Cluster Computing (2024) 27:3433–3449
https://doi.org/10.1007/s10586-023-04151-2

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[2] · Vicente Blanco[1]

Received: 4 July 2023 / Revised: 8 September 2023 / Accepted: 9 September 2023 / Published online: 10 October 2023

Abstract
The growing demand for more computing resources has increased the overall energy consumption of computer systems. To support this increasing demand, power and energy consumption must be considered as a constraint on software execution. Modern architectures provide tools for managing the power constraints of a system directly. The Intel Power Cap is a relatively new tool developed to give users fine-grained control over power usage at the central processing unit (CPU) level. The complexity of these tools, in addition to the high variety of modern heterogeneous architectures, hinders predictions of the energy consumption and the performance of any target software. The application of power capping technologies usually leads to the bi-objective optimization problem for energy efficiency and execution time but optimal power constraints could also produce exceeding performance losses. Thus, methods and tools are needed to calculate the proper parameters for power capping technologies, and to optimize energy efficiency. We propose a methodology to analyze the performance and the energy efficiency trade-offs using this power cap technology for a given application. A Pareto front is extracted for the multi-objective performance and energy problem, which represents multiple feasible configurations for both objectives. An extensive experimentation is carried out to categorize the different applications to determine the overall optimal power cap configurations. We propose the use of machine learning (ML) clustering techniques to categorize each application in the target architecture. The use of ML allows us to automate the process and simplifies the effort required to solve the optimization problem. A practical case is presented where we categorize the applications using ML techniques, with the possibility of adding a new application into an 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

Alberto Cabrera, Francisco Almeida, Dagoberto Castellanos–Nieves, Ariel Oleksiak and have contributed equally to this work.

✉ Vicente Blanco
vblanco@ull.es

Alberto Cabrera
Alberto.Cabrera@ull.es

Francisco Almeida
fameida@ull.es

Dagoberto Castellanos-Nieves
dcasterll@ull.es

Ariel Oleksiak
ariel@man.poznan.pl

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

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

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 computational hardware, more efficient architectures have been introduced over the last few 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 the analysis of both the performance and energetic behavior of the computational hardware to avoid detrimental configurations. This new opportunity for energetic tuning in applications adds a new layer of complexity, particularly in parallel environments where algorithms may have different sections of code executing simultaneously.

The application of power capping technologies allows new parameters to be introduced to manually control power draw within certain limits. This parameterization would allow a significantly reduced power consumption without performance loss and the end result could be more energy efficient. The influence of these parameters and how they affect the system when modified can be studied using different techniques, such as analytical modelling or performance analysis, to provide different mechanisms for limiting power with specific parameters that directly affect power or energy consumption.

The complexity of these tools, combined with the high variety of modern heterogeneous architectures, hinders any prediction about the energy consumption and the performance of any target software. Optimal power constraints could also produce excessive performance losses. Increasing our knowledge of the energy performance of these architectures in advance is key to maximize our use of resources. Without predictive techniques, the outcome of applying power capping techniques is uncertain, and the resulting effect of power management may ultimately degrade time performance and energy consumption. Therefore, methods and tools are needed to apply appropriate parameters to power capping technologies, and optimize energy efficiency.

Analytical modeling allows us to study and provide knowledge of complex systems. Models give insight on the complexity of the computational environments and facilitate decision making to achieve our goals. Different objectives can be selected to adapt to a given environment, such as minimizing time to solution, reducing energy consumption, or finding the best performance under a power constraint. However, analytical models are quite difficult to attain and usually give very deep insight of a specific application. Performance analysis is a discipline focused on enhancing the effectiveness of a given procedure by observing its behavior on a target. The knowledge gained provides insight into the complexity of computational systems. 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 us to find better configurations to the resource usage in a given architecture.

Manual analysis of the problem becomes a costly task that requires deep insight on the architecture and the software to be tuned, the complexity and amount of work carried out by the experts increases, and the volume of data to manage entails additional difficulties. The application of power capping technologies usually leads to the bi-objective optimization problem for energy efficiency and execution time. Theoretical models are hard to obtain since there is a high trade-off between model complexity and accuracy. However, as systems increase in size and complexity, statistical techniques get an increased significance.

In [3], we performed a preliminary 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 paper, we present a generalized methodology, based on the Pareto front configuration, to analyze the performance and energy consumption in a given system when executing applications under a power limit. This analysis allows us to extract a set of power cap configurations to adapt the energy efficiency of a given system. The configurations obtained allow, at runtime, the definition of policies to satisfy a power or performance constraint. To illustrate and justify our proposal, we present our extensive experimentation and discuss the different

feasible power configurations that optimize our target architecture. We analyzed the effect of power capping technologies for a specific architecture, and obtained the Pareto front of different NAS Parallel Benchmark (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 thus able to reconfigure the hardware power limits taking into account different power and performance constraints. This is a hard and computational intensive task.

In this paper we also propose the use of unsupervised machine learning (ML) techniques. They provide an automated and statistical approach to extract information from big volumes of data, and offer deep insights on how to define the behavior of complex systems [4]. Machine learning is a subfield of artificial intelligence that enables computers to acquire knowledge from data without the need for explicit programming. There are three main categories of machine learning algorithms: supervised, semi-supervised, and unsupervised. In supervised learning, labeled data is used to establish relationships between inputs and outputs. Semi-supervised learning involves both labeled and unlabeled data to enhance accuracy, while unsupervised learning has the algorithm discovering patterns in unlabeled data, useful in tasks such as clustering similar data or detecting anomalies. These approaches empower computers to learn and make autonomous decisions based on data. In our case, applying unsupervised ML techniques to the obtained data allows us to extract hidden features and simplify the volume of data [5]. Additionally, the use of ML allows us to reduce the computational efforts to obtain the Pareto front. We propose the use of ML clustering techniques [6, 7] to automate the process. Clusterization allows a set of applications to be grouped into different categories according to their energy resource requirements. The set of benchmark applications will be clusterized through ML according to their energy consuption patterns and when a new application needs to be executed, the cluster to which it belongs holds the information for optimal executions.

The main goal of using clustering techniques in our proposal is to minimize the manual input of an expert to identify the power configurations shown in Sect. 4. In addition, it makes it possible to suggest the inclusion of new target applications to existing categories without the need for intensive computation. The ML approach has a fixed computational and energy cost (only one execution is needed), which is low when compared to the potential benefit of its use on applications that run continuously. The time performance presented in Sect. 5 shows that the energy consumption is negligible in relation to the benefits. 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 main contributions of our paper are as follows:

- 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 achieved for these objectives, and how increasing the size of the problem does not affect the power cap parameters of the Pareto front. The Pareto front also provides the ability to predict the trade-offs between execution time and energy when using a power cap technology. In addition, we formalize our methodology to illustrate the independence between our proposal and the specific technology chosen to validate this research.
- We have performed a large number of experiments using different power cap settings, resulting in a rich data set of time and energy measurements in both serial and parallel executions, using a variety of kernel applications from the NPB [8], which are derived from computational fluid dynamics (CFD) applications.
- We have applied this methodology to provide an in-depth experimental validation of the energetic behavior of the NPB. We analyzed the effect of power capping technologies over multiple NPB kernels in a given architecture. Due to the previous analysis, we were able to reduce the amount of experimentation. We gained deep insight from the feasible power capping configurations for each application, which we use to categorize them. We simplified the optimization problem using these categories and obtain one optimal power configuration per category, rather than optimizing each application 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 a comprehensive set of clustering machine learning 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.

