Energy-efficient power cap configurations through Pareto Front Analysis and Machine Learning Categorization

Alberto Cabrera
University of La Laguna
Francisco Almeida
University of La Laguna
Dagoberto Castellanos-Nieves
University of La Laguna
Ariel Oleksiak
Poznan Supercomputing and Networking Center
Vicente Blanco (vblanco@ull.es)
University of La Laguna

Research Article

Keywords: Energy Aware Computing, Power Capping, Machine Learning, Clustering algorithms

Posted Date: February 13th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2066435/v2

Additional Declarations: No competing interests reported.

Energy efficient power cap configurations through Pareto Front Analysis and Machine Learning Categorization

Alberto Cabrera[1†], Francisco Almeida[1†], Dagoberto Castellanos–Nieves[1†], Ariel Oleksiak[1†] and Vicente Blanco[1*]

[1*]Computer Science and Systems Department, Universidad de La Laguna (ULL), San Francisco de Paula s/n, La Laguna, 38270, Spain.

[2]Poznan Supercomputing and Netwoking Center, Institute of Bioorganic Chemistry PAS, ul. Jana Pawła II 10, Poznan, 61-139, Poland.

*Corresponding author(s). E-mail(s): vblanco@ull.es;
Contributing authors: Alberto.Cabrera@ull.es; fameida@ull.es; dcasterll@ull.es; ariel@man.poznan.pl;
†These authors contributed equally to this work.

Abstract

The growing demand for more computational resources has increased the overall energy consumption in computer systems. To support the increasing requirements, power and energy consumption have to be considered as a constraint to execute software. Modern architectures provide tools to manage directly the power constraints of a system. The Intel Power Cap is a relatively recent tool developed to offer fine control of power usage to users at a central processing unit (CPU) level. We propose a methodology to analyze the performance and the energy efficiency trade–offs using this power cap technology for a given algorithm. We extract a Pareto front for the multi–objective performance and energy problem to represent multiple feasible configurations for both objectives. We perform an extensive experimentation to categorize the

different algorithms to reduce the total amount of optimal power cap configurations. We propose the use of Machine Learning (ML) clustering techniques to categorize any algorithm in the target architecture. The use of Machine Learning allows to automate the procedure and simplify the effort required to solve the optimization problem. We present a practical case where we categorize the kernels using the ML techniques, with the option to include a new algorithm into an already existing categorization.

Keywords: Energy Aware Computing, Power Capping, Machine Learning, Clustering algorithms

1 Introduction

Energy and power limitations are constraints that affect multiple fields in computer science. The growth in internet services, artificial intelligence, mobile devices, Internet of Things devices and the computational demands in the High Performance Community provide multiple complex environments where huge amounts of energy are consumed, which are currently estimated at 200 terawatt hours each year [1]. Infrastructures, cooling systems, intercommunication networks, virtualization clusters, long term storage, computational hardware, software applications and schedulers are examples of all the gears that need to fit and work together towards improving the energy efficiency of information and communication technologies [2].

For the computational hardware, more efficient architectures have been introduced over the last years. Heterogeneous systems have become a great candidate to maximize energy efficiency, and there are efforts towards shifting to low power processors using highly efficient co–processors in System–on–chips, or the usage of GPUs to solve highly parallel applications as is the case of Nvidia general purpose graphic processing units (GPGPU), and the now discontinued Knights Landing Intel architecture.

Power limiters are introduced by manufacturers as a power management tool to limit the power draw of computational elements. These tools provide mechanisms to easily define power constraints and better control energy usage in a computational system. As the energetic environment can be configured at runtime, programmers can add a new layer of optimization in applications development. Two examples of power limiters have been provided by Intel and Nvidia. Intel incorporated the Power Cap technology to assign multiple advanced power constraints to their CPUs using different parameters. Nvidia on the other hand, incorporated in their drivers an option to assign a hard power limit in their high–end cards.

Using these tools to define a power constraint requires to analyze both the performance and energetic behavior of the computational hardware to avoid detrimental configurations. This new opportunity for energetic tuning in applications incorporates a new layer of complexity, specially in parallel environments where algorithms may have different sections of code executing simultaneously.

Increasing the knowledge of the energetic behavior in these architectures is key to maximize the usage of resources. Performance analysis is a discipline focused on enhancing the effectiveness of a given procedure by observing its behavior on a target. The knowledge extracted provides insight to the complexity of computational environments. As multiple objectives can be studied, this discipline can be applied to different objectives, such as time to solution or energy consumption. By combining the analysis with power capping tools, the improved understanding of the system allows to find better configurations to affect the resource usage in a given architecture.

However, as systems increase in size and complexity, statistical techniques get an increased significance. Manual analysis of the problem becomes a costly

task that requires deep insight of the architecture and the software we intend to tune up, the complexity and amount of work carried out by the experts increases, and the volume of data to manage entail additional difficulties. Theoretical models are hard to obtain since there is a high trade–off between model complexity and accuracy. Also, newer technologies and algorithms require adaptation of existing models.

Machine Learning (ML) techniques, offer an automated and statistical approach to extract information from big volumes of data, and offer deep insight in order to define the behavior of complex systems [3]. Moreover, applying unsupervised ML techniques to the attained data allows to extract hidden features or simplify the volume of data [4].

In [5], we performed a prelimary analysis of the capability to find a Pareto front configuration for power consumption and performance. Heatmaps are used as an initial approximation to seek feasible configurations using the Intel power Cap. In this work, we present a generalized methodology to analyze the performance and energy consumption in a given system when executing applications under a power limit. This analysis allows to extract a set of power cap configurations to adapt the energy efficiency of a given system. The obtained constraints allow, at runtime, to define policies aimed to fulfill a power or performance constraint. To illustrate and justify our proposal, we perform an extensive experimentation and discuss the different feasible power configurations that optimize our target architecture. Additionally, to reduce the efforts of this methodology, we introduce the usage of ML clustering techniques to automate the procedure.

Following are the main contributions of our paper:

- We present a general framework to analyze the effect of power cap technologies in a given architecture, parameterized by a configuration with a

certain number of k parameters, and detailed evaluations are given for the minimal case of $k = 2$, performance and energy efficiency. We illustrate how a Pareto front of solutions can be attained for these objectives, and how increasing the size of the problem does not affect the power cap parameters within the Pareto front. The Pareto front also provides the ability to predict the trade–offs between execution time and energy when applying a power cap technology. Additionally, we formalize our methodology to illustrate the independence between our proposal and the specific technology chosen to validate this research.

- We perform a large number of experiments using different power cap settings, which translates into a rich data set of time and energy measurements in both serial and parallel environments, using a variety of kernel applications from the NAS Parallel Benchmarks (NPB), derived from computational fluid dynamics (CFD) applications.
- We apply this methodology to provide an in–depth experimental validation of the energetic behavior of the NPB. We analyze the effect of power capping technologies over multiple NPB kernels in a given architecture. Due to the previous analysis, we are able to reduce the amount of experimentation. We extract deep insight from the feasible power capping configurations for each algorithm, which we use to categorize them. We simplify the optimization problem using this categories and have an optimal power configuration per category, instead of optimizing each algorithm separately.
- We propose the use of ML clustering techniques to provide an automated procedure to categorize different algorithms with the insight of the presented methodology. We present the application of an extensive amount of clustering techniques using multiple configurations, and compare it to our Pareto front based analysis.

- Finally, we illustrate the procedure with a specific ML clustering technique to perform the categorization of the NPB. Using ML, we are able to classify the different algorithms automatically. ML is also used to categorize the cost of a new algorithm based on the training of a selected number of clusters, thus simplifying the optimization problem.

