Journal of Scheduling (2021) 24:489–505
https://doi.org/10.1007/s10951-020-00649-4

An approach to reduce energy consumption and performance losses on heterogeneous servers using power capping

Tomasz Ciesielczyk[2] · Alberto Cabrera[1] · Ariel Oleksiak[2] · Wojciech Piątek[2] · Grzegorz Waligóra[3] · Francisco Almeida[1] · Vicente Blanco[1]

Published online: 20 May 2020

Abstract
Rapid growth of demand for remote computational power, along with high energy costs and infrastructure limits, has led to treating power usage as a primary constraint in data centers. Especially, recent challenges related to development of exascale systems or autonomous edge systems require tools that will limit power usage and energy consumption. This paper presents a power capping method that allows operators to quickly adjust the power usage to external conditions and, at the same time, to reduce energy consumption and negative impact on performance of applications. We propose an optimization model and both heuristic and exact methods to solve this problem. We present an evaluation of power capping approaches supported by results of application benchmarks and experiments performed on new heterogeneous servers.

Keywords Power capping · Energy-aware computing · Energy efficiency

1 Introduction

Recent rapid growth of demand for remote computational power, storage, and network bandwidth has led to fast development of data centers and, consequently, significant increases in energy consumption by these systems. A large portion of global IT energy consumption is used by data centers, reaching around 1.5–3% of energy, depending on regions.

The need for highly efficient and low power usage is essential for emerging systems such as high-performance computing (HPC) exascale systems and edge computing. The former requires an extremely large power supply that makes building such systems unfeasible or very expensive. Edge computing requires low power and dynamic management of systems, as they may have limited power availability and a need for fast autonomous reaction in case of changing conditions in remote locations. In all these cases, the power availability depends on the installation's electrical infrastructure and cooling system.

To address these requirements, new server architectures have emerged that allow achieving higher densities of servers, lower energy and cost overhead, and better adaptation to specific applications. One of the approaches for providing such a solution is the microserver platform supplied by the M2DC project Oleksiak et al. (2017). M2DC systems integrate heterogeneous hardware such as x86 CPUs, AMR64 CPUs, GPUs, and field-programmable gate array (FPGA) chips as small form factor microservers. An M2DC system can host up to 27 high performance or 144 low

✉ Ariel Oleksiak
ariel@man.poznan.pl

Tomasz Ciesielczyk
tomaszc@man.poznan.pl

Alberto Cabrera
Alberto.Cabrera@ull.es

Wojciech Piątek
piatek@man.poznan.pl

Grzegorz Waligóra
grzegorz.waligora@cs.put.poznan.pl

Francisco Almeida
falmeida@ull.es

Vicente Blanco
Vicente.Blanco@ull.es

[1] HPC Group. Escuela Superior de Ingeniería y Tecnología, Universidad de La Laguna, 38270 San Cristóbal de La Laguna, Tenerife, Spain

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

[3] Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznań, Poland

power nodes. On top of this hardware platform, there is a middleware layer responsible for integration and access by management systems and applications. Specific configurations of the platform with preinstalled software libraries and tools, or appliances, are prepared for given classes of applications such as image processing, IoT data analytics, cloud computing or HPC. The platform can reduce facility costs even up to almost 70% due to lower space costs, lower redundancy of components, and less air volume to cool down. To enable accurate control of such a high density, complex, and reconfigurable platform, the heterogeneous microserver system comes along with intelligent power management techniques. These methods deal with heterogeneity and workload specifics during power capping, and model their impact on costs across the whole data center. A preliminary analysis and approach to power capping and reducing energy costs in data centers by the use of the M2DC platform was presented in Oleksiak et al. (2018).

To enable appropriate management of such a platform, the power capping method must meet key requirements such as supporting heterogeneous nodes, automated adaptation of models in the case of changes in the microserver platform configuration, fast reaction to changing conditions, while taking into account application profiles (and impact of power changes on application performance), and priorities of nodes. In this paper, we propose a power capping method meeting these challenges. Using this approach, a trade-off between lower performance of power capped hardware, changes in efficiency of applications, and node priorities is found. The presented approach to server architecture and management concentrates on meeting the challenges of the high-density M2DC microserver platform and its specific appliances, but can also help with adaptation to specific conditions and autonomous management for other more typical server architectures and in the emerging case of edge systems.

This work is structured as follows. Related works in the field of power capping methods and energy efficiency are described in Sect. 2. The considered problem of power distribution in a heterogeneous microserver appliance is formulated in Sect. 3. Section 4 describes the overall proposed power capping method, including applied algorithms and basic power capping algorithms used as a reference in the evaluation. Section 5 presents the methodology for building server power usage models and applying power capping for target hardware and applications. Benchmarks used for tuning the models and the impact of power capping on energy efficiency and performance of applications are studied. Section 6 presents the evaluation of the proposed power capping method. Based on computational experiments, we demonstrate the trade-off between performance and efficiency when our methodology is applied. Finally, Sect. 7 contains conclusions.