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 Sect. 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, Sect. 6 summarizes our conclusions and future work.

2 Related work

Energy efficiency is one of the multiple challenges in several contexts 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 [9]. 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 [10, 11]. 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 [12], or energy-efficient strategies for tracking a target in a field of obstacles [13].

Dynamic voltage and frequency scaling (DVFS) [14–16] 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 [16–18]. *E-AMOM* [19] 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 has 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 to be a possible replacement for the DVFS techniques present in the literature [20, 21]. Using DVFS along power capping technologies through the RAPL interface, *Conductor* [22] is introduced as a runtime system to optimize application performance within a given power constraint. It is capable of improving the performance of applications under a power budget by allocating and shifting the power consumed by the application.

In computer science, ML is gaining traction due to the effectiveness of deep neural networks, and the scientific community is applying these techniques using multiple approaches to improve performance [23], to solve complex problems [23, 24], to automatically tune experimentation [24], to design experiments [25] or to evaluate the quality of energy profiles compared to measurements obtained from external power meters [26]. As it has been proven to work in countless cases, we apply ML techniques to manage decision-making automatically, focused on the energy efficiency and performance trade-offs of a given application.

There are also multiple methodologies and algorithms that analyze and solve power-performance trade-offs in the literature [12, 27–30], as modern processors offer multiple power and performance optimization options by changing core and uncore frequencies individually, affecting the overall voltage and power. NORNIR [27] focuses on a single application and, with the help of linear regression and monitorization, configures an application at runtime to satisfy user needs. A methodology was also introduced in [28] 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 [29] 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 [30] 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 work introduces a methodology to analyze the use of power cap technologies in a specific system. Within the HPC community, analytical and statistical models are frequently employed to identify optimal configurations to achieve performance and energy efficiency. The application of a Pareto front multi-objective approach has been widely used to represent workload redistribution in manycore architectures [31]. In our study, we adopt an experimental approach to analyze our target architecture, with a particular focus on investigating the impact of power capping technologies rather than workload redistribution.

Regression based methods have also been used to explore Pareto efficient configurations to account for the trade-off between time and energy efficiency [32], where DVFS and parallelization is analyzed to formulate a model for a given application. Similarly, we share the goal of finding the optimal trade-off zone and perform an in-depth analysis of our target applications. However, our approach is applied to a wide range of applications, and our main contribution is the categorization of multiple Pareto fronts to reduce the number of optimal power configurations in a given system. 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 clustering techniques, rather than defining new methods, algorithms or analytical models. This simplifies the level of expertise required by the user to implement our solution.

3 Power cap performance analysis

We propose a methodology for using power management tools to extract optimal configurations for a set of power constraints. To introduce the reader to the power capping concepts, an analysis of the available power capping technologies is presented in Sect. 3.1. In Sect. 3.2, we present a step-by-step methodology to apply power management technologies to a given architecture executing a given application, using the Intel power cap.

We provide deep insight into the interpretation of the data and the different power configurations extracted from the study. Based upon the knowledge obtained through this experimental methodology, we are able to classify and categorize applications that, despite being different in nature, can share the same 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 consumption in current hardware architectures. While not available for every architecture, the use of implemented vendor solutions provides a simple solution for computing with a defined power constraint. In the case of ARM processors, power management relies on their on-chip management and the use of dynamic on-chip voltage and frequency scaling across the chip [33], unless custom designs give control to the user. State-of-the-art processors allow the user to change the frequency of individual cores [34, 35] as well as a selection of C-states of the OS [36, 37]. AMD included a Thermal Design Power (TDP) management tool in their older architectures, the 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 [38]. On the other hand, Nvidia and Intel provide their own approach to give users control over their power management.

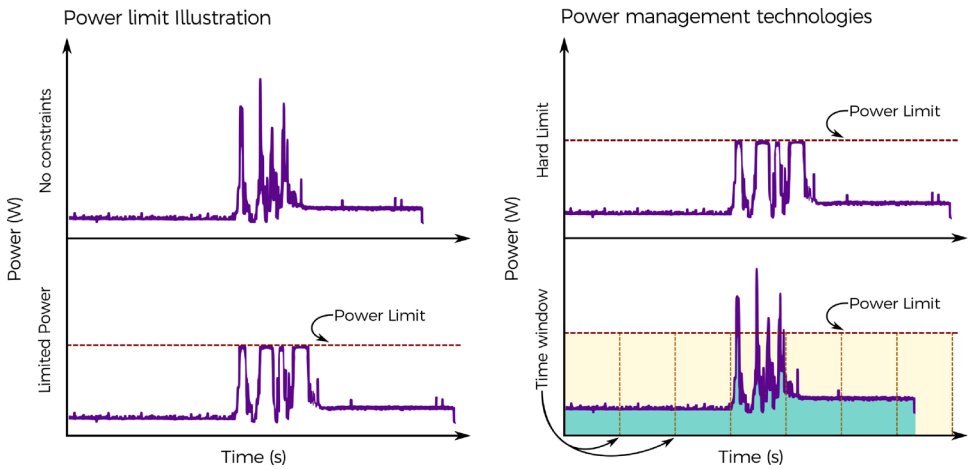
Nvidia's drivers allow to set a power limit for different GPUs [39, 40]. However, their hardware requires a minimum power, which results in less control to the power consumption management. High-end and modern cards offer a better range to control this power limitation. The Nvidia power limit implementation sets a strict constraint as a hard limit that the hardware cannot exceed. Figure 1 showcases the impact of hard power management enforcement using a simple example. Figure 1a depicts the power profile of a generic execution without any power management tool (at the top) and shows how setting a strict power cap affects the overall execution. The lower portion of Fig. 1a illustrates a hypothetical power limit that is applied, resulting in longer execution times for both the power intensive sections and the overall application.

The impact of these power limits on overall energy consumption will always depend on the nature of the application. While some applications will lose performance and consume 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 for their chips through the Running Average Power Limit (RAPL) [41–43]. RAPL provides mechanisms to set average power limits on different zones of Intel processors. These zones, known as Power Planes (PP) [44], 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 allows 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 that our power constraints are satisfied. Both time windows and power limits are limited by the manufacturer specifications and must be adapted for the specific processor model we are using.