We analyzed the effect of power capping technologies for a specific architecture, and obtained the Pareto front of different NPB kernels, to then categorize them. Using this categorization, we are able to reduce the total number of power configurations in our target system, and are able to reconfigure the hardware power limits taking into account different power and performance constraints. With the ML approach based in clustering techniques, we prove that the automation of this work is feasible and discuss how it could be applied in a real scenario.

The rest of the paper is structured as follows. Section 2 covers the background and related work. Section 3 describes our methodology to analyze the energy usage and the performance of our target application under a power cap. In Section 4, we present the NPB use–case, and how we apply our methodology to perform the in–depth analysis of a target architecture. Section 5 presents a detailed justification of the use of ML clustering techniques and its application to our use–case. Finally, Section 6 summarizes our conclusions and future work.

2 Related Work

Energy efficiency is one of the multiple challenges in multiple environments in computer science. In the Internet of Things the growth of devices has an unprecedented rate and energy consumption is critical to support its expansion [6]. On the other hand, in parallel computing and High Performance

Computing environments, the constant demand of computational resources and the increasing costs to run highly parallel systems has shifted the efforts from optimizing only performance to a multi–objective approach where energy efficiency is also critical [7, 8]. In the field of optimization, energy constraints have been included in meta-heuristics as part of multi–objective complex problems, such as the Flexible job shop scheduling problem [9], or energy–efficient strategies for tracking a target in a field of obstacles [10].

Dynamic voltage and frequency scaling (DVFS) is a common approach to improve the energy efficiency of software applications. In HPC, methodologies and tools are developed using models with the help of DVFS, reducing the overall energy consumption while also improving performance for scientific applications [11–13]. *E–AMOM* [14] is an example of a framework developed upon these methodologies, using models, DVFS, and dynamic concurrency throttling (DCT) to reach these goals.

However, the increasing importance of energy efficiency and power constraints have led to the development of more specialized tools. The interest in these tools is increasing as their usage relates directly to power limitations instead of affecting other parameters, such as frequency. These tools have been proven valid as a possible replacement for the DVFS techniques present in the literature [15, 16]. Using DVFS along power capping technologies through the RAPL interface, *Conductor* [17] was developed as a middleware for post Sandy Bridge Intel architectures. It is capable of improving the performance of applications that are under a power budget allocating and shifting the power of the application.

In computer science, ML has been gaining traction due to the effectiveness of deep neural networks, and the scientific community is applying these techniques using multiple approaches to improve performance [18], to

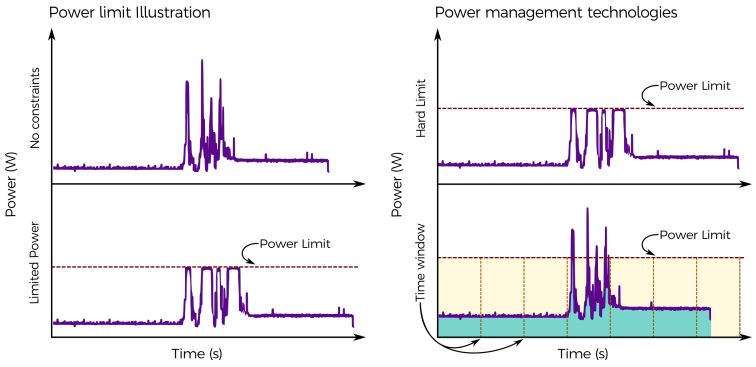
solve complex problems [18, 19], to automatically tune experimentation [19], to design experiments [20] or to evaluate the goodness of energy profiles compared to measurements obtained from external power meters [21]. As it has proven to work in countless cases, we are applying ML techniques to manage decision–making automatically, focused in the energy efficiency and performance trade–offs of a given application.

In our work, we present a methodology to analyze the usage of power cap technologies in a given system. In the HPC community, analytical and statistical models are often developed to find optimal configurations for performance and energy efficiency. The use of a Pareto front multi–objective approach has been used to represent the workload redistribution in manycore architectures [22]. Our work follows the experimental approach to analyze our target architecture, but focuses in the effect of power capping technologies instead of workload redistribution. Regression based methods have also been used to explore Pareto efficient configurations to consider the trade–off between time and energy efficiency [23], where DVFS and parallelization is analyzed to formulate a model for a given application. Likely, we share the objective of finding the optimal trade–off zone and perform an in–depth analysis of our target applications. However, our approach is applied to multiple algorithms, and our main contribution yields in the categorization of multiple Pareto fronts to reduce the amount of optimal power configurations in a given system.

There are also multiple methodologies and algorithms that analyze and solve power–performance trade–offs in the literature [9, 24–27], as modern processors offer multiple power and performance optimization options by changing core and uncore frequencies individually, affecting the overall voltage and power. NORNIR [24] focus on a single application and, with the help of linear

regression and monitorization, configures an application at runtime to satisfy user needs. Coutinho et al. [25] also introduced a methodology to find Pareto–optimal configurations in Heterogeneous Multi–processing systems. They combine an analytical model for a given application and an analytical power model for a given hardware in a unique model that predicts the energy consumption of a given application. OPAD [26] is a methodology to solve power budgeting problem using Pareto optimality. In this case, a parallel dynamic programming network is proposed to calculate the optimal frequency configuration of the cores in a manycore system. PGCapping [27] is also proposed to optimize the performance of a system under a power cap, in this case, through the use of power gating to shut down idling cores instead of using only DVFS.

Our proposals are applied at a higher level, using a technology specifically designed for power–capping, and the decision making is performed using existing ML techniques, instead of defining new methods, algorithms or analytical models, simplifying the expertise required by the user to implement our solution.



(a) Power cap application (b) Power policy comparison

Fig. 1: Illustration of different power management tools

3 Power Cap Performance Analysis

We propose a methodology to use power management tools and extract optimal configurations for a set of power constraints. To justify our approach, a comparison of different power capping technologies is presented in the following Subsection 3.1. In Subsection 3.2, we introduce a step–by–step methodology to apply power management technologies to a given architecture executing a given application, using the Intel power cap.

We offer deep insight of the data interpretation and the different power configurations extracted from the study. Based upon the knowledge obtained through this experimentation methodology, we are able to classify and categorize algorithms that, despite being of different nature, could share energetic behaviors between them, thus minimizing the number of optimal power management configurations.

3.1 Power management technologies

Power management technologies are an alternative proposal for controlling power usage in current hardware architectures. While not available for every architecture, exploiting the implemented manufacturer solutions provide an easy solution to compute with a defined power constraint. In the case of ARM processors, power management relies in their on–chip management and the use of dynamic voltage and frequency scaling (DVFS) across the chip, unless individual designs expose control to the user. State-of-the-art processors let the user change the frequency of individual cores as well as a selection of C-states from OS. AMD included in their older architectures a Thermal Design Power (TDP) management tool, TDP Power Cap. However in their latest architecture, known as Zen, they provide tools to control the C–States and to set the amount of active cores in their CPU. On the other hand, Nvidia

and Intel provide their own approach to give users control over their power management.

Nvidia allows to set a power limit through their drivers to different GPUs. However, their hardware requires a minimum power, which results in less control to the power consumption management. High–end and modern cards offer a better range to exert control over this energetic limitation. Nvidia power limit implementation sets a strict constraint as a hard limit that the hardware cannot surpass. Fig. 1 illustrates the effects of hard power management enforcement using a naive example. Fig. 1a depicts the power profile of a generic execution without any power management tool (at the top) and how setting a strict power cap affects the overall execution. The bottom of the Fig. shows a hypothetical power limit applied and, as a result, the power intensive sections, and the overall application, take more time to complete.

The effect of these power limits in the overall energy consumption will always depend on the nature of the application. While some applications will lose performance and spend more energy in the process, other applications will improve their overall work per watt, improving the energy efficiency of the execution.

