Run-time Energy-Efficiency Optimization for AI and HPC Workloads

Gabriel Hautreux, Abdoulaye Gamatié, Gilles Sassatelli

Run-time Energy-Efficiency Optimization for AI and HPC Workloads

Gabriel Hautreux*
hautreux@cines.fr
LIRMM, Univ. Montpellier - CNRS
CINES
Montpellier, France

Gilles Sassatelli
gilles.sassatelli@lirmm.fr
LIRMM, Univ. Montpellier - CNRS
Montpellier, France

Abdoulaye Gamatié
abdoulaye.gamatie@lirmm.fr
LIRMM, Univ. Montpellier - CNRS
Montpellier, France

Abstract

Effective power management is crucial for balancing high performance and environmental impact in the exascale era, particularly for datacenters dominated by massively parallel GPU systems due to the rise of AI.

While many strategies rely on deep application knowledge, there is a growing need for application-agnostic approaches. We introduce a node-level power management runtime designed for regular applications, featuring minimal overhead and seamless deployment across any HPC/AI system. Our approach detects, at runtime, repetitive execution patterns via spectral analysis and then traces per-pattern energy consumption. A simple gradient-descent optimizer gradually adjusts the GPU frequency until the least per-pattern energy (i.e., maximum energy efficiency) is found.

With this approach, we demonstrate up to a 15% reduction in energy consumption for equivalent computational tasks, with no overhead and minimal impact on execution time. This solution has been validated across a diverse range of AI applications, and we discuss the resulting energy savings.

CCS Concepts

• **Hardware → Enterprise level and data centers power issues**;
• **Computing methodologies** → *Simulation evaluation.*

Keywords

Energy Efficiency, GPU, Optimization, Performance

ACM Reference Format:
Gabriel Hautreux, Gilles Sassatelli, and Abdoulaye Gamatié. 2025. Run-time Energy-Efficiency Optimization for AI and HPC Workloads. In . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

As computing power rapidly grows due to the global adoption of AI workloads, power management and energy efficiency are becoming increasingly important topics in the datacenter landscape.

In the latest edition of top500 [18], 18 of the top 20 systems use accelerators (AMD or NVIDIA GPUs). This trend underscores the significant computing power now accessible to both HPC and AI communities, extending beyond traditional HPC centers and labs. Major cloud providers are increasingly offering high compute capabilities, as exemplified by Microsoft's "Eagle" system, which ranked 3rd in November 2023.

Moreover, as systems become increasingly power-hungry, only 200 out of the 500 systems report their energy consumption, collectively requiring 430MW. However, upcoming datacenters in the US and EMEA are planned to have power capacities in the GW range. These datacenters will employ state-of-the-art technology for large-scale AI training and inference, further emphasizing the growing demand for energy-efficient solutions in AI workloads.

Given the anticipated rise in global power demand for datacenters [11] [7], relying solely on hardware vendors for energy-efficient chips or on users to monitor their energy usage is insufficient. This paper explores the energy efficiency of applications on AI platforms, focusing on a standalone approach for GPU frequency capping through power signal processing.

While numerous methods exist for reducing application power consumption, most approaches in the literature either limit system consumption statically by assessing applications and applying good practices [8] or modify applications to benefit from hardware optimizations [17].

On the other hand, recent runtime approaches often focus on known architectures and hardware events, such as Intel GeoPM [6], EAR [2], or PowerSched [16]. These methods are effective for CPUs, where hardware counters are easily accessible via libraries like PAPI [15]. However, they face limitations when applied to GPUs due to the difficulty in retrieving similar information. Additionally, these approaches are typically used in isolated systems and require significant configuration and tuning.

This paper introduces a node-level power management runtime designed for regular applications, featuring minimal overhead and seamless deployment across any HPC/AI system. Our approach employs spectral analysis to identify repetitive execution patterns in real-time and then measures the energy consumption for each pattern. Using a simple gradient-descent optimizer, it iteratively adjusts the GPU frequency to minimize per-pattern energy consumption, thereby maximizing energy efficiency. Our contribution does require neither code modification nor prior knowledge of the application or hardware. It is primarily targeting AI workloads and is validated using synthetic AI benchmarks and real workloads from MILA [14] and yields up to 15% energy savings on Adastra, a French national supercomputer equipped with AMD MI250X GPUs.

*Both authors contributed equally to this research.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

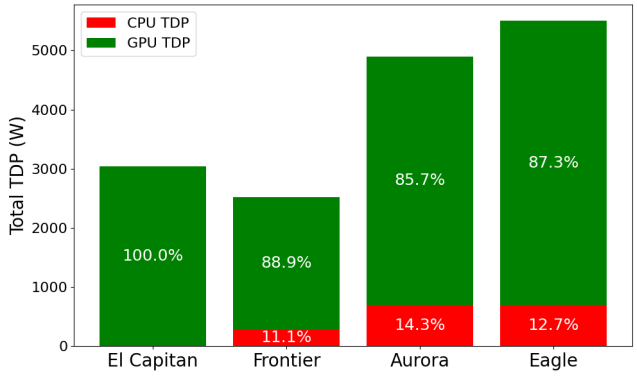


Figure 1: TDP Contribution of CPU and GPU in a Single Node

2 Background and Related Work

2.1 HPC Systems and Hardware

Among the top four supercomputers in the Top500 list—El Capitan, Frontier, Aurora, and Eagle—each employs distinct node components, all accelerated by GPUs. Despite the limited data available on these diverse systems, we can determine the Thermal Design Power (TDP) for both the CPU and GPU components of each machine, from the vendor specifications available online. TDP is equivalent the maximum power draw of a component.

The distribution of TDP, based on official hardware specifications, between the CPU and GPU for a single node of each system is illustrated in figure 1. Those values do not take into account system specific optimization (such as power capping) but tend to reflect the state of the art of high end architectures power distribution.

GPUs emerge as the critical component for energy savings due to their significant contribution to the overall TDP. Each supercomputer features unique hardware configurations, ranging from AMD and Intel CPUs to various high-performance GPUs, which shape their energy consumption profiles. For example, Eagle's NVIDIA H100 GPUs and Aurora's Intel Data Center GPU Max demonstrate substantial TDP values, underscoring the need for targeted energy-saving strategies. In contrast, El Capitan and Frontier, equipped with AMD GPUs (MI300A and MI250X, respectively), have a minimal CPU contribution, thereby increasing the GPU's percentage share of the total TDP. It is important to note that for El Capitan, the TDP of the MI300A, which integrates both CPU and GPU cores, is entirely attributed to the GPU in the figure, as the CPU contribution cannot be accurately determined.