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

Along the paper, we will use two different architectures to illustrate our methodology. An Intel i5-6200U CPU (i5-6200CPU) for the sequential computations, it is 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 runs an Ubuntu 18.04 LTS, with kernel version 4.18 and GCC 7.4. And a second platform, an Intel Xeon Gold 6230N (Intel Xeon) to illustrate the method over parallel applications on a multicore sytem. The Intel Xeon Gold 6230N has 20 cores at 2.30 GHz, running Debian 10, kernel version 5.4.0-0.bpo.2-amd64, and GCC 8.3.0.



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

Fig. 1 Illustration of different power management tools

Energy measurements were collected using the EML [45], a driver based energy measurement library. In our experiments, we used the appropriate drivers to extract total energy metrics from the RAPL interface. RAPL has been criticized for not being very accurate in measuring the energy consumption of applications [46]. However, in terms of the implications of validating our methodology there is no loss of generality when using RAPL, as the methodology essentially obtains a Pareto front or apply ML clustering techniques from a dataset to model the behavior of a set of applications. The Pareto front and the obtained clusters will model the dataset, be it accurate or not, so using the RAPL tool for measurements does not affect the validity of the analysis. The methodology will not change even if the dataset changes.

3.2 Multi-objective optimization using power limits

We can use power limiters for different constraints in our infrastructure, addressing multiple objectives simultaneously. Performance, power consumption, energy efficiency, quality of service or temperature are examples of different metrics that we might consider in decision making. These multiple objectives may or may not be in conflict with each other. Temperature and power are highly correlated, while performance and quality of service are similarly linked. However, energy efficiency and performance can have opposite effects for a given power cap configuration.

In other words, for any given power configuration, a number k of different objectives can be considered for optimization. Each power configuration has a direct impact on these k objectives and directly affects how power management changes the hardware behaviour. Formally, we can formulate a multi-objective optimization problem in which we consider:

- k, the number of objective functions to address.
- f_i, optimization function for the i-th objective.
- p, power cap configuration, which consists of various parameters specific to each 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.

In this Section, we will illustrate our formalization with a specific instance of the NPB. These benchmarks provide a standardized suite of programs (or kernels) that represent various computational problems. One of the kernels within NPB is the BT kernel, which stands for Block Tridiagonal solver. The BT kernel involves solving a system of equations with a block tridiagonal coefficient matrix, which is a common method used in many applications. In NPB, the problem sizes used for evaluation are standardized and called as classes. The kernel nomenclature in NPB follows a consistent format, typically denoted as (AA.B), where AA is the abbreviation for the specific kernel, and B is the class of the kernel. For example, BT.W refers to the BT kernel class, or size, W, that solves a 3-dimensional matrix of size (24, 24, 24). Throughout this section, we will provide examples and analysis specifically tailored to the case of sequential executions on the i5-6200CPU for the BT.W An example for $k = 2$ conflicting objectives is shown in Table 1, where each power configuration p is defined by an average power limit, a time window and a power plane. A variety of power limits and time windows have been selected, while the power plane is fixed to the PSYS. A static PSYS, is not optimal for every algorithm, but is sufficient for our homogeneous codes. The results of BT.W produce a set of metrics $(f_1, f_2) = (execution\ time,\ energy\ consumption)$ for each configuration p, obtained over

Table 1 Sequential NPB, BT.W Kernel (execution time (s), energy consumption (J)) using Intel power cap. i5-6200CPU

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)
4.5	(3.55, 16.00)	...	(3.58, 16.16)	(3.57, 16.10)
⋮	⋮	⋱	⋮	⋮
8.0	(2.31, 18.54)	...	(2.33, 18.70)	(2.32, 18.67)

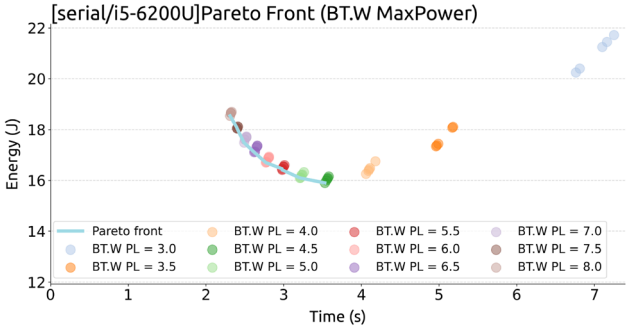


Fig. 2 Sequential NPB, BT.W Kernel. Pareto front for k = 2 objectives. Energy and execution Time. PL = Power Limit in Watts. i5-6200CPU

different power limits and time windows. In subsequent Sections some other sequential and parallel kernels of the NPB will be considered.

The contents of Table 1 are shown in Fig. 2 using a different color for each power limit to facilitate data analysis. We have chosen to optimize execution time and energy consumption, which are represented in both axes of Fig. 2.

In Fig. 2 we can see the multi-objective nature of these objective functions and multiple conflicting solutions. Since it is unfeasible to minimize both objectives simultaneously, a set of Pareto optimal solutions is required. A Pareto front is obtained with all the non-dominated tuples, represented by a segment. The dominated solutions, are discarded as there is, at least, one tuple in the Pareto set that improves both objective functions. In addition, Fig. 2, also shows how the time window parameter for the Intel architecture has a much smaller impact than the power limit itself. To simplify the data, we could average the metrics for each power limit, which would also reduce the total number of experiments required. However, despite the apparent uniformity of this Pareto front, we are not able to use a binary search approach to sample because we cannot guarantee this behavior for any application running on modern hardware. In Sect. 4, we also present results for a parallel version using 20 cores, where some kernels are not as uniform as in example shown.

In order to find the different configurations for the target architectures, we have defined a simple benchmarking process to determine the behavior of each application. Systems can be configured using the power management tools and the results may vary with different hardware and software. We learn the effects of applying power limits to our heterogeneous system by running small and simple instances of software. The experimentation is relatively simple and can be extended to different applications and architectures without great effort.

The final goal is to obtain a Pareto front for a number of conflicting objectives to optimize the applications. After the experimentation, a variety of power configurations p will be available for each application in the Pareto front, so that we can optimize our objective functions.

We can formally define our experimental 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. For example, for the Intel architecture i5-6200CPU that we are considering, p can be defined as a 3-tuple of an average power, a time window, and a power zone to which the configuration applies, i.e. $p = (avgpower, timewindow, zone)$. While multiple limits can be set during an execution, we will use a single power limit for each case. In another example, the Nvidia GPU architecture would only need the maximum power value, i.e. $p = (maxpower)$.
- $M_{h,s}$, hardware metrics collected during the experiments. 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 the case presented in Table 1 and Fig. 2, for $k = 2$, we have a 3-tuple with the experimental metrics: p, execution time and energy consumption, although we could also consider other objectives.
- $P_{h,s} \subseteq P_{H,S}$, Pareto front set of power configurations that can be used for a given s and h to optimize our multiple objectives. Extracting this set depends on how we want 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 for a given s and h. Along the paper we

will use the notation $p^t_{h,s} \rightarrow (execution\ time^*, energy\ consumption)$, to denote the best *execution time* found for the h, s configuration, and at the same time indicate the energy consumed for this optimal execution time.