2 Related work

As limiting power usage and total energy consumption are essential for development of large systems, many approaches for improving efficiency have been proposed. For instance, HEROS Guzek et al. (2015) introduces a load balancing algorithm for energy-efficient resource allocation in heterogeneous systems. PVA Takouna et al. (2013), peer VMs aggregation, proposes to enable dynamic discovery of communication patterns and reschedule VMs based on the acquired knowledge with virtual machine migrations. Several heuristic approaches (based on evolutionary computation) for resource management in cloud computing have been explored in Guzek et al. (2015). As an example, it is worth mentioning a work presented by Jeyarani et al. (2012). The authors adopt self-adaptive particle swarm optimization to address the VM placement problem. They start with finding the proper solution from the performance point of view (choosing candidates that satisfy the performance constraints) and then select the one with the lowest energy consumption. Energy-aware allocation heuristics have also been proposed for a complete data center Beloglazov et al. (2012). Energy-aware dynamic load balancing was proposed in Cabrera et al. (2017), and job power consumption in HPC in Borghesi et al. (2016).

Limiting required power for computations is an important specific problem caused by constraints on available power in a given computing system. Many approaches to power capping have been studied up to now. In Liu et al. (2016), the authors present an optimization approach for power capping called *FastCap*. The proposed solution benefits from both CPU and memory DVFS (Dynamic Voltage and Frequency Scaling) capabilities. The idea of FastCap is to maximize the system performance within the given power budget together with a fair allocation of computational power to running applications. The algorithm considers the total time of one memory access for the particular core as a performance metric. It also utilizes power models of processors and memory in order to find optimal settings. Finally, frequency selection is based on the collected performance counters. Another approach—ALPACA, described in Krzywda et al. (2018), optimizes power settings of the system from the application point of view. The main goal is to minimize quality of service (QoS) violations and electricity costs while fulfilling power budget constraints for servers. ALPACA allows specifying performance metrics, thresholds, performance degradation, and cost models for each application and takes these characteristics into account while calculating QoS degradation. Then, it harnesses RAPL (Running Average Power Limit) and a “cgroups” (control groups) mechanism to control the application power draw. In order to use ALPACA, prior application benchmarking is required that collects measurements from real hardware and creates corresponding power models.

An extension to power capping techniques based on frequency scaling is presented in Bhattacharya et al. (2012). The authors propose supporting existing solutions for power capping with an admission control mechanism. They claim that this combined approach can address the fast reaction requirements engendered by rapid power peaks in data centers. The idea behind admission control is to reduce power demand by reducing the amount of performed work, and thus the amount of used computational resources. The presented solution cannot be applied as a standalone power capping mechanism, but can improve the effectiveness and performance of existing ones. It is suggested to implement admission control on the application or the load balancer level. In Krzywaniak et al. (2018), the authors discuss the trade-offs between performance and energy consumption while using power capping. They evaluate the behavior of four hardware architectures and three applications running on them. The authors measure power consumption for various power settings (enforced using RAPL) and their corresponding execution times. Based on that, they identify the most efficient power limit settings that result in the highest energy savings. The presented work could be a good starting point for proposing a power capping approach that takes into account application profiles while reducing the power for particular servers.

Multiple papers also present how existing tools and libraries can be applied to power capping techniques. The use of the Intel RAPL library for power management is described in Rountree et al. (2013), for analyzing energy/performance trade-offs with power capping for parallel applications on modern multi- and many-core processors in Krzywaniak et al. (2018), or used with Intel Power Governor in Tiwari et al. (2015).

Some papers have studied power capping applied to specific applications, for example Fukazawa et al. (2014). An interesting analysis of the impact, in terms of performance and energy usage, that power caps have on a system running scientific applications is described in Haidar et al. (2019). The authors explore how different power caps affect the performance of numerical algorithms with different computational profiles in order to characterize the types of algorithms that benefit most from power management schemes.

A comparison of hardware, software, and hybrid techniques is summarized in Zhang and Hoffmann (2016).

The intelligent power capping method proposed in this paper follows experiences from the presented state of the art and adds a contribution that is an integration of an approach to operating a system subject to power usage constraints (caps) with energy saving features. The method combines a greedy heuristic that quickly adapts to the power thresholds with an energy optimization algorithm. The method uses dynamic priorities and appropriate models to minimize potential performance losses and reliability issues, supports heterogeneous nodes, and adapts power capping decisions to characteristics of applications in order to optimize their efficiency. An example of how limits of the power usage may help to reduce energy costs of cooling and infrastructure for data center using the capacity management was presented in Da Costa et al. (2015). A preliminary study and a proposal of the power capping method for the M2DC heterogeneous microserver platform is described in Oleksiak et al. (2018); the platform to which we apply the power capping method is presented in Oleksiak et al. (2017).