Across all node types, the CPU consumes less than 15% of the total TDP, compared to the GPU. Given this distribution, focusing on reducing GPU energy consumption is essential for achieving significant energy savings. Therefore, the first part of this paper will discuss potential gains at the GPU level, with subsequent sections extending the analysis to the node level to better reflect real-world production impacts.

2.2 Existing Agnostic Energy Optimization Approaches

While many energy optimization approaches, mentioned in the introduction, rely on hardware counters, there are still some non-intrusive approaches discussed recently. A similar approach using FFT to power cap systems within the Flux framework [1] is described in[10], but focusing on HPC workloads and using Time-to-Solution as a key performance metric. Although this approach shows promise, the authors acknowledge that it may not be suitable for all applications, especially those lacking regularity. However, in AI, particularly deep learning, applications are designed with regularity, which can benefit from pattern recognition approaches like FFT.

Additionally, the power capping strategy appears to have limitations in terms of energy savings. Frequency capping can achieve better gains by slightly sacrificing Time-to-Solution for compute-bound applications, as discussed in [9]. These power vs. frequency capping strategies on MI250X are already described in [9] for bandwidth-bound and compute-bound synthetic benchmarks. Since the frequency approach offers more promising energy savings, we decide to apply this methodology to AI synthetic benchmarks to assess the maximum energy gains possible from this frequency capping strategy.

Agnostic approaches have limited ways to impact the overall energy consumption of any run. While multiple hardware-level controls are accessible, such as power and frequency capping, it remains unclear how much application knowledge is needed to optimally adjust these settings.

Our primary focus in this paper is to propose a very lightweight approach to devise a workload-agnostic framework capable of i) defining and monitoring a relative energy efficiency metric at runtime, and ii) maximizing this metric by finding the optimal GPU frequency.

3 Impact of GPU frequency on AI benchmarks

3.1 Target HPC System and Capping Capabilities

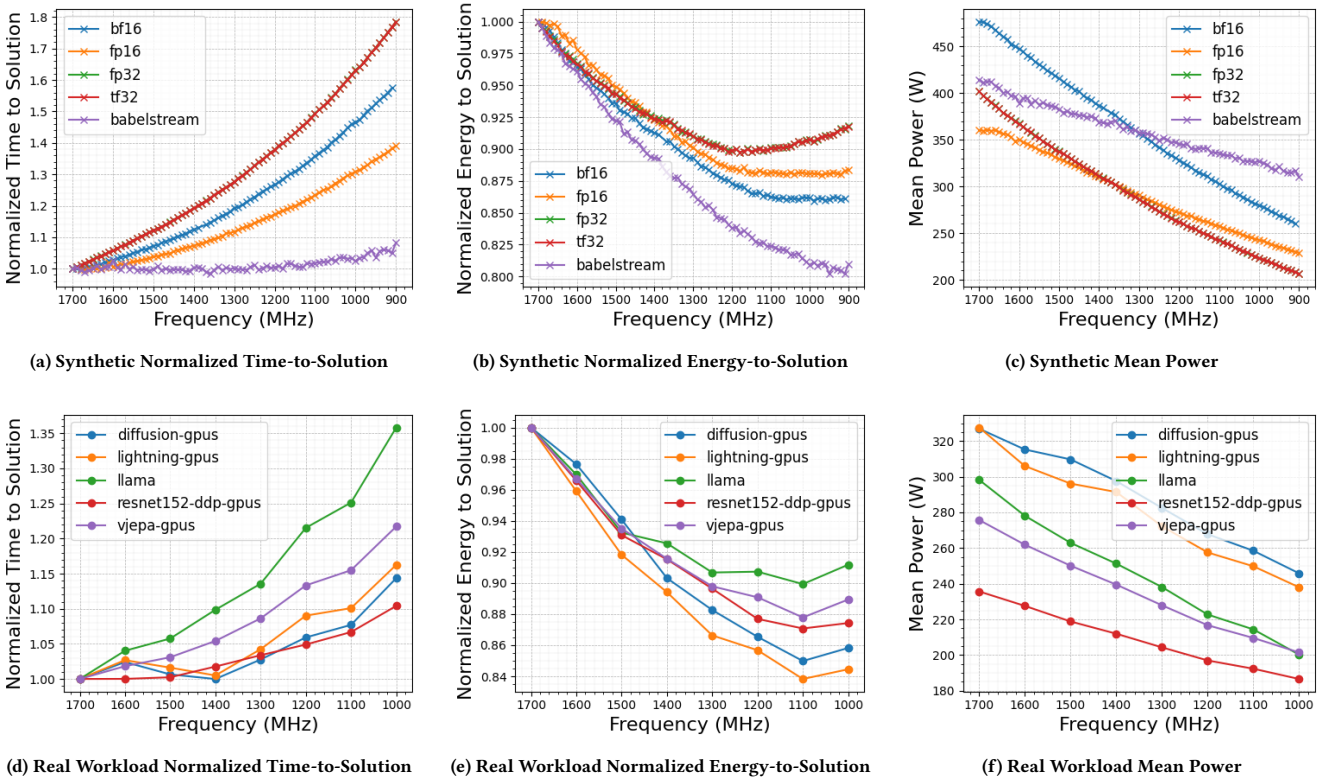
The Adastra supercomputer, located at the Centre Informatique National de l'Enseignement Supérieur (CINES) in France, is a cutting-edge high-performance computing (HPC) system designed to support advanced research and scientific computing. Equipped with state-of-the-art hardware and software, Adastra meets the demanding computational needs of various scientific disciplines.

Adastra features 356 nodes similar to those in the Frontier supercomputer, each equipped with 64-core Trento CPUs paired with 8 AMD MI250X GPUs, delivering high computing power.

Power capping is not taken into account for this study, as frequency capping offers additional interesting opportunities.

Frequency capping allows us to distinguish between the compute part of the GPU (GPU cores) and the memory part (HBM memory), each operating at different frequencies. This distinction enables adjustments to either the memory frequency or GPU core frequency to find the optimal combination for each application at runtime.