- $p^e_{h,s} \in P_{h,s}$, power configuration that achieves the most efficient energy consumption for a given s and h, i.e., minimizes energy consumption of a given task. Again, in what follows, we will use the notation $p^e_{h,s} \rightarrow (execution\ time, energy\ consumption^*)$, to denote the best *energy consumption* found for the h, s configuration, why also denoting the execution time required for this optimal energy consumption.

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

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 discuss three different approaches to manage the power consumption of a system using the BT.W kernel as an illustrative case:

- A first approach would be to consider having k different configurations defined by the user, so $|P_{h,s}| = k$, one configuration per optimization objective. In the experiments presented for the BT, these values set the power limit at 8 or 4.5 Watts respectively to minimize execution time or maximizing energy efficiency. This would be the situation if one wanted to simplify the multi-objective problem by solving the single-objective problems individually. Our methodology aims to solve the multi-objective problem as a whole, so this option is discarded.
- A second approach is to consider every possible value between 4.5 and 8 Watts, including all the different solutions to achieve better energy efficiencies with less impact on the execution time trade-off if necessary. This is the option we will consider along the paper.
- A third approach could be to consider a power limited situation where we would configure power limits from 4.5 Watts to a maximum available power. For example, in an environment where we only have 6 Watts available for our CPU, we would consider configurations between 4.5 and 6 Watts. The methodology applied to this case would be the same as in the former one, except that the Pareto front would be a subset of the former since the problem has an additional power limitation. Therefore, without loss of generality, this approach will not be considered throughout the paper.

In each case, the end result is a set of power configurations from which we can make decisions to meet different energy and execution time constraints.

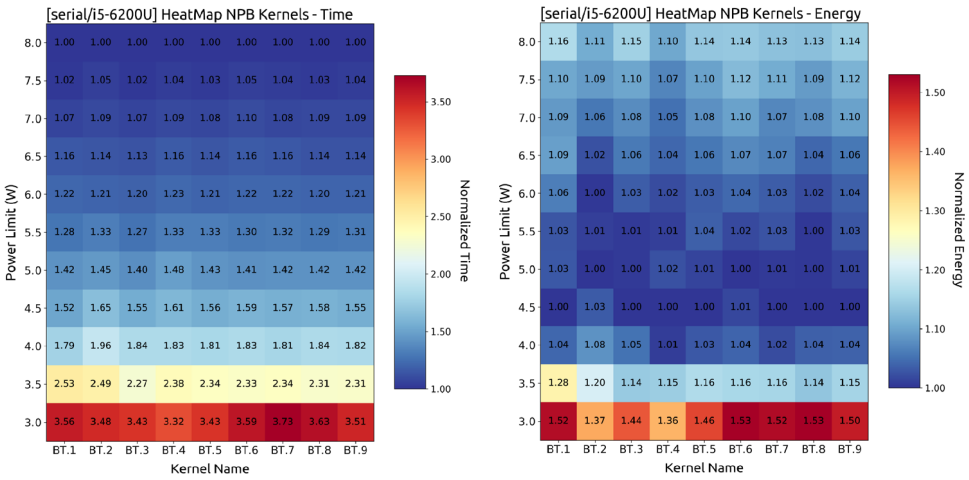
The Pareto front represents the optimal solutions to the Multi-objective problem for all the objective functions simultaneously. In order to perform sensitivity analysis and also trying to get some predictive ability from the methodology, we now introduce an alternative representation model based on heat diagrams (see Fig. 3). This allows the analysis of the individual objectives without losing the Multi-objective perspective. Continuing with our studio case, 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 because of the low dependence observed from the time window. At the same time, the data were normalized to show the effect of the power configuration for each objective function. This allows us to compare data for different sizes or problems in an unified heatmap, as shown in Fig. 3.

The NPB predefined sizes, or classes, have a 4-fold increment between them, as they are designed for highly parallel systems. In order to have a smaller range of different sizes for the BT kernel, we have created a new set of classes, from BT.1 to BT.9. These custom classes allow for a more gradual increase in problem size. The smallest class, BT.1, performs calculations on a 3-dimensional matrix of size (16, 16, 16), while the largest class, BT.9, processes a (32, 32, 32) size matrix, with increments of (2, 2, 2) between each class, or size. In other words, BT.1 size is (16, 16, 16), BT.2 (18, 18, 18), BT.3 (20, 20, 20), and so on, up to BT.9. As a reference with the predefined classes, BT.W is of size (24, 24, 24) which corresponds to BT.5.

In Fig. 3, we illustrate the execution time and energy consumption of various custom classes of the BT kernel using this alternative representation. The Y represents the maximum power allowed to the CPU, each power level is the average of the results of several experiments conducted over different time windows. The columns of the heatmap are normalized, with a value of 1.00 indicating the optimal measurement for each problem size. In both figures, dark blue indicates the best values, while red indicates 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 again observe negligible



(a) Execution Time (b) Energy Efficiency

Fig. 3 Sequential NPB. Alternative representation for the Pareto front. Several BT custom classes. Normalized values, the closest the cell value is to 1, the better. i5-6200CPU

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

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

differences between the different problem sizes. The optimal power limits in Fig. 3 are 8 W and 4.5 W respectively, where 8W improves the execution time by 36.4% over 4.5W and 4.5W improves energy consumption by 11.6% over 8W, which are the extreme values in the Pareto front set.

In this bi-objective case, optimal values can be obtained 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 required to improve the energy efficiency. This data representation helps to perform a sensitivity analysis for different software and hardware configurations to accurately discard impractical power values and slightly reduce the total number of executions required to study an architecture.

In this first use-case, we could discard values above 8.0 W and below 3.0 W as the analysis for BT has shown that these limits are reasonable in our system. However, this is only true in this particular 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 this section we will develop our experimental methodology by testing several kernels of the NPB. In the Sect. 3.2 we illustrated the methodology using the BT.W kernel as an example, and the custom executions for BT.1 to BT.9, now we extend the experiments to some other NPB kernels.

Using the RAPL interface, we specify multiple time windows for each specified power average to the CPU. We have approached the problem with simplicity in mind, so we limited the power consumption of the entire CPU using the PSYS zone. In Sect. 3.2 we introduced several power configurations using $p = (powerlimit, timewindow, PSYS)$ for the BT kernel and we observed that the time window had less impact for this kernel. This behavior also occurs in each kernel individually, and is repeated for each of the problem sizes in the serial version executions of the NPB kernels considered in the computational experiments. In order to facilitate the analysis of the behavior for each kernel, we will condense the data by taking an average of three executions for each different power limit.

