Increasing Energy Efficiency of Astrophysics Simulations Through GPU Frequency Scaling

Osman Seckin Simsek
University of Basel
Basel, Switzerland
osman.simsek@unibas.ch

Jean-Guillaume Piccinali
Swiss National Supercomputing Centre
Lugano, Switzerland
jgp@cscs.ch

Florina M. Ciorba
University of Basel
Basel, Switzerland
florina.ciorba@unibas.ch

***Abstract*—The growing demand for high performance computing (HPC) necessitates significant energy consumption, posing a sustainability challenge for HPC centers, users, and society at large, especially in the face of stricter environmental regulations. While efforts exist to reduce overall system energy consumption, optimizations for GPU-based workloads, increasingly prevalent in HPC, has received insufficient attention for workload-specific energy efficiency optimizations. This work addresses this critical gap by proposing dynamic approaches to increase energy efficiency by instrumenting the simulation code with an open-source toolkit that enables accurate power and energy measurements for a wide range of CPU-GPU node architectures, as well as the instrumentation of the code controlling the GPU frequency dynamically. We further investigate the energy-performance trade-off by comparing both static and dynamic GPU frequency scaling strategies within SPH-EXA, a newly developed, open-source, and GPU-centric simulation framework specializing in astrophysical and cosmological simulations. Our findings demonstrate that code instrumentation enables detailed energy consumption acquisition beyond traditional HPC system monitoring, while dynamic frequency scaling of computational kernels achieves energy reduction with limited performance loss. This approach empowers researchers to develop more sustainable large-scale scientific simulations running mainly on GPUs.**

***Index Terms*—HPC, GPUs, frequency scaling, energy efficiency, large-scale simulations**

I. Introduction

Energy efficiency is a critical issue in high performance computing (HPC) due to its substantial operational costs and environmental impact. To reduce energy consumption in HPC systems, collaborative efforts are needed from supercomputing centers, hardware vendors, and HPC users. Although HPC centers have made strides by incorporating renewable energy sources, there is still potential for further progress, particularly from users. Hardware vendors have also significantly advanced the energy efficiency of HPC systems, resulting in more than 15-fold increase in the energy efficiency of the first ten systems in the Green500 list over the last decade [1]. HPC users can also contribute by optimizing their simulations to enhance both performance and energy efficiency.

In Astronomy and Astrophysics, the recently published Astronet roadmap [2] offers guidelines for research activities within the astronomical community that have a considerable environmental impact. The report advocates for a comprehensive approach to reducing the carbon footprint. Firstly, researchers need to be more conscious of the environmental costs associated with their work, such as the computing and power required for simulations and data analysis, as well as the energy consumption of observatories. They should aim to minimize energy use and promote the adoption of renewable energy sources in the facilities they utilize. Secondly, the importance of code optimization is emphasized. While user-friendly interpreted languages are useful, researchers should also have proficiency in compiled languages for greater efficiency. The Astronet report strongly encourages the use of efficient programming languages and computational architectures for intensive computations to lower environmental costs. However, switching from interpreted to compiled languages is only part of the solution; further energy savings can be achieved through better-optimized code and more efficient use of system resources.

There is a positive correlation between the performance of a parallel application and its energy consumption, which presents a significant challenge for optimization. While higher performance can often be obtained by allocating additional resources, this approach may lead to reduced energy efficiency. The complexity of this trade-off is further exacerbated by the evolving landscape of computing hardware, with modern systems featuring an increasing number of CPU cores and specialized accelerators like GPUs. These diverse processing elements have varying power consumption characteristics, making it increasingly difficult to achieve optimal performance without excessive energy use. However, the performance benefits provided by GPUs can lead to improved overall energy efficiency, especially for compute-bound components of scientific applications. Thus, greater energy efficiency in scientific applications can be attained by optimizing GPU resource utilization and focusing on both performance and energy efficiency.

In this work, we demonstrate the advantages of enabling precise measurement of energy consumption as well as using this information to adjust the frequency of GPUs both dynamically through code instrumentation to assess the energy-performance trade-off of an astrophysical simulation framework. We apply our approach to a real-world case in computational astrophysics, specifically the SPH-EXA simulation framework [3], which runs on various CPU+GPU node architectures. SPH-EXA is equipped with extensions to measure energy consumption using HPE/Cray-specific *pm_counters* for systems built by HPE, and for other systems, it employs an open-source power measurement toolkit [4] (PMT) to collect energy consumption data and generate reports that users can analyze to develop energy-efficient code and conduct computational experiments. This information is then used to determine the best frequencies for each computational kernel for increasing the energy efficiency of the simulation. We enabled user-level GPU frequency adjustment, allowing users to set GPU frequencies within a cluster without needing superuser privileges, solving the issue of restricted access typically required for GPU frequency changes. Our results show that dynamic GPU frequency setting through code instrumentation decreases the energy consumption of our simulations up to 7.82% per GPU while the performance loss is limited to 2.95%.

The rest of this paper is organized as follows. Section II discusses the importance of energy efficiency and gives background information on how energy is measured in systems and within applications. Section III presents our methodology for measuring the energy consumption and performance of each SPH-EXA function online and discusses how this information can be exploited to increase the energy efficiency of large scale simulations. Section IV shows our results for validating our instrumented energy measurements as well as the benefits of GPU compute frequency scaling using static and dynamic methods. Section V concludes this paper.

II. Background and Related Work

The changes in supercomputers from CPU-only to CPU-GPU heterogeneous architectures, started an effort to port SPH simulation codes to GPUs, enabling higher performance, larger, and higher resolution simulations [5]–[8]. Although the motivation for this switch was due to the immense performance benefits that GPUs can provide, it was also beneficial from the energy efficiency and environmental impact perspectives. Portegies Zwart [9] evaluates how different programming languages affect the energy consumption and carbon footprint for various N-body codes where CUDA-based implementations prevail with the most energy-efficient and producing the least carbon footprint, performing almost an order of magnitude better than Fortran or C++ as shown in Figure 1. However, the increased energy efficiency by better utilizing GPUs is only an afterthought, as research on energy efficiency of scientific applications is lacking.