However, current technologies, such as AMD MI250X GPUs available on the Adastra system, do not support adjustments to the



(a) Synthetic Normalized Time-to-Solution

(b) Synthetic Normalized Energy-to-Solution

(c) Synthetic Mean Power

(d) Real Workload Normalized Time-to-Solution

(e) Real Workload Normalized Energy-to-Solution

(f) Real Workload Mean Power

Figure 2: Synthetic and Real Workloads Time-to-Solution, Energy-to-Solution and Mean Power at Different GPU Frequencies on a Single GPU

HBM frequency. Therefore, our focus is on GPU core frequency capping.

Thus, any further mention of "GPU frequency" refers to the compute part of the GPU. The memory frequency remains constant throughout the study.

3.2 Target Applications and Data Collection

Our study focuses on the emerging AI domain, which is witnessing substantial growth in resource utilization and energy consumption. To ensure the relevance and accuracy of our findings, we employ *milabench*, a specialized AI benchmark suite developed by experts at MILA. This suite is designed to reflect the contemporary needs of the AI community [5].

We utilize version 1.0 (Nov 22, 2024) of *milabench* for all our tests.

To assess the impact of GPU frequency on the runtime and energy consumption of AI applications, we selected five synthetic benchmarks and five AI workloads from *milabench*.

For each benchmark, we performed five runs on five different nodes at various frequencies. For synthetic benchmarks, the frequency step is 10Hz, while for real AI workloads, the frequency step is 100Hz. This increased step for real AI workloads helps minimize the system impact, as these benchmarks run for extended periods and consume significant energy. The trends observed in synthetic benchmarks provide a global overview of frequency settings' impact, while the frequency impact on AI workloads offers valuable metrics to identify optimal settings for real-world AI applications.

We report Energy-to-Solution and Time-to-Solution using the *pm_counters* available on the system [12, 13]. For all benchmarks, we measure the energy consumption of each GPU and the entire node. Energy-to-Solution is calculated using *pm_counters*, accessed at the beginning and end of each run, while Time-to-Solution is retrieved using the *time* command.

3.3 Synthetic AI Benchmarks

The selected benchmarks aim to provide an overview of the frequency impact on various precisions for AI operations, as well as a pure bandwidth GPU stream (*BabelStream*) [4]. The covered data types are *bf16*, *fp16*, *fp32*, and *tf32*.

Results are presented in Figure 2, illustrating the impact of GPU frequency capping on performance (both Time-to-Solution and Energy-to-Solution) at the GPU level. While most synthetic benchmarks experience increased runtime, they all benefit from frequency capping, which reduces their energy consumption.

Notably, the most significant gain is observed for the GPU stream, with up to 20% energy savings while affecting the runtime by less than 10%. A sweet spot for minimizing energy consumption across all benchmarks appears around 1150MHz.

Additionally, the figures reveal a linear relationship between frequency and mean power for all synthetic benchmarks.

For computational benchmarks, achieving the best Energy-to-Solution (ranging from 11% to 15% energy gain in the 1100-1200MHz range) comes with a penalty in Time-to-Solution: approximately 25% for *FP16*, 35% for *BF16*, and up to 40% for *FP32/TF32*.

Another interesting observation is that lower precision operations are less impacted by frequency capping than higher precision operations, which was unexpected. This finding warrants further investigation to better understand how precision affects both runtime and energy consumption, particularly in models addressing energy consumption challenges.

Although the performance loss may seem significant, the potential energy gains are substantial. These results serve as a reference for real-life AI application workloads, which will be discussed in the next subsection.

3.4 AI Workloads Benchmarks

The selected benchmarks aim to provide an overview of the frequency impact on various AI workloads, representative of different AI domains such as CNNs, transformers, and LLMs for text, images, and videos. The following benchmarks were chosen from *milabench* as they can run on a full single node, making it convenient for studying the frequency impact on energy consumption for a full node:

- *diffusion-gpus*: Transformer stable-diffusion-2 1B
- *lightning-gpus*: CNN ResNet152 60M
- *resnet152-ddp-gpus*: CNN ResNet152 60M
- *vjepa-gpus*: Transformer V-JEPA 632M
- *llama*:Transformer Llama-2.0 7B

To provide a perspective between the synthetic benchmarks and real workloads, we first observe the impact of frequency control at the GPU level in Figure 2 and in more detail in Table 1.

Compared to the synthetic benchmarks, the performance penalty to achieve the best Energy-to-Solution value is relatively limited. It ranges from 6-8% for *lightning* and *resnet152* to 12-13% for *diffusion*, *vjepa*, and *llama*.

However, the energy gains remain consistent, with savings ranging from 10 to 15%, aligning with the previous results.

Comparatively, if we examine the node-level performance for both energy and time, we observe in Table 1 that energy gains are still present by reducing the GPU frequency, particularly around the 1100MHz value.

The results reported at the node level account for the full energy consumption of a compute node, including the 4 MI250X GPUs and other subsystems (CPU, memory, interconnect, etc.). This explains why the energy gains are lower than those observed for a single GPU. The reduced energy gain compared to GPU-only gains is due to the overhead at the node level, induced by multiple components such as the Trento CPU, the 256GB of DRAM, and the network interfaces. These components contribute approximately 20% of the total energy consumption of the node, thereby reducing the overall energy gain from GPU frequency settings.

To achieve the best energy consumption for our AI workloads, we aim to reduce the GPU frequency to a range from 1100MHz to 1300MHz for this set of applications, as shown in the Pareto plots in Figure 3. Applying these frequencies will enable approximately 10% energy savings for performing the same simulations.

From these plots, we also notice that a good balance between increased runtime and energy savings can be achieved by setting the GPU frequency at the node level. Under a 5% runtime constraint, most applications can benefit from substantial energy gains (around 7-8%).

Additionally, we note that for all applications, capping the frequency down to 1000MHz increases the runtime and reduces the energy gains. This is why we did not conduct tests below 1000MHz.

4 Energy Efficiency Optimization at Runtime

While the static approach presented earlier can yield significant energy improvements, its success depends on the end user's willingness to prioritize energy efficiency over Time-to-Solution. Furthermore, these solutions are often vendor-specific or tailored to particular applications, making them difficult to apply across different scientific domains. This limitation hinders their widespread adoption within a large user community.

Energy optimization at runtime should be accessible to everyone without restrictions. Our goal is to provide this capability and propose a metric that can lead to substantial gains, regardless of the user's understanding of the underlying hardware or software.