Our first approximation shows that proper management of the Time Window improves the overall efficiency and execution time of a target application. However, in-depth analysis is required to draw the right conclusions on how to adjust the parameter to suit our needs.

Following the same data treatment to normalize values, Fig. 4 shows the dataset for 8 NPB kernels in their serial version, similar to Fig. 3. Again, the best values are normalized to 1 to adequately compare different applications or problem sizes. While performance is still addressed, the following discussion will mainly focus on the energy efficiency of the power cap configurations.

The target sequential architecture (the i5-6200CPU) allows a wider range for power capping, below 3 Watts and

above 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, the metrics from the experiments with a 3 Watts limit showed an extreme decrease in both performance and energy efficiency.

We can apply the conclusions drawn from the BT study to all the kernels studied. 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 are affected by the power cap limit with tuples of the form of $(executiontime,\ energyconsumed)$. For example, for the BT.W kernel presented in Fig. 4 the best performance is achieved with a power limit > 8 Watts, $p^t_{h,s} > 8W \rightarrow (1.00,\ 1.16)$, which translates to a 16% increase in energy consumption, while the best energy efficiency is achieved at 4.5 Watts, $p^e_{h,s} = 4.5W \rightarrow (1.53,\ 1.00)$, with a 53% increase in execution time.

Three different patterns are observed for the remaining kernels in the data shown in Fig. 4:

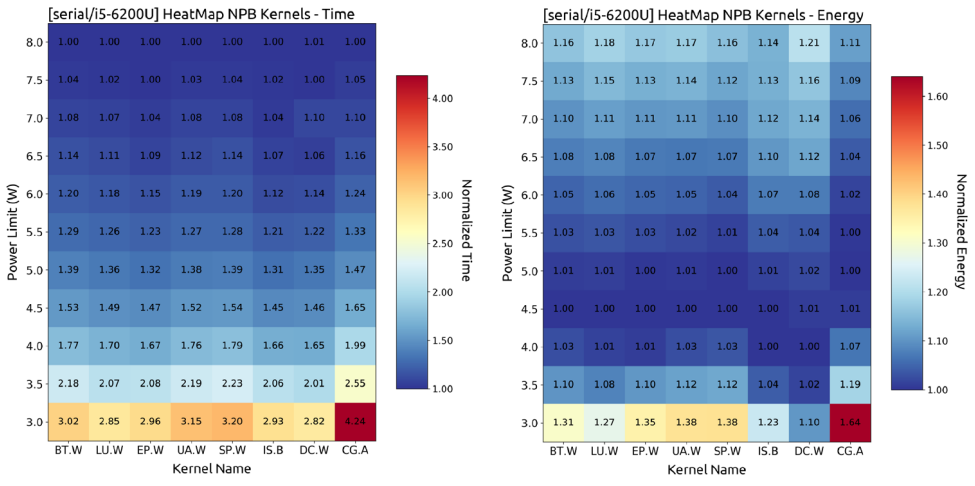
- The first case includes the DC.W and the IS.B. These kernels reach the best energy efficiency at 4 Watts, where for DC.W: $p^t_{h,s} > 8W \rightarrow (1.00, 1.21)$, $p^e_{h,s} = 4W \rightarrow (1.65, 1.00)$ and, for IS.B: $p^t_{h,s} > 8W \rightarrow (1.00, 1.14)$, $p^e_{h,s} = 4W \rightarrow (1.66, 1.00)$. Both algorithms are memory intensive, and while data does not indicate why, we suspect the similar power configuration has something to do with it.
- The CG.A reaches the best energy efficiency at 5 Watts: $p^t_{h,s} > 8W \rightarrow (1.00, 1.11)$ and $p^e_{h,s} = 5W \rightarrow (1.46, 1.00)$.
- The remaining kernels achieve 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 many differences between them, these kernels hardly require different power limits in their serial versions.

In addition, Fig. 4 shows that multiple applications can have similar energetic behavior. With this information, instead of having one set of power policies per application, any decision that improves execution time or energy efficiency could be applied to a category of applications.

The analogous study using the *OpenMP* version of the NPB kernels has also been performed to analyze a different target CPU, where we observe a similar energy and performance behavior. We analyzed the Intel Xeon parallel architecture described in Sect. 3.1 using the same experimental methodology. The variation in hardware power consumption increases the range of feasible power limits to manage the energy efficiency for the NPB kernels. We have also selected classes with larger problem instances, in order to take advantage of the increased computational capacities. For DC, however, the selected class is still *W*, since the performance is I/O dependent.

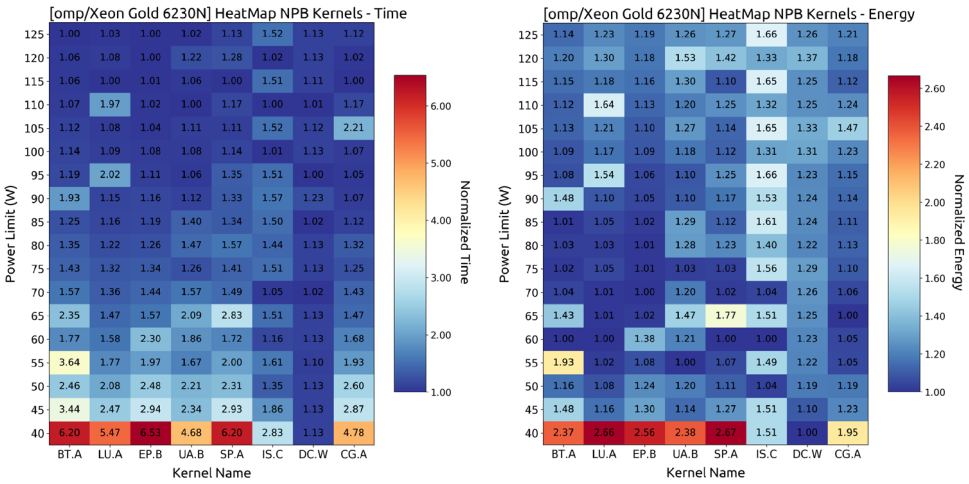
Figure 5 represents the *OpenMP* parallel execution using the same kernels presented in the previous section. Here, we take advantage of the twenty cores available in our Intel Xeon parallel architecture, extending our power limits from 30 W to a TDP of 125 W.

