SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving

Kaijie Fan
kfan@unisa.it
TU Berlin, Germany
University of Salerno, Italy

Marco D'Antonio
University of Salerno, Italy

Lorenzo Carpentieri
University of Salerno, Italy

Biagio Cosenza
University of Salerno, Italy

Federico Ficarelli
CINECA, Italy

Daniele Cesarini
CINECA, Italy

ABSTRACT

Energy-efficient computing uses power management techniques such as frequency scaling to save energy. Implementing energy-efficient techniques on large-scale computing systems is challenging for several reasons. While most modern architectures, including GPUs, are capable of frequency scaling, these features are often not available on large systems. In addition, achieving higher energy savings requires precise energy tuning because not only applications but also different kernels can have different energy characteristics. We propose SYnergy, a novel energy-efficient approach that spans languages, compilers, runtimes, and job schedulers to achieve unprecedented fine-grained energy savings on large-scale heterogeneous clusters. SYnergy defines an extension to the SYCL programming model that allows programmers to define a specific energy goal for each kernel. For example, a kernel can aim to minimize well-known energy metrics such as EDP and ED2P or to achieve predefined energy-performance tradeoffs, such as the best performance with 25% energy savings. Through compiler integration and a machine learning model, each kernel is statically optimized for the specific target. On large computing systems, a SLURM plugin allows SYnergy to run on all available devices in the cluster, providing scalable energy savings. The methodology is inherently portable and has been evaluated on both NVIDIA and AMD GPUs. Experimental results show unprecedented improvements in energy and energy-related metrics on real-world applications, as well as scalable energy savings on a 64-GPU cluster.

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → *Power estimation and optimization.*

KEYWORDS

Frequency scaling, Heterogeneous Computing, Energy efficiency, Modeling

ACM Reference Format:
Kaijie Fan, Marco D'Antonio, Lorenzo Carpentieri, Biagio Cosenza, Federico Ficarelli, and Daniele Cesarini. 2023. SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3581784.3607055

SC '23, November 12–17, 2023, Denver, CO, USA

1 INTRODUCTION

Energy-efficient computing has been identified as a major technology challenge to optimize the performance of exascale applications under power or energy constraints [18]. Rising electricity costs, power constraints, and the diminishing efficiency benefits of Moore's Law have further exacerbated this challenge and increased the need for energy-efficient technology.

One of the most effective technologies for energy-efficient computing is Dynamic Voltage and Frequency Scaling (DVFS), which improves energy efficiency by changing the core or memory frequency until it reaches a voltage and frequency point that is close to the threshold voltage, after which the energy efficiency decreases again [23].

Bringing the benefits of frequency scaling to today's large heterogeneous systems is challenging. First, while modern GPUs now broadly support frequency scaling through hardware vendor libraries such as Intel's RAPL [20], NVIDIA's NVML [32], and AMD's ROCm SMI [2], to the best of our knowledge, there is no portable way to support frequency scaling between CPUs, GPUs, and accelerators that would enable portable power-efficient approaches. The second challenge comes from the need for fine-grained tuning approaches. Related works [8, 15] have shown how different kernels can have a strong energy characterization, therefore leading e.g., to a different energy-optimal frequency. While this has largely been studied on micro-benchmarks and single kernels, large applications cannot simply set the same frequency for all kernels if they want higher energy savings. In terms of energy modeling, it is also important to provide the user with a simple and portable interface that facilitates the selection of the best energy-efficient solution without exposing technical details. Unfortunately, frequency scaling is usually not available to users on large production systems due to potential technical problems: for example, one user can set a frequency too low and the next user will unknowingly experience a slowdown.

All of these arguments call for an energy-efficient approach that does not only focus on a single aspect but rather addresses the energy holistically across the software stack, including the programming model, compiler, library, and job scheduler. To this end, we propose SYnergy a novel software approach to fulfill energy-efficient computing in large-scale computing systems by integrating

a language interface with extensions to compilers, energy models, and job schedulers for scalable energy savings. In summary, the contributions of this paper are as follows:

- A new portable energy interface based on SYCL that allows programmers to define energy targets on a per-kernel rather than per-application basis. The interface enables portable energy profiling and frequency scaling on a wide range of heterogeneous hardware.
- A set of energy targets, which expands traditional energy-delay metrics (EDP and ED2P) with new ones that better express energy tradeoffs, such as the best performing configuration while saving 25% energy (ES_25), or the most energy-efficient while losing only 25% of performance (PL_25).
- A fine-grained energy methodology consisting of a set of energy models integrated into a SYCL compilation toolchain, that enables per-kernel tuning for specific energy targets.
- An energy plugin for SLURM, which enables GPU frequency scaling on HPC production systems.
- An experimental evaluation including energy-saving results on 23 benchmarks on different heterogeneous nodes equipped with AMD MI100 and NVIDIA V100, and a large-scale energy-scalability evaluation on two real-world applications on the Marconi-100 GPU cluster at CINECA.

The rest of this paper is organized as follows. Section 2 presents background and motivations. Section 3 provides an overview of our approach. Section 4, 5 and 6 introduce, respectively, the energy interface, the energy metrics and the modeling methodology. Section 7 describes the energy-aware job scheduler. Section 8 presents the experimental evaluation of our approach. Section 9 and 10 conclude the paper with related work and final conclusion.

2 MOTIVATION

2.1 Energy Interface

Modern processors provide an interface to enable a number of power/energy capabilities offered by the hardware. Typically, we are interested in two functionalities: an interface that allows us to report on either the current power draw or the accumulated energy consumption of a defined power domain; the ability to dynamically scale the frequency of a core or memory. The vendor's power/energy interfaces, however, are very different from each other, and there is no common interface to provide portable common functionality. For example, Intel CPU provides the Running Average Power Limit (RAPL) interface [20], which is capable of very accurate cumulative energy consumption estimation at different levels, including package, core, uncore, and DRAM. Recently, GPUs have begun to provide power/energy interfaces. The NVIDIA Management Library (NVML) reports on the current board power draw and power limits, as well as enables frequency scaling. While current high-end NVIDIA GPUs only enable frequency change for the core frequency, a few models also enable to select one out of four different memory frequencies, e.g., NVIDIA Titan X. Similar interfaces are also supported by AMD and Intel through ROCm SMI [2] and Level Zero [21], respectively.

Figure 1 shows the available core and memory frequencies for three major GPUs from NVIDIA and AMD. For NVIDIA V100, the memory frequency is set to 877 MHz, and we have a total of 196

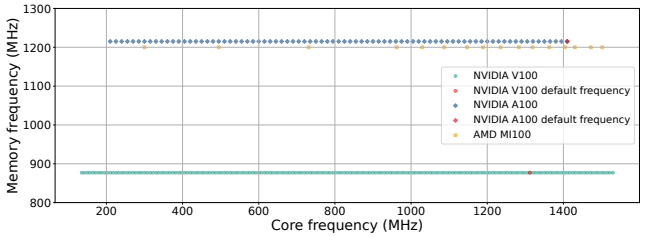
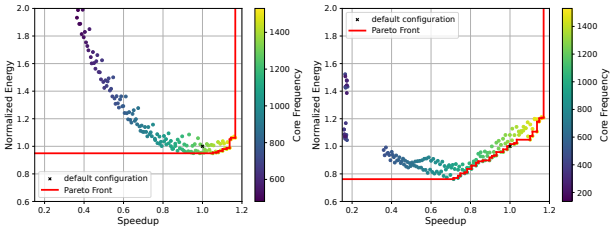


Figure 1: Available frequencies for NVIDIA V100, A100 and AMD M100.

core frequency configurations, from 135 to 1530 MHz. For NVIDIA A100, the memory frequency is set to 1215 MHz, and we have a total of 81 core frequency configurations, from 210 to 1410 MHz. On AMD MI100, the memory frequency is set to 1200 MHz, and we have a total of 16 core configurations, from 300 to 1502 MHz. Differently from NVIDIA, AMD MI100 does not provide a default frequency configuration, as the frequency is automatically adjusted based on the workload. Unfortunately, GPU power interfaces are rather limited when compared to their CPU counterparts. They have limited support for power/energy domains, e.g., frequency scaling cannot be applied to a specific compute unit or streaming multiprocessor. In addition, GPU power reads are asynchronous and, one should be aware of potential delays, especially when measuring the energy of small kernel executions. Not only is there no portable interface for power and energy, but *current programming models do not allow for integration with existing power capabilities.*

2.2 Fine-grained Energy Tuning

It is important to understand that energy consumption is largely dependent on the kernel. Figure 2 shows two kernels, a linear regression, and a median filter, executed on an NVIDIA V100 GPU. The graph shows the essential multi-objective nature of the energy problem, where the x-axis represents the speedup and the y-axis is the normalized per-task energy consumption. The baseline is the default configuration (1312 MHz core and 877 MHz memory frequency). The red line is the Pareto front.



