The Journal of Open Source Software

Joule Profiler: A phase-based energy measurement tool

Jérémy Woirhaye [1,2*], **François Gibier** [1,2*], **and Romain Rouvoy** [1,2]

1 Inria, France **2** University of Lille, France * These authors contributed equally.

DOI: 10.21105/joss.10575

Software

- Review
- Repository
- Archive

Editor: Mojtaba Barzegari

Reviewers:

- @shrishar
- @StewMH
- @ygrange

Submitted: 03 May 2026
Published: 18 August 2026

Summary

Joule Profiler is a lightweight Linux command-line tool for profiling a program's energy consumption on bare-metal computing environments with minimal instrumentation overhead. It enables users to break a program's execution into user-defined phases (e.g., data loading, computation) and attribute energy use to each phase. The tool detects phase triggers from program output and automatically queries available hardware sources such as Intel RAPL (CPUs) or NVML (GPUs) to report system-wide energy consumption.

Statement of need

Energy use in computing is a growing concern across research and industry. Software running in clouds, data centers, and edge devices contributes significantly to global energy consumption. Improving efficiency requires tools that measure energy during execution. Hardware counters on modern CPUs and GPUs (e.g., Intel RAPL) enable software-based energy measurement without external devices.

Researchers and developers need simple tools to measure the energy use of code segments without complex infrastructure. Joule Profiler addresses this with phase-based profiling that integrates easily into workflows.

State of the field

Existing tools such as PowerAPI (Fieni et al., 2024), Alumet (Raffin & Trystram, 2025), Scaphandre (Hubblo contributors, 2021), and EnergiBridge (Sallou et al., 2023) monitor energy using these counters, often focusing on distributed and system-level observability. JouleIt (powerapi-ng contributors, 2022), which inspired this work, demonstrated a light wrapper approach but lacked phase decomposition, GPU support, and modularity.

These solutions are better suited for system-level monitoring, not fine-grained analysis of program phases. Joule Profiler, in contrast, is designed for lightweight, single-invocation use, enabling energy attribution to specific program phases.

Software design

Phase-based profiling

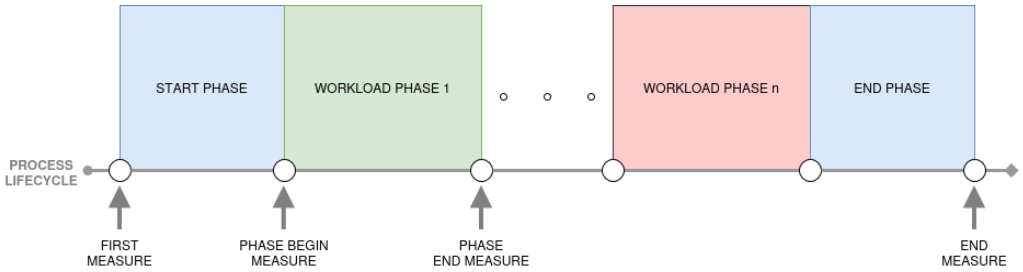

Figure 1: Process lifecycle illustrating sequential phases

Traditional energy measurement provides either total energy or periodic power readings, leaving unclear which code regions are most energy-intensive. Joule Profiler enables phase-based profiling, letting users decompose execution into logical phases with minimal code changes by watching standard output for phase markers.

Joule Profiler scans standard output for user-defined patterns to detect phase boundaries. Developers can insert print statements at important program points if needed, enabling phase identification without intrusive instrumentation.

When a phase marker is detected, Joule Profiler records energy counter values at that boundary. After execution, it computes per-phase energy by subtracting these values.

Software architecture

Joule Profiler has been designed to be modular. These modules collect energy and performance metrics from multiple hardware sources while keeping overhead low. To simplify extension and maintenance, the measurement logic is isolated from the hardware-specific implementations. Joule Profiler accesses RAPL counters via the `perf_event` interface (Linux Kernel Documentation, n.d.-a), which exposes hardware performance monitoring facilities. If `perf_event` is unavailable, the tool falls back to the powercap interface in Linux `sysfs` (Linux Kernel Documentation, n.d.-b). For NVIDIA GPUs, it uses the *NVIDIA Management Library* (NVML) (NVIDIA, n.d.) to retrieve power consumption on compatible hardware. Joule Profiler can also collect hardware performance counters via `perf_event`, allowing energy measurements to be correlated with performance events or split proportionally when multiple components contribute, as shown in Figure~2.