For this hardware, the fastest configurations still require the removal of any power limitations, as would be expected from the serial versions. For optimal energy efficiency, we again find different patterns in the data presented in Fig. 5. The minimum in both heatmaps was increased to 40 Watts because the loss of efficiency at lower power levels reduces



(a) Execution Time (b) Energy Efficiency

Fig. 4 Sequential NPB. Alternative representation for the Pareto front. Several NPB standard kernels. Normalized values, the closest the cell value is to 1, the better. i5-6200CPU



(a) Execution Time (b) Energy Efficiency

Fig. 5 Parallel NPB Benchmarks. Alternative representation for the Pareto front. Several NPB standard kernels. Normalized values, the closest the cell value is to 1, the better. Intel Xeon

the clarity of the chart in most of the cases. Nevertheless, each omitted value was strictly worse than the value at 40 W. Three patterns are observed for the parallel kernels shown in Fig. 5, as in the sequential case, however in the parallel case the kernels are grouped in a different manner: 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 categorization process is applicable to both serial and parallel versions. We can define a small number of power and performance policies for a given system, and adapt the execution to the needs of the user. By introducing problem categorization, we also move from looking at specific trade-offs in seconds or joules, to normalized values that determine if a given configuration is better or worse for a given resource. Without generalization, we could only determine how much execution time is increased to save a given amount of energy for a given application.

The former methodology could also be applied in a power constrained highly parallel or distributed system, where multiple hardware platforms are available. A resource allocation problem can be formulated for the platform to partially utilize the available hardware according to the multi-objective criteria. However, our goal here is to recommend configurations based on their performance and energy efficiency, rather than estimating the exact amount of energy consumed for each resource, as analytical and statistical models are better tools for this purpose. Parallelization also adds enough complexity to the process, so that the expert has to increase the effort to properly categorize the kernels. A simpler approach could be achieved using machine learning techniques.

5 Machine learning based analysis

Machine learning (ML), a subfield of artificial intelligence, empowers computers to acquire knowledge from data without the need for explicit programming. Within ML, unsupervised learning involves training the algorithm on an unlabeled dataset. The algorithm utilizes this dataset to discover patterns and structures within it. In particular, clustering techniques [6, 7] are used to divide a dataset into groups or clusters, where the data in each cluster is more similar to each other than to data in other clusters. Clustering models are among the most recognized unsupervised learning techniques, employed to group objects based on the similarity of their characteristics.

Various clustering techniques exist, each with its strengths and weaknesses. Some common techniques include K-Means [47], Agglomerative [48], and Divisive [49]. K-Means is a straightforward clustering algorithm that randomly selects K centroids and then assigns data to the nearest centroids. Agglomerative and Divisive are more complex algorithms that combine or split clusters successively until the desired number of clusters is achieved.

Distances play a pivotal role in cluster analysis as they measure the similarity or dissimilarity between individuals or groups. Different types of distances can be chosen based on the data type and target analysis. Distances are used to calculate clustering metrics, which indicate how clusters are formed. It is essential to compare these metrics and select the one that best fits the data and the researcher's criteria. In predictive analytics, clusters can be utilized to build predictive models or as part of a segmentation process. In the former case, clusters serve as independent variables in predictive models, allowing the model to capture similarities or dissimilarities between data points, thereby improving prediction accuracy. In the latter case,

clusters are used to divide the dataset into smaller, more homogeneous groups, which can facilitate pattern identification and the construction of more precise models. Additionally, it is possible to classify a new dataset using a trained clustering model. The clustering model can be employed to assign the new data to the most similar cluster, simplifying the problem-solving process.

In Sect. 4, we proved that a categorization of different applications is possible in order to reduce the total number of power configurations in a given system, and that deep insight is possible for a given problem in a given architecture. However, manually determining the membership of each new application for an existing set of power configurations is a costly task, as we have shown throughout our computational experience. In order to improve the management of the performance and energy efficiency in a given system, we propose the use of ML clustering techniques to reduce the complexity of the analysis. Moreover, when a new application requires execution, it is associated to a cluster to identify the parameters for efficient executions. The applications will be clusterized according to their energy power usage. 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, using the same previous use-case, we will also show how a new application can be added to an existing clusterization, so that previous knowledge can be used in a new case.

Our proposal works into two steps. In the first step (Sect. 5.1), we use clustering techniques to define a set of application categories by using different algorithms, indices and distances from each cluster centroid. As a result of this step, a number K of clusters and their μ centroids are obtained. This step is performed once. The attained number of clusters is compared with the experimental results for the Pareto front in Section 4, which are presented in Figs. 4 and 5. In a second step (Sect. 5.2), we determine the distance from a new kernel or application to each centroid μ using an algorithm selected in the first step. This step would be repeated for each new application we would like to analyze.

5.1 Application categorization through clustering

A cluster can be defined as a collection of datasets that have a high degree of "natural association" while the clusters are "relatively distinct" from each other [50]. 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 that have been analyzed to determine their association.

Clustering techniques serve the purpose of segmenting data into groups that have not been previously defined. These techniques help to find the properties of similar items and use these properties to make recommendations.

In unsupervised clustering, determining the optimal number of clusters is a widely studied problem by itself. In various proposals, such as [51, 52], are introduced methods to identify a good enough representation of how the data is intrinsically grouped. Clustering validation can be achieved using a variety of techniques. Internal validation of the clustering technique uses only data information, which allows us to evaluate our clustering structure without the need for an external data input. This analysis can be complemented with any other additional knowledge, such as the manual clustering achieved in Sect. 4, to select the optimal clustering algorithm for a given dataset. In our particular case, clustering techniques can be used to extrapolate relationships between the different NPB kernels.

The chosen software implementation of the clustering algorithms is the R library *NbClust* [53]. This package provides 30 statistical indices to determine the number of clusters, allowing the selection of different distance measures such as Euclidean, Maximum, Manhattan, Canberra and Minkowski distances, and various agglomeration methods: Ward, Single, Complete, Average, McQuitty, Median and Centroid [53]. In our work, we also combine various similarity measures, grouping and validity indices, such as the KL index, Tau Index and SDindex [53].

To determine which combination of application and parameters produces better results in our environment, we have performed an in depth analysis using the 20 core study of the NPB, shown in Fig. 5. The input datasets for our clustering algorithm correspond to each column in Fig. 5, i. e., each kernel execution with the different power limits applied corresponds to one data set.

Table 3 collects the best number of clusters obtained after combining the different clustering methods, validity indices and distances. The best number of clusters obtained is as follows:

- 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 using 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.

The most common result for the best number of clusters is 3, a result slightly different from our analysis in Sect. 4.

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

Clustering	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
	SDindex	3	2	3	3	3
	Tau	3	4	3	3	3
Median	KL	6	2	4	2	6
	SDindex	3	3	3	3	3
	Tau	3	4	3	3	3
Centroid	KL	6	2	2	3	6
	SDindex	3	3	3	3	3
	Tau	3	4	3	3	3
K-Means	KL	6	6	6	6	6
	SDindex	2	2	2	2	2
	Tau	2	3	2	2	2

In Sect. 4, we categorized the NPB kernels into 4 different categories due to two specific power values that show some strange behavior. However, since the optimal time and energy configurations are similar in both categories, the machine learning algorithms categorize them equally, as these methods have been proven statistically robust. These results justify the incorporation of ML algorithms as a methodology to determine the optimal number of power configurations for a given software and hardware. The most influential parameter is the choice of the validity index, where the KL, Tau and SD indices yield 3 different clusters. Note that selecting of 2 or 6 clusters is still a valid clusterization for the input NPB kernels, where the power configurations provided for 2 or 6 clusters could also satisfy the needs of the user.