Finally, Intel offers a more advanced power management of their chips through the Running Average Power Limit (RAPL). RAPL provides mechanisms to set average power limits on different zones of their processors. These zones, known as Power Planes (PP), allow to set different power constraints to the cores, the uncore, the package itself and/or the DRAM. Sky Lake architectures introduced a new PP, the Power of System (PSYS), to monitor and set constraints at the platform level. RAPL also provides the option to set short and long term windows to enforce the power limits. Short term averages can be used to adapt power constraints to workload bursts, while the longer

term limit would ensure to meet our power constraints. Both time windows and power limits are limited to the manufacturer specifications and have to be adapted to the specific processor model we are using.

Fig. 1b illustrates a different power profile using the naive application example. It illustrates the Intel approximation, where the power limit is not enforced, but guaranteed over a period of time, known as *time window*. As shown in the chart, the power profile of the application could exceed the power limit momentarily, as long as the average power at the end of the time window is below or equal to the specified constraint. While it is not shown in the chart, a short term and a long term window can be applied to better control the power profile of a given application.

3.2 Multi–objective optimization using power limits

We can apply power limiters to address different limitations sets for our infrastructure, addressing multiple objectives simultaneously. Performance, power draw, energy efficiency, quality of service or temperature are examples of different metrics we could consider for our decision making. These multiple objectives may or may not enter in conflict with each other. Temperature and power draw will have high correlation, while performance and quality of service will be similarly tied. However, energy efficiency and performance may have opposite effects for a given power cap configuration.

In other words, for any power configuration, a number k of different objectives can be considered for optimization. Each power configuration has a direct effect on these k objectives and directly affects how power management changes the hardware behavior. In formal terms, we can formulate a multi–objective optimization problem where we consider:

- k, the number of objective functions to address.

Power Limit (W)	Time Window (s) 0.2	...	0.8	1.0
3.0	(6.81, 20.39)	...	(7.25, 21.72)	(7.16, 21.44)
3.5	(4.99, 17.44)	...	(5.18, 18.11)	(5.17, 18.07)
4.0	(4.10, 16.43)	...	(4.18, 16.75)	(4.11, 16.48)
⋮	⋮	⋱	⋮	⋮
8.0	(2.31, 18.54)	...	(2.33, 18.70)	(2.32, 18.67)

Table 1: NPB BT.W Kernel (execution time (s), energy consumption (J)) using Intel power cap.

- f_i, optimization function for the i–th objective.
- p, power cap configuration, comprised of different parameters specific to the appropriate power management tool.

Where we have to find

$$min(f_1(p), f_2(p), \ldots, f_k(p)),\ max(f_1(p), f_2(p), \ldots, f_k(p))$$

or a mix of both cases.

An example for $k = 2$ conflicting objectives is illustrated in Table 1, where execution time and energy are to be minimized for the Block Tri–diagonal solver (BT) kernel from the NAS parallel benchmarks (NPB). In Intel architectures, as explained before, every power configuration p is obtained by applying an average power limit, a time window and a zone. A variety of power limits and time windows have been chosen, while the zone is fixed to the power of system (PSYS). An static PSYS, while not optimal for every algorithm, is enough for our homogeneous codes.

The results of every execution of this kernel using the size *class W*, a 3-dimensional matrix of size (18, 18, 18), generate a set of metrics $(f_1, f_2) =$ (*execution time*, *energy consumption*) for each configuration p, which are obtained through different power limits and time windows.

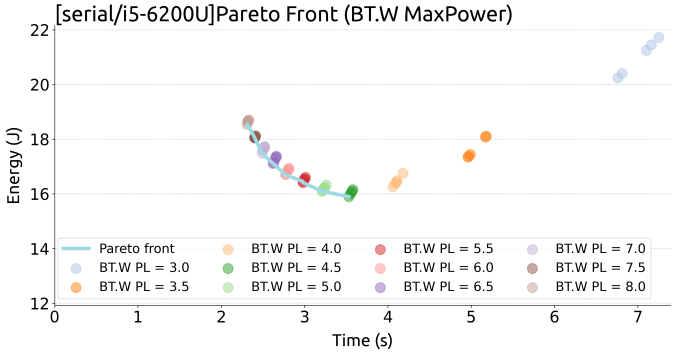


Fig. 2: Pareto front for k = 2 objectives. Energy and execution Time. PL = Power Limit in Watts

We also assigned a different color to each power limit to facilitate data analysis. We illustrate the contents of Table 1 in Fig. 2 using these colors. Reflected by both axis of the plot, we have chosen to optimize execution time (x axis) and energy consumption (y axis). In the figure, we also observe the multi–objective nature of these objective functions and multiple conflicting solutions. As it is unfeasible to minimize both objectives at the same time, a set of Pareto optimal solutions is necessary. A Pareto front is obtained with all the non–dominated tuples, represented using a segment in the Fig. The dominated solutions, are discarded as there is, at least, one tuple in the Pareto set that improves both objectives functions. In Fig. 2, we also observe how the time window parameter for the Intel architecture has much less impact compared to the power limit itself. To simplify data, we could average the metrics for each power limit, which also could be considered to reduce the total experimentation needed. Still, despite the apparent uniformity of this Pareto front, we are not able to perform a binary search approach to sample as we cannot guarantee this behavior for every algorithm executed in modern hardware. In Section 4, we also present a parallel version using 20 cores, where some kernels are not as uniform as the illustrated example.

To find the different configurations for the target architectures, we defined a simple benchmarking process to determine the behavior of any application. Systems are reconfigurable using the power management tools and its outcome varies for different hardware and software. We learn the effects of applying power limits to our heterogeneous system by executing small and simple instances of software.

However, experimentation is relatively simple, allowing to extend the software pool or the available hardware without complex efforts. The final objective is to obtain a Pareto front for multiple conflicting objectives to optimize algorithms. After the experimentation, a variety of power configurations p will be available for each algorithm in the Pareto front, so that we can optimize our objective functions.

We can formally define our experimentation methodology with the following parameters:

- H, set of architectures in our experimental environment.
- S, set of software applications to evaluate under power constraints.
- $P_{H,S}$, set of non–dominated power cap configurations, a subset of the Pareto front of optimal solutions.
- $h \in H$, a computational node of our environment.
- $s \in S$, a specific application or computational kernel.
- $P_{h,exp}$, experimental set of power cap configurations to test system h.
- $p \in P_{h,exp}$, power cap configuration. Extending the previous formalization, Intel architectures require to define a 3–tuple of average power, a time window and a power plane to apply the configuration, i.e. $p = (avgpower, timewindow, zone)$. While multiple limits can be established simultaneously, we will limit ourselves to a single power limit at a given

time. For Nvidia GPUs a single maximum power value is needed, i.e. $p = (maxpower)$.

- $M_{h,s}$, hardware metrics gathered during the experimentation. Every element $m \in M_{h,s}$ is a k–tuple in the form $m = (p, f_1(p), f_2(p), \cdots, f_k(p))$, for a given $p \in P_{h,exp}$. In our case for $k = 2$, we would have a 3–tuple with the experimental metrics: p, execution time and energy consumption, though we could consider other objectives.
- $P_{h,s} \subseteq P_{H,S}$, Pareto front set of power configurations that can be applied to a given s and h to optimize our multiple objectives. Extracting this set from the acquired knowledge will depend on how we intend to optimize our objectives. By definition, $P_{h,s} \subseteq P_{h,exp}$.
- $p^t_{h,s} \in P_{h,s}$, power configuration that achieves the best performance in a given s and h.
- $p^e_{h,s} \in P_{h,s}$, power configuration that achieves the most efficient energy consumption in a given s and h, i.e. minimizes energy consumption of an specific task.