3 Problem formulation

Let us denote a set of nodes in a heterogeneous microserver appliance as N, and a single node as n_i, where:

$$n_i \in N,\ i = 0, 1, \ldots, |N| - 1. \tag{1}$$

A node's significance in the appliance can be determined by a priority r_i, which is assigned to each node $n_i \in N$. The main aim of using the priorities is to enable user to maintain critical services under high performance settings. Unless required by the power limit imposed by the user, the highest possible priority ($r_i = 1$) assignable to a node results in no power capping actions applied to the node. Power actions are applied to a node with the highest priority only if available power actions for other nodes with lower priorities ($r_i < 1$) are not sufficient for the defined power limit. The lowest priority ($r_i = 0$) designates a set of nodes for which power usage is limited before other nodes in the power reduction process. Definition of a node's priority is given by Formula (2):

$$\begin{cases} r_i = 1 & \textit{the highest priority} \\ r_i \in (0, 1) & \textit{standard priority} \\ r_i = 0 & \textit{the lowest priority} \end{cases} \tag{2}$$

Each node n_i in the heterogeneous appliance has its own predefined set of available power settings, and exactly one of those power settings is applied to a given node. Exemplary power setting policies are: DVFS, RAPL (Intel's power settings providing additional power management) or GPU settings (e.g., Nvidia-smi tool). The set of available power settings is limited (e.g., not all frequencies are available) in order to reduce the problem complexity. The set of all available power settings of all nodes n_i, $i = 0, 1, \ldots, |N| - 1$, is denoted by P, and the set of available power settings available for node n_i is denoted by C_i, where:

$$P = \{C_0 \cup C_1 \cup \ldots \cup C_{|N|-1}\}. \tag{3}$$

The nodes in a heterogeneous platform differ in terms of energy consumption and architecture; therefore, the number of available power settings may vary for particular nodes. A single power setting j applied to node n_i is denoted by c_{ij}, i.e.,

$$c_{ij} \in C_i,\ \ j = 0, 1, \ldots, y_i, \tag{4}$$

where $y_i = |C_i| - 1$ is the node's default power setting corresponding to its state without power limit and, at the same time, with the highest performance. Thus, the default power setting applied to the node n_i is c_{iy_i}. On the other hand, the power setting which determines the node's maximum power savings and also its lowest performance is denoted as c_{i0}. Each power setting c_{ij} has a corresponding performance value determined by performed benchmark computations. A node's measured computational performance is assumed, in general, to be inversely proportional to the time needed to complete a predefined, repetitive benchmark computation. The time necessary to finish a predefined benchmark by node n_i with power setting c_{ij} is defined as $time_{ij}$ and, consequently, the node's performance $perf_{ij}$ with this power setting is defined as:

$$perf_{ij} = \frac{1}{time_{ij}}. \tag{5}$$

The normalized performance value of node n_i for power setting c_{ij} is denoted as ω_{ij}, such that $\omega_{ij} \in \langle 0, 1\rangle$, which is a ratio between the relevant performance and the maximum performance of the node with the default power setting c_{iy_i}:

$$\omega_{ij} = \frac{perf_{ij}}{perf_{iy_i}} = \frac{time_{iy_i}}{time_{ij}}, \tag{6}$$

where $time_{iy_i}$ is, of course, the minimum time needed by node n_i to complete the benchmark computation, corresponding to its highest performance $perf_{iy_i}$. The current utilization ratio of node n_i with the power setting c_{ij} is denoted by u_{ij}, where:

$$u_{ij} \in \langle 0\%, 100\%\rangle\,. \tag{7}$$

The utilization value is used to estimate the power usage of a node. According to the power usage data analyzed at the Google data center Fan et al. (2007), power consumption increases linearly with CPU utilization. Therefore, in this research it is assumed that power usage is linearly proportional to utilization. The power usage for node n_i with power setting c_{ij} is denoted by p_{ij}, and defined by the following formula:

$$p_{ij} = (pmax_{ij} - pidle_{ij})u_{ij} + pidle_{ij}, \tag{8}$$

in which $pmax_{ij}$ and $pidle_{ij}$ are the maximum power usage (i.e., power usage with 100% utilization) and the power usage within the idle state (i.e., with utilization equal to 0%) for node n_i with power setting c_{ij}, respectively. Nodes which are shut down (their power usage is equal to 0) are not taken into consideration in the power optimization process; therefore, the power usage of each node is assumed to be positive:

$$\underset{n_i \in N}{\forall}\ \underset{c_{ij} \in C_i}{\forall}\ p_{ij} > 0. \tag{9}$$

The utilization value of node n_i with power setting c_{ij} can be estimated in two ways (Eqs. (10) and (11)):

1. When a node's active time $active_{ij}$ (the time when node's CPU is not in the idle state) and idle time $idle_{ij}$ (the time when node's CPU is in the idle state) are known:

$$u_{ij} = \frac{\sum active_{ij}}{\sum active_{ij} + \sum idle_{ij}} * 100\%. \tag{10}$$

2. When a node's current power usage is known (see Eq. (8)):

$$u_{ij} = \frac{p_{ij} - pidle_{ij}}{pmax_{ij} - pidle_{ij}}. \tag{11}$$

It is assumed that the change of current power setting applied to node n_i has some influence on its utilization. The predicted utilization of a node is required later to estimate power usage after a power setting change. For the purpose of this research, we describe three basic properties related to the utilization change estimation during a power setting change on node n_i from a power setting c_{ij} to power setting c_{ik} ($c_{ij} \rightarrow c_{ik},\ j \neq k$). These properties do not take into consideration the application execution state (e.g., an application may finish computations and start a data synchronization process which is much less power consuming):