(a) Linear Regression **(b) Median Filter**

Figure 2: Two kernels with different energy characterization.

The two kernels have very different energy characterization: `Linear Regression` (Figure 2a) has overall high energy consumption, there are no configurations that allow saving more than 10% of the energy and, in general, low core-frequency configurations are very inefficient in performance. On the other hand,

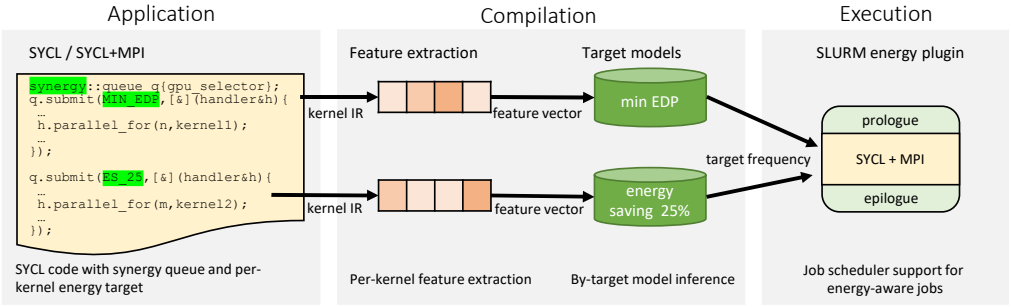


Figure 3: SYnergy Approach Overview.

`Median Filter` (Figure 2b) has the potential for more than 20% energy savings, and low-frequency configurations do not lose too much performance. Energy characterization depends on many factors, for example, computationally bounded kernels are very sensitive to changes in the core frequency. Overall, it is clear that setting the same frequency for all the kernels in an application, i.e., *coarse-grained tuning is not optimal*; ideally, *for greater energy savings, we prefer a fine-grained solution that explores per-kernel energy tuning.*

2.3 Energy Support by Job Scheduler

Modern job schedulers provide modules to deal with power management. SLURM [43] provides an integrated power management system for energy accounting [9] and power capping [11], which takes the configured power cap for the system and distributes it across the nodes controlled by SLURM. SLURM lowers the power caps on nodes that are consuming less than their cap and redistributes that power to other nodes, with configurable power thresholds. SLURM also provides a power-saving mechanism for powering down idle nodes. Custom SLURM plugins provide energy accounting for specific infrastructure such as Cray's per-node power and energy data from the head node [40], and IBM's systems director active energy manager [24], which provides power and energy measurements on each node. However, a scheduler can only operate at the job level, because it can only see the job as a whole and cannot implement per-kernel specific energy/power techniques. In other words, a *job scheduler can only implement coarse-grained energy/power techniques*; *implementing fine-grained techniques necessarily requires providing an energy interface to the users.*

Unfortunately, SLURM installations on large-scale clusters typically do not provide advanced energy capabilities to the users. The main reason for this limitation is due to technical problems that may arise if, for example, any user is allowed to change the frequency: a user may potentially set the core frequency to a too-low core frequency, and unintentionally affect the next users of the same resources, e.g., the GPU.

3 SYNERGY DESIGN

SYnergy is a software layer for general-purpose, cross-stack, energy-efficient computing on large-scale HPC clusters. It takes advantage of the frequency-scaling capabilities of modern hardware, especially GPUs, and exposes them to the user through a high-level, easy-to-use interface.

3.1 Overview

SYnergy is built around three basic components: a SYCL-based API, a compilation framework integrated with a set of target models, and a SLURM plugin. Figure 3 illustrates the relationships between these components.

A SYnergy application is either a SYCL or SYCL+MPI application that uses the SYnergy API. SYnergy's interface is basically a wrapper of the `sycl::queue`, providing additional features to allow frequency scaling and energy/power measurements at the kernel level. In particular, each kernel is annotated with a specific energy target.

At compile time, the SYCL compilation toolchain is extended with an additional feature extraction pass that statically extracts a feature vector from each kernel, which is later fed into a target machine learning model that computes the target frequency. Model inference depends on the energy target provided by the user, and the predicted frequency configuration is made available to the SYCL library at runtime.

After compilation, a SYnergy application is already energy-aware and instrumented with the frequency settings specified by the per-kernel user annotation. In the case of an MPI+SYCL application, to enable the frequency scaling and measurement capabilities on a cluster, a SLURM job scheduler plugin extends the job execution policy with a specific prologue and epilogue, enabling energy-efficient computing on all devices in the job.

3.2 Deployment

The current version of SYnergy has been installed on the Marconi-100 cluster at CINECA. However, the whole approach can be easily deployed on a new cluster with very little effort.

Since the programming interface is based on SYCL and is completely transparent to the hardware, a program written in SYnergy can easily be executed on another machine without any changes.

Installing SYnergy on a new system requires two steps. First, the energy models must be built for each target device. For this purpose, each code in the micro-benchmark is executed with different core frequency configurations. The execution data, i.e., per-task energy and runtime, represent the training set on which the energy-target models are built. Details regarding the modeling can be found in Section 6, while details about the deployment are available in the AD/AE Appendix. The second step is related to the availability of energy capabilities for per-device power/energy readings and frequency scaling. The SYnergy approach provides a SLURM plugin that must be installed in order to give any user such capabilities.

The current implementation of the SYnergy's SLURM plugin has a prologue and epilogue logic to support NVIDIA GPUs, but it can be easily extended to other vendors. See Section 7 for details.

4 SYNERGY PROGRAMMING INTERFACE

The SYnergy energy-aware SYCL API is one of the building blocks upon which our system is built. The API, inspired by the SYCL extension Celerity [37, 39], provides a common interface that allows energy profiling and frequency scaling on devices from different vendors, without requiring developers to work with vendor-specific libraries. The API is released as a header-only library that can be integrated into an existing C++ building environment. This paper uses the interface bindings for NVIDIA and AMD GPUs, which are mapped into the NVML [32] and ROCm SMI [2], respectively.

4.1 SYCL

SYCL [14] is a programming model for heterogeneous computing that builds on modern C++. While SYCL follows the execution and memory model of OpenCL [12, 13], implementations can have a non-OpenCL mapping, e.g., the Open SYCL CUDA backend [1]. SYCL supports single-source programming where both kernel and host codes are stored in the same source file. In a SYCL program, the kernel code to be executed on a device such as a GPU, is expressed by the `parallel_for` function from the command group handler object. The device on which the kernel code executes is represented by a queue.

4.2 Profiling Interface

The main entry point for using the SYnergy interface is the `synergy::queue` class, which extends the standard SYCL queue with energy capabilities. The SYnergy API provides coarse-grained and fine-grained energy profiling capabilities that allow measuring, respectively, the energy consumption of the whole device and the energy consumption of each kernel executed in a SYCL application.