Algorithm 1 summarizes the experimental steps followed to analyze how power capping affects a given software and hardware. It is a generalized benchmark that can be applied to different applications in various platforms. The process can be summarized as performing multiple executions for various power cap configurations in target architecture. Once the experimentation is over, every pair h, s of hardware and software has an associated $P_{h,s}$ to address different objective functions depending on the user needs.

For the bi–objective case presented in Fig. 2, we can use the information contained in the Pareto front $P_{h,s}$ and manage the power to improve the energy efficiency of an execution. We will explain three different approaches to manage

Algorithm 1 Benchmark process

input: $H \rightarrow$ Hardware Pool, $S \rightarrow$ Software Pool
output: $P_{H,S} \rightarrow$ set of power settings

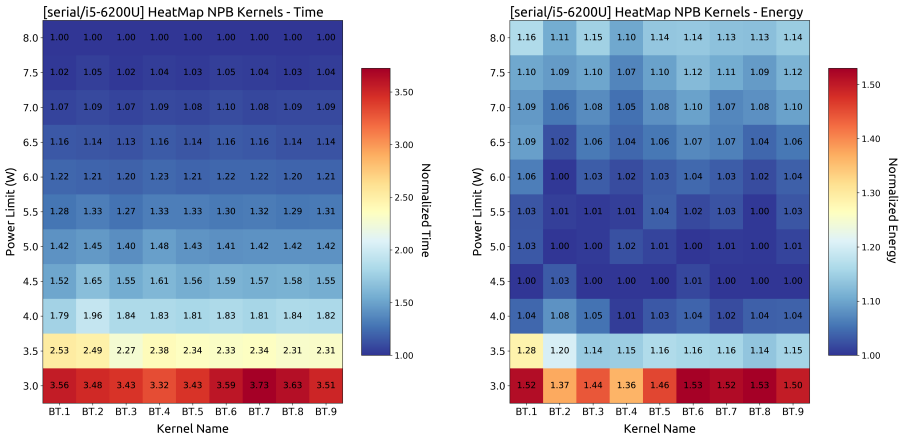
$P_{H,S} \leftarrow \varnothing$
for all $h \in H$ **do**
  $P_{h,exp} \leftarrow experimental\_interval(h)$
    ▷ $P_{h,exp}$: Power experimental parameters
  for all $s \in S$ **do**
    $M_{h,s} \leftarrow \varnothing$ ▷ Hardware Metrics
    $P_{h,s} \leftarrow \varnothing$ ▷ Optimal configurations for (h, s)
    for $p \in P_{h,exp}$ **do**
      $m \leftarrow measure\_execution(h, s, p)$
      $M_{h,s} \leftarrow M_{h,s} \bigcup \{m\}$
    end for
    $P_{h,s} \leftarrow get\_pareto\_front(M_{h,s})$
    $P_{H,S} \leftarrow P_{H,S} \bigcup P_{h,s}$
  end for
end for
return $P_{H,S}$

the power consumption of a system using the BT.W kernel as the illustrating case:

- A first approach would consider having k different configurations defined by the user, thus $|P_{h,s}| = k$, one configuration per optimization objective. In the presented experimentation for the BT, these values are setting the power limit at 8 or 4.5 Watts to minimize execution time or maximizing energy efficiency respectively.
- A second approach is to consider each possible value between 4.5 and 8 Watts, including all the different solutions to achieve better energy efficiencies with less impact in the execution time trade-off if necessary.
- A third approach is to consider a power constrained situation where we would configure power limits from 4.5 Watts to a maximum available power. As an example, in an environment where we only have 6 Watts available for our CPU, we would consider configurations ranging from 4.5 to 6 Watts.

Power Limit (W)	3.0	3.5	4.0	4.5	···	8.0
Time	3.02	2.18	1.77	1.53	···	1.00
Energy	1.31	1.10	1.03	1.00	···	1.16

Table 2: NPB BT.W Kernel normalized execution time and energy consumption averages



(a) Execution Time (b) Energy Efficiency

Fig. 3: NPB comparison for different size using BT custom classes.

In every case, the end result is a set of power configurations we can use to make decisions to satisfy different energetic and execution time constraints.

In a power constrained highly parallel or distributed environments, resource allocation problems can be formulated using any of the previous cases to use the available hardware partially, following the multi–objective criteria.

The Pareto front represents the optimal solutions for the Multi-objective problem for all the objective functions simultaneously. In order to perform sensitivity analysis and also trying get some predictive ability from the methodology, we introduce now an alternative representation model based on heat diagrams (see Fig. 3). This allows the analysis of the individual objectives

without losing the Multi-objective perspective. Following with our case of studio, the BT.W problem, Table 2 presents an alternative representation of the data shown in Table 1. Average values for the different power limits have been calculated given the low dependence observed from the time window. At the same time, data have been normalized to show the effects of the power configuration for each objective function. By doing so, we can compare data for different sizes or problems in an unified heatmap, as shown in Fig. 3a and 3b.

While the NPB presents multiple predefined classes, the problem size increment between these provided instances is very high, as it is thought for highly parallel systems. As an example, the A, B and C classes have around a 4 times size increase going from one class to the next, while classes D, E and F have a 16 times size increase between each problem. We defined custom classes, labeled in the X axis, from BT.1 to BT.9 to study a softer increment in the kernel sizes. The smallest size performs calculations over a 3-dimensional matrix of sizes (12, 12, 12), while the biggest problem size processes (32, 32, 32). For comparison, we already stated that BT.W is of size (24, 24, 24). In other words, the problems perform calculations over 1728, 32768 and 5832 elements respectively.

In Fig. 3a and 3b, we represent the execution time for different custom classes for the BT kernel using this alternative representation. The Y axis represents the maximum power allowed to the CPU, and each power has an average for several experiments using different time windows. As each column of the heatmap is normalized, 1.00 represents the best value measured for each problem size. In both figures, dark blue represents the best values, while red illustrates the opposite.

For the execution time objective, there is no difference between the selected problem sizes, and the better performance is achieved by removing the power

limit. For the energy objective function, we observe again negligible differences between the multiple problem sizes. The optimal power limit in Figures 3a and 3b are 8 W and 4.5 W respectively, the extreme values in the Pareto front set.

In this bi–objective case, optimal values can be extracted from the heat maps by observing the behavior of each single objective individually. By studying both heatmaps, we can compare the normalized data to understand the energy cost of the fastest configuration or the extra execution time needed in order to improve the energy efficiency. This data representation helps to perform a sensitivity analysis for different software and hardware configurations to accurately discard impracticable power values and slightly reduce the total number of executions required to study an architecture.

In this initial use–case, we could discard values over 8.0 W and under 3.0 W as the BT analysis has shown these limits are reasonable in our environment. However, this is only applicable in this specific architecture for single–core executions, and daemons or services installed in our system could affect our power consumption. Still, this delimits the experimentation for future software applications and significantly reduces the effort required to incorporate the knowledge of these applications.

4 Pareto Front based Analysis

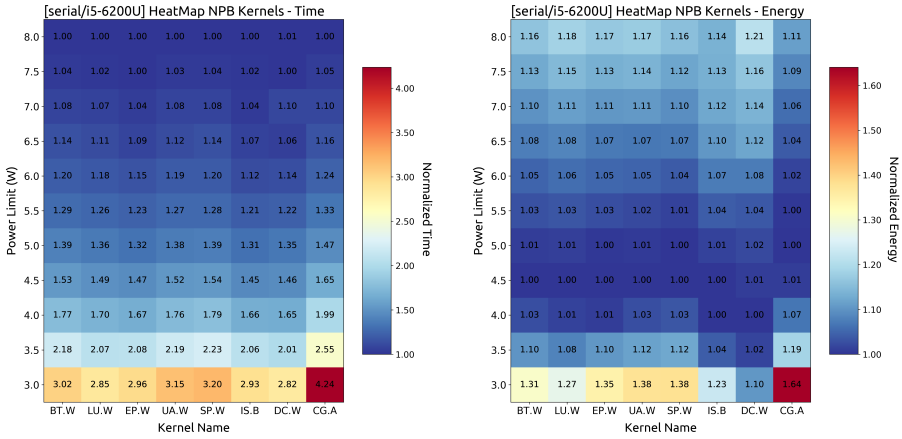
In order to illustrate the proposed methodology, this section presents an in–depth analysis of an Intel i5-6200U CPU, a Skylake processor with power cap capabilities. It has 2 cores at 2.30 GHz, with a thermal design power of 15 Watts. The node has an Ubuntu 18.04 LTS, with kernel version 4.18 and GCC 7.4. We also present a second analysis using a Intel Xeon Gold 6230N to illustrate a multicore environment. The Intel Xeon Gold 6230N has 20 cores at 2.30GHz, with Debian 10, kernel version 5.4.0-0.bpo.2-amd64 and GCC 8.3.0.