1. Property related to performance decrease:

$$\begin{aligned} &\omega_{ik} < \omega_{ij} \\ &u_{ik} > u_{ij} \rightarrow u_{ik} \in \big(u_{ij}, 100\big\rangle\,. \end{aligned} \tag{12}$$

   This property claims that the utilization of a node grows with a performance decrease.
2. Property related to performance increase:

$$\begin{aligned} &\omega_{ik} > \omega_{ij} \\ &u_{ik} < u_{ij} \rightarrow u_{ik} \in \big\langle 0, u_{ij}\big)\,. \end{aligned} \tag{13}$$

   This property maintains that when the performance of a node increases, its utilization goes down.
3. Property related to equal performance:

$$\omega_{ik} = \omega_{ij}$$

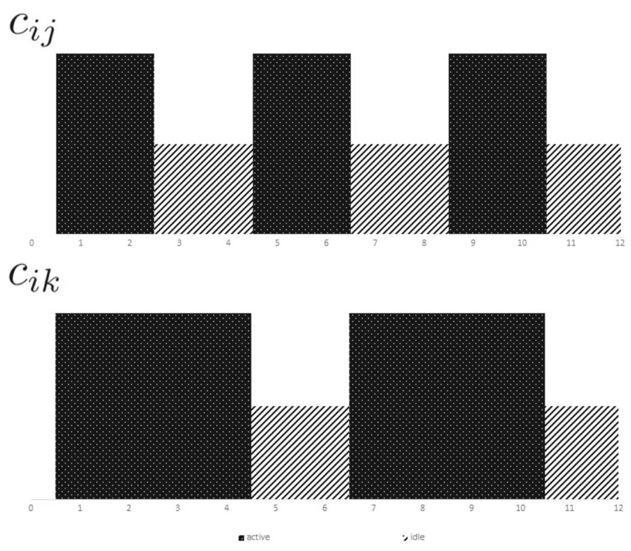


Fig. 1 Power settings activity and idle time comparison

$$u_{ik} = u_{ij}. \tag{14}$$

In the case, when the performance of a node does not change, its utilization remains the same.

Exemplary active times and idle state times for node n_i for two different power settings are presented in Fig. 1. The ratio between the active time for power setting c_{ij} and the active time for power setting c_{ik} is assumed to be linearly and inversely proportional to the corresponding performance (Eq. (15)):

$$\frac{active_{ik}}{active_{ij}} = \frac{\omega_{ij}}{\omega_{ik}}. \tag{15}$$

In order to simplify the power distribution model and to enable us to easily predict and estimate the utilization after changing the current power setting, it is assumed that the idle time for computations performed on node n_i is equal for all power settings (Eq. (16)). This assumption is based on a presumption that operations executed during the idle state (e.g., operations such as memory synchronization, data download, await for client request, etc.) are independent of power setting changes.

$$\underset{n_i \in N}{\forall} \underset{c_{ij} \in C_i}{\forall} idle_{ij} = idle_i. \tag{16}$$

The performance ratio of node n_i when changing from power setting c_{ij} to power setting c_{ik} ($c_{ij} \rightarrow c_{ik},\ j \neq k$) is defined as:

$$\tau = \frac{\omega_{ik}}{\omega_{ij}}. \tag{17}$$

Taking into account Eq. (15), we can also write:

$$\tau = \frac{active_{ij}}{active_{ik}}. \tag{18}$$

Let us now analyze the impact of a power setting change from c_{ij} to c_{ik}, where $j \neq k$ on the utilization of node n_i. Let us start with writing Eq. (10) for power settings c_{ij} and c_{ik}:

$$u_{ij} = \frac{\sum active_{ij}}{\sum active_{ij} + \sum idle_{ij}} * 100\% \tag{19}$$

$$u_{ik} = \frac{\sum active_{ik}}{\sum active_{ik} + \sum idle_{ik}} * 100\%. \tag{20}$$

Then denote:

$$a_j = \sum active_{ij} \tag{21}$$

$$a_k = \sum active_{ik} \tag{22}$$

$$b = \sum idle_{ij} = \sum idle_{ik} = idle_i, \tag{23}$$

and by substitution of (21) and (23) in (19), as well as (22) and (23) in (20) we obtain:

$$u_{ij} = \frac{a_j}{a_j + b} * 100\% \tag{24}$$

$$u_{ik} = \frac{a_k}{a_k + b} * 100\%. \tag{25}$$

Deriving b from Eq. (24), we get:

$$b = \frac{a_j * 100\%}{u_{ij}} - a_j. \tag{26}$$

Next, by using (18) in Eq. (22) we can write:

$$a_k = \sum \frac{active_{ij}}{\tau} = \frac{a_j}{\tau}. \tag{27}$$

Finally, by substituting (26) and (27) into (25) we obtain:

$$u_{ik} = \frac{1}{1 + \tau(\frac{100\%}{u_{ij}} - 1)} * 100\%. \tag{28}$$

Equation (28) relates the current utilization of node n_i to its previous utilization when changing from power setting c_{ij} to power setting c_{ik}. An example of the utilization estimation according to Eq. (28) is presented in Formula (29). The calculations for this example are related to active and idle times of a node for two different power settings ($c_{ij} \rightarrow c_{ik}$, $\omega_{ik} = \frac{1}{2}\omega_{ij}$), presented in Fig. 1 ($u_{ij} = 50\%$).