Increasing the energy efficiency of simulations can be achieved by not only increasing the performance, but also better utilizing the computational resources. Decreasing the energy consumption of a simulation without changing the application is possible through altering the operational voltage and/or frequency of the computational units.

Dynamic Voltage and Frequency Scaling (DVFS), is a power management technique that dynamically adjusts the voltage and frequency of processors and other devices in systems. By lowering voltage and frequency during low-intensity

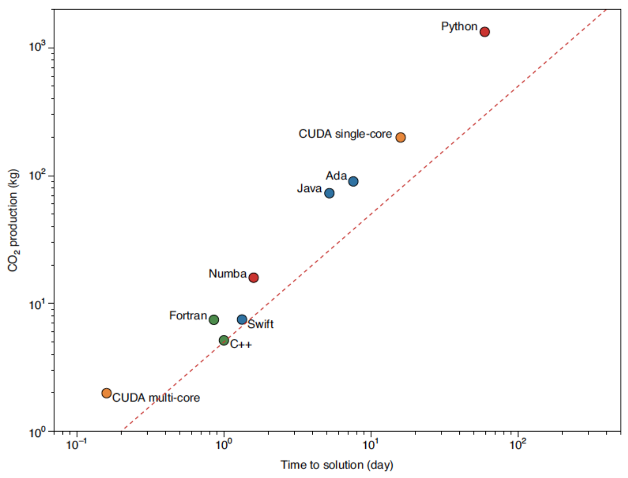

Fig. 1. Programming language efficiency as a function of the time to solution (image reproduced from [9]).

tasks, DVFS can significantly reduce power consumption. The energy efficiency benefits of DVFS has been studied extensively for GPUs [10]–[13], but application specific energy efficiency studies have not seen a lot of attention. Moreover, decreasing the voltage may result in computational inaccuracies and decreasing the frequency negatively affects the performance and these effects are different for each application, so the energy efficiency concerns might differ depending on the application. Overall, increasing the energy efficiency without impacting the performance or the correctness of the applications is of paramount importance.

The multi-dimensional energy optimization problem can be solved by finding the sweet spot frequencies for applications, thus increasing the overall energy efficiency [14]. The first step towards finding such sweet spots is measuring the energy consumption of the application in a detailed way. This step can then be followed by optimizing for the best frequency for parts of the application.

A. Application-level Energy Measurement

Measuring the energy of a simulation requires power and time measurements. Although external power meters are the most accurate method to monitor the power consumption of computing systems [15], [16], power meters are generally not integrated in production systems. Instead, the systems provide the power and energy measurements through embedded sensors using IPMI or ACPI interfaces. However, the power consumption data is rarely available to the system users.

In HPC systems, resource and job management systems, such as Slurm [17], can be configured to provide information about the energy consumption of submitted jobs by adding the *energy* label to the **AccountingStorageTRES** list of Trackable REsources (TRES). Energy accounting data can be accessed with the *sacct* command and *ConsumedEnergy* parameter. Depending on the system, the Slurm's energy reporting backend can be *ipmi*, *pm_counters* or *rapl*. The *ipmi* uses the baseboard management controller (BMC) and *pm_counters* is an HPE/Cray-specific BMC, while *rapl* uses the hardware sensors to read the power consumption of the CPUs.

The Cray systems incorporate Out-Of-Band (OOB) data collection capabilities for monitoring power and energy at the node level, with a default collection rate of 10 Hz. Collected power and energy data are published via special */sys/cray/pm_counters/* sysfs files, as described by Martin in [18] and [19]. The pm_counters files are read-only and available to users for monitoring purposes and to report energy usage. The energy usage of node, CPU and memory of a compute node can be read from the **energy**, **cpu_energy** and **memory_energy** pm files. The energy usage of accelerators can be read from the **accel[0,1,2,3]_energy** pm files. It is important to note that the energy consumption of the auxiliary parts of a node can also be calculated by subtracting the GPU, CPU, and memory measurements from the node-level energy measurement.

While energy consumption data provided as part of a job report is useful for users, it does not provide specific information about which parts of the job consumed most energy, depriving the user of the opportunity to reduce their job's energy consumption. When information is available about devices that consume more energy, or as a functional breakdown, users can then employ various techniques to reduce their application's energy consumption.

PMT [4] is an open-source library that provides a common interface to measure power consumption of various devices. For GPUs it relies on NVML [20] for Nvidia and rocm-smi [21] for AMD. CPUs are monitored through the RAPL (Running Average Power Limit) [22] interface, or through LIKWID [23]. Using the power measurements taken from the vendor provided APIs, PMT calculates the energy consumption of the instrumented code and reports the measurements to the user. The advantage of PMT compared to previous efforts for power measurements, e.g., LIKWID [23], is that it provides an interface to a comprehensive set of back-ends which reduces the frequent code changes within the application code base. PMT also supports the HPE/Cray-specific back-end, thus providing easy and accurate measurements for compute nodes entirely built by HPE/Cray.

B. Static and Dynamic Frequency Scaling

Frequency scaling is a key power management technique in both CPUs and GPUs. Processors dynamically adjust their clock speed based on workload. During demanding tasks, the frequency increases for faster processing, but consumes more power. Conversely, for simpler tasks, the frequency dips to conserve power while maintaining adequate performance. This ensures the processor is not constantly running at maximum power, optimizing efficiency without sacrificing performance.