Listing 1 shows how the API can be used to query the energy consumed by a kernel, i.e., a `parallel_for` and a device using `kernel_energy_consumption` and `device_energy_consumption` functions, respectively. Since device computation is asynchronous, we wait for the kernel to finish before querying energy consumption.

```
synergy::queue q{gpu_selector_v};
buffer<float, 1> x_buf{x};
buffer<float, 1> y_buf{y};
buffer<float, 1> z_buf{z};
event e = q.submit([&](handler& h) {
  accessor<float, 1, read> x_acc{x_buf, h};
  accessor<float, 1, read> y_acc{y_buf, h};
  accessor<float, 1, write> z_acc{z_buf, h};
  float a{alpha};
  h.parallel_for(range<1>{n}, [=](id<1> id) {
    z_acc[id] = a * x_acc[id] + y_acc[id];
  });
});
e.wait_and_throw();
double kernel_energy = q.kernel_energy_consumption(e);
double device_energy = q.device_energy_consumption();
```

Listing 1: Energy profiling with the SYnergy API

With the coarse-grained approach, the device energy consumption is measured by sampling the instantaneous power in a time window that begins when the SYnergy queue is built and ends when it is destroyed. This feature is useful whereas an application is made up of a mix of large and small kernels, the latter of which are not easy to profile because of the short execution time as explained in 4.4. In order to enable fine-grained energy profiling the SYnergy API leverages the SYCL event features that allow us to query the execution status of a kernel (i.e. submitted, running, complete). Using an asynchronous thread to poll the kernel status we sample the power of a kernel until it is complete. In this way, we measure only the kernel energy consumption rather than the entire device.

4.3 Frequency Scaling Interface

A SYnergy queue can be constructed as a conventional SYCL queue or by manually defining the memory and core frequencies configuration for the kernels that will be submitted to the queue. Listing 2 illustrates a SYnergy queue construction with a memory frequency of 1215 MHz and a core frequency of 210 MHz.

```
synergy::queue q{1215, 210, gpu_selector_v};
... // setup buffers
event e = q.submit([&](handler& h) {
  ... // setup accessors
  h.parallel_for(n, kernel);
});
```

Listing 2: SYnergy queue with target frequencies

For fine-grained frequency scaling, each kernel submitted to the queue can be executed with a specified frequency configuration, which is set just before the kernel starts.

When integrated into the whole SYnergy architecture, the API can be used to specify an energy target (energy target metrics are defined in Section 5): MIN_EDP, MIN_ED2P, ES_x, PL_x. Listing 3 shows this type of kernel submission. The API can be used without these targets, potentially as a standalone API for any SYCL application where frequency scaling and energy profiling are desired.

```
synergy::queue q{gpu_selector_v};
... // setup buffers
q.submit(MIN_EDP, [&](handler& h) {
  ... // setup accessors
  h.parallel_for(n, kernel);
});
```

Listing 3: SYnergy kernel submitted with MIN_EDP target

Finally, all of these approaches can be mixed, allowing for multiple queues with different target configurations, with the ability to specify a target for each kernel submission, as shown in Listing 4.

```
synergy::queue low_freq{877, 810, gpu_selector_v};
synergy::queue default_freq{gpu_selector_v};
... // setup buffers
low_freq.submit([&](handler& h) {
  ... // setup accessors
  h.parallel_for(n, kernel1);
});
default_freq.submit(877, 1530, [&](handler& h) {
  ... // setup accessors
  h.parallel_for(m, kernel2);
});
```

Listing 4: Queues and kernels with different targets

4.4 Limitations

Since SYCL has no way to execute instructions just before a kernel starts executing on the device, SYnergy implements frequency

scaling in the command group. However, as the kernel execution is asynchronous, and the corresponding kernel may not have been executed yet; therefore, we wait for the completion of the task.

The energy profiling and frequency scaling are also affected by issues related to the underlying vendor-specific libraries. Accurate fine-grained energy profiling is limited by the fact that the kernel execution must be long enough in order to produce meaningful results, due to the maximum sampling frequency supported by the hardware, which needs around 15 ms long sampling intervals [5].

Our experiments have outlined that the frequency scaling using the NVML library introduces an overhead that becomes significant as the number of submitted kernels grows.

5 ENERGY METRICS

Our approach focuses on delivering energy-efficient configurations without sacrificing performance. By trying to minimize energy consumption and maximize performance, we formulate a multi-objective problem. While applying frequency scaling, energy consumption and performance are not strongly correlated, which means that there is not a single optimal solution, but a set of Pareto optimal dominant solutions.

Dealing with multi-objective problems and Pareto sets makes the optimization process more complicated for users, who need to understand the different tradeoffs at stake. To provide a high-level, easy-to-use interface, it would be nice to provide a simple tuning interface that returns the relevant Pareto optimal configuration to the user. Scalar metrics are a solution to this problem.

5.1 Energy Delay Products

An example of a scalar metric is the Energy-Delay Product (EDP) [19], which was originally proposed to provide insight into some of the basic trade-offs in low-power design. EDP is defined as the product of energy consumed and the execution time: $EDP = et$. A variant of EDP is the Energy-Delay Square Product (ED2P) metric, which gives more importance to the execution time and is defined as $ED2P = et^2$. In the context of software optimization based on frequency scaling, both EDP and ED2P metrics provide only limited insight into the overall distribution of the Pareto optimal set.

Figure 4 shows energy and performance data for the Black-Scholes benchmark. In Figure 4a and 4b we can see how EDP and ED2P change as the core frequency increases. The blue and green points represent the core frequency value that minimizes, respectively, EDP and ED2P. As expected, ED2P is very close to the configuration that delivers maximum performance at maximum core frequency, and therefore should not be considered a tradeoff metric. By looking at the entire distribution, we can see that there are many Pareto optimal solutions between the minimum energy point and the one with maximum performance. The EDP optimal point lies in between, but users may be interested in finding more tradeoffs in this interval.

5.2 Energy Saving Metric

To provide a better understanding of the tradeoff solutions that can potentially be selected, we define a set of new metrics that aim to provide easy-to-use and interpretable energy tradeoffs. In defining a new metric, we start with two observations. First, the default core frequency configuration is typically very close to the maximum

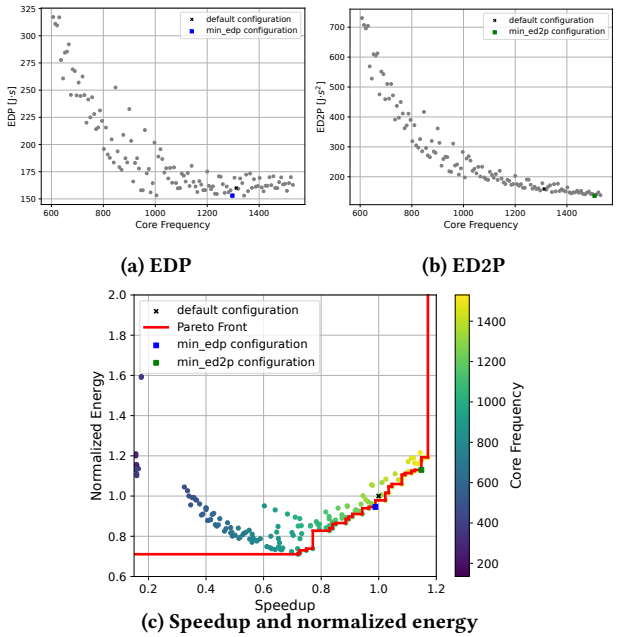


Figure 4: Black-Scholes benchmark.

available frequency, which means that by default GPU drivers are tuned for maximum performance. The second consideration is that looking at the multi-objective distribution, interesting tradeoff solutions lie in the interval between the core frequency that delivers the minimum energy and the default one.

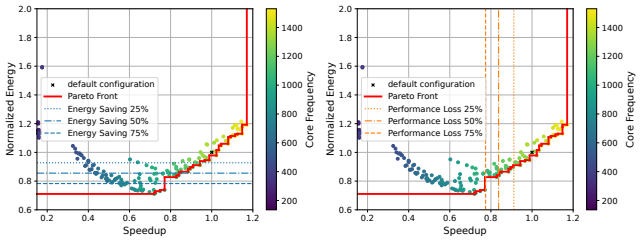
We define the energy savings metric ES_x, the frequency configuration that delivers the x% energy savings, in terms of the potential savings using the default frequency as a baseline. For example, ES_100 is the frequency with the lowest power consumption. Figure 5a shows the energy saving metric for the Black-Scholes benchmark. The blue dotted lines represents, respectively, ES_25, ES_50 and ES_75.