$$\tau = \frac{\omega_{ik}}{\omega_{ij}} = \frac{0.5\omega_{ij}}{\omega_{ij}} = 0.5$$
$$u_{ik} = \frac{1}{1 + 0.5(\frac{100\%}{50\%} - 1)} * 100\% \approx 66.67\%. \tag{29}$$

In order to determine whether the power setting c_{ij} is applied to node n_i, a binary variable s_{ij} is used, defined as:

$$\begin{cases} s_{ij} = 1 & if\ c_{ij} \text{ is applied to } n_i \\ s_{ij} = 0 & \text{otherwise.} \end{cases}$$

For each node n_i there can be only one power setting applied to it at a time (Eq. (30)):

$$\sum_{j=0}^{y_i} s_{ij} = 1,\ i = 0, 1, \ldots, |N| - 1. \tag{30}$$

The value which determines the efficiency (in terms of power usage and energy savings) of an applied power setting is the efficiency value. The efficiency for power setting c_{ij} applied to node n_i is denoted as eff_{ij}, and it is inversely proportional to energy (the product of power and time) used to execute the predefined benchmark computation:

$$eff_{ij} = \frac{1}{energy_{ij}} = \frac{1}{p_{ij}time_{ij}}. \tag{31}$$

The normalized value of the efficiency ratio for power setting c_{ij} is denoted as $\triangle eff_{ij}$, and the formula of this ratio is defined by Eq. (32):

$$\triangle eff_{ij} = \frac{eff_{ij}}{eff_{iy_i}}. \tag{32}$$

Taking into account Eqs. (31) and (6), we can also write:

$$\triangle eff_{ij} = \frac{p_{iy_i}\omega_{ij}}{p_{ij}}. \tag{33}$$

Moreover, considering Formulas (6) and (9), it is easy to see that:

$$\triangle eff_{ij} > 0. \tag{34}$$

In order to evaluate the power setting configuration in terms of emission of carbon dioxide and reducing the costs of running an appliance, it is necessary to estimate the power savings for each configuration change. Let us assume that the initial time to finish jobs commissioned to each node is $time_{ij}$, and that each node after finishing all commissioned jobs is turned off (its power usage after completing the jobs is equal to 0). Under these assumptions, the power savings ratio for each node is a difference between energy used by node n_i with applied power setting c_{ij} ($energy_{ij}$) and energy after applying a new power setting c_{ik} ($energy_{ik}$). Let us define the power savings on node n_i as $savings_i$:

$$savings_i = energy_{ij} - energy_{ik}. \tag{35}$$

Taking into account Eqs. (31) and (6), it is possible to obtain the following savings formula for node n_i:

$$savings_i = t(p_{ij} - p_{ik} * \frac{\omega_{ij}}{\omega_{ik}}), \tag{36}$$

where t is a fixed time or the time between power setting changes. The total power savings of a current configuration of nodes can be designated by Eq. (37):

$$savings = \sum_{i=0}^{|N|-1} savings_i. \tag{37}$$

The evaluation score of the state of node n_i with power setting c_{ij} is denoted by $eval_{ij}$. The higher the evaluation value, the more profitable the state of the node. The impact of the provided node's priority r_i is defined as ρ_i:

$$\rho_i = \frac{r_i}{1 - r_i}. \tag{38}$$

The r_i parameter determines the end user's computation significance. The higher r_i, the more crucial is the node for the user. The trade-off between performance and efficiency provided by the appliance's administrator is defined as α, where $\alpha \in \langle 0, 1 \rangle$. The greater the value of the α parameter, the higher the impact its performance has on the state's evaluation. On the other hand, the smaller the value of α, the higher the impact of its efficiency. The way of calculating the state evaluation is given by Formula (39):

$$\begin{cases} eval_{ij} = p_{iy_i} * \rho_i(\omega_{ij})^{\alpha}(\triangle eff_{ij})^{1-\alpha} & r \in (0, 1) \\ eval_{ij} = p_{iy_i} * (\omega_{ij})^{\alpha}(\triangle eff_{ij})^{1-\alpha} & r = 0 \vee 1. \end{cases} \tag{39}$$

