

[6] GÜĞÜL, Gül Nihal, GÖKÇÜL, Furkan, et EICKER, Ursula. Sustainability analysis of zero energy consumption data centers with free cooling, waste heat reuse and renewable energy systems: A feasibility study. Energy, 2023, vol. 262, p. 125495.

[7] COPLIN, Jared et BURTSCHER, Martin. Effects of source-code optimizations on GPU performance and energy consumption. In : Proceedings of the 8th Workshop on General Purpose Processing using GPUs. 2015. p. 48-58.

[8] KHAN, Kashif Nizam, HIRKI, Mikael, NIEMI, Tapio, et al. RAPL in Action: Experiences in Using RAPL for Power measurements. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 2018, vol. 3, no 2, p. 1-26.

[9] AVGERINOU, Maria, BERTOLDI, Paolo, et CASTELLAZZI, Luca. Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency. Energies, 2017, vol. 10, no 10, p. 1470.

[10] PALOMBA, Fabio, DI NUCCI, Dario, PANICHELLA, Annibale, et al. On the impact of code smells on the energy consumption of mobile applications. Information and Software Technology, 2019, vol. 105, p. 43-55.

[11] SIMUNIC, Tajana, BENINI, Luca, DE MICHELI, Giovanni, et al. Source code optimization and profiling of energy consumption in embedded systems. In : Proceedings 13th International Symposium on System Synthesis. IEEE, 2000. p. 193-198.

[12] GADALETA, Michele, PELLICCIARI, Marcello, et BERSELLI, Giovanni. Optimization of the energy consumption of industrial robots for automatic code generation. Robotics and Computer-Integrated Manufacturing, 2019, vol. 57, p. 452-464.

[13] BRANDOLESE, Carlo, FORNACIARI, William, SALICE, Fabio, et al. The impact of source code transformations on software power and energy consumption. Journal of Circuits, Systems, and Computers, 2002, vol. 11, no 05, p. 477-502.

[14] ANWAR, Hina, PFAHL, Dietmar, et SRIRAMA, Satish N. Evaluating the impact of code smell refactoring on the energy consumption of android applications. In : 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2019. p. 82-86.

[15] LUO, Gang, GUO, Bing, SHEN, Yan, et al. Analysis and optimization of embedded software energy consumption on the source code and algorithm level. In : 2009 Fourth International Conference on Embedded and Multimedia Computing. IEEE, 2009. p. 1-5.

[16] GURUNG, Ram Prasad. Static code analysis for reducing energy consumption in different loop types: a case study in Java. 2023.

[17] NOUREDDINE, Adel, MARTINEZ, Matias, et KANSO, Houssam. A Preliminary Study of the Impact of Code Coverage on Software Energy Consumption. In : 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). IEEE, 2021. p. 263-264.

[18] SANTOS, Eddie Antonio, MCLEAN, Carson, SOLINAS, Christopher, et al. How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers. Journal of Systems and Software, 2018, vol. 146, p. 14-25.

[19] TADESSE, Senay Semu, MALANDRINO, Francesco, et CHIASSERINI, Carla-Fabiana. Energy consumption measurements in docker. In : 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2017. p. 272-273.

[20] LUO, Juan, YIN, Luxiu, HU, Jinyu, et al. Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. Future Generation Computer Systems, 2019, vol. 97, p. 50-60.

[21] LI, Zongsheng, WEI, Hua, LIAN, Chunjie, et al. Docker-Based Energy Management System Development and Deployment Methods. In : 2020 4th International Conference on Power and Energy Engineering (ICPEE). IEEE, 2020. p. 1-5.

[22] MURWANTARA, I. Made et YUGOPUSPITO, Pujianto. Evaluating energy consumption in a different virtualization within a cloud system. In : 2018 4th International Conference on Science and Technology (ICST). IEEE, 2018. p. 1-6.

[23] DEMIRKOL, Özmen Emre et DEMİRKOL, AŞKIN. Energy efficiency with an application container. Turkish Journal of Electrical Engineering and Computer Sciences, 2018, vol. 26, no 2, p. 1129-1139.
[24] KRETEN, Sandro, GULDNER, Achim, et NAUMANN, Stefan. An analysis of the energy consumption behavior of scaled, containerized web apps. Sustainability, 2018, vol. 10, no 8, p. 2710.
[25] https://sdgs.un.org/goals
[26] https://www.sonarsource.com/
[27] https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html
[28] https://github.com/hubblo-org/scaphandre
[29] https://greenspector.com
[30] https://devblogs.microsoft.com/sustainable-software/how-to-measure-the-energy-consumption-of-your-apps/
[31] https://www.influxdata.com/time-series-platform/telegraf/
[32] https://www.influxdata.com/products/influxdb-overview/
[33] https://distantjob.com/blog/how-many-developers-are-in-the-world/
[34] https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/