5.3 Performance Loss Metric

Another way to look at energy tradeoffs is to focus on performance loss rather than energy savings. In this view, we define the performance loss metric PL_x as the frequency configuration that has a x% performance degradation, in terms of the potential performance loss using the default frequency as a baseline. This metric operates on the same interval as ES; however, it scales on the performance values rather than the energy values. Figure 5b shows the performance loss metric for the Black-Scholes benchmark. The orange dotted lines represents, respectively, PL_25, PL_50 and PL_75.

6 MODELING METHODOLOGY

One of SYnergy's key design features is finding an optimal frequency configuration to minimize execution time, energy consumption, and energy-delay products, and furthermore, achieve the energy saving and performance loss metric. This process requires modeling how the energy metrics (in Section 5) change with different workload characterization and frequency scaling. Our proposed



(a) Energy Saving **(b) Performance Loss**

Figure 5: Energy Metrics for Black-Scholes.

methodology is based on the typical two-phase modeling with supervised learning: each metric model is built in the training phase; later, when a new input workload is provided, a predicting phase infers the optimal frequency configuration.

The description of the two phases is presented below.

6.1 Training Phase

The goal of the training phase is to build separate models for execution time, energy consumption, EDP, and ED2P. Figure 6 illustrate the workflow for the training phase.

We first, instead of using existing benchmarks, construct a set of micro-benchmarks and extract a set of static features of each micro-benchmarks to build the training set ❶. Formally, any micro-benchmark is represented by a static feature vector

$$\vec{k} = (k_{int\_add}, k_{int\_mul}, k_{int\_div}, k_{int\_bw}, k_{float\_add}, k_{float\_mul}, k_{float\_div}, k_{sf}, k_{gl\_access}, k_{loc\_access})$$

where each element represents a specific instruction type. The description of each feature is listed in Table 1.

Table 1: Static code features.

Feature	Description
k_{int_add}	integer additions and subtractions
k_{int_mul}	integer multiplications
k_{int_div}	integer divisions
k_{int_bw}	integer bitwise operations
k_{float_add}	floating point additions and subtractions
k_{float_mul}	floating point multiplications
k_{float_div}	floating point divisions
k_{sf}	special functions
k_{gl_access}	global memory accesses
k_{loc_access}	local memory accesses

Meanwhile, each micro-benchmark is executed with various frequency configurations ❷ to obtain execution time (t) and energy consumption (e) measurements and then calculate EDP (edp) and ED2P ($ed2p$). Frequency configurations, represented as a frequency vector $\vec{f}$, together with the different metric measurements are also used to build the training set T.

Once the training set $T = (\vec{k}, \vec{f}, e, t, edp, ed2p)$ is prepared, different machine learning methods are applied to build four single-target models ❸ of execution time $F_t(\vec{k}, \vec{f})$, energy consumption $F_e(\vec{k}, \vec{f})$, EDP $F_{edp}(\vec{k}, \vec{f})$ and ED2P $F_{ed2p}(\vec{k}, \vec{f})$.

6.2 Prediction Phase

The final goal of the modeling workflow is to predict several frequency configurations which bring optimal energy metrics separately for a new workload. Figure 6 explains how our modeling workflow is capable of predicting the optimal frequency setting for a new input SYCL workload.

For the target of execution time, energy consumption, EDP, and ED2P metric, we first extract the new static code features ❹ and generate a feature vector $\vec{k}'$ of a new workload. Together with the frequency configuration vector $\vec{f}$, we build the prediction dataset $P = (\vec{k}', \vec{f})$ to represent each new input SYCL workload. Combining the four single-target models obtained in the training phase ❺, the four metrics with different frequency configurations are predicted: $\hat{t} = F_t(\vec{k}', \vec{f})$, $\hat{e} = F_e(\vec{k}', \vec{f})$, $\hat{edp} = F_{edp}(\vec{k}', \vec{f})$, and $\hat{ed2p} = F_{ed2p}(\vec{k}', \vec{f})$.

Once we have the above four predictions, we can easily find the frequency configuration $\hat{f}$ by using a search algorithm ❻ for different energy targets, e.g., MIN_EDP, ES_25 or PL_25, which is able to be defined by users.

7 ENERGY-AWARE JOB SCHEDULER

To allow users to collect energy and power measurements and programmatically set the operating frequencies of GPU boards for this work, appropriate facilities had to be put in place. Given the selected platform for running experiments [6], the GPU board vendor provides interfaces for monitoring, profiling, and performance bounds manipulation. NVML [32] is a C language programming interface for monitoring and managing various device states within NVIDIA data center class GPUs. It exposes through its APIs several board management actions, e.g., initialization and cleanup, board status queries, control, and events monitoring.

7.1 HPC Cluster Energy Monitoring Challenges

For system integrity reasons, all of the statechanging API calls are normally restricted to the root user. It's possible though, through documented API calls, to allow unprivileged users to access some otherwise privileged NVML calls with a setting that controls permissions on a per-GPU granularity. The active production status of the Marconi100 HPC system poses additional challenges. In the context of an HPC production machine, where the system performs multiple jobs on the same nodes in an ordered fashion, low-level control APIs of devices pose potential configuration hazards for the nodes: via uncontrolled access to the devices, a user could easily change performance states, lowering frequencies or capping the power consumption, and potentially leaving these configurations at the termination of a job. In this case, the next job allocated on the node will run at the GPU performance state left by the previous job, without the possibility of exploiting the maximum performance that is usually granted by the system. In order to leave the node in a consistent performance state at the end of a privileged job, we decided to pursue an approach based on temporary privilege raising by the SLURM [44] job scheduler, the infrastructure component is responsible for managing the entirety of machine resources.

The approach can be outlined as follows. First, the user explicitly requests privileged GPU access in a specific batch job submitted to

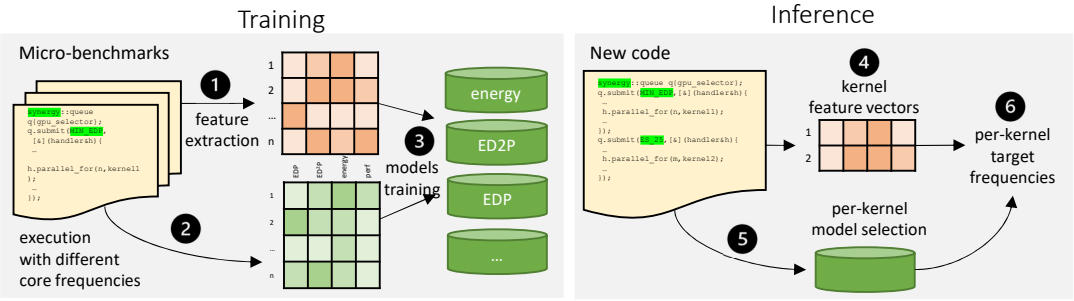


Figure 6: Machine learning based energy models

the scheduler. Then, the scheduler ensures that the job has requested for exclusive allocation of nodes: this ensures that while the user has temporary access to privileged interfaces, they cannot affect the performance or infer details of other jobs running on the same node. When the job starts and tasks are assigned, the SLURM plugin ensures that privileges are elevated on the devices. When the job ends, the same SLURM plugin removes the privileges granted to the job's user and restores the default performance bounds.

For each GPU installed on the system, there are three clock frequency levels available for SM cores (while the GPU memory frequency is fixed for HBM-based devices): application clock frequency, the frequency that the GPU uses as a target when executing instruction streams from both compute and shader kernels; minimum and maximum clock frequency, hard bounds that cannot be overridden by application frequency in any way. These are set by the root user only and privileges for these bounds cannot be lowered.