The overall evaluation of the appliance's state is defined by Formula (40):

$$\sum_{i=0}^{|N|-1} \sum_{j=0}^{y_i} eval_{ij}s_{ij}. \tag{40}$$

Let us stress that the overall evaluation does not take into account the previous power settings configuration; it is based only on the currently applied power settings and the future workload state. The new power settings configuration refers to the current system state. If the workload changes, the evaluation may also change and the new best power settings allocation may be different. Taking the above into account, we can formulate the following mathematical programming problem for solving the defined power capping problem: *maximize*

$$\sum_{i=0}^{|N|-1} \sum_{j=0}^{y_i} eval_{ij} s_{ij} \tag{41}$$

subject to

$$\sum_{i=0}^{|N|-1} \sum_{j=0}^{y_i} p_{ij} s_{ij} \leq power\_limit. \tag{42}$$

The formulated problem is, obviously, nonlinear, and taking into account the binary character of variables s_{ij}, we obtain a binary nonlinear programming (BNLP) problem. In this problem formulation, the overall evaluation function (41) is maximized subject to constraint (42), assuring that the imposed power limit is not exceeded.

4 Power management procedure

4.1 Power management background

The introduced power distribution model is utilized within two power management procedures, namely the power capping procedure (PCM) and the energy saving procedure (ESM). The main difference between them is that PCM's main aim is keeping the total power usage below the defined power limit provided by the appliance's administrator or other systems—PCM is triggered only if the defined power limit is exceeded. Furthermore, if current power usage does not satisfy the constraint defined in Formula (43), the PCM performs intrusive actions. The available intrusive actions consist of actions that may heavily influence the current workload, i.e., may result not only in the job's delay (performance reduction) but also in computation progress loss, the job's rejection, service unavailability, etc. Exemplary intrusive actions are suspend and shutdown.

$$\sum_{i=0}^{|N|-1} p_{i0} \leq power\_limit. \tag{43}$$

On the other hand, the ESM also takes into consideration the provided power limit, but its operation is based only on non-intrusive power actions—performance reduction should be the only negative consequence. The ESM's main aim is to optimize current workload in terms of energy efficiency and performance—the trade-off between the two is defined by the

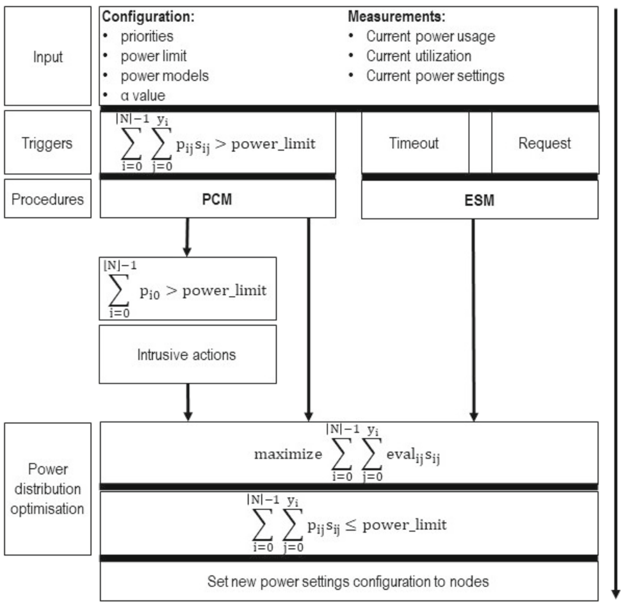

Input
Configuration:
• priorities
• power limit
• power models
• α value
Measurements:
• Current power usage
• Current utilization
• Current power settings
Triggers
$\sum_{i=0}^{|N|-1}\sum_{j=0}^{y_i} p_{ij}s_{ij} > power\_limit$
Timeout
Request
Procedures
PCM
ESM
$\sum_{i=0}^{|N|-1} p_{i0} > power\_limit$
Intrusive actions
Power distribution optimisation
maximize $\sum_{i=0}^{|N|-1}\sum_{j=0}^{y_i} eval_{ij}s_{ij}$
$\sum_{i=0}^{|N|-1}\sum_{j=0}^{y_i} p_{ij}s_{ij} \leq power\_limit$
Set new power settings configuration to nodes

Fig. 2 Power management concept

α parameter (39) set by the administrator. Unlike the PCM, which should provide a new power configuration as soon as possible due to the importance of the power limit, the ESM can utilize exact and time-consuming methods. The ESM is triggered periodically after a defined timeout or directly on the user's demand. The general power management idea is presented in Fig. 2. Power models provided as input to power management procedures are used in the power usage and performance estimation for each power setting c_{ij}. The detailed PCM and ESM approaches are presented below as Algorithms 1 and 2, respectively. The architecture and implementation of PCM and ESM were introduced formerly in the Resource and Thermal Management Module developed within the M2DC project Oleksiak et al. (2017).

4.2 Power distribution algorithms

We propose two algorithms in our method, namely the greedy algorithm (Algorithm 5) and the exact optimization algorithm using a MILP solver (Algorithm 6). The implementation of the MILP solver has been delivered by the ojAlgo http://ojalgo.org/ library. However, since the PCM should react to the load and power usage change as soon as possible, the algorithm using the MILP solver is not recommended for this purpose—the expected execution time of the optimization procedure for 2000 states (200 servers with 10 power states) is approximately 30 s. Therefore, we use the greedy heuristic for PCM and the more time-consuming optimization for ESM. The description of variables and functions used in the algorithms is presented in Table 1.