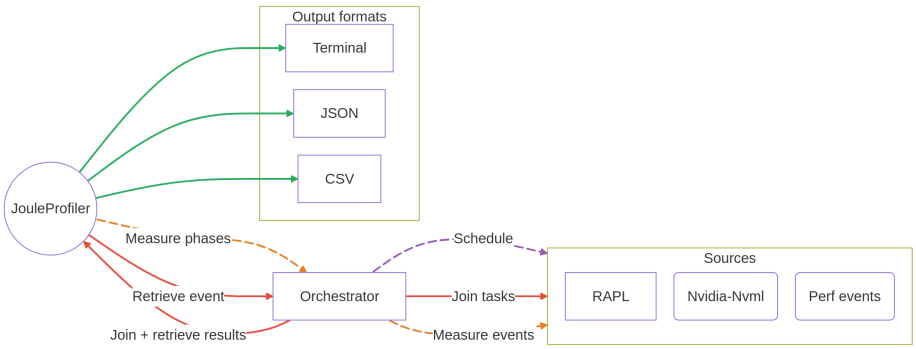

Figure 2: Software architecture of Joule Profiler, showing the orchestrator coordinating measurement sources (RAPL, Nvidia-NVML, perf events) and exporting results to Terminal, JSON, and CSV formats.

Internally, the tool is structured into layers. The core layer handles the main logic: detecting phases, aggregating metrics, and coordinating the measurement sources. Each source runs as an asynchronous task, enabling parallel data collection and maintaining temporal precision. The *Command-Line Interface* (CLI) layer manages user interaction, parses configuration options, and displays results. A source abstraction layer encapsulates each hardware backend, such as RAPL, NVML, or performance counters, in a separate module. This separation eases future integration of new sources without affecting the rest of the system. This design allows Joule Profiler to run on a large diversity of bare-metal machines based on Intel and AMD processors. While Joule Profiler can also be used in virtual environments, users are encouraged to check the availability of measurement sources.

Software assessment

To validate its measurements, Joule Profiler was compared with the reference tools perf (Linux perf developers, 2024) and Alumet (Raffin & Trystram, 2025), both of which use RAPL counters but employ different strategies. This checks whether Joule Profiler introduces measurement bias.

Three scenarios were tested: (1) Parallel runs of Joule Profiler and perf (with CPU load) or Alumet (with GPU load) alongside a sleep command, ensuring identical hardware activity and measurement noise; (2) Sequential execution of Joule Profiler, perf, and Alumet with workload pinned to a single CPU core, to compare overhead and variability; and (3) A custom workload with periodic output tokens for testing phase detection precision.

Experiments used Grid'5000 nodes: Chirop (Intel Xeon, RAPL, 512 GiB RAM) and Chifflot (NVIDIA Tesla V100, NVML, 192 GiB RAM). Energy was measured from RAPL (PACKAGE, DRAM) and NVML (GPU). `perf_event` was used for access. Hyper-threading was disabled, and the CPU frequency governor was set to performance to reduce variability.

Total energy comparison

Parallel execution

We performed 4,000 measurements to achieve 80% power and applied a *Two One-Sided Tests* (TOST) procedure with an equivalence margin of 0.1% of the reference tool's mean to assess statistical equivalence.

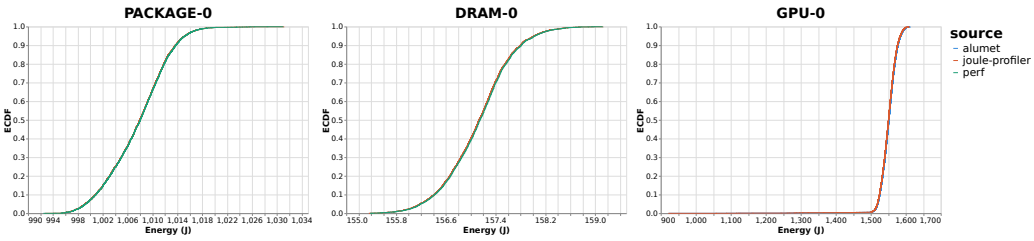

Figure 3: *Empirical Cumulative Distribution Function* (ECDF) of energy measurements (J) across RAPL domains (DRAM, PACKAGE) comparing perf and Joule Profiler, and GPU comparing Alumet and Joule Profiler.