Energy measurements were gathered using the EML [28], a driver based energy measurement library. In our experimentation, we used the appropriate drivers to extract total energy metrics from the RAPL interface. RAPL has been criticized of being not too accurate in measuring the energy consumption by applications [29]. However, to the effects of validating our methodology there is no loss of generality when using RAPL since, basically, the methodology obtains a Pareto front or apply ML clustering techniques from a dataset to model the behavior of a group of algorithms. The Pareto front and the obtainded clusters will model the dataset be it accurate or not, so the use of the RAPL tool for measurements does not affect the validity of the analysis. The methodology will not change even if the dataset changes.

This use–case illustrates our experimental methodology by testing version 3.3.1 of the NAS Parallel Benchmarks (NPB) [30]. NPB is a collection of kernels derived from Computational Fluid Dynamics (CFD) applications. Using the RAPL interface, we set multiple time windows for each specified power average to the CPU. We have approached the problem having simplicity in mind so, we limited the power consumption of the whole CPU, using the PSYS zone.

As stated in the previous section, to compare different algorithms or problem sizes appropriately, normalization of the data is required. Each kernel from the NPB is referred as $AA.B$, with AA as the standard abbreviation of the kernel name and B as the size, or class, of the problem. In the previous section we presented an example with the BT.W, and the custom BT.1 through BT.9 executions. Experimental data is illustrated using Table 1 and Fig. 2, 3a, and 3b.

We presented multiple power configurations using $p = (powerlimit,\ timewindow,\ PSYS)$ and observed how the time window had



(a) Execution Time (b) Energy Efficiency

Fig. 4: Intel power cap application to serial NPB Benchmarks. Normalized values, the closest the cell value is to 1, the better.

less impact for this kernel. This behavior appears in each kernel individually and its repeated as the problem size in these serial version executions.

We will condense the data by obtaining an averaged value of three executions for each different power limit, to facilitate the study of the behaviors for each kernel. Our first approximation shows that proper management of the Time Window improves the overall efficiency and execution time of a target application. However, an in–depth analysis is required to draw proper conclusions on how to adjust the parameter to our needs. In this work we will only consider the power limit, since it affects the energy consumption of the application more directly.

Table 2 shows an example of the normalized and averaged data over all the windows for the kernel BT.W. Following the same data treatment to normalize values, Fig. 4 depicts the energy dataset for 8 NPB kernels in their serial version, similarly to Fig.3b. Again, the best energy efficiency is the normalized value 1. Since the time heatmap is analogous to Fig. 3a, time data will still

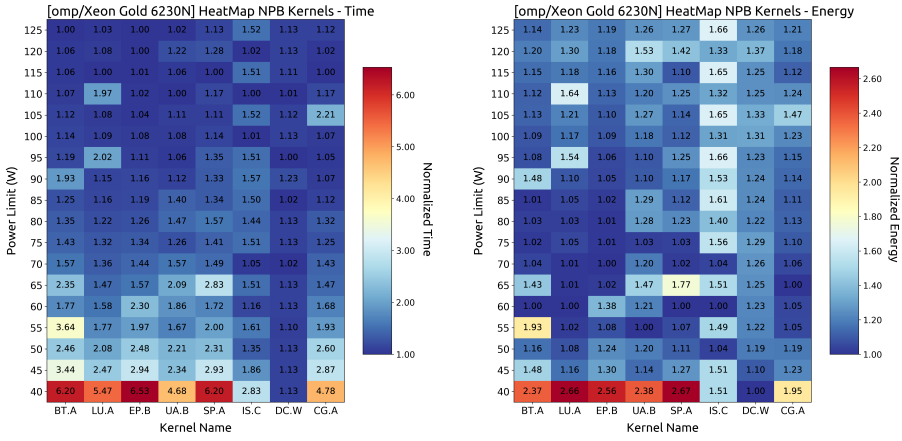
be discussed, but we will mainly focus on the energy efficiency of power cap configurations.

The target architecture allows a broader range for power capping, below 3 Watts and over 8 Watts. However, these executions never required more than 8 Watts, so it represents the absence of power capping (> 8 Watts). On the other side, metrics from the experimentation with a 3 Watts limit have shown an extreme decrement in both performance and efficiency.

We can apply the conclusions drawn from the BT study to all the studied kernels. This includes knowing the values for the Pareto front set, which are the power values in between the power limit for the best execution time (> 8 Watts), and the power limit that achieves the best energy efficiency. We will also denote how energy consumption and execution time is affected using power cap with tuples in the form of (*execution time*, *energy comsumed*). Both execution time and energy consumed will be the normalized values where 1.00 represents the best possible outcome. To exemplify, in the BT.W kernel presented in Fig. 4b, the best performance is achieved with a power limit > 8 Watts, $p^t_{h,s} > 8W \rightarrow (1.00,\ 1.16)$, which translates to increasing energy consumption by 16% , while the best energy efficiency is achieved at 4.5 Watts, $p^e_{h,s} = 4.5W \rightarrow (1.53,\ 1.00)$, an increased execution time of 53%.

Three different patterns can be observed in the data presented in Fig. 4b:

- The first case includes the DC.W and the IS.B. These kernels reach the best energy efficiency at 4 Watts, where DC $p^t_{h,s} > 8W \rightarrow (1.00, 1.21)$, $p^e_{h,s} = 4W \rightarrow (1.65, 1.00)$ and, IS $p^t_{h,s} > 8W \rightarrow (1.00, 1.14)$, $p^e_{h,s} = 4W \rightarrow (1.66, 1.00)$. Both algorithms are memory intensive, and while data does not indicate why, we suspect the similar power configuration is related to it.
- The CG.A reaches the best energy efficiency at 5 Watts, $p^t_{h,s} > 8W \rightarrow (1.00, 1.11)$ and $p^e_{h,s} = 5W \rightarrow (1.46, 1.00)$.



(a) Execution Time (b) Energy Efficiency

Fig. 5: Intel power cap application to parallel NPB Benchmarks. Normalized values, the closest the cell value is to 1, the better.

- The rest of the kernels reach the best energy efficiency at 4.5 Watts. These kernels have a mix of memory accesses and computation, and operate at a similar power limit. Despite the multiple differences between them, these algorithms barely require different power limits in their serial version.

Moreover, Fig. 4b indicates that multiple algorithms could have a similar energetic behavior. With this information, instead of having a set of power policies per algorithm, every decision that improves execution time or energy efficiency could be applied to a problem category.

The analogous study using the *OpenMP* version of the NPB kernels was also performed to analyze a different target CPU, where we observe a similar energetic and performance behavior. We analyzed a Xeon Gold 6230N CPU following the same experimentation methodology. The different hardware power consumption increases the range of feasible power limits to manage the energy efficiency for the NPB kernels. We also selected classes with bigger problem

instances, to properly adapt the problems the increased computational capacities. For DC, however, the selected class is still W, as performance is dependent on I/O.