In supercomputing centres, CPU and GPU frequencies of the nodes are generally set to the highest frequency values for performance reasons. However, due to the increasing energy costs, some centres have been introducing ways to reduce the overall energy consumption by setting lower CPU frequencies. In 2022, the ARCHER2 Supercomputing Center decided to decrease the default CPU frequencies [24] in order to reduce the power consumption with limited performance loss for a variety of applications. Additionally, CPU and GPU frequencies can be controlled by Slurm and be set to a specific value or a range of values. For example, the *--cpu-freq=1800000* flag would set the CPU frequency to 1.8 GHz, and the *--gpu-freq=900* flag would set the GPU frequency to 900 MHz. This is possible under the condition that the supercomputing centre is allowing users to change default values.

On the other hand, due to the correlation between performance and energy, setting the operating frequency statically to a lower value will impact the performance of the application negatively. Although DVFS tries to decrease the frequency when the computational unit is not in use to save energy, this is dependent on the utilization of the compute unit. Specifically for GPUs, the utilization is calculated by the time taken to execute one or more computational kernels and thus using Nvidia-smi and rocm-smi APIs result in high-level and unrepresentative results for actual GPU utilization [25]. Therefore, the frequency set by the DVFS on GPUs is also not exactly representative of the underlying kernel execution, leading to reduced energy efficiency. This inefficiency can be solved by finding out the best frequency for the performance-energy trade-off and dynamically setting the GPU frequency for different kernels from within the application through code instrumentation.

III. Methodology

A. Application

SPH-EXA [3][1] is a simulation framework that leverages state-of-the-art SPH method implementation. It is written in modern C++, with minimal software dependencies and can execute on CPUs and GPUs at extremely large scales [26], making it well-suited for extreme-scale astrophysical and cosmological simulations.

SPH-EXA employs MPI+X for parallelization, where X is either CUDA, HIP if the system offers CPUs and GPUs, or OpenMP if the system offers only CPUs. On CPU+GPU systems, SPH-EXA moves all the simulation data from the CPU memory to the GPU memory at the beginning of the simulation and runs entirely on GPUs, leaving CPUs available for handling auxiliary tasks, such as performance or energy profiling. This way, the performance of the instrumented code is unaffected by performance or energy profiling.

B. Measurement of Application Energy Consumption

SPH-EXA provides hooks within the code that can be used for low-overhead profiling, enabling third party tools to be integrated into the framework for performance and energy consumption analysis. The hooks are normally used to measure the timings for each function within the framework, which consists of functions that perform domain decomposition, halo exchanges, and physics computational kernels.

[1]https://github.com/unibas-dmi-hpc/SPH-EXA

We instrumented the SPH-EXA code using these hooks with native support for HPE/Cray built systems to collect energy consumption data from the start of each function call until its completion. In case the target production system is not HPE/Cray built, we use PMT to get the measurements from CPU and GPU separately for each device. Energy consumption is measured per each MPI rank throughout the simulation, gathered at the end of the execution, and stored into a file for post-hoc analysis, to avoid perturbing the actual simulation. The provided hooks are also used to instrument the code for setting the GPU frequency within the code before each computational kernel in order to increase the energy efficiency of the overall execution. If the kernel is not computationally intensive, the GPU frequency is reduced back. The benefit of instrumenting the code is that the developer has prior knowledge about the computational kernels, hence can select the best frequency that benefits the performance-energy trade-off.

The general rule-of-thumb in GPU-centric applications is to use one MPI rank to drive one GPU. However, this assignment does not favor energy measurements. For example, HPE/Cray *pm_counters* measure power consumption and report it in a file per GPU card. On systems where each GPU card has two GCDs (GPU Complex Dies), such as LUMI-G (Table I), an MPI rank only drives a GPU half-card, while power consumption is measured and reported for the entire card which corresponds to two MPI ranks. This is not an issue on systems where each GPU card has one GCD, such as the CSCS-A100 system (Table I). In both cases, *pm_counters* reports 4-or-8 power consumption measurements on CPUs, one for each MPI rank executing on one node. The CPU measurements for both systems take into account the entire CPU, and all MPI ranks on the same node report the same energy measurement, hence only one measurement needs to be used for the calculations. To overcome these discrepancies, our analysis scripts take the system's hardware configuration and MPI rank-to-GPU assignment into consideration. However, two GCDs on one GPU card still creates certain measurement inaccuracies, as discussed in Section IV-A.

With energy measurement enabled in SPH-EXA, we executed two astrophysical simulations and collected the time and energy measurements throughout the runs. The information gathered is exploited to draw insights from the energy consumption of the simulation at device-level and code function-level. We show that functional-level energy consumption can be used to designate which part of the simulation code can benefit from GPU frequency down-scaling for reducing the energy consumption.

C. Tuning Application Functions for Energy Efficiency

KernelTuner [27] is a tool designed to help developers create optimized applications for GPUs. It allows users of KernelTuner to define the GPU kernels and specify which parts of the code can be tuned to improve performance. KernelTuner then tries out different combinations of these adjustments and measures how well each one performs.

The tool provides tune_kernel main functionality which includes kernel_name, kernel_source, problem_size, and params arguments. By supplying the kernel_name and kernel_source, the users can specify which kernel to tune. The problem_size is used to determine the grid dimensions while benchmarking different kernel configurations. The final argument params is a dictionary that specifies the tunable parameters of the kernel. For each tunable parameter, specified as a key in the dictionary, it contains a list of all possible values. KernelTuner offers a lot of strategies for the search space, brute-force search being the default, which simply iterates over the entire search space. This may seem inefficient, but for small number of tunable parameters, this can be done in a reasonable amount of time.

Since we are interested in finding the most energy-efficient frequency for specific kernels, which itself is not a tunable parameter for a kernel, but rather a device-wise parameter, we simply employ the ease-of-use that KernelTuner provides for running the same kernel multiple times and reporting the time-to-solution and energy consumption metrics to choose the best option.

We keep the problem_size parameter fixed for this optimization problem and change the GPU frequency using Nvidia Management Library (NVML) [20]. Specifically for our experiments, we set the problem size to 450^3 particles for each kernel in the SPH-EXA simulation framework.