For the purpose of this work, the API call that allows for application clock frequency privilege lowering is `nvmlDeviceSetAPIRestriction`: the SLURM plugin developed here leverages this API call to temporarily lower privilege requirements for jobs that are both using nodes exclusively and request for a specific runtime feature according to requirements defined according to CINECA's system administration team.

7.2 SLURM Plugin

The developed `nvgpufreq` plugin allows accessing the NVML privileged clock settings by regular users when an exclusive job run on a node marked with a specific tag, we used the Generic RESources (GRES) of SLURM to tag nodes with these capabilities. The plugin operates by intercepting the *prologue* and *epilogue* of each job submitted to the cluster (via SLURM own extension hooks).

In the *prologue* hook, the plugin performs the following steps. First, it retrieves the node info from the scheduler *daemon* (`slurmctld`). If the plugin fails to do so, it terminates its execution. Then, it checks if the node is tagged with the `nvgpufreq` GRES. If the node is not tagged, the plugin terminates its execution. Another check is performed if NVML shared object is available for dynamic loading (via `dlopen`), otherwise, the plugin terminates its execution; it also checks if the job is tagged with `nvgpufreq` GRES (if the job does not specify the aforementioned GRES, it terminates its own execution) and if the job runs exclusively on the node (if the node can be shared among multiple jobs the plugin terminates its execution). If all of the above checks are successful, the actual privilege requirements for application clock settings are lowered on the GPU boards allocated to the specific job otherwise the plugin terminates its execution without applying any configuration. When the job terminates for any reason, the *epilogue* hook is run and the plugin performs a full cleanup procedure restoring the node's GPUs at the maximum frequency and removing the privileged access.

The infrastructure, after configuration auditing, has been deployed on the Marconi100 [6] system to allow the user base to run large-scale measurement campaigns for any standard users.

8 EXPERIMENTAL EVALUATION

In this section, we first present the experimental setup that includes the SYCL benchmarks and hardware platform. The evaluation consists of an analysis of SYCL benchmark characterization, followed by a prediction accuracy analysis of different ML algorithms. Finally, we analyze the energy scalability on a multi-node computing system.

8.1 Experimental Setup

For the experimental evaluation of our approach, we rely on the Intel DPC++ [22] SYCL implementation. Since SYnergy wraps vendor-specific libraries to target different architectures we need NVML on the nodes with an NVIDIA GPU and ROCm SMI on the AMD nodes.

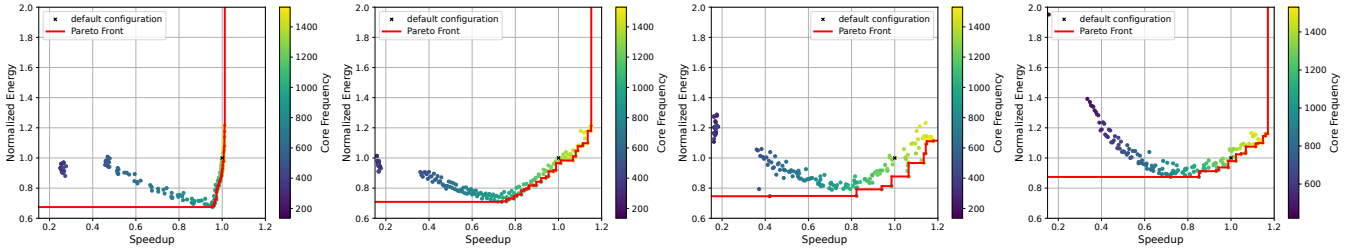
Single node. We conduct the evaluation on 23 applications from the SYCL benchmark suite using two different nodes: one equipped with an AMD EPYC 7313 processor, and an AMD MI100 GPU; the other node is equipped with a Power9 processor and four NVIDIA V100 GPUs.

Multi-node. The multi-node experiment is carried out on the Marconi100, an accelerated cluster based on IBM Power9 processors and NVIDIA V100 GPUs. The nodes are connected by a Mellanox Infiniband EDR with a DragonFly+ topology. Tests have been performed with up to 16 nodes (4 GPUs per node) using two real-world SYCL + MPI applications: CloverLeaf [17] and MiniWeather [31].

8.2 SYCL Benchmark Characterization Analysis

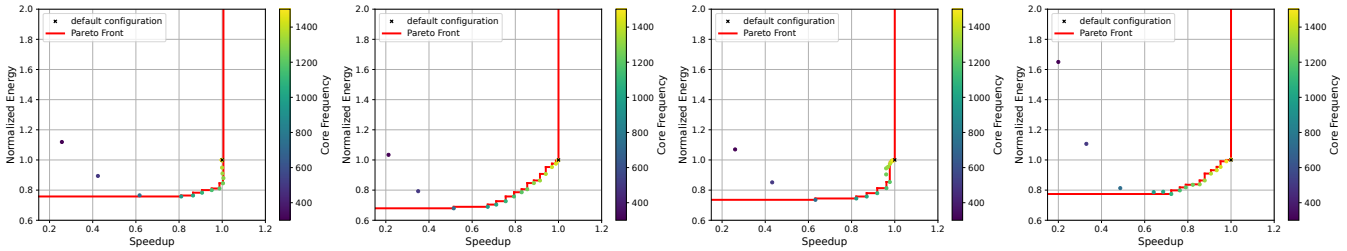
This section analyzes the SYCL benchmarks characterization in terms of speedup, normalized energy consumption, and Pareto Front on both NVIDIA V100 and AMD MI100 GPUs.

In Figure 7 and Figure 8, we show a selection of four significant benchmarks taken from the 23 SYCL benchmarks executed on NVIDIA V100 and AMD MI100, separately. For each benchmark, we present speedup (x-axis) and normalized energy (y-axis) with



(a) Matrix Multiplication **(b) Sobel 3** **(c) Mersenne Twister** **(d) FTLE**

Figure 7: Multi-objective characterization of the selected benchmarks from the SYCL benchmark suite on NVIDIA V100.



(a) Matrix Mutliplication **(b) Sobel 3** **(c) Mersenne Twister** **(d) FTLE**

Figure 8: Multi-objective characterization of the selected benchmarks from the SYCL benchmark suite on AMD MI100.

different frequency configurations; the reference baseline (black cross point) for both corresponds to the energy and performance value of the default frequency configuration. Based on the speedup and normalized energy profiling, we derive the Pareto Front (red line).

Speedup. In terms of experiment on NVIDIA V100, `Sobel3` (Figure 7b) shows a high variance with respect to the core frequencies lying in the Pareto Front: speedup goes from 0.73 up to 1.15. At the other extreme, `Matrix Multiplication` (Figure 7a) shows very little speedup difference while increasing the core frequency lying in the Pareto Front: speedup only goes from 0.95 to 1.01, which means there is less performance improvement space in Pareto Front. Other benchmarks behave within those two extreme cases. However, for the experiment on AMD MI100 (Figure 8), the default configuration always brings the best performance for all the SYCL benchmarks.

Normalized energy. Pareto Front of `Matrix Multiplication` on NVIDIA V100 (Figure 7a) and AMD MI100 (Figure 8a) show a larger slope than other benchmarks. In other words, `Matrix Multiplication` tested on NVIDIA V100 saves 33% energy with only 5% performance loss compared with the default configuration. `Sobel3` could also save 30% energy but with 27% performance loss. In general, we find that the default configuration on NVIDIA V100 is not the optimal choice, and it is even not a Pareto-optimal solution in some cases. There exists more space to find performance-energy tradeoffs on NVIDIA V100.