5.2 Adding a new application into a cluster

In order to categorize our use-cases, and only for illustrative purposes, we have chosen the K-Means clustering using the Manhattan distance, and set the number of clusters to $k = 3$. Running the selected algorithm with the data obtained in the previous experiment (illustrated in Fig. 4 and 5) gives the results shown in Tables 4 and 5.

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

i5-6200U Categorization output					
Added		Pareto	Distance to centroid		
Kernel	K-means	Analysis	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	

For each individual kernel we performed the experiment using the other 7 NPB kernels as the training set and use the remaining kernel as the test set, so that we perform a k-fold cross validation. In order to add a new element (an application) to an existing categorization we have to perform these exact steps, which strengthens our proposal. Each row in both Tables contains the data related to each of the training and test sets, with the kernel used for testing referenced the *Added Kernel* column.

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 test kernel in bold. The column *Analysis* indicates the *Added Kernel* cluster from our analysis derived from the Pareto front in Sect. 4, for easy comparison with the conclusions extracted previously. Finally, the distance of the kernel to each cluster centroid is shown to better understand the data presented.

For better explanation, we will consider the first row in Table 4, the serial version of BT.W. After running the K-Means on 3 clusters using all the kernels except 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, indicate that the BT.W belongs to cluster number 1, in bold in our K-Means vector of membership. If we compare the results obtained with our manual analysis, we can observe that the automated procedure achieves the same results that we obtained using the heatmaps.

Table 5 Cluster membership comparative, NPB parallel version in Intel Xeon

Xeon gold 6230N categorization output					
Added		Pareto	Distance to centroid		
Kernel	K-Means	Analysis	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

For the i5-6200U case, the Table 4 shows exactly the same results for the Pareto analysis and the categorization algorithm for the majority of the kernels. *LU.W* when used as a test, switches from the cluster 1 to cluster 2. Another notable case, the *CG.A*, has the peculiarity of being the only member of cluster 3 in the Pareto Analysis, and the K-Means has a different categorization proposal using only 2 clusters. In this last case, we conclude that *CG.A* is not significantly distant from the other kernels when is added afterwards and categorized in one of the existing categories.

For the Intel Xeon parallel platform, 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. 5. Again we observe a case, DC.W, where the Pareto analysis provides a different number of clusters to the ML algorithm but, in practice, the efficiency obtained is not very distant. Nevertheless, each result in both tables is a valid solution, as our manual approximation, and each of the presented categorizations is 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 into the existing clusters to illustrate the performance of this methodology. For the case presented, the calculation the membership and number of clusters using the 7 kernels required 1.8 ± 0.8 ms in the Intel i5-6200U. On the other hand, the addition of a new element to the existing classification required 1.7 ± 0.5 ms, and had a straightforward implementation. This confirms that the process to find the parameters for the efficient execution is also efficient in terms of performance and energy efficiency.

Comparing the low computational effort required and the relatively low implementation difficulty to applying a clustering algorithm with the costly task of manually studying all the data, we have found that ML algorithms facilitate the process of obtaining different power configurations to optimize software execution in a target architecture and automate the data analysis to simplify the work required to apply our proposal. The initial categorization of the applications, could be done automatically with a small benchmark when installing the system and could also be integrated 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, into an existing category for future executions. Furthermore, when the volume of newly classified items increases significantly, a recalculation of the clusters can be executed to refine the existing classification of the applications.

6 Conclusion

We have analyzed the impact of power capping technologies in a given architecture, and proposed a methodology to categorize the feasible power configuration of a target application. As the energy behavior depends on both hardware and software, we have illustrated a use-case using the NPB kernels in a specific hardware. We extracted a Pareto front of solutions for execution time and energy consumption for each kernel. The analysis of the Pareto front allowed us to classify the different kernels in order to understand the power configurations in our system. This simplifies the optimization problem as we obtain an optimal power configuration per category, rahter than optimizing each application individually. Then, in order 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.

Because of the data-driven approach of Machine Learning, our methodology, when automated, will not yield perfect results in all cases. However, our experiments strongly suggest the automation capabilities are promising, and they illustrate how new applications can be added to an existing categorization so that previous knowledge can be reused for a better resource allocation.

In the near future we plan to extend our methodology to highly parallel systems and introduce it into power constrained environments where the power draw of the hardware is higher than the total available power. This will likely require the incorporation of some other Machine Learning techniques. Additionally, we are also interested 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 deal with the difficulties introduced by heterogeneous environments.

Author contributions These authors contributed equally to this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. 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).

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

Declarations

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

Ethical approval Not applicable.

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. 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
4. 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 arxiv:abs/1902.10810 (2019)
5. Lison, P.: An introduction to machine learning. Lang. Technol. Group (LTG) **35**, 1 (2015)
6. Hennig, C., Meila, M., Murtagh, F., Rocci, R.: Handbook of Cluster Analysis. CRC Press, New York (2015)
7. Karlsson, C.: Handbook of Research on Cluster Theory, vol. 1. Edward Elgar Publishing, Blekinge Institute of Technology, Cheltenham (2010)
8. 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)
9. 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
10. 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
11. Ahmed, K., Bull, J., Liu, J.: Contract-based demand response model for high performance computing systems. In: Chen, J., Yang, L. (eds.) ISPA/IUCC/BDCloud/SocialCom/SustainCom, pp. 580–589. IEEE, Melbourne (2018)
12. 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. Cybern. **49**(3), 1097–1109 (2019)
13. Mahboubi, H., Masoudimansour, W., Aghdam, A.G., Sayrafian-Pour, K.: An energy-efficient target-tracking strategy for mobile sensor networks. IEEE Trans. Cybern. **47**(2), 511–523 (2017)
14. Herbert, S., Marculescu, D.: Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In: Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED '07), pp. 38–43 (2007). https://doi.org/10.1145/1283780.1283790
15. Rauber, T., Rünger, G.: A scheduling selection process for energy-efficient task execution on dvfs processors. Concurr. Comput. **31**(19), 5043 (2019). https://doi.org/10.1002/cpe.5043
16. 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
17. 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)
18. 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)
19. 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. Comput. Sci. **29**(3), 197–210 (2014). https://doi.org/10.1007/s00450-013-0239-3
20. 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 International Parallel and Distributed Processing Symposium Wksh. PhD Forum, pp. 947–953 (2012). https://doi.org/10.1109/IPDPSW.2012.116