Figure 2 shows the outcome of the search space for all SPH-EXA kernels with the GPU compute frequency parameter between 1410 MHz and 1005 MHz. We have not experimented with frequencies below 1005 MHz, due to limited benefits for energy efficiency. This range also provides a reasonable search space for KernelTuner to find the best GPU compute frequency for energy efficiency.

The frequency selection is also in line with the frequency settings for production supercomputers. For example, Piz Daint supercomputer [28] at the Swiss National Supercomputing Centre (CSCS) has the GPU frequency between to the minimum value of 1189 MHz and maximum value of 1328 MHz.

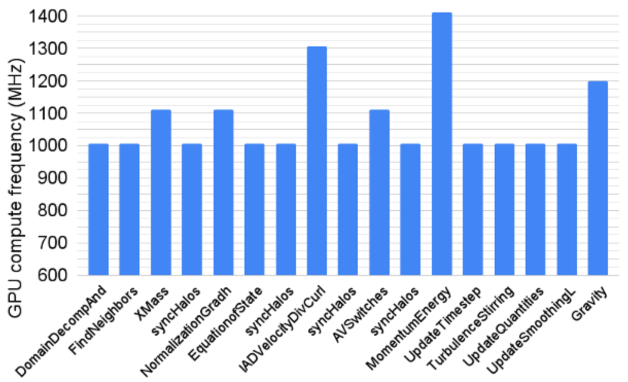

Fig. 2. GPU frequencies per function optimized for the best EDP outcome for Subsonic Turbulence simulation using 450^3 particles.

D. Application Instrumentation for Dynamic GPU Frequency Scaling

For changing the GPU frequency dynamically during the application execution, we instrumented SPH-EXA using API calls from NVML. Before each SPH-EXA function, we call the following code below which enables setting both the GPU compute frequency and memory frequency, though we keep the memory frequency as is for all cases.

```
nvmlDevice_t nvmlDeviceId;
getNvmlDevice(&nvmlDeviceId)

int smClock = 1410; // 1305, 1200, 1110, 1005
nvmlDeviceSetApplicationsClocks(nvmlDeviceId,
                                memClock,
                                smClock));
```

Since each MPI-rank is bound to only one GPU, *getNvmlDevice* returns the corresponding device ID which is then used to set the GPU compute frequency. Depending on the best frequency values that are found using KernelTuner, we set each function's frequency using the *nvmlDeviceSetApplicationsClocks* API call.

IV. Results

We conducted experiments with the SPH-EXA simulation framework and executed Subsonic Turbulence and Evrard Collapse simulations with 150 and 80 million particles per GPU, respectively, evaluating the value added by energy measurement instrumentation, across three systems with GPUs as shown in Table I. We chose these two workloads due to the different functionality, such that Evrard Collapse includes *gravity* calculations whereas Subsonic Turbulence does not.

We first validate the energy measurements from energy instrumented SPH-EXA against Slurm provided measurements on LUMI-G and CSCS-A100 systems. We then showcase application energy consumption per-device on the system as well as per-function during the simulation to a level of detail which is not normally available to the users. Furthermore, we use the miniHPC system for exploiting information provided by measuring the energy per-function to evaluate the effect of static and dynamic GPU frequency down-scaling on performance, energy consumption, and energy-delay-product (EDP) which is a combined metric for understanding the trade-off between performance and energy consumption.

A. Validation of PMT Energy Measurements

Integration of instrumentation-based power measurement tools, such as PMT, does not guarantee the validity of the measurements. Since users at most only have access to Slurm measurements, we validated our PMT instrumented simulations to this baseline. To this end, we validate the PMT measured energy with Slurm provided energy by running the Subsonic Turbulence experiments with energy measurement enabled, using 8-to-48 GPU cards with 1 GPU per MPI rank and 150 million particles per GPU as shown in Figure 3.

We observe a strong match between PMT measured and Slurm measured energy values. The difference between PMT and SLURM measurements is due to the timing of the energy

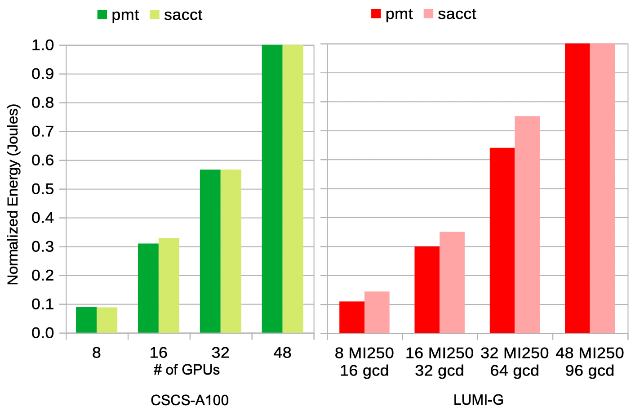

Fig. 3. Comparison between PMT measured and Slurm reported energy, normalized to 48 GPUs in CSCS-A100 and 96 GCDs in LUMI-G respectively.

measurement. Slurm starts measuring energy as soon as the job is submitted, while PMT starts the measurement when the time-stepping loop begins in SPH-EXA. This means that PMT does not measure the job and application setup phases, such as job launching or allocating the required data structures for the simulation. This difference in measurements does not pose a problem for the actionable insights users can take, as users have limited control over the job setup time and reducing the energy consumption of application initialization has a limited impact on the total amount of energy consumed. This is because the computational units that consume the most energy, namely GPUs, are idle during job and application setup phases.