Algorithm 1 Power capping procedure

```
power_capping_procedure:
  input: $N_{active\_nodes}, N_{unused\_nodes}, power\_limit, current\_power$
  output: set of power actions
  $suspend\_savings :=$ max_suspend_savings($N_{unused\_nodes}$)
  $power\_budget := power\_limit - current\_power$
  if $suspend\_savings + power\_budget > 0$ then
    return suspend_actions($N_{unused\_nodes}, power\_budget$)
  $max\_savings :=$ estimate_power_savings($\bigcup_{n_i \in N_{active\_nodes}} c_{i0}$)
  if $max\_savings + suspend\_savings + power\_budget > 0$ then
    return suspend_actions($N_{unused\_nodes}$) $\cup$ power_distribution_algorithm($N_{active\_nodes}$, $power\_budget - suspend\_savings$)
  while $max\_savings + power\_budget + suspend\_savings < 0$ and $N_{active\_nodes} \neq \emptyset$ do
    $node :=$ least_important($N_{active\_nodes}$)
    $N_{active\_nodes} := N_{active\_nodes} \setminus node$
    $N_{unused\_nodes} := N_{unused\_nodes} \cup node$
    $suspend\_savings :=$ max_suspend_savings($N_{unused\_nodes}$)
    $max\_savings :=$ estimate_power_savings($\bigcup_{n_i \in N_{active\_nodes}} c_{i0}$)
  return suspend_actions($N_{unused\_nodes}$) $\cup$ power_distribution_algorithm($N_{active\_nodes}$, $power\_budget + suspend\_savings$)
```

Algorithm 2 Energy efficiency procedure

```
energy_efficiency_procedure:
  input: $N_{active\_nodes}, N_{unused\_nodes}, power\_limit, current\_power$
  output: set of power actions
  $suspend\_savings :=$ max_suspend_savings($N_{unused\_nodes}$)
  $power\_budget := power\_limit - current\_power$
  if $suspend\_savings + power\_budget > 0$ then
    return suspend_actions($N_{unused\_nodes}$, $power\_budget$) $\cup$ power_distribution_algorithm($N_{active\_nodes}$, $power\_budget + suspend\_savings$)
```

In order to evaluate results of the proposed method, we also define two simple power capping algorithms as references for comparison. These reference algorithms are denoted as random (Algorithm 3) and simple (Algorithm 4).

5 Impact of power capping on performance and efficiency

In this section, we use different hardware configurations in order to illustrate a methodology to determine the performance and the energy efficiency for a given application. We center our work around the usage of the power capping technologies, such as those provided by Intel and Nvidia drivers. While DVFS has been a frequently used technique for improving energy efficiency and reducing power consumption at lower levels, the latest technologies allow us to specify a power budget in watts, greatly improving the usability and reflecting the need for a power budget.

Intel, in the architectures used in this work, allows us to set various power budgets in their processors. These limits can be applied to the cores, the uncore, the dram, to each package, and, depending on the architecture, to the whole processor. Nvidia, on the other hand, allows us to set a power

Table 1 Definitions of variables and functions

Variable/function name	Description
Power_limit	Maximum power usage level defined by the user.
Current_power	Current power usage of all appliance components (current total power usage).
Set of power actions	The power actions set determines actions to be applied to nodes after the optimization process. This set consists of power actions which may influence the node's performance (power settings), suspend it or shut it down.
Max_suspend_savings	Function which estimates power savings for given set of nodes.
Suspend_actions	Function which returns the minimum required set of suspend actions for a given set of nodes and power budget. The minimum suspend actions set is determined by the power budget which should be a non-=negative value.
Estimate_power_savings	Function which estimates power savings achieved by applying provided power settings set.
N_{active_nodes}	Set of nodes which are executing committed jobs. ($N_{active\_nodes} \cup N_{unused\_nodes} = N$)
N_{unused_nodes}	Set of nodes in idle state without any committed jobs.
Order_power_settings	Function which orders nodes' available power settings by evaluation value ($eval_{ij}$).
Power_change	Function which calculates power usage change after applying a provided Power setting argument.
Least_important	Function which designates the least important node within the provided set of nodes. The node's importance is determined by the node's priority r_i and its current utilization (the lower utilization the lower importance).
Find_best_power_allocation	Function responsible for optimizing power settings allocation of the provided set of available power settings related to nodes. This function utilizes the ojAlgo expression based model with MILP http://ojalgo.org/.
Power_distribution_algorithm	Function which determines power settings for the current system state and power budget.