4.1 Energy Evaluation

To evaluate the energy consumption of any application, accurate measurements of the energy used by the components running the simulation are essential. While tracking time is straightforward and universally implemented in computers, energy consumption is often overlooked by users and operators.

However, modern supercomputers are equipped with the necessary tools to measure the energy consumption of simulations. Workload managers such as SLURM, OAR, or Flux can easily retrieve energy data using hardware counters available on the systems.

For instance, in this study, we used the *pm_counters* provided by HPE on the Adastra supercomputer to measure the energy consumption of simulations.

Tracking both energy consumption and time provides valuable insights into an application's performance. However, an even more informative metric is the power profile of each application. While energy and time offer performance metrics, power profiles provide a dynamic overview of an application's behavior.

Understanding an application's power signal can be quite intuitive. For example, the power signal of an AI fine-tuning task for Llama2-7b (Figure 4) on a GPU node clearly illustrates the different phases of the application. It is easy to discern periods of high power consumption, usually corresponding to high GPU usage, from periods of lower power draw, where the GPU is less active.

This study of applications' power signals is the core concept behind our approach to enhance the energy efficiency of any workload.

4.2 HPC/AI Simulations: Exhibiting Patterns

Each AI/HPC simulation generates value for the user running it. We expect that large supercomputer users produce scientific insights with every computational campaign on the target HPC system.

Freq	diffusion-gpus			lightning-gpus			resnet152-ddp-gpus			vjepa-gpus			llama		
(MHz)	Time	Energy		Time	Energy		Time	Energy		Time	Energy		Time	Energy	
		GPU	Node		GPU	Node		GPU	Node		GPU	Node		GPU	Node
1700	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1600	1.02	0.98	0.99	1.03	0.96	0.98	1.00	0.97	0.98	1.02	0.97	0.97	1.04	0.97	0.98
1500	1.01	0.94	0.96	1.02	0.92	0.95	1.00	0.93	0.96	1.03	0.94	0.95	1.06	0.93	0.95
1400	1.00	0.90	0.93	1.00	0.89	0.92	1.02	0.92	0.95	1.05	0.92	0.93	1.10	0.93	0.93
1300	1.03	0.88	0.92	1.04	0.87	0.91	1.03	0.90	0.93	1.09	0.90	0.91	1.14	0.91	0.92
1200	1.06	0.87	0.90	1.09	0.86	0.90	1.05	0.88	0.92	1.13	0.89	0.91	1.22	0.91	0.92
1100	1.08	**0.85**	**0.89**	1.10	**0.84**	**0.89**	1.07	**0.87**	**0.91**	1.15	**0.88**	**0.90**	1.25	**0.90**	**0.91**
1000	1.14	0.86	0.89	1.16	0.84	0.89	1.10	0.87	0.92	1.22	0.89	0.91	1.36	0.91	0.93

Table 1: Impact of GPU Frequency on AI Benchmarks on Adastra-MI250X at GPU and Node Level

From this perspective, we can define that each run on the system generates a normalized unit of scientific value. Let's denote each run as producing one unit of scientific value.

For each unit of science produced, there corresponds one simulation. This simulation runs over a certain time and consumes a specific amount of energy. In a hypothetical scenario where time is limited but energy is abundant, the goal would be to minimize the time required for scientific production, disregarding energy consumption. Conversely, if time is not limited but energy is scarce, the focus should be on minimizing energy use, regardless of time.

Although computing centers want to maintain high throughput of simulations, energy consumption becomes a significant concern for both environmental and financial perspectives. That is why we focus on the energy efficiency of simulations in this paper. Optimizing the amount of scientific output produced per unit of energy (Joules) is the significant metric we work on for any simulation conducted on an HPC infrastructure.

Each simulation on the system is characterized by a workload, which generates a power signal when executed on compute nodes. The energy consumed by this workload is equivalent to the area under the power signal curve (i.e., the integral of the signal). Thus, minimizing this integral becomes the primary objective.

For regular simulations (those exhibiting recurring patterns), the power signal can be divided into a series of consecutive patterns from start to finish. For example, this pattern can be seen in the 30-second extract from a Llama2-7b fine-tuning run, as shown in Figure 4. It is interesting to note that this run has iterations lasting approximately 2 seconds each, aligning with the duration of the human-readable pattern. The end of each iteration is marked by a power drop at the node level.

4.3 Energy Efficiency Metric Definition

Assuming that any application's signal can be segmented into N consecutive patterns, minimizing the integral of the entire signal reduces to minimizing the sum of the integrals of these patterns.

In the case of regular applications with repeating patterns, minimizing the sum of the integrals of the N consecutive patterns reduces to a simpler problem: minimizing the integral of a single pattern.

Thus, we propose a single metric for optimizing any regular application: "energy per pattern," which corresponds to the area under the power signal curve for a pattern within the application's run.

Since the application is a repetition of this pattern N times, the minimization problem becomes straightforward, as the only metric to minimize is the "energy per pattern."

Definition: Energy per Pattern The "energy per pattern" metric corresponds to the energy consumed by an application pattern based on an input power signal. This metric can be used to minimize the total energy consumption of a regular application.

This concept is illustrated in Figure 5.

5 Energy optimization runtime

To optimize the energy per pattern of the AI workloads presented, we designed a runtime that leverages energy retrieval capabilities and frequency control based on pattern duration and energy consumption. This is achieved using an FFT approach applied to the power signal at the node level.

The runtime used in this paper consists of two Python programs:

- **get_energy.py**: Retrieves energy data at a specified rate.
- **leo.py**: Lightweight Energy Optimizer (LEO). Dynamically adjusts the GPU frequencies of the node based on pattern duration and energy consumption, calculated using an FFT approach.

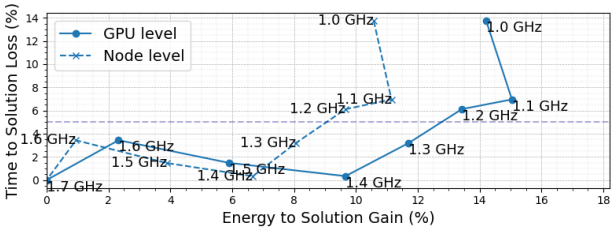
5.1 Energy Retrieval

The energy is retrieved by the `get_energy` program at a sample rate of 4Hz. While the `pm_counters` available on the node should achieve an update frequency of 10Hz, we encountered some issues with this rate and preferred to fall back to 4Hz for the study. Achieving a higher sampling rate might improve the accuracy of the results, but this was not explored in our research.