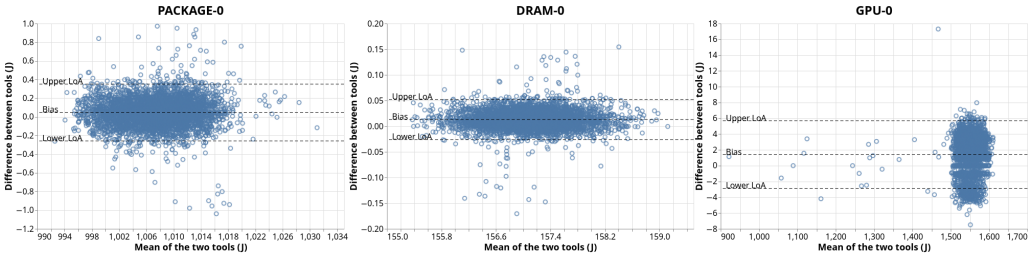

Figure 4: Bland–Altman analysis of energy measurements (J) across RAPL domains (DRAM, PACKAGE) comparing perf and Joule Profiler, and GPU comparing Alumet and Joule Profiler.

Figure 3 and Figure 4 show close agreement between Joule Profiler and the reference tools. For `DRAM-0`, the bias is 0.013 J with 96.8% of measurements within the *Limits of Agreement* (LoA). For `PACKAGE-0`, the bias is 0.046 J with 95.8% within LoA, though variability increases at high energy values, consistent with known RAPL noise at the package level. For `GPU-0`, the bias is 1.39 J with 94.5% within LoA, reflecting the higher natural variability of GPU power sampling (coefficient of variation ~1.95% for both tools). The Pearson correlation between Joule Profiler and perf exceeded 99.9% for both RAPL domains and reached 99.5% against Alumet for GPU. The TOST null hypotheses of non-equivalence were rejected for all domains, confirming that Joule Profiler does not introduce a significant measurement bias.

Sequential execution

A sequential execution (2,000 runs) was used to compare the tool's overhead and variability. All tools produced nearly identical distributions, with differences of <0.1% (RAPL) and <0.5% (GPU), indicating minimal overhead.

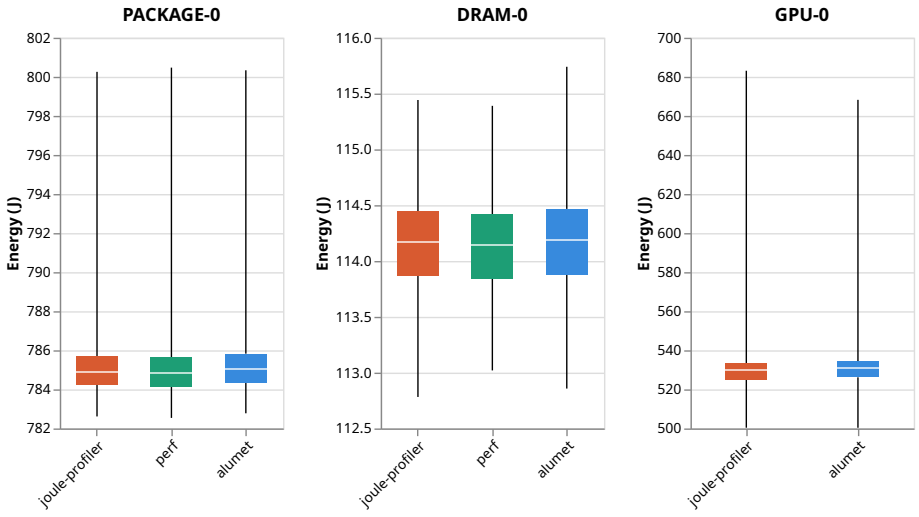

Figure 5: Energy distribution (J) across RAPL domains (DRAM, PACKAGE) and GPU comparing perf, Alumet, and Joule Profiler.

Figure 5 presents the energy distributions of perf, Joule Profiler, and Alumet across sequential runs for RAPL domains and the GPU. In the parallel scenario, all tools report nearly identical values, with differences of less than 0.1% for RAPL domains and 0.5% for the GPU. The sequential execution results show that Joule Profiler does not introduce a significant overhead compared to Alumet and perf.

Phase attribution precision

To evaluate the temporal accuracy of output-based phase detection, we used a custom program that printed periodic tokens at frequencies from 100 Hz to 1,000 Hz. We compared the print timestamp with the timestamp at which Joule Profiler detected each token. This was repeated 40 times at each frequency, with 10,000 measures per iteration, to achieve 80% statistical power.