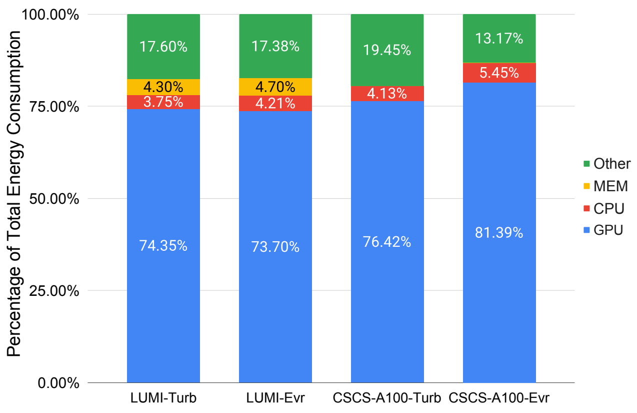

Fig. 4. Breakdown of energy consumption by GPU device for the Subsonic Turbulence and Evrard Collapse simulations executing on two systems using 150 million and 80 million particles per GPU on 32 MPI ranks.

B. Measuring and Reporting Energy Consumption per-Device and per-Function

Figure 4 shows the percentage of energy consumed by each device on the two test systems for the Subsonic Turbulence and Evrard Collapse simulations, both running with a total of 32 Ranks. The LUMI-G system shows separate measurements

TABLE I
SIMULATION AND COMPUTING SYSTEM PARAMETERS.

Simulation	**Parameters**	**Info**
Subsonic Turbulence	-n 0.6 | 1.2 | 2.4 | 4.9 | 7.4 | 9.2 | 14.7 Billion particles -s 100	150 million particles per GPU | 100 time-steps
Evrard Collapse	-n 0.6 | 1.2 | 2.4 | 3.2 | 4.8 | 7.7 Billion particles -s 100	80 million particles per GPU | 100 time-steps
System	**Hardware of each Node**	**GPU Frequencies**
LUMI-G	1 × 64 cores AMD EPYC 7A53 CPU with 512 GB Memory 8 × AMD Mi250X GPUs half cards with 64 GB Memory	AMD GPU compute frequency: 1700 MHz AMD GPU memory frequency: 1600 MHz
CSCS-A100	1 × 64 cores AMD EPYC 7113 with 4 × Nvidia A100-SXM4 with 80 GB Memory	Nvidia GPU compute frequency: 1410 MHz Nvidia GPU memory frequency: 1593 MHz
miniHPC	2 × 28 Core Intel Xeon Gold 6258R CPU with 1.5 TB Memory 2 × Nvidia A100-PCIE with 40 GB Memory	Nvidia GPU compute frequency: 1410 MHz Nvidia GPU memory frequency: 1593 MHz

for GPU, CPU, and memory while CSCS-A100 system does not provide separate measurements for the memory, hence the *Other* reported for CSCS-A100 also includes the energy consumed by the memory. The amount of total energy consumed in mega-Joules (MJ) is 24.4, 15.2, 12.5, and 10.7 for LUMI-Turb, LUMI-Evr, CSCS-A100-Turb, and CSCS-A100-Evr, respectively.

The energy consumption breakdown by device shows that the GPU consumes the most energy, 74.3% on LUMI-G and 76.4% on CSCS-A100 system. This information already provides insight about where the greatest potential lies for reducing energy consumption. Since SPH-EXA computations are predominantly executed on the GPU, the optimizations focusing on energy efficiency need to be applied to the computational kernels executing on the GPU.

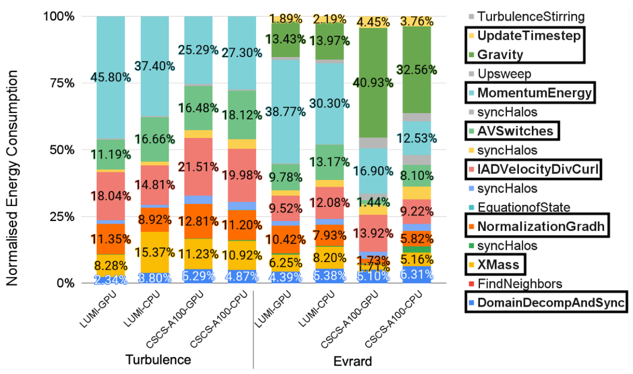

Fig. 5. Breakdown of energy consumption by SPH-EXA functions for the Subsonic Turbulence and Evrard Collapse simulations executing on two GPU systems, using 150 million and 80 million particles per GPU, respectively.

The energy consumed by auxiliary parts of the node, named as *other*, is a calculated value as explained in Section II. Even though it is the second-most energy consuming part of the simulation, we do not have additional information to insightfully analyze it. For example, it would be important to know if the energy consumed is attributed to the network interface card, so that the communication operations become a target for future optimizations.

The instrumented code also enables energy measurements at code functional-level. Figure 5 shows all SPH-EXA functions called in the time-stepping loop of the Subsonic Turbulence and Evrard Collapse simulations using 150 million and 80 million particles per GPU respectively.. The functions that consume the most energy overall are enclosed in a box in the legend. The reason these functions also consume the most energy on the CPU, as well as on the GPU, is that the CPU consumes energy which is proportional to the execution time of each function, even though the CPU is mostly idle during the execution of these functions.

The energy consumption per function normalized to the total energy consumed per device varies greatly depending on the system. For example, for the Turbulence simulation on the CSCS-A100 system, the function *MomentumEnergy* only consumes 25.29% of the total energy consumed by the GPU (3.1 MJ) while on LUMI-G it consumes 45.80% (11.2 MJ). This is a clear indication that *MomentumEnergy* function can further be optimized for AMD GPUs.

C. Statically Setting the GPU Frequency

The information made available by the integration of energy measurements into SPH-EXA shows where the most energy is consumed both device-wise and function-wise and is paramount for the energy efficiency optimizations. Efficiency metrics such as energy-delay product, calculated by multiplying the total amount of energy with the execution time, can be used to quantify the impact of applied optimizations.

Previous studies [9] show the effect of the choice of programming languages and compute device frequencies for running the simulations in an astrophysical context. Portegies Zwart et. al. [9], shows that GPUs are the most energy-efficient for large scale simulations and they evaluate the effect of CPU frequency changes on the energy-to-solution. Since SPH-EXA is a GPU-native simulation framework, we employed frequency down-scaling as a way to reduce the GPU power consumption as other studies [29], [30] show the benefits of reducing the energy consumption through reduced power consumption.