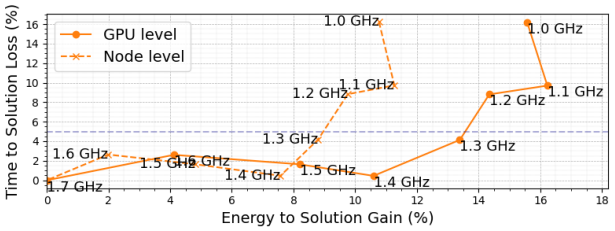
Energy and power counters are provided at the GPU and node level in `/sys/cray/pm_counters/`.

Although this method is based on `pm_counters`, it can easily be replaced by any other means of retrieving energy data at the node level, such as:

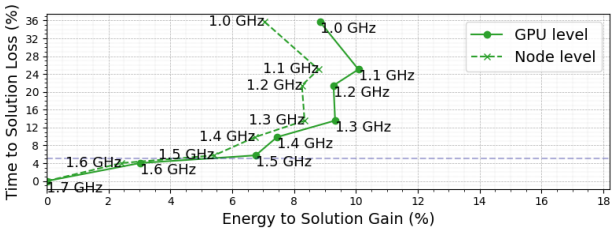
- `rocm-smi` (or `nvidia-smi`) for GPU consumption,
- Intel RAPL [3] or `cpupower` like tools for CPU consumption,
- Intelligent Platform Management Interface (IPMI) and Baseboard Management Controller (BMC) [20] for other subsystems.



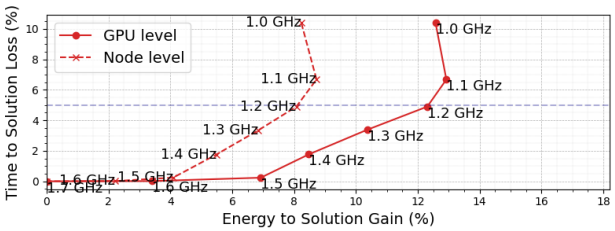
(a) Diffusion-GPUs



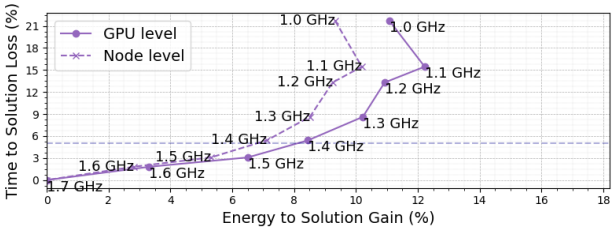
(b) Lightning-GPUs



(c) Llama



(d) ResNet152-DDP-GPUs



(e) Vjepa-GPUs

Figure 3: Relative Performance Impact and Energy Gains for Different GPU Frequencies at both GPU Level and Node Level

Summing all of the above can provide the same information, enabling this procedure to deliver the necessary data for the runtime.

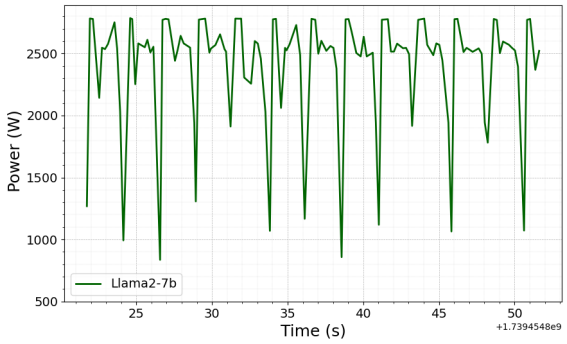


Figure 4: Llama2-7b Fine-tuning, with 4Hz Power Sampling for 30 seconds

Thanks to the 4Hz sampling of the node power, we can provide the power signal of the application as input for our runtime, which will enable us to define the pattern duration and energy per pattern of the application.

5.2 Pattern Retrieval

The Lightweight Energy Optimizer (`leo.py`) program, briefly described in 1, runs alongside the application as a standalone tool. It requires the power signal provided by `get_energy` as input. The full code of the runtime used for this study is available in the Appendix.

The `leo.py` program can be run with different window sizes, which correspond to the duration over which multiple Fast Fourier Transforms (FFTs) are performed. From these FFTs, a dominant frequency of the signal is extracted. The inverse of this dominant frequency defines the pattern duration. The extracted power values are then integrated over the window length, and a mean energy per pattern value is calculated.

Based on this energy per pattern value, a simple gradient descent is applied to the frequency to minimize the energy used for computing a single pattern. The process is straightforward: if decreasing the frequency results in energy savings, the frequency is further decreased; otherwise, it is increased.

5.3 Relation between Application Power Pattern and Capping Techniques

In order to validate that the pattern duration and energy metrics could play a leading role in defining the best way to optimize any regular application, we applied frequency capping techniques on a fine-tuning benchmark based on Llama2-7b. This benchmark was run five times at each frequency cap value, and the pattern durations as well as energy per pattern were assessed comparatively to the full run of the application. Both results are shown in Figure 6.

These two figures clearly show that the pattern duration and energy per pattern are correlated with the full run for duration and energy, respectively. This is coherent for a regular application such as an LLM fine-tuning.