Fig. 5 represents the *OpenMP* parallel execution using the same kernels presented in the previous section. Here, we execute using the twenty cores available in our Intel Xeon Gold 6230N, thus expanding our power limits from 30 W up to the TDP of 125 W.

For this hardware, the fastest configurations still require removing any power limitations, as expected from the serial executions. For the optimal energy efficiency, we find again different patterns in the data presented in Fig. 5a and 5b. The minimum in both heatmaps was incremented to 40 Watts, since the efficiency loss using less energy in most of the studied cases reduced the clarity of the chart. Still, every omitted value was strictly worse than the value at 40 W. Similar patterns are observed in the data presented in Fig. 5a and 5b, which are: DC.W; LU.A, EP.B, UA.B, and CG.A; BT.A and SP.A; and IS.C. We conclude that the addition of parallelism does not affect the methodology to analyze the behavior of the target applications using power limits.

A classification process is applicable for both serial and parallel versions. We can determine a small number of power and performance policies for a given system, and adapt the execution to the needs of a user. By introducing problem categorization, we also change from considering specific trade–offs in seconds or joules, for normalized values that determine if a given configuration is better or worse for a given resource. Without generalization, we could determine for a given algorithm how much execution time is increased to save a given amount of energy. However, our objective here is to provide configurations based on their performance and efficiency, rather than estimate the exact amount consumed for each resource, as analytical and statistical models are better tools for that

purpose. Parallelization also adds enough complexity to the process, so that the expert has to increase the effort to properly categorize the kernels, hence, a simpler approach could be achieved with help of Machine Learning procedures.

5 Machine Learning Based Analysis

Users often want the best available performance, but as energy has become a major issue, less critical applications can be executed more efficiently over a longer period without degrading the user experience. In the previous section, we proved that a categorization of different algorithms is possible in order to reduce the total number of power configurations in a given system, and deep insight is achievable for a given problem in a given architecture. However, manually determining the membership of every new algorithm to an existing set of power configurations is a costly task, as we have shown through the whole computational experience. To further improve the management of the performance and energy efficiency in a given system, we propose the usage of ML algorithms to reduce the analysis complexity. We present a use–case using the data from our previous Pareto front analysis to clusterize the power configurations of the different NPB kernels. Furthermore, we will also present, through the same previous use–case, how to insert a new algorithm into an already existing clusterization so that previous knowledge can be reapplied to a new algorithm.

A cluster can be defined as a collection of records that have a high degree of "natural association" while the clusters are "relatively distinct" from one another [31]. In our context, the energetic behavior of each algorithm determines how we collect them into a cluster. The amount of power required for optimal energy efficiency and for optimal performance are the characteristics we have been analyzing to determine their association.

Clustering techniques serve the purpose of segmenting data in groups that have not been previously defined. As their counterpart, classification algorithms assigns the records in our data to previously defined groups. With the help of clustering, we can find the properties of similar items and use these properties to make recommendations. The main purpose of using clustering techniques in our proposal is to minimize the manual input of an expert to designate the power configurations illustrated in Section 4. Additionally, suggesting new target algorithms to existing categories without the need of intensive computation.

In unsupervised classification, determining the optimal number of clusters is a widely studied problem by itself. In diverse proposals, such as [32, 33], methodologies to identify a good enough representation of how data is intrinsically grouped. Clustering validation is achievable with different techniques. Internal validation of the clustering technique uses data information exclusively, which allows to evaluate our clustering structure without the need of an external data input. Including external data, such as the manual clustering achieved in Section 4, is often used to choose the optimal clustering algorithm over a specific dataset. In our particular case, clustering techniques can be used to extrapolate relationships between the different NPB kernels.

The proposal in this Section is divided in two steps. In the first step, we use clustering techniques to define a number of kernel categories by using different algorithms, indexes and distances from each cluster centroid. The attained number of clusters is compared to the experimental results from Section 4, presented in Fig. 4b, 5a and 5b. As a result from this step, a number k of clusters and their μ centroids are obtained. This step is performed once. Finally, in a second step, we determine the distance from a new kernel or algorithm

to each centroid μ using an algorithm selected during the first step. This step would be repeated for each new kernel we would like to analyze.

The chosen software implementation of the clustering algorithms is the R library *NbClust*. This package provides 30 statistical indices to determine the number of clusters, allowing the selection of various distance measurements such as Euclidean, maximum, Manhattan, Canberra and Minkowski distances, and various agglomeration methods: Ward, single, complete, average, McQuitty, median and centroid. In our work, we also combine various similarity measures, grouping and validity indexes, such as, KL index, Silhouette, Tau Index and SDindex

To determine which combination of algorithm and parameters yields better results in our environment, we have performed an in depth study using the 20 core study of the NPB, represented in Fig. 5a and 5b. The input records for our clustering algorithm correspond to each column in Fig. 5a and 5b, i. e. each kernel execution with the different applied power limits conforms a record.

We obtained the following best number of clusters after the combination of parameters for the executions illustrated in Table 3:

- 2 different clusters were obtained using 26 different clustering algorithms.
- 3 different clusters were obtained using 68 different clustering algorithms.
- 4 different clusters were obtained by only 7 different clustering algorithms.
- 5 different clusters were obtained using 2 different clustering algorithms.
- 6 different clusters were obtained using 32 different clustering algorithms.

Table 3 also contains the specific results obtained by each of the studied parameter combinations. The most common outcome for the best number of clusters is 3, which offers a counterpart to the analysis performed in Section 5. In Section 4, we categorized the NPB kernels in 4 different categories due to two specific power values presenting some strange behavior. Still, as the optimal

Agglomeration	Index	Distance Measurement				
		Eucl	Max	Manh	Canb	Mink
ward.D	kl	6	5	6	3	6
ward.D2	sdindex	3	6	3	6	3
	tau	3	2	3	3	3
single	kl	6	2	6	3	6
	sdindex	3	2	3	3	3
	tau	3	2	3	3	3
complete	kl	6	4	6	2	6
	sdindex	3	6	3	3	3
	tau	3	4	3	3	3
average	kl	6	2	6	2	6
	sdindex	3	2	3	2	3
	tau	3	4	3	3	3
mcquitty	kl	6	2	6	2	6
	sindex	3	2	3	3	3
	tau	3	4	3	3	3
median	kl	6	2	4	2	6
	sindex	3	3	3	3	3
	tau	3	4	3	3	3
centroid	kl	6	2	2	3	6
	sindex	3	3	3	3	3
	tau	3	4	3	3	3
k–means	kl	6	6	6	6	6
	sindex	2	2	2	2	2
	tau	2	3	2	2	2

Table 3: Number of power configurations suggested by each ML clustering algorithm, index and distance parameters

time and energy configurations in both categories are similar, the machine learning algorithms categorize them equally, as they are proven statistic and robust methods. These results justify the use of ML algorithms as part of a methodology to extract the optimal number of power configurations for a given software and hardware. The most influential parameter is the selection of the validity index, where the KL, Tau and SD indices yield 3 different clusters. It is important to remark that the selection of 2 or 6 clusters is still a valid clusterization for the input NPB kernels, where having more or less power configurations could satisfy special needs for specific environments.