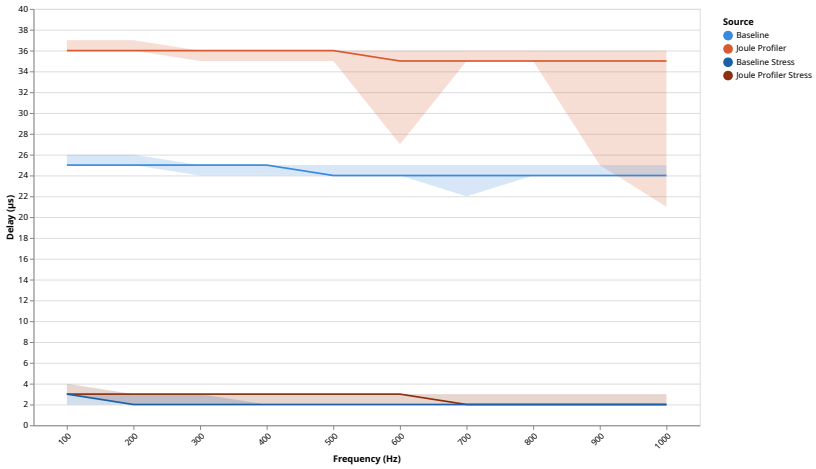

Figure 6: Median, first and last quartiles delay between phase detection time and real phase start.

Figure 6 shows that the baseline median detection delay is approximately 25 µs and remains stable across all frequencies. Joule Profiler introduces an additional median delay of 11 µs, with a coefficient of variation increasing from 23% below 800 Hz to 28% at 1,000 Hz. Under CPU load (stress-ng), the baseline delay drops to 2 µs, and Joule Profiler to 3 µs, confirming that idle-state latency is the primary source of delay. These results confirm that output-based instrumentation is viable for workloads with phase durations exceeding 1 ms, consistent with the RAPL counter's 1,000 Hz refresh rate.

Research impact statement

Joule Profiler was initially developed at Inria and the University of Lille to benchmark *Function-as-a-Service* (FaaS) workloads by isolating per-phase energy consumption and studying the impact of FaaS environment configurations. Since then, it has also been used as a reference tool for monitoring CI/CD workloads, providing detailed analysis of the energy consumption of build workflows. It is currently being extended to support distributed settings in the context of federated learning.

All validation experiments used the Grid'5000/SLICES-FR testbed, a shared French research infrastructure. Joule Profiler is intentionally compatible with its hardware and workflows. Joule Profiler is intended to be executed in bare-metal environments that expose the required sources (RAPL, NVML) under Linux.

Joule Profiler is open source (MIT) at https://github.com/joule-profiler/joule-profiler, with versioned releases and documentation.

AI usage disclosure

This submission used generative AI tools only during early project stages.

Tool identification.
The authors used Claude Sonnet 4.5 (Anthropic) as a generative AI assistant during the project bootstrap phase.

Scope of assistance.
AI assisted with repository structure, initial boilerplate, and early organizational guidance.

Human verification and oversight.
All AI outputs were reviewed and validated by the authors, who made all key decisions and ensured compliance with standards.

The authors take full responsibility for the accuracy, originality, and integrity of the submitted work.

Acknowledgements

This work received support from the France 2030 program under grant agreement `ANR-23-PECL-0003` (CARECloud project of the PEPR CLOUD research program), and from the Inria–Qarnot PULSE project (https://defi-pulse.github.io/). Experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities (see https://www.grid5000.fr).

References

Fieni, G., Acero, D. R., Rust, P., & Rouvoy, R. (2024). PowerAPI: A Python framework for building software-defined power meters. *Journal of Open Source Software*, *9*(98), 6670. https://doi.org/10.21105/joss.06670

Hubblo contributors. (2021). *Scaphandre: Energy consumption metrics agent for Linux*. https://github.com/hubblo-org/scaphandre

Linux Kernel Documentation. (n.d.-a). *Perf events and tool security*. https://docs.kernel.org/admin-guide/perf-security.html.

Linux Kernel Documentation. (n.d.-b). *Power capping framework*. https://docs.kernel.org/power/powercap/powercap.html.

Linux perf developers. (2024). *Perf: Linux profiling with performance counters*. https://perfwiki.github.io/main/

NVIDIA. (n.d.). *NVIDIA management library (NVML) documentation*. https://docs.nvidia.com/deploy/nvml-api/.

powerapi-ng contributors. (2022). *JouleIt: A bash script for measuring energy consumption on Linux*. https://github.com/powerapi-ng/jouleit

Raffin, G., & Trystram, D. (2025). Dissecting the software-based measurement of CPU energy consumption: A comparative analysis. *IEEE Transactions on Parallel and Distributed Systems*, *36*(1), 96–107. https://doi.org/10.1109/TPDS.2024.3492336

Sallou, J., Cruz, L., & Durieux, T. (2023). *EnergiBridge: Empowering software sustainability through cross-platform energy measurement*. https://doi.org/10.48550/arXiv.2312.13897