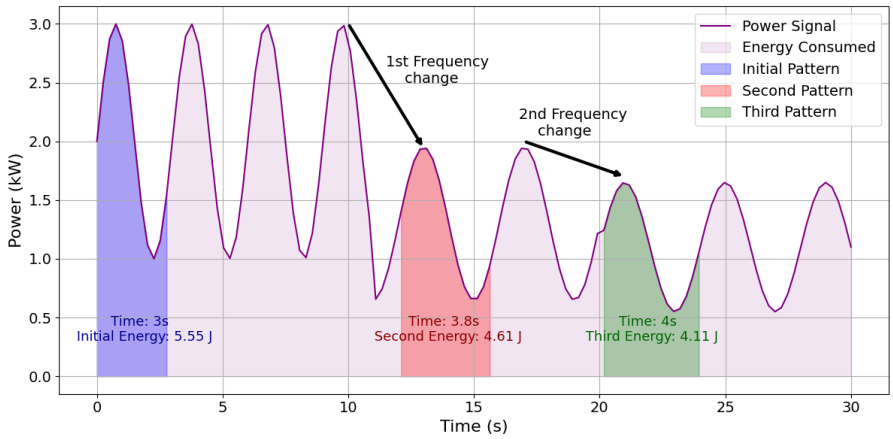


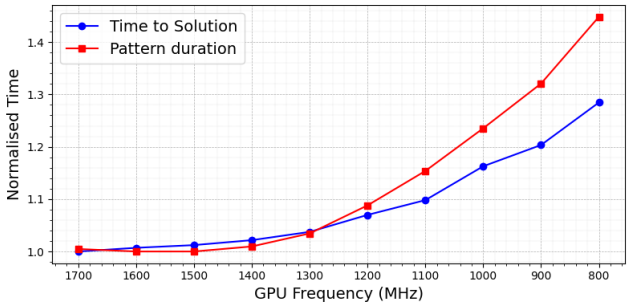
Figure 5: Illustration of Pattern Duration and Energy per Pattern Control based on Frequency Changes

Algorithm 1 Dynamic Frequency Control at Runtime based on Energy per Pattern

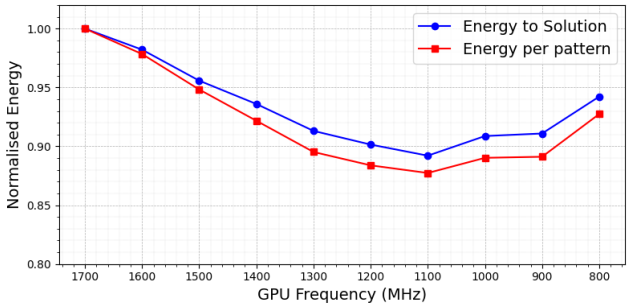
Procedure AdjustFrequency()
if frequency increased **then**
  if energy per pattern increased **then**
    freq ← *lower_freq*
  else
    freq ← *higher_freq*
  end if
else if frequency decreased **then**
  if energy per pattern decreased **then**
    freq ← *higher_freq*
  else
    freq ← *lower_freq*
  end if
end if
return freq
End Procedure

Procedure AnalyzePowerSignal()
  Read and process the power signal file
  Compute pattern duration and energy using FFT
End Procedure

Procedure Main
while True **do**
  AnalyzePowerSignal
  AdjustFrequency
  Wait for next analysis
end while
End Procedure



(a) Pattern Duration vs Time-to-Solution



(b) Energy per Pattern vs Energy-to-Solution

Figure 6: Pattern Duration vs Time-to-Solution and Energy per Pattern vs Energy-to-Solution for a Llama2-7b Fine-tuning

6 Experimental Validation

This runtime has been applied to all the previously mentioned AI workloads. The baseline results available in Table 1 serve as a reference to validate that the runtime converged to a suitable GPU frequency in order to minimize node-level consumption.

6.1 Experimental Protocol

The experimental protocol is designed to ensure the reliability and reproducibility of our AI benchmarking results. Each step serves a specific purpose:

- **Warm-up**:
  - The benchmark is run once to fetch the data and feed the caches.
  - This step mitigates the impact of initialization times on subsequent tests, ensuring that further measurements reflect steady-state performance.
- **Standalone**:
  - The benchmark is run in standalone mode, repeated 10 times.
  - This steps measures the benchmark's typical performance for both Time-to-Solution and Energy-to-Solution.
- **Dry-Run**:
  - The benchmark is run 10 times with the runtime enabled, but without frequency control.
  - This step measures the overhead introduced by the runtime computations, crucial for understanding the computational cost associated with the real-time signal processing.
- **Runtime On**:
  - The benchmark is run 10 times with the runtime continuously active and controlling the GPU frequency.
  - This step evaluates the runtime impact on the application for both Time-to-Solution and Energy-to-Solution.

By following this protocol, we aim to achieve consistent and comparable results, ensuring that the runs correctly reflect the behavior of the currently implemented runtime.

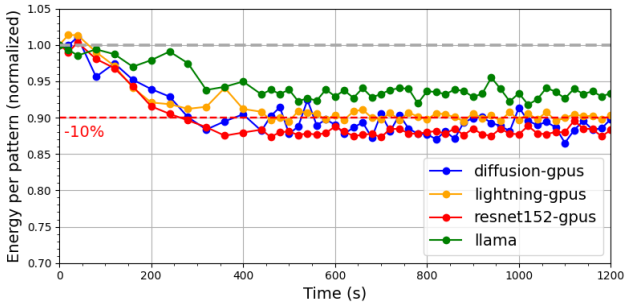
6.2 Energy Optimization Results

Results are reported in Table 2. Most of the benchmarks behave as expected, benefiting from a performance gain in terms of energy of approximately 10%. However, the vjepa-gpus benchmark did not benefit from the runtime, as no pattern could be found in the power signal. Consequently, the frequency did not change. This is still an interesting result, as it shows that the runtime did not affect the behavior of the application.

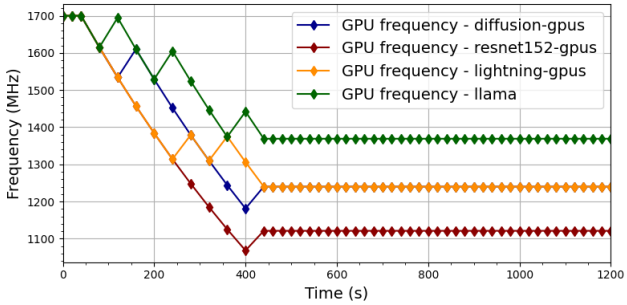
6.3 Zoom on Pattern Duration and Energy per Pattern

Our runtime enabled us to follow the value of our main metrics: pattern duration and energy per pattern. The convergence of the energy per pattern is shown in figure 7. We can observe that for all the workload we obtained the target energy per pattern in about 400 seconds. This shows that this naïve approach is still interesting in terms of convergence time, as it can find a sweet spot in just a few minutes, where runs could last for hours or even days.

Our runtime enabled us to track the values of our main metrics: pattern duration and energy per pattern. The convergence of the energy per pattern is shown in Figure 7. We can observe that for all the workloads, we achieved the target energy per pattern in approximately 400 seconds. This demonstrates that this naïve approach is still effective in terms of convergence time, as it can find a sweet spot in just a few minutes, even though runs could last for hours or even days. In that perspective, it enables rapid energy savings for long-running simulations.