As the systems we used to gather the energy measurements in larger simulations do not allow user control over the GPU frequencies, we used the GPU node of a local research cluster named *miniHPC* which includes a node with Nvidia A100 GPU and conduct experiments with different GPU compute frequencies. Since the GPU memory of miniHPC is smaller than the GPU memory on the other two test systems, we were

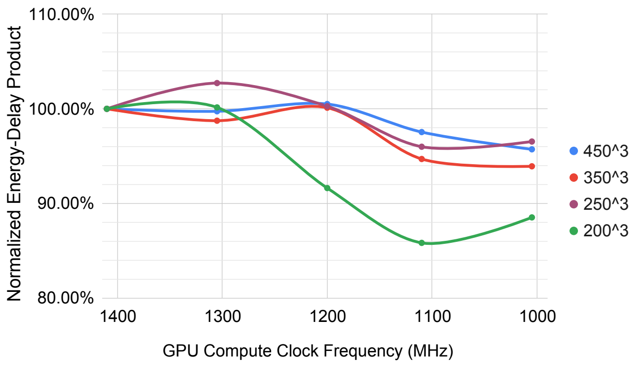

Fig. 6. Effect of statically down-scaling the GPU frequency on the energy-delay product of Subsonic Turbulence SPH-EXA simulation with different GPU particle counts, running on a single Nvidia A100 GPU.

forced to execute smaller simulations, starting at 91 million particles per GPU.

We first evaluate the effect of the change in the simulation problem size on the EDP, by varying the simulated particles per GPU between 450^3 (91 million) particles down to 200^3 (8 million) particles. We observe that the EDP drops significantly when the GPUs are not fully utilized as evidenced by the 200^3 particle case in green. These results show that when the problem size is small, leaving the GPU underutilized, i.e. 200^3 case, using lower frequency such as 1110 MHz results in the best energy efficiency configuration.

Figure 6 shows the energy-delay product (EDP) of SPH-EXA turbulence simulations normalized to the baseline, when the GPU compute frequency is set to 1410 MHz. We compare the EDP of SPH-EXA simulations when varying the GPU compute frequency between 1410 MHz and 1005 MHz. As frequencies are reduced, the time-to-solution of the simulation increases, but the power consumption reduction is so significant that the overall EDP decreases.

Figure 8 shows the execution time, energy, and EDP values of the most time consuming functions in SPH-EXA simulations with 450^3 particles per GPU. All the results here are normalized to the baseline, when the GPU compute frequency is set to 1410 MHz. The most computationally intensive functions, i.e., *MomentumEnergy* and *IADVelocityDivCurl*, has high execution time increase of more than 20% while the energy reductions are limited to 13% and 19% respectively for the 1005 MHz GPU frequency as shown in Figure 8(a) and Figure 8(b).

The combined metric of EDP thus shows that decreasing the frequency for these functions has limited benefits as shown in Figure 8(c). On the other hand, all the other functions have at least 10% reduction in their corresponding EDP metric.

These results corroborate the findings presented using KernelTuner in Section II-B in Figure 2. As shown in both Figures 2 and 8(c), the functions that are less computationally intensive such as *XMass* and *NormalizationGradh* can benefit from lower frequencies than the most compute-bound functions such as *MomentumEnergy*.

D. Dynamically Setting GPU Frequency for Different Functions

Down-scaling the GPU frequency during the entire simulation to save energy is not the best solution as it increases the overall time-to-solution. However, dynamic approaches can be employed for identifying Pareto-optimal solutions that provide acceptable performance and lower energy consumption.

Figure 7 shows the comparison of time-to-solution, energy consumption, and EDP metrics. The values are normalized to 1410 MHz baseline. Statically setting the GPU frequency to a lower value increases the energy efficiency of some of the functions in SPH-EXA. However, if we look at the overall time-to-solution and energy consumption reduction for the frequency range between 1005-1410 MHz, we see that time-to-solution of our simulations are affected. Although the energy savings are significant, the most important metric for the users remains time-to-solution. As the frequency decreases, time-to-solution increases, impacting overall performance. The energy reduction however is more significant, resulting in the decrease in the combined EDP metric.

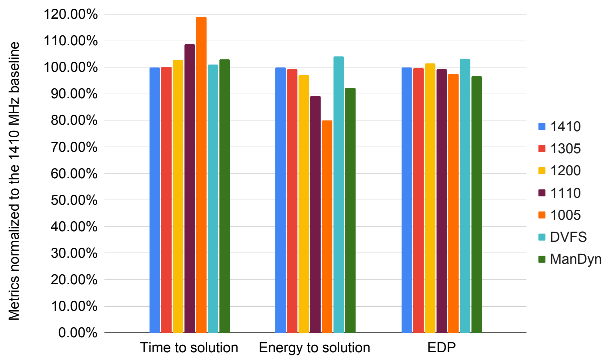

Fig. 7. Effect of dynamic frequency changes limits performance loss while increasing energy efficiency in Subsonic Turbulence simulation using 450^3 particles on a single Nvidia A100 GPU. ManDyn denotes manually changing the GPU frequency through code instrumentation.

The DVFS usage does not decrease the time-to-solution, providing similar values with the baseline. However, the energy-to-solution metric is also higher compared to the baseline. Since the compute-intensive kernels also benefit from DVFS effects, non-compute dominated kernels consume more energy, hence the overall energy consumption increases.

On the other hand, for the proposed method of manually changing the GPU frequency (ManDyn) we clearly see the benefit of reduced energy and the performance loss is limited to a negligible degree as shown in Figure 7.