i5-6200U Categorization output

Added Kernel	K–means	Analysis	Distance to Centroid 1	2	3
BT.W	$< \mathbf{1}, 1, 1, 1, 1, 2, 2, 3 >$	1	**7.99**	15.14	20.70
LU.W	$< 1, \mathbf{2}, 1, 1, 1, 2, 2, 3 >$	1	10.67	**7.30**	28.82
EP.W	$< 1, 1, \mathbf{1}, 1, 1, 2, 2, 3 >$	1	**8.51**	13.81	22.59
UA.W	$< 1, 1, 1, \mathbf{1}, 1, 2, 2, 3 >$	1	**8.15**	17.85	17.83
SP.W	$< 1, 1, 1, 1, \mathbf{1}, 2, 2, 3 >$	1	**8.74**	19.44	12.76
IS.B	$< 1, 1, 1, 1, 1, \mathbf{2}, 2, 3 >$	2	10.78	**10.26**	26.75
DC.W	$< 1, 1, 1, 1, 1, 2, \mathbf{2}, 3 >$	2	10.39	**7.26**	27.25
CG.A	$< 1, 1, 1, 1, 1, 2, 2, \mathbf{1} >$	3	**9.95**	19.82	

Table 4: Cluster membership comparative, NPB serial version in i5-6200U

Xeon Gold 6230N Categorization output

Added Kernel	K–means	Analysis	Distance to Centroid 1	2	3
BT.A	$< \mathbf{1}, 2, 1, 1, 1, 3, 1, 1 >$	1	**20.70**	25.72	39.80
LU.A	$< 1, \mathbf{1}, 2, 1, 1, 3, 2, 2 >$	2	**20.86**	22.73	42.81
EP.B	$< 1, 2, \mathbf{1}, 1, 1, 3, 1, 1 >$	2	**16.86**	44.11	25.22
UA.B	$< 1, 2, 1, \mathbf{1}, 1, 3, 1, 1 >$	2	**21.20**	31.58	43.13
SP.A	$< 1, 2, 1, 1, \mathbf{1}, 3, 1, 1 >$	1	**21.73**	23.16	42.35
IS.C	$< 1, 2, 3, 1, 1, \mathbf{3}, 3, 3 >$	3	27.25	35.81	**24.81**
DC.W	$< 1, 1, 2, 1, 1, 3, \mathbf{2}, 2 >$	4	24.90	**23.08**	33.87
CG.A	$< 1, 1, 1, 1, 1, 3, 2, \mathbf{2} >$	2	19.44	**18.88**	37.18

Table 5: Cluster membership comparative, NPB parallel version in Xeon Gold 6230N

For illustrative purposes, we have chosen the K–means agglomeration using the Manhattan distance, and will set the number of clusters to $k = 3$ in order to categorize our use–cases. The execution of a selected algorithm with the data obtained in the previous experimentation and illustrated in Fig. 4b, 5a and 5b yield the results presented in Tables 4 and 5.

For each individual kernel we performed the experimentation using the other 7 NPB kernels as training set and use the remaining kernel as testing set, so that we perform a k–fold cross validation. To introduce a new element to an existing categorization we have to perform these exact steps, which reinforces

our proposal. Each row in both Tables contains the data related to each of the training and testing sets, with the kernel used for testing referenced in column *Added Kernel*.

The column *K–means* shows a vector of the kernel membership to the clusters, where each position of the vector indicates which kernel in the tuple ($< BT, LU, EP, UA, SP, IS, DC, CG >$) belongs to which cluster (1, 2 or 3), with the testing kernel in bold. The column *Analysis* determines the *Added Kernel* cluster from our previous analysis for easy comparison with the conclusions extracted in the previous Section. Finally, the distance of the kernel to every cluster centroid is shown to better understand the presented data.

For better explanation, we will discuss the first row in Table 4, the serial version of BT.W. After executing the K–means for 3 clusters using all the kernels but the BT.W, we calculated the distance of the kernel to the existing centroids using the Manhattan distance. These distances, 7.99, 15.14, 20.70, determine that the BT.W belongs to cluster number 1, in bold in our K–means vector of membership. If we compare the obtained results and our manual analysis, we can observe that the automated procedure achieves the same results we achieved using the heatmaps.

For the i5-6200U case, in Table 4, the exact same results are obtained for the Pareto analysis and the categorization algorithm for the majority of the kernels. $LU.W$ when used as test, switches from the cluster 1 to cluster 2. Another remarkable case, the $CG.A$, has the peculiarity that it is the only member of cluster 3 in the other instances, and the k–means has a different suggested categorization using only 2 clusters. In this last case, we cannot conclude that $CG.A$ is significantly distant to the other kernels when added afterwards, and is categorized in one of the existing categories.

For the Xeon Gold 6230N case, Table 5 shows more variability for some kernels such as the LU.A, CG.A and DC.W, that is different from what we analyzed using the heatmaps, Fig. 5a and 5b. Still, each result in both tables is a valid solution, as our manual approximation, and each of the presented categorizations are reasonable to improve the efficiency of our target systems.

We profiled our R code and calculated the execution time for both clusterization and introduction of a new element to the existing clusters to illustrate the performance of this methodology. For the presented case, calculating the membership and number of clusters using the 7 kernels required $1.8 \pm 0.8\ ms$ in the Intel i5-6200U. In the other hand, incorporating a new element to the existing classification required $1.7 \pm 0.5\ ms$, and had an straightforward implementation.

Comparing the low computational effort required and the relatively low implementation difficulty to apply a clustering algorithm, and the costly task of studying all the data manually, we have determined that ML algorithms facilitate the process to obtain different power configurations to optimize software execution in a target architecture and automatize the data analysis to simplify the work necessary to apply our proposal. The initial categorization of the applications, could be automatically gathered together with a small benchmark when installing the system and also it could be incorporated as part of a queue manager or an scheduler in the target system. When new tasks are sent to the queue manager or scheduler, the resource usage of these tasks is collected and used to perform off–line calculations to classify the new items, software or tasks, to an existing category for future executions. Furthermore, when the volume of new items classified increases considerably, a recalculation of the clusters could be executed to refine the existing classification of the algorithms.

6 Conclusion

We have analyzed the effect of power capping technologies in a given architecture, and proposed a methodology to categorize the feasible power configuration of a target application. As the energetic behavior depends on both hardware and software, we have illustrated a use–case using the NPB kernels in a specific hardware. We extract a Pareto front of solutions for execution time and energy consumption for each kernel. The analysis of the Pareto front allows us to classify the different kernels to understand the power configurations in our system. This simplifies the optimization problem as we obtain an optimal power configuration per category, instead of optimizing each algorithm individually. Then, to automate and reduce the effort of applying this methodology, we presented a Machine Learning approach based in clustering to reduce the number of unique power limit configurations.

Due to the data–driven approach of Machine Learning, we cannot use our work as proof of truth to automate our methodology. However, our experimentation strongly supports the automation capabilities and illustrates how to incorporate new software to an existing categorization so that previous knowledge can be reused for a better resource allocation.

For the near future we plan to extend our ideas to highly parallel systems and introduce the methodology in power constrained environments where the power draw of the hardware is higher than the total available power. This likely will involve to incorporate some other Machine Learning techniques. Additionally, we hold interest in heterogeneous benchmarks, such as Rodinia, and architectures, including GPU processing. We have analyzed homogeneous codes in CPU environments that can be used as a starting methodology to better manage the difficulties introduced by heterogeneous environments

Declarations

Ethical Approval

not applicable

Competing interests

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Authors' contributions

These authors contributed equally to this work.

Funding

This work was supported by the Spanish Ministry of Science and Innovation with the PID2019-107228RB-I00 project; by the Government of the Canary Islands, with the project ProID2021010012 and the grant TESIS2017010134, which is co-financed by the Ministry of Economy, Industry, Commerce and Knowledge of Canary Islands and the European Social Funds (ESF), operative program integrated of Canary Islands 2014-2020 Strategy Aim 3, Priority Topic 74(85%); and the Spanish network CAPAP-H. This work was also supported by the European Union's Horizon 2020 research and innovation program under the FET-HPC grant agreement 801137 (RECIPE).