8.3 Prediction Accuracy Analysis

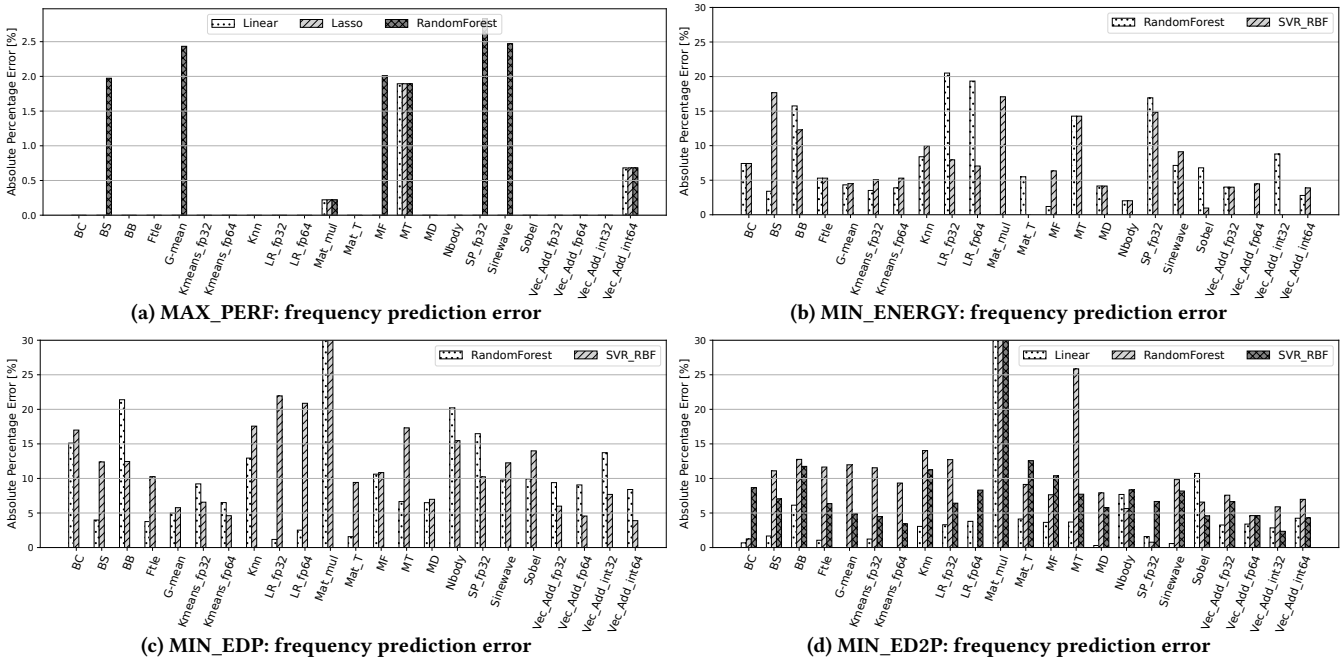
In our work, one main goal is to build models accurately predicting frequency configuration according to the user-defined objectives, e.g., MAX_PERF, MIN_EDP, ES_x, or PL_x. In this section, we discuss the prediction accuracy of different ML algorithms for our energy metrics and select the best ML algorithm to predict the optimal frequency configuration of the two real-world applications.

The tested ML regression algorithms include linear regression, least absolute shrinkage and selection operator (Lasso), Random Forest, and support vector machine regression with RBF kernel (SVR_RBF). As our four modeling targets have different behavior with frequency scaling, we use linear regression, Lasso, and Random Forest algorithms to train the performance model, while using Linear, Random Forest, and SVR_RBF to train energy, EDP, and ED2P models.

To better analyze the prediction accuracy, we measure absolute percentage error (APE), mean absolute percentage error (MAPE), and root mean square error (RMSE) between predicted and actual values. It is notable that the error metrics are not between the predicted and actual objectives, e.g., performance or energy, but between the predicted and actual optimal frequency. In fact, the actual value is one objective obtained from the training set according to the actual optimal frequency. The predicted value is the same objective obtained from the training set but corresponds to the predicted optimal frequency.

Figure 9 shows the APE results of frequency predictions for each SYCL benchmark using different ML algorithms. Each subfigure represents the error of predicted optimal frequency achieving the user-defined energy objectives. In a few cases, especially in Figure 9a, some of the APE results are zero which represents the predicted frequency is the same as the actual optimal frequency.

Table 2 lists the RMSE and MAPE results of predicted frequency targeting the user-defined objectives. From Figure 9 and Table 2, we can find that for modeling performance, ED2P, and performance loss (PL_x) metrics, Linear regression performs better, while using the Random Forest algorithm has a better prediction of energy, EDP and energy saving (ES_x) metrics.



(a) MAX_PERF: frequency prediction error

(b) MIN_ENERGY: frequency prediction error

(c) MIN_EDP: frequency prediction error

(d) MIN_ED2P: frequency prediction error

Figure 9: Frequency prediction error for various benchmarks using different ML algorithms.

Table 2: Error analysis of each objective by using different ML algorithms.

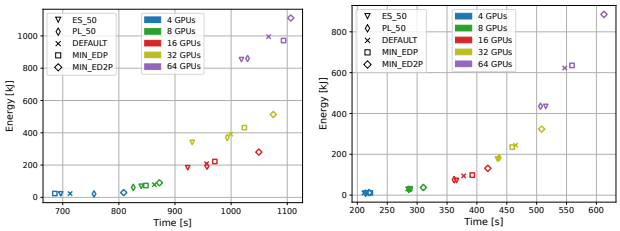
	Linear		Lasso		RandomForest		SVR		
Objective	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	**Best**
MAX_PERF	0.0014	0.001	0.0014	0.0012	0.0107	0.0072	-	-	Linear
MIN_ENERGY	-	-	-	-	3.29	0.066	5.65	0.071	RandomForest
MIN_EDP	-	-	-	-	6.06	0.104	7.42	0.130	RandomForest
MIN_ED2P	4.67	0.056	-	-	5.58	0.099	5.26	0.092	Linear
ES_25	-	-	-	-	1.95	0.029	2.98	0.04	RandomForest
ES_50	-	-	-	-	2.87	0.040	3.24	0.043	RandomForest
ES_75	-	-	-	-	5.17	0.061	4.21	0.043	RandomForest
PL_25	0.0391	0.0443	0.0391	0.0443	0.0519	0.047	-	-	Linear
PL_50	0.0596	0.0686	0.0596	0.0686	0.0771	0.0759	-	-	Linear
PL_75	0.1246	0.0986	0.1246	0.0986	0.1239	0.0986	-	-	Linear

8.4 Energy Scalability on Multiple Nodes

We validate our methodology on large-scale clusters by presenting an energy scaling characterization of the two real-world applications CloverLeaf and MiniWeather. CloverLeaf is an application that solves Euler's equations of compressible fluid dynamics in two spatial dimensions. MiniWeather simulates weather-like flows, it uses the YAKL [30] library for launching kernel, in order to provide portability across different programming models. We integrate both CloverLeaf and MiniWeather existent SYCL versions in SYnergy to apply the fine-grained frequency scaling. Figure 10a and Figure 10b show the energy scaling of the two applications up to 64 NVIDIA V100 GPUs using a weak scaling approach. For both applications, performance scalability is limited by GPU memory constraints that affect MPI applications. The performance scaling behavior of CloverLeaf is in line with previous scaling analysis on the application [28].

We present time on the x-axis and energy on the y-axis. Each point in the plot corresponds to a version of the application in which a frequency configuration for each kernel is selected according to one of the metrics defined in Section 5. The reference baseline is represented by a cross and corresponds to the energy and time value of the default frequency configuration for all kernels. The energy consumption regards only the GPU devices, while the execution time captured by the applications includes computation and communication.

Regarding the EDP metric, both applications show an energy behavior that is similar to the default frequency. The results shown in the single-node characterization analysis are also confirmed in the large-scale environment. In fact, the ES_x and PL_x targets allow us to explore other Pareto-optimal solutions, providing energy improvements and performance gains as we can see from ES_50 and PL_50.



(a) CloverLeaf **(b) MiniWeather**

Figure 10: Real-world applications energy scaling.

Overall, ES_50 and PL_50 achieve good energy savings, allowing an energy save of around 20% in the CloverLeaf application and up to 30% in the MiniWeather application.