Dynamically setting the GPU frequency leads to limited increase in time-to-solution, as much as 2.95%, with 8% reduction in energy consumption, leading to 4% reduction in EDP compared to setting the frequency statically to 1005 MHz which provides 2.5% decrease in EDP overall. Only comparing the static frequency of 1005 MHz with setting

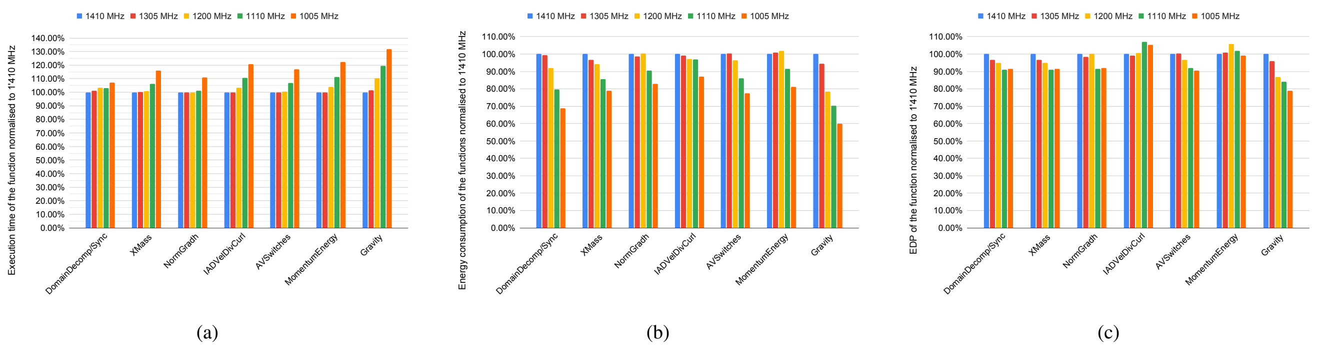

Fig. 8. Effect of statically down-scaling the GPU frequency on (a) execution time, (b) energy, (c) EDP for Subsonic Turbulence SPH-EXA simulation running on different GPU compute frequencies for 450^3 particles.

the frequency dynamically, we see a 16% decrease in time-to-solution, showing clear benefits of dynamically setting the compute frequency during execution.

E. Delving into GPU DVFS

In order to better understand the comparison between the effects of DVFS on performance and energy, we setup another experiment for measuring the frequencies set by the DVFS during the simulation execution. Figure 9 shows the measured frequencies set by DVFS on a single A100 GPU during Subsonic Turbulence simulation execution for 10 time-steps. We clearly see the pattern produced by the DVFS in each time-step the frequencies climb to the maximum of 1410 MHz for *MomentumEnergy* and above 1350 MHz for *IADVelocityDivCurl* functions. While executing the kernels in between, the frequency is set between 1300 and 1350.

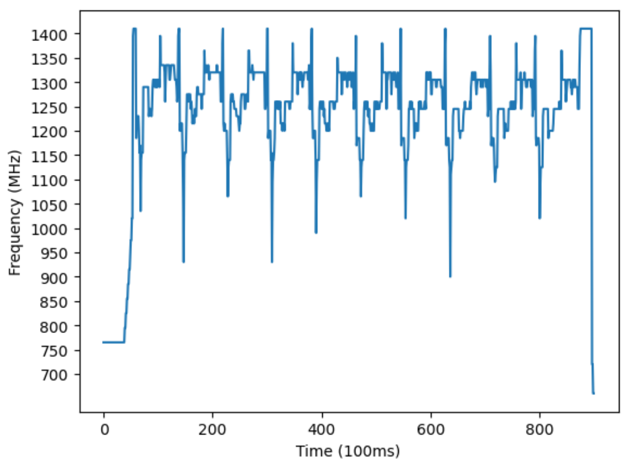

Fig. 9. Device frequencies set by DVFS on a single A100 GPU during Subsonic Turbulence simulation execution using 450^3 particles for 10 time-steps.

Moreover, at the beginning of each time-step, during *DomainDecompAndSync* function we see the frequency around 1200 MHz. This specific function launches a lot of lightweight kernels for calculating the particle keys which do not utilize the GPU and do not necessarily require such high frequencies. However, each kernel launch boosts the GPU frequency since the kernel does not yet have any information on how much utilization is achieved, leading to a higher than necessary frequency value which translates into increased energy consumption. These results are in line with findings reported in the literature [25] since this work also discusses the reported GPU utilization is overestimated.

At the end of each time-step, the frequency dips due to communication operations in order to calculate the physical time of the current time-step which is a collective communication function. These small functions at the end of each time-step decreases the GPU frequency below 1000 MHz in some cases, but due to the execution times of these functions being very low, they do not affect the overall results much. However, this is an indicator that for such functions, the frequency could be lowered further.

V. Conclusion and Future Work

This paper presents the benefits of dynamically adapting the GPU frequency through code instrumentation to enhance energy efficiency on an astrophysics and cosmology simulation framework, SPH-EXA. We analyzed energy consumption across multiple compute devices and simulation functions, identifying areas for performance and energy improvements. By adjusting GPU frequencies dynamically, we evaluated performance and energy impacts, comparing our results with against static frequency setting as well as GPU DVFS usage. Experiments with Subsonic Turbulence and Evrard collapse simulations validated our approach, limiting the performance loss of dynamic frequency down-scaling, to a mere 2.95% while the reduction in energy consumption is as high as 8%.

We further investigated the DVFS of usage on a single Nvidia A100 GPU and measured the frequencies during the simulation execution. We observed that the computationally intensive kernels do not always reach the highest possible frequency, while a high number of lightweight kernel launches negatively affect the energy efficiency.

Through our results, we show that application developers can help increase the energy efficiency of their code through instrumentation, since developers have expert knowledge on the applications they develop, helping with the global effort of decreasing the energy usage their codes as well as supercomputers.

Future work includes the adaptation of the proposed method on AMD and Intel GPUs, and studying the effect of different architectures and frequencies. Additionally, the proposed method will be applied to other simulation codes that use GPU acceleration to increase the overall energy efficiency and decrease the energy-delay product.