(a) Energy per Pattern



(b) GPU Frequency Evolution

Figure 7: Energy per Pattern and GPU Frequency Evolution during Application Runs

All the final metrics for pattern duration and energy per pattern are reported in Table 3. We observe that these metrics follow the same trends as the total energy gain of the application, as expected.

6.4 Overhead Evaluation

For all the runs, we used an FFT sliding window of 20 seconds, with an FFT computation every 2 seconds. Our sampling rate for the `pm_counters` is 4 Hz, resulting in 80 samples per 20 seconds.

In total, we compute 30 FFTs over 80 values per minute. Additionally, we set the frequency of the GPUs every time we find a suitable spot for minimizing the energy per pattern metric.

As all of these actions could impact the global performance of the applications, the runs were conducted in dry-run mode to assess the overhead of the runtime.

The results of the overhead estimation are reported in Table 4.

The maximum overhead observed is 0.7% for the time metric and 0.2% for the energy metric. Due to the very low computation performed during the runs, it was expected that the overhead would be close to 0 for both time and energy. The near-zero overhead on all the applications makes this approach a perfect lightweight energy optimization runtime for regular applications.

AI Benchmark	Standalone Time to Solution (s)	Standalone Energy to Solution (Wh) Node	Standalone Energy to Solution (Wh) GPU	Runtime on Time (s) Solution (s)	Runtime on Energy to Solution (Wh) Node	Runtime on Energy to Solution (Wh) GPU	Performance **Time to Solution (s)**	Performance **Energy to Solution (Wh) Node**	Performance **Energy to Solution (Wh) GPU**
diffusion-gpus	2117	1202	1041	2385	1095	932	**+12.7%**	**-9.0%**	**-10.5%**
lightning-gpus	2227	1413	1124	2436	1275	969	**+9.4%**	**-9.8%**	**-13.8%**
resnet152-ddp-gpus	2635	1469	1161	2984	1320	998	**+13.2%**	**-10.1%**	**-14.0%**
vjepa-gpus	4713	1874	1442	4724	1878	1446	**+0.2%**	**+0.2%**	**+0.3%**
llama	5108	3039	2545	5331	2859	2316	**+4.3%**	**-5.9%**	**-9.0%**

Table 2: Impact of our Runtime on Application Duration and Energy

Benchmark	Pattern duration (s) Initial	Pattern duration (s) optimized	Energy per pattern (J) Initial	Energy per pattern (J) optimized	GPU frequency Converged value (MHz)
diffusion-gpus	1.47	1.70(**+16%**)	3465	3049(**-12%**)	1240
lightning-gpus	0.64	0.72(**+12%**)	1627	1450(**-11%**)	1240
resnet152-ddp-gpus	0.63	0.76(**+20%**)	1609	1400(**-13%**)	1121
vjepa-gpus	Not found		Not found		1700
llama	1.20	1.24(**+3%**)	2797	2695(**-4%**)	1369

Table 3: Impact of our Runtime on Pattern Duration and Energy

Benchmark	Standalone Time (s)	Standalone Energy (Wh)	Runtime off (dry-run) Time (s)	Runtime off (dry-run) Energy (Wh)	overhead (%) Time	overhead (%) Energy
diffusion-gpus	2117	1202	2117	1203	+0.0%	+0,0%
lightning-gpus	2227	1413	2230	1414	+0.1%	+0.1%
resnet152-ddp-gpus	2635	1469	2655	1472	+0.7%	+0.2%
vjepa-gpus	4713	1874	4713	1874	+0%	+0%
llama	5108	3039	5117	3039	+0.2%	+0%

Table 4: Overhead of FFT Runtime over AI benchmarks on Adastra-MI250X

6.5 Focus on Vjepa-gpus, the Non-Energy Optimized Application

As discussed previously, the Vjepa-gpus benchmark is not optimized by our runtime. To understand why, we plot 2 minutes of the power signal of Vjepa-gpus in Figure 8.

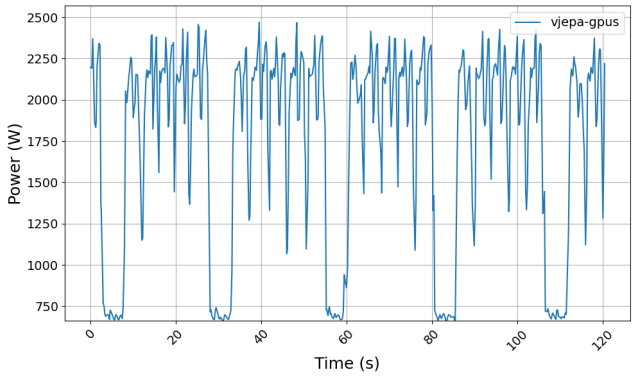


Figure 8: 2 minutes of 4Hz Sampling of Vjepa-gpus

We observe a 22 seconds pattern in this plot. Within this pattern, smaller variations can be observed, but perhaps not as distinctly as those seen for Llama2-7b. This could explain why we cannot identify a smaller pattern within the signal. Since our FFT window was set to 20 seconds for our tests, we cannot identify the 22 seconds pattern either.

To address this issue and still optimize the application, we could easily tune our implementation to extend its FFT window if no pattern is found after a certain time. This would allow us to detect larger patterns and enable frequency control for energy optimization.

The fact that Vjepa-gpus is not optimized by the runtime highlights a limitation of the current implementation, which can be easily fixed. The 4Hz sampling rate for power could also be a limiting factor.

6.6 Insights and Discussions

The proposed runtime demonstrates a promising opportunity for significant energy optimization in HPC and AI workloads, without introducing overhead or requiring specific hardware or software prerequisites. While the study focuses entirely on energy optimization, it acknowledges potential performance trade-offs in terms of execution time. The proposed runtime can be easily adapted to set a pattern duration threshold, ensuring that performance remains within acceptable limits during power optimization. This feature was not implemented, as it was beyond the scope of the current study.

Additionally, although we highlight limitations for applications where patterns cannot be identified using our basic FFT approach, it opens avenues for further research and immediate tuning of our runtime. Future work could explore more sophisticated pattern recognition techniques or machine learning methods to predict and optimize energy consumption based on historical data. This would extend the benefits of energy optimization to a broader range of applications including those multi-patterns non regular applications.