Availability of data and materials

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Jones, N.: How to stop data centres from gobbling up the world's electricity. Nature **561**(7722), 163–167 (2018)

[2] Andrae, A.S., Edler, T.: On global electricity usage of communication

technology: trends to 2030. Challenges **6**(1), 117–157 (2015)

[3] Fox, G.C., Glazier, J.A., Kadupitiya, J.C.S., Jadhao, V., Kim, M., Qiu, J., Sluka, J.P., Somogyi, E.T., Marathe, M., Adiga, A., Chen, J., Beckstein, O., Jha, S.: Learning everywhere: Pervasive machine learning for effective high-performance computation. CoRR **abs/1902.10810** (2019) arXiv:1902.10810

[4] Lison, P.: An introduction to machine learning. Language Technology Group (LTG), 1 **35** (2015)

[5] Cabrera, A., Almeida, F., Blanco, V., Castellanos–Nieves, D.: Finding energy efficient hardware configurations under a power cap. In: Avances en Arquitectura Y Tecnología de Computadores: Actas de Jornadas SARTECO, Cáceres, 18 a 20 de Septiembre de 2019, pp. 253–258 (2019). Servicio de Publicaciones

[6] Jalali, F., Khodadustan, S., Gray, C., Hinton, K., Suits, F.: Greening iot with fog: A survey. In: 2017 IEEE International Conference on Edge Computing (EDGE), pp. 25–31 (2017). https://doi.org/10.1109/IEEE.EDGE.2017.13

[7] Jin, C., de Supinski, B.R., Abramson, D., Poxon, H., DeRose, L., Dinh, M.N., Endrei, M., Jessup, E.R.: A survey on software methods to improve the energy efficiency of parallel computing. IJHPCA **31**(6), 517–549 (2017). https://doi.org/10.1177/1094342016665471

[8] Ahmed, K., Bull, J., Liu, J.: Contract-based demand response model for high performance computing systems. In: Chen, J., Yang, L.T. (eds.) ISPA/IUCC/BDCloud/SocialCom/SustainCom

2018, Melbourne, Australia, December 11-13, 2018, pp. 580–589. IEEE, ??? (2018). https://doi.org/10.1109/BDCloud.2018.00091. https://doi.org/10.1109/BDCloud.2018.00091

[9] Lei, D., Li, M., Wang, L.: A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold. IEEE Trans. on Cybernetics **49**(3), 1097–1109 (2019)

[10] Mahboubi, H., Masoudimansour, W., Aghdam, A.G., Sayrafian-Pour, K.: An energy-efficient target-tracking strategy for mobile sensor networks. IEEE Trans. on Cybernetics **47**(2), 511–523 (2017)

[11] Curtis-Maury, M., Shah, A., Blagojevic, F., Nikolopoulos, D.S., de Supinski, B.R., Schulz, M.: Prediction models for multi-dimensional power-performance optimization on many cores. In: 2008 International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 250–259 (2008)

[12] Wu, X., Taylor, V., Cook, J., Mucci, P.J.: Using performance-power modeling to improve energy efficiency of hpc applications. Computer **49**(10), 20–29 (2016)

[13] Rauber, T., RÜnger, G.: Dvfs rk: Performance and energy modeling of frequency-scaled multithreaded runge-kutta methods. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 392–399 (2019). https://doi.org/10.1109/EMPDP.2019.8671593

[14] Lively, C., Taylor, V., Wu, X., Chang, H.-C., Su, C.-Y., Cameron, K.,

Moore, S., Terpstra, D.: E-amom: an energy-aware modeling and optimization methodology for scientific applications. Computer Science - Research and Development **29**(3), 197–210 (2014). https://doi.org/10.1007/s00450-013-0239-3

[15] Rountree, B., Ahn, D.H., de Supinski, B.R., Lowenthal, D.K., Schulz, M.: Beyond dvfs: A first look at performance under a hardware-enforced power bound. In: 2012 IEEE 26th Intl. Parallel and Distributed Processing Symposium Wksh. PhD Forum, pp. 947–953 (2012). https://doi.org/10.1109/IPDPSW.2012.116

[16] Lawson, G., Sundriyal, V., Sosonkina, M., Shen, Y.: Runtime power limiting of parallel applications on intel xeon phi processors. In: 4th International Workshop on Energy Efficient Supercomputing, E2SC@SC 2016, Salt Lake City, UT, USA, November 14, 2016, pp. 39–45. IEEE Computer Society, ??? (2016). https://doi.org/10.1109/E2SC.2016.011

[17] Marathe, A., Bailey, P.E., Lowenthal, D.K., Rountree, B., Schulz, M., de Supinski, B.R.: A run-time system for power-constrained hpc applications. In: Kunkel, J.M., Ludwig, T. (eds.) High Performance Computing, pp. 394–408. Springer, Cham (2015)

[18] Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences **115**(34), 8505–8510 (2018)

[19] Kadupitiya, J., Fox, G.C., Jadhao, V.: Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles. Indiana University, Nov (2018)

[20] Yager, K.: Autonomous experimentation as a paradigm for materials discovery. In: Big Data and Extreme-Scale Computing Workshop (2018)

[21] Fahad, M., Shahid, A., Manumachu, R.R., Lastovetsky, A.: A novel statistical learning-based methodology for measuring the goodness of energy profiles of applications executing on multicore computing platforms. Energies **13**(15), 3944 (2020). https://doi.org/10.3390/en13153944

[22] Reddy, R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Transactions on Comp. **PP**(99), 1–1 (2017). https://doi.org/10.1109/TC.2017.2742513

[23] Endrei, M., Jin, C., Dinh, M.N., Abramson, D., Poxon, H., DeRose, L., de Supinski, B.R.: 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, pp. 17–11713. IEEE / ACM, ??? (2018). http://dl.acm.org/citation.cfm?id=3291679

[24] De Sensi, D., Torquati, M., Danelutto, M.: A reconfiguration algorithm for power-aware parallel applications. ACM Trans. Archit. Code Optim. **13**(4) (2016). https://doi.org/10.1145/3004054

[25] Coutinho Demetrios, A.M., De Sensi, D., Lorenzon, A.F., Georgiou, K., Nunez-Yanez, J., Eder, K., Xavier-de-Souza, S.: Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. Energies **13**(9) (2020). https://doi.org/10.3390/en13092409

[26] Wang, X., Zhao, B., Wang, L., Mak, T., Yang, M., Jiang, Y., Daneshtalab, M.: A pareto-optimal runtime power budgeting scheme for many-core systems. Microprocessors and Microsystems **46**, 136–148 (2016). https://doi.org/10.1016/j.micpro.2016.03.006

[27] Ma, K., Wang, X.: Pgcapping: Exploiting power gating for power capping and core lifetime balancing in cmps. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. PACT '12, pp. 13–22. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2370816.2370821. https://doi.org/10.1145/2370816.2370821

[28] Cabrera, A., Almeida, F., Arteaga, J., Blanco, V.: Measuring energy consumption using eml (energy measurement library). Computer Science - Research and Development **30**(2), 135–143 (2014). https://doi.org/10.1007/s00450-014-0269-5

[29] Fahad, M., Shahid, A., Manumachu, R.R., Lastovetsky, A.: A comparative study of methods for measurement of energy of computing. Energies **12**(11), 2204 (2019). https://doi.org/10.3390/en12112204

[30] Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., Yarrow, M.: The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center (1995)

[31] Anderberg, M.R.: Cluster Analysis for Applications: Probability and Mathematical Statistics: a Series of Monographs and Textbooks vol. 19. Academic press, ??? (2014)

[32] Davies, D.L., Bouldin, D.W.: A cluster separation measure. IEEE Trans.

on pattern analysis and machine intell. (2), 224–227 (1979)

[33] Handl, J., Knowles, J., Kell, D.B.: Computational cluster validation in post-genomic data analysis. Bioinformatics **21**(15), 3201–3212 (2005)