21. 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, November 14, 2016, pp. 39–45. IEEE Computer Society, Salt Lake City (2016). https://doi.org/10.1109/E2SC.2016.011
22. 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 Perform. Comput., pp. 394–408. Springer, Cham (2015)
23. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Nat. Acad. Sci. **115**(34), 8505–8510 (2018)
24. 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, Bloomington (2018)
25. Yager, K.: Autonomous experimentation as a paradigm for materials discovery. In: *Big Data and Extreme-Scale Computing Workshop* (2018)
26. 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
27. De Sensi, D., Torquati, M., Danelutto, M.: A reconfiguration algorithm for power-aware parallel applications. ACM Trans. Archit. Code Optim. **4**, 13 (2016). https://doi.org/10.1145/3004054
28. 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 **9**, 13 (2020). https://doi.org/10.3390/en13092409
29. 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. Microprocess. Microsyst. **46**, 136–148 (2016). https://doi.org/10.1016/j.micpro.2016.03.006
30. 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 (2012). https://doi.org/10.1145/2370816.2370821
31. Reddy, R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Trans. Comp. **99**, 1–11 (2017). https://doi.org/10.1109/TC.2017.2742513
32. 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, pp. 17–11713. IEEE / ACM, Dallas (2018). http://dl.acm.org/citation.cfm?id=3291679
33. ARM: ARM DynamIQ technology: power management. https://www.arm.com/technologies/dynamiq
34. Wysocki, R.J.: intel_pstate CPU Performance Scaling Driver. https://www.kernel.org/doc/html/v4.12/admin-guide/pm/intel_pstate.html
35. Rui, H.: amd-pstate CPU Performance Scaling Driver. https://docs.kernel.org/admin-guide/pm/amd-pstate.html
36. DELL: Controlling Processor C-State Usage in Linux. https://wiki.bu.ost.ch/infoportal/_media/embedded_systems/ethercat/controlling_processor_c-state_usage_in_linux_v1.1_nov2013.pdf. A Dell technical white paper describing the use of C-states with Linux operating systems (2013)
37. Wysocki, R.J.: intel_idle CPU Idle Time Management Driver. https://docs.kernel.org/admin-guide/pm/intel_idle.html
38. Devices, A.M.: HPC Tuning Guide for AMD EPYC Processors (2018). http://developer.amd.com/wp-content/resources/56420.pdf
39. Nvidia SMI documentation. https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf, Accessed Aug. 2020 (2016)
40. NVIDIA: Nvidia Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml
41. Corporation, I.: Intel® 64 and IA-32 Architectures Software Developer Manuals. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
42. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: memory power estimation and capping. In: Oklobdzija, V.G., Pangle, B., Chang, N., Shanbhag, N.R., Kim, C.H. (eds.) Proceedings of the 2010 International Symposium on Low Power Electronics and Design, 2010, pp. 189–194. ACM, Dallas (2010). https://doi.org/10.1145/1840845.1840883
43. Almeida, F., Pérez, V.B., Gonzalez, I., Cabrera, A., Giménez, D.: Analytical energy models for mpi communications on a sandy-bridge architecture. In: 42nd International Conference on Parallel Processing. ICPP, pp. 868–876. IEEE, Lyon (2013)
44. Google: 8402290 (2013). https://patentcenter.uspto.gov/applications/12263421
45. Cabrera, A., Almeida, F., Arteaga, J., Blanco, V.: Measuring energy consumption using eml (energy measurement library). Comput. Sci. **30**(2), 135–143 (2014). https://doi.org/10.1007/s00450-014-0269-5
46. 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
47. Na, S., Xumin, L., Yong, G.: Research on k-means clustering algorithm: An improved k-means clustering algorithm. In: 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, pp. 63–67 (2010). IEEE
48. Nugraha, A., Perdana, M.A.H., Santoso, H.A., Zeniarja, J., Luthfiarta, A., Pertiwi, A.: Determining the senior high school major using agglomerative hierarchial clustering algorithm. In: 2018 International Seminar on Application for Technology of Information and Communication, pp. 225–228 (2018). IEEE
49. Kavitha, V., Punithavalli, M.: Comparing odac and hierarchical algorithm using time series data streams. In: 2010 IEEE International Conference on Computational Intelligence and Computing Research, pp. 1–4 (2010). IEEE
50. Anderberg, M.R.: Cluster Analysis for Applications: Probability and Mathematical Statistics: a Series of Monographs and Textbooks, vol. 19. Academic press, U. Michigan (2014)
51. Davies, D.L.: A cluster separation measure. IEEE Trans. Pattern Anal. Mach. Intell **2**, 224–227 (1979)
52. Handl, J., Knowles, J., Kell, D.B.: Computational cluster validation in post-genomic data analysis. Bioinformatics **21**(15), 3201–3212 (2005)
53. Charrad, M., Ghazzali, N., Boiteau, V.: Nbclust: an r package for determining the relevant number of clusters in a data set. J. Stat. Softw. **61**(6), 1–36 (2014). https://doi.org/10.18637/jss.v061.i06

Alberto Cabrera received the B.A. and M.S. degrees in computer sciences from Universidad de La Laguna, Spain, in 2010 and 2013 respectively. He is currently a Ph.D. Student at Universidad de La

Laguna. His research interests are primarily in parallel system analysis and prediction, parallel computing, heterogeneous computing and energy efficiency in high-performance systems.

Francisco Almeida received an M.Sc. in mathematics and a Ph.D. in computer science from the University of La Laguna, Spain, in 1989, 1992, and 1996, respectively. He is a Professor at the Department of Computer Science and Systems, University of La Laguna. His research interests are primarily in parallel computing, parallel algorithms for optimization problems, parallel system performance analysis and prediction, skeleton tools for parallel programming, and web services for HPC and Grid technology.

Dagoberto Castellanos-Nieves received his B.A. and M.Sc. degrees in Mechanical Engineering from the University of Holguin, Cuba, and his Ph.D. from the University of Murcia, Spain. He is an Associate Professor at the Department of Computer Engineering and Systems at the Universidad de La Laguna and is collaborating on various research projects. He has published over thirty articles in journals, conferences, and book chapters and is the author or coauthor of several books.

Ariel Oleksiak received his M.Sc. degree in Computer Science in 2001, and Ph.D. in Computer Science in 2009 from Poznań University of Technology. In 2019, he obtained a D.Sc. degree. His research interests include the efficiency of data centres and clouds, intelligent resource management and data processing methods, and computer simulations. He has contributed to many research projects. Among others, he coordinated the EU-funded CoolEmAll and M2DC projects dealing with the energy efficiency of data centres.

Vicente Blanco received the B.A. and the M.S. degrees in Physics from the University of Santiago de Compostela in 1992 and 1993, respectively. He obtained his PhD in Physics in 2002. He is an Associate Professor in the Department of Computer Science and Systems at the University of La Laguna. His research interests are performance evaluation, prediction and visualization of parallel codes, parallel algorithms for dense and sparse algebra, Grid and GPGPU technology, and Energy-aware algorithms/systems.