Algorithm 3 Random power distribution algorithm

```
random_power_distribution:
  input: $N_{active\_nodes}$, $power\_budget$
  output: $C''$ $\rightarrow$set of power actions
  $C'' := \emptyset$
  $savings :=$ estimate_power_savings($\bigcup_{n_i \in N_{active\_nodes}} c_{i0}$)
  $power\_budget := power\_budget + savings$
  $C' := \bigcup_{n_i \in N_{active\_nodes}, c_{ij} \in C_i} c_{ij}$
  shuffle($C'$)
  $N' := N_{active\_nodes}$
  for each $c_{ij}$ in $C'$ do
    if $n_i \cap N' \neq \emptyset$ and power_change($c_{ij}$) $< power\_budget$ then
      $C'' := C'' \cup c_{ij}$
      $power\_budget := power\_budget -$ power_change($c_{ij}$)
      $N' := N' \setminus n_i$
  $C'' := C'' \cup \bigcup_{n_i \in N_{active\_nodes} \setminus N'} c_{i0}$
  return $C''$
```

Algorithm 4 Simple power distribution algorithm

```
simple_power_distribution:
  input: $N_{active\_nodes}$, $power\_budget$
  output: $C''$ $\rightarrow$set of power actions
  $i := 0$
  $N' := N_{active\_nodes}$
  while $i < |N'|$ and $power\_budget < 0$ do
    $j = y_i$
    while $j \geq 0$ and $power\_budget - power\_change(c_{ij}) < 0$ do
      $j := j - 1$
    $power\_budget := power\_budget - power\_change(c_{ij})$
    $C'' := C'' \cup c_{ij}$
    $i := i + 1$
  return $C''$
```

Algorithm 5 Greedy power distribution algorithm

```
greedy_power_distribution:
  input: $N_{active\_nodes}$, $power\_budget$
  output: $C''$ $\rightarrow$set of power actions
  $C'' := \emptyset$
  $savings :=$ estimate_power_savings($\bigcup_{n_i \in N_{active\_nodes}} c_{i0}$)
  $power\_budget := power\_budget + savings$
  $C' := \bigcup_{n_i \in N_{active\_nodes}, c_{ij} \in C_i} c_{ij}$
  $N' := N_{active\_nodes}$
  order_power_settings($C'$)
  for each $c_{ij}$ in $C'$ do
    if $n_i \cap N' \neq \emptyset$ and power_change($c_{ij}$) $< power\_budget$
then
      $C'' := C'' \cup c_{ij}$
      $power\_budget := power\_budget -$ power_change($c_{ij}$)
      $N' := N' \setminus n_i$
  $C'' := C'' \cup \bigcup_{n_i \in N_{active\_nodes} \setminus N'} c_{i0}$
  return $C''$
```

Algorithm 6 MILP power distribution algorithm

```
milp_power_distribution:
  input: $N_{active\_nodes}$, $power\_budget$
  output: set of power actions
  $C' := \bigcup_{n_i \in N_{active\_nodes}, s_{ij} \in C_i} c_{ij}$
  return find_best_power_allocation($C'$, $power\_budget$)
```

limit through their drivers to their GPUs. However, the Nvidia cards have restraints on the minimum power limit allowed, although this aspect has improved for the Nvidia Volta cards. Tables 2 and 3 illustrate the minimum and maximum power consumptions for various CPUs and GPUs.

To apply the power saving methodology, we designed a benchmarking process to determine the behavior of our target architectures. Using this scheme, we determine the effects of applying power budgets to our heterogeneous system by executing small and simple instances of software. This limits our power saving to the software and hardware combinations that have already been studied. However, experimentation is relatively simple, allowing us to extend the software pool or the available hardware without complex efforts. Benchmarking is performed once, as it provides good enough results to extract the general power behavior of the application. While the obtained data contain variability, we have to take into account the trade-off between resources spent during the benchmark and the resources gained for our target application and architecture. Hence, critical software will require more executions if more precise data are required.

Algorithm 7 defines the few steps required to determine the behavior of executing a software stack on a given hardware platform. Once we have determined the minimum and maximum power required by a system, a series of small tests are executed to measure the performance of the application in *measure_execution* for a set of different power configurations. The specific set can be defined by the server provider or administrator; however, they are always limited to the hardware constraints. Once all power configurations are measured in every case for the software stack and the target hardware, this datum D has to be treated to obtain the possible C configurations that compose P.

5.1 Benchmark procedure

An example of this benchmark procedure, using the serial version of the NAS Parallel Benchmarks (NPB) https://www.nas.nasa.gov/ as the software stack, was performed over a simple Intel architecture. The chosen architecture is an i5-6200U CPU, a sixth-generation processor with the *Intel PowerCap* capabilities. Power capping, performed through the RAPL interface, allows us to control different power-related settings in the CPUs, such as the maximum power

Table 2 Intel CPU power constraints

CPU name	# Cores	Base frequency (GHz)	TDP (W)
i5-6200U	2	2.30	15
Intel Xeon E3-1505M	4	2.80	45
Intel Xeon D-1548	8	2.00	45
Intel Xeon D-1577	16	1.30	45

Table 3 Nvidia GPU power constraints

GPU name	Min. power(W)	TDP(W)
Tesla P40	125	250
Volta V100	100	250

allowed during a time window, and the zone of the chip it is applied to, which allows us to set power limits to the cores or the DRAM separately. We used a simple approach where we limited the power consumption of the whole chip, using the *PACKAGE* zone. Energy measurements were