A key concern is ensuring that all users have access to relatively fine-grained power measurement logs to utilize this approach effectively. While the accuracy of power measurements may not be a critical issue, lower measurement frequencies might require increasing the window size to identify larger patterns, potentially taking more time to converge but remaining feasible. Moreover, the ability to modify the frequency cap of GPUs and other hardware components should be available to users interested in energy studies. However, security policies in many centers may restrict this

functionality, limiting the application of energy efficiency strategies. Nonetheless, system operators with appropriate rights can use this approach to minimize the Energy-to-Solution for all workloads, introducing no overhead.

7 Conclusion and Future Work

This paper demonstrates the feasibility of an application-agnostic runtime designed to enhance the energy efficiency of HPC and AI simulations. On top of being applications agnostic, this approach is extremely lightweight (with almost no overhead in this naive implementation), and tends to be hardware agnostic as well. Although some limitations can exist for irregular applications, the study shows that defining a global metric based only on the power signal of any application can minimize its energy consumption. For all of the presented benchmarks, we are able to identify patterns and derive this metric to follow the "energy per pattern" values. The most important point is that "energy per pattern" metric is easily portable and understandable for any users and operators of HPC and AI systems.

The energy benefits of this approach can be substantial, as many AI workloads can achieve approximately 10% energy savings while producing the same scientific output. In a world where energy production is increasingly constrained, such lightweight, hardware- and application-agnostic approaches can help balance the growing demand for AI with the need for effective energy management by HPC centers and cloud providers.

Finally, it is important to notice that, even if all the benchmarks we run have a Time-to-Solution penalty, the Energy-to-Solution is always better than the reference at the node level for all the GPU frequencies we applied during this study. This demonstrates that the GPU frequency control is extremely beneficial for Energy-to-Solution and should be more taken care of by the HPC and AI community.

Future work will focus on extending the "energy per pattern" metric to irregular applications by treating the power signal as a linear combination of multiple patterns, thereby enhancing the runtime's applicability and efficiency across a broader spectrum of HPC and AI applications. Additionally, the runtime will be adapted to support any GPU hardware using agnostic libraries such as Variorum [19]. Finally, the impact of an increased power measurement frequency on the overall performance of the runtime will also be studied.

References

[1] Dong H. Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. 2014. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. In *2014 43rd International Conference on Parallel Processing Workshops*. 9–17. doi:10.1109/ICPPW.2014.15

[2] Julita Corbalán and Luigi Brochard. 2018. EAR: Energy management framework for supercomputers. https://api.semanticscholar.org/CorpusID:252911624

[3] Intel Corporation. 2023. Intel 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html Accessed: 2025-04-09.

[4] Tom Deakin and Simon McIntosh-Smith. 2019. BabelStream. https://hpc.tomdeakin.com and https://uob-hpc.github.io. University of Bristol.

[5] Pierre Delaunay, Xavier Bouthillier, Olivier Breuleux, Satya Ortiz-Gagné, Olexa Bilaniuk, Fabrice Normandin, Arnaud Bergeron, Bruno Carrez, Guillaume Alain, Soline Blanc, Frédéric Osterrath, Joseph Viviano, Roger Creus-Castanyer, Darshan Patil, Rabiul Awal, and Le Zhang. 2024. Introducing Milabench: Benchmarking Accelerators for AI. *arXiv preprint arXiv:2411.11940* (2024). https://arxiv.org/abs/2411.11940v1

[6] GEOPM Team. 2024. GEOPM: Global Extensible Open Power Manager. https://github.com/geopm/geopm Accessed: 2024-04-03.

[7] IEA. 2024. What the data centre and AI boom could mean for the energy sector. https://www.iea.org/commentaries/what-the-data-centre-and-ai-boom-could-mean-for-the-energy-sector Licence: CC BY 4.0.

[8] Adrian Jackson, Alan Simpson, and Andrew Turner. 2023. Emissions and energy efficiency on large-scale high performance computing facilities: ARCHER2 UK national supercomputing service case study. arXiv:2309.05440 [cs.DC] https://arxiv.org/abs/2309.05440

[9] Ahmad Maroof Karimi, Matthias Maiterth, Woong Shin, Naw Safrin Sattar, Hao Lu, and Feiyi Wang. 2024. Exploring the Frontiers of Energy Efficiency using Power Management at System Scale. In *SC24*. 1835–1844.

[10] Naman Kulshreshtha, Tapasya Patki, Jim Garlick, Mark Grondona, and Rong Ge. 2024. Vendor-neutral and Production-grade Job Power Management in High Performance Computing. In *SC24*. 1845–1855.

[11] Lawrence Berkeley National Laboratory. 2024. *United States Data Center Energy Usage Report*. Technical Report. Lawrence Berkeley National Laboratory. https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report.pdf

[12] S. Martin. 2014. Cray XC30 Power Monitoring and Management. https://api.semanticscholar.org/CorpusID:208012529.

[13] S. Martin and G. J. Koprowski. 2018. Cray ®XC ™Advanced Power Management Updates. https://api.semanticscholar.org/CorpusID:202663809.

[14] Mila - Quebec AI Institute. 2024. About Mila. https://mila.quebec/en/about/about-mila Accessed: 2024-04-03.

[15] Phil Mucci, Shirley Moore, Christine Deane, and George Ho. 1999. PAPI: A Portable Interface to Hardware Performance Counters. (01 1999).

[16] Christian Simmendinger, Marcel Marquardt, Jan Mäder, and Ralf Schneider. 2024. PowerSched - Managing Power Consumption in Overprovisioned Systems. In *2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. 1–8. doi:10.1109/CLUSTERWorkshops61563.2024.00012

[17] Osman Seckin Simsek, Jean-Guillaume Piccinali, and Florina M. Ciorba. 2025. Increasing Energy Efficiency of Astrophysics Simulations Through GPU Frequency Scaling. In *Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (Atlanta, GA, USA) *(SC-W '24)*. IEEE Press, 1826–1834. doi:10.1109/SCW63240.2024.00229

[18] TOP500 Organization. 2024. TOP500 Supercomputer Sites. https://top500.org/ Accessed: 2024-04-03.

[19] Variorum Contributors. 2024. Variorum Documentation. https://variorum.readthedocs.io/en/latest/ Accessed: 2025-04-15.

[20] Wikipedia. 2025. Intelligent Platform Management Interface. https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface Accessed: 2025-04-09.