9 RELATED WORK

The significant growth in computing ability of large computing clusters and supercomputers is clearly accompanied by a huge increase in energy consumption. Hence, power and energy constrains have become more and more crucial. A significant amount of research has been done for improving the energy efficiency issue.

DVFS-based technique. DVFS, as one of the most widely used strategies for improving energy efficiency, has been investigated by numerous studies. Some focused on the different performance-energy metrics, including performance, energy or energy-delay-product [4, 15, 26, 33, 41], while others used Pareto Front to find Pareto-optimal solutions for achieving performance-energy trade-offs [7, 8, 29, 47]. Among them, Sourouri et al. [38] presented an energy-conserving methodology, which combines the strengths of fine-grained autotuning with DVFS in a real-world HPC application but only on a single compute node. These approaches, however, are limited to CPUs or GPUs. Even though there is some relevant research based on DVFS on heterogeneous architectures, the DVFS technology is actually implemented separately. There are still gaps in DVFS portability between CPUs, NVIDIA GPUs, AMD GPUs, and other accelerators.

Scheduler. The latest research demonstrated the importance of scheduling, including workload distribution and resource allocation, as a key decision variable in energy efficiency optimization both on homogeneous multicore CPU clusters and heterogeneous systems. Examples include workload distribution prediction to achieve optimal energy and performance [25, 27], job scheduler to manage power constraints [3, 10, 35, 36, 42]. Besides, Power capping allocation is becoming essential for keeping large-scale systems within a fixed power constraint and has become a major concern for HPC operating systems, especially the future exascale supercomputers. Zhang *et al.* [45, 46] presented *PowerShift* and *PoDD*, which are dynamic, hierarchical, distributed power management system for coupled applications in large-scale systems, to determine optimal power and performance tradeoffs. Ramesh *et al.* [34] modeled the impact of dynamic power capping schemes on progress for a set of online HPC applications. Hao *et al.* [16] combined the powercap with uncore frequency scaling and proposed a machine learning modeling to predict the Pareto-optimal powercap configurations for achieving trade-offs among performance and energy consumption.

Table 3 compares the capability of several related works and SYnergy. By comparison, our proposal SYnergy offers a novel software approach for energy-efficient computing in heterogeneous systems, integrating a language interface with extensions to compilers, energy models, and job schedulers, fulfilling fine-grained tuning for scalable energy savings. To the best of our knowledge, our work is the first programming interface that allows programmers to annotate kernels with energy targets.

Table 3: Comparison against the state-of-the-art.

	Sourouri et al. [38]	**Hao et al.** [16]	**PoDD** [46]	**SYnergy**
Scope	a real-world HPC application	MPI and OpenMP applications	coupled applications	SYCL benchmarks and two real-world applications
Approach	dynamically tune core frequency, uncore frequency, and the number of threads, exhaustive search	powercap allocation based on ML	powercap, dynamically tune power allocation by classifying applications and online model	language support, compilation with per-kernel target energy model, job scheduler extension
Granularity	fine-grained (kernel)	fine-grained (kernel)	coarse-grained (application)	fine-grained (kernel)
GPU	×	×	×	✓
Objective	Energy, EDP, ED^2P	Pareto front	1/runtime	EDP, ED^2P, ES_x, PL_x

10 CONCLUSION

This paper presents SYnergy, a novel approach for energy-efficient computing that embraces programming model, compiler and job scheduler for energy scalability on large-scale production clusters.

SYnergy makes three major breakthroughs: a SYCL-based energy interface that extends sycl::queue with energy capabilities, allowing programmers to specify per-kernel energy targets; a compilation and modeling approach that automatically predicts the energy-performance tradeoff based on user-defined targets; a SLURM scheduler plugin that enables energy features on large clusters.

The SYnergy approach has been experimentally evaluated on NVIDIA V100 and AMD MI100, on a collection of 23 benchmarks. A large-scale experimental evaluation has also been performed on the Marconi100 cluster, showing scalable energy savings with up to 64 GPUs. The SYnergy approach and newly defined metrics allowed us to discover solutions with up to 30% and 20% energy saving with respect to the default configuration on MiniWeather and CloverLeaf.

ACKNOWLEDGMENTS

This research has been funded by the European High-Performance Computing Joint Undertaking (JU) under grant agreement No.

956137 (LIGATE project) and No. 956560 (REGALE project). We wish to thank the SC reviewers and artifact evaluators for their extremely insightful and helpful suggestions that have significantly improved the quality of this paper and its artifact.

REFERENCES

[1] Aksel Alpay and Vincent Heuveline. 2020. SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL. In *IWOCL '20: International Workshop on OpenCL*. 8:1. https://doi.org/10.1145/3388333.3388658

[2] AMD. 2023. ROCm System Management Interface. https://github.com/RadeonOpenCompute/rocm_smi_lib

[3] Eishi Arima, Minjoon Kang, Issa Saba, Josef Weidendorfer, Carsten Trinitis, and Martin Schulz. 2022. Optimizing Hardware Resource Partitioning and Job Allocations on Modern GPUs under Power Caps. In *Workshop Proceedings of the 51st International Conference on Parallel Processing, ICPP Workshops 2022, Bordeaux, France, 29 August 2022 - 1 September 2022*. ACM, 9:1–9:10.

[4] Wenlei Bao, Changwan Hong, Sudheer Chunduri, Sriram Krishnamoorthy, Louis-Noël Pouchet, Fabrice Rastello, and P. Sadayappan. 2016. Static and Dynamic Frequency Scaling on Multicore CPUs. *ACM Trans. Archit. Code Optim.* 13, 4 (2016), 51:1–51:26.

[5] Martin Burtscher, Ivan Zecena, and Ziliang Zong. 2014. Measuring GPU Power with the K20 Built-in Sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs* (Salt Lake City, UT, USA) *(GPGPU-7)*. Association for Computing Machinery, New York, NY, USA, 28–36.

[6] CINECA. 2023. The Marconi100 Supercomputer. https://www.hpc.cineca.it/hardware/marconi100

[7] Mark Endrei, Chao Jin, Minh Ngoc Dinh, David Abramson, Heidi Poxon, Luiz DeRose, and Bronis R. de Supinski. 2018. Energy efficiency modeling of parallel applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*. IEEE / ACM, 17:1–17:13.

[8] Kaijie Fan, Biagio Cosenza, and Ben H. H. Juurlink. 2019. Predictable GPUs Frequency Scaling for Energy and Performance. In *Proceedings of the 48th International Conference on Parallel Processing, ICPP, Kyoto, Japan, August 05-08*. 52:1–52:10.

[9] Yiannis Georgiou, Thomas Cadeau, David Glesser, Danny Auble, Morris Jette, and Matthieu Hautreux. 2014. Energy Accounting and Control with SLURM Resource and Job Management System. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8314)*, Mainak Chatterjee, Jiannong Cao, Kishore Kothapalli, and Sergio Rajsbaum (Eds.). Springer, 96–118. https://doi.org/10.1007/978-3-642-45249-9_7

[10] Neha Gholkar, Frank Mueller, and Barry Rountree. 2016. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016*, Ayal Zaks, Bilha Mendelson, Lawrence Rauchwerger, and Wen-mei W. Hwu (Eds.). ACM, 179–191.

[11] David Glesser, Yiannis Georgiou, Matthieu Hautreux, and Denis Trystram. 2014. *Introducing Power-capping in Slurm scheduling*. Technical Report. Lugano, Switzerland. https://hal.science/hal-01102285

[12] Khronos OpenCL Working Group. 2021. *OpenCL 3.0 API Specification*. Technical Report. Khronos Group.

[13] Khronos OpenCL Working Group. 2021. *OpenCL 3.0 C Language Specification*. Technical Report. Khronos Group.

[14] Khronos SYCL Working Group. 2021. *SYCL 2020 Specification, revision 3*. Technical Report. Khronos Group.

[15] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU Power Modelling for Multi-Domain Voltage-Frequency Scaling. In *24th IEEE International Symposium on High-Performance Computing Architecture, HPCA*.