Acknowledgment

This work is supported by the Swiss Platform for Advanced Scientific Computing (PASC) project "SPH-EXA2: Optimizing Smoothed Particle Hydrodynamics for Exascale Computing", and as part of SKACH consortium through funding with The State Secretariat for Education, Research and Innovation (SERI) of Switzerland. The numerical calculations were supported by the EuroHPC-JU Extreme Scale Access Mode through "TGSF: The role of turbulence and gravity in star formation" (proposal ID: EHPC-EXT-2023E01-031). We also acknowledge support from the LUMI Supercomputer Center and the Swiss National Supercomputing Center (CSCS - allocation c32).

References

[1] G. List, 2023, accessed 9 August 2023. [Online]. Available: https://www.top500.org/lists/green500/

[2] ASTRONET, 2022, accessed 24 April 2024. [Online]. Available: https://www.astronet-eu.org/?page_id=521

[3] A. Cavelan, R. M. Cabezón, M. Grabarczyk, and F. M. Ciorba, "A smoothed particle hydrodynamics mini-app for exascale," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2020, pp. 1–11.

[4] S. Corda, B. Veenboer, and E. Tolley, "Pmt: Power measurement toolkit," in *2022 IEEE/ACM International Workshop on HPC User Support Tools (HUST)*, 2022, pp. 44–47.

[5] A. J. Crespo, J. M. Domínguez, B. D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal, "Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph)," *Computer Physics Communications*, vol. 187, pp. 204–216, 2015.

[6] G. Bilotta, A. Hérault, A. Cappello, G. Ganci, and C. Del Negro, "Gpusph: a smoothed particle hydrodynamics model for the thermal and rheological evolution of lava flows," *Geological Society, London, Special Publications*, vol. 426, no. 1, pp. 387–408, 2016.

[7] G. Oger, E. Jacquin, M. Doring, P. Guilcher, R. Dolbeau, P. Cabelguen, L. Bertaux, D. Le Touze, and B. Alessandrini, "Hybrid cpu-gpu acceleration of the 3d parallel code sph-flow," in *Proceedings of the 5th Spheric Workshop, Manchester, UK*, 2010.

[8] J.-M. Cherfils, G. Pinon, and E. Rivoalen, "Josephine: A parallel sph code for free-surface flows," *Computer Physics Communications*, vol. 183, no. 7, pp. 1468–1480, 2012.

[9] S. Portegies Zwart, "The ecological impact of high-performance computing in astrophysics," *Nature Astronomy*, vol. 4, no. 9, pp. 819–822, 2020.

[10] X. Mei, L. S. Yung, K. Zhao, and X. Chu, "A measurement study of gpu dvfs on energy conservation," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, 2013, pp. 1–5.

[11] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of gpu dvfs on energy conservation," *Digital Communications and Networks*, vol. 3, no. 2, pp. 89–100, 2017.

[12] M. H. Santriaji and H. Hoffmann, "Grape: Minimizing energy for gpu applications with performance requirements," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[13] P. Zou, A. Li, K. Barker, and R. Ge, "Indicator-directed dynamic power management for iterative workloads on gpu-accelerated systems," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 559–568.

[14] S. Götz, T. Ilsche, J. Cardoso, J. Spillner, T. Kissinger, U. Aßmann, W. Lehner, W. E. Nagel, and A. Schill, "Energy-efficient databases using sweet spot frequencies," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 871–876.

[15] J. W. Romein and B. Veenboer, "Powersensor 2: A fast power measurement tool," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2018, pp. 111–113.

[16] D. Abdurachmanov, P. Elmer, G. Eulisse, R. Knight, T. Niemi, J. K. Nurminen, F. Nyback, G. Pestana, Z. Ou, and K. Khan, "Techniques and tools for measuring energy efficiency of scientific software applications," in *Journal of Physics: Conference Series*, vol. 608, no. 1. IOP Publishing, 2015, p. 012032.

[17] "Slurm: Simple linux utility for resource management," https://slurm.schedmd.com, accessed: (2024-04-01).

[18] S. Martin, "Cray xc30 power monitoring and management," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:208012529

[19] S. Martin and G. J. Koprowski, "Cray ® xc tm advanced power management updates," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:202663809

[20] NVML, 2024, accessed 24 April 2024. [Online]. Available: https://docs.nvidia.com/deploy/nvml-api/index.html

[21] R. system management interface (rocm smi) library, 2024, accessed 24 April 2024. [Online]. Available: https://github.com/ROCm/rocm_smi_lib

[22] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, 2010, pp. 189–194.

[23] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *2010 39th international conference on parallel processing workshops*. IEEE, 2010, pp. 207–216.

[24] ARCHER2, 2022, accessed 24 April 2024. [Online]. Available: https://www.archer2.ac.uk/news/2022/12/12/CPUFreq.html

[25] E. Yousefzadeh-Asl-Miandoab, T. Robroek, and P. Tozun, "Profiling and monitoring deep learning training tasks," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*, 2023, pp. 18–25.

[26] S. Keller, A. Cavelan, R. Cabezon, L. Mayer, and F. Ciorba, "Cornerstone: Octree construction algorithms for scalable particle simulations," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2023, pp. 1–10.

[27] B. van Werkhoven, "Kernel tuner: A search-optimizing gpu code auto-tuner," *Future Generation Computer Systems*, vol. 90, pp. 347–358, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18313359

[28] "Piz daint — cscs," www.cscs.ch, 05 2024. [Online]. Available: https://www.cscs.ch/computers/piz-daint/

[29] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of dynamic voltage and frequency scaling on a k20 gpu," in *2013 42nd International Conference on Parallel Processing*. IEEE, 2013, pp. 826–833.

[30] O. S. Simsek, J.-G. Piccinali, and F. M. Ciorba, "Accurate measurement of application-level energy consumption for energy-aware large-scale simulations," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1881–1884.