[16] Meng Hao, Weizhe Zhang, Yiming Wang, Gangzhao Lu, Farui Wang, and Athanasios V. Vasilakos. 2021. Fine-Grained Powercap Allocation for Power-Constrained Systems Based on Multi-Objective Machine Learning. *IEEE Trans. Parallel Distributed Syst.* 32, 7 (2021), 1789–1801.

[17] JA Herdman, WP Gaudin, Simon McIntosh-Smith, Michael Boulton, David A Beckingsale, AC Mallinson, and Stephen A Jarvis. 2012. Accelerating hydrocodes with OpenACC, OpenCL and CUDA. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 465–471.

[18] Michael A. Heroux, Lois McInnes, Xiaoye Sherry Li, James Ahrens, Todd Munson, Kathryn Mohror, Terece Turton, Jeffrey Vetter, and Rajeev Thakur. 2022. *ECP Software Technology Capability Assessment Report*. Technical Report. https://doi.org/10.2172/1888898

[19] M. Horowitz, T. Indermaur, and R. Gonzalez. 1994. Low-power digital design. In *Proceedings of 1994 IEEE Symposium on Low Power Electronics*. 8–11.

[20] Intel. 2014. RAPL Running Average Power Limit Power Meter. https://01.org/blogs/2014/running-average-power-limit-âĂŞ-rapl

[21] Intel. 2022. Level Zero Specification documentation. https://spec.oneapi.io/level-zero/latest/index.html

[22] Intel. 2022. oneAPI Data Parallel C++ compiler. https://github.com/intel/llvm/releases/tag/2022-09 Online; accessed 6 Apr 2023.

[23] Shailendra Jain, Surhud Khare, Satish Yada, V. Ambili, Praveen Salihundam, Shiva Ramani, Sriram Muthukumar, M. Srinivasan, Arun Kumar, Shasi Kumar, Rajaraman Ramanarayanan, Vasantha Erraguntla, Jason Howard, Sriram R. Vangal, Saurabh Dighe, Gregory Ruhl, Paolo A. Aseron, Howard Wilson, Nitin Borkar, Vivek De, and Shekhar Borkar. 2012. A 280mV-to-1.2V wide-operating-range IA-32 processor in 32nm CMOS. In *IEEE International Solid-State Circuits Conference, ISSCC*. 66–68.

[24] Morris Jette and Danny Auble. 2012. SLURM Integration with IBM Parallel Environment. SLURM User Group Meeting.

[25] Hamidreza Khaleghzadeh, Muhammad Fahad, Arsalan Shahid, Ravi Reddy Manumachu, and Alexey Lastovetsky. 2021. Bi-Objective Optimization of Data-Parallel Applications on Heterogeneous HPC Platforms for Performance and Energy Through Workload Distribution. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2021), 543–560. https://doi.org/10.1109/TPDS.2020.3027338

[26] Karlo Kraljic, Daniel Kerger, and Martin Schulz. 2022. Energy Efficient Frequency Scaling on GPUs in Heterogeneous HPC Systems. In *Architecture of Computing Systems - 35th International Conference, ARCS 2022, Heilbronn, Germany, September 13-15, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13642)*. Springer, 3–16.

[27] Alexey Lastovetsky and Ravi Reddy Manumachu. 2017. New Model-Based Methods and Algorithms for Performance and Energy Optimization of Data Parallel Applications on Homogeneous Multicore Clusters. *IEEE Transactions on Parallel and Distributed Systems* 28, 4 (2017), 1119–1133.

[28] A Mallinson, D Beckingsale, W Gaudin, J Herdman, and S Jarvis. 2013. Towards portable performance for explicit hydrodynamics codes. In *The International Workshop on OpenCL (IWOCL)*, Vol. 2013.

[29] Ravi Reddy Manumachu and Alexey L. Lastovetsky. 2018. Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy. *IEEE Trans. Computers* 67, 2 (2018), 160–177.

[30] Matthew Norman, Isaac Lyngaas, Abhishek Bagusetty, and Mark Berrill. 2022. Portable C++ Code that can Look and Feel Like Fortran Code with Yet Another Kernel Launcher (YAKL). *International Journal of Parallel Programming* (2022), 1–22.

[31] Matthew R Norman and USDOE. 2020. miniWeather. https://doi.org/10.11578/dc.20201001.88

[32] NVIDIA. 2023. NVIDIA NVML API Reference Guide. https://docs.nvidia.com/deploy/nvml-api/index.html

[33] George Papadimitriou, Athanasios Chatzidimitriou, and Dimitris Gizopoulos. 2019. Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs. In *HPCA*. IEEE, 133–146.

[34] Srinivasan Ramesh, Swann Perarnau, Sridutt Bhalachandra, Allen D. Malony, and Peter H. Beckman. 2019. Understanding the Impact of Dynamic Power Capping on Application Progress. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. IEEE, 793–804.

[35] Haris Ribic and Yu David Liu. 2016. AEQUITAS: Coordinated Energy Management Across Parallel Applications. In *Proceedings of the 2016 International Conference on Supercomputing, ICS 2016, Istanbul, Turkey, June 1-3, 2016*, Ozcan Ozturk, Kemal Ebcioglu, Mahmut T. Kandemir, and Onur Mutlu (Eds.). ACM, 4:1–4:12.

[36] Issa Saba, Eishi Arima, Dai Liu, and Martin Schulz. 2022. Orchestrated Co-scheduling, Resource Partitioning, and Power Capping on CPU-GPU Heterogeneous Systems via Machine Learning. In *Architecture of Computing Systems - 35th International Conference, ARCS 2022, Heilbronn, Germany, September 13-15, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13642)*, Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck (Eds.). Springer, 51–67.

[37] Philip Salzmann, Fabian Knorr, Peter Thoman, Philipp Gschwandtner, Biagio Cosenza, and Thomas Fahringer. 2023. An Asynchronous Dataflow-Driven Execution Model For Distributed Accelerator Computing. In *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023, Bangalore, India, May 1-4, 2023*. IEEE, 82–93.

[38] Mohammed Sourouri, Espen Birger Raknes, Nico Reissmann, Johannes Langguth, Daniel Hackenberg, Robert Schöne, and Per Gunnar Kjeldsberg. 2017. Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017*. ACM, 41.

[39] Peter Thoman, Philip Salzmann, Biagio Cosenza, and Thomas Fahringer. 2019. Celerity: High-Level C++ for Accelerator Clusters. In *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, G"ottingen, Germany, August 26-30, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11725)*. Springer, 291–303.

[40] David Wallace. 2014. SLURM on Cray Systems. Slurm Birds of a Feather, International Conference for High Performance Computing, Networking, Storage, and Analysis, SC.

[41] Qiang Wang and Xiaowen Chu. 2020. GPGPU Performance Estimation With Core and Memory Frequency Scaling. *IEEE Trans. Parallel Distributed Syst.* 31, 12 (2020), 2865–2881.

[42] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *19th International Conference on Parallel Architectures and Compilation Techniques, PACT 2010, Vienna, Austria, September 11-15, 2010*, Valentina Salapura, Michael Gschwind, and Jens Knoop (Eds.). ACM, 29–40.

[43] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers (Lecture Notes in Computer Science, Vol. 2862)*, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer, 44–60. https://doi.org/10.1007/10968987_3

[44] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Vol. 2862. Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60. Series Title: Lecture Notes in Computer Science.

[45] Huazhe Zhang and Henry Hoffmann. 2018. Performance & Energy Tradeoffs for Dependent Distributed Applications Under System-wide Power Caps. In *Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, Eugene, OR, USA, August 13-16, 2018*. ACM, 67:1–67:11.

[46] Huazhe Zhang and Henry Hoffmann. 2019. *PoDD*: power-capping dependent distributed applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*. ACM, 28:1–28:23.

[47] Marcela Zuluaga, Andreas Krause, and Markus Püschel. 2016. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. *Journal of Machine Learning Research* 17 (2016), 104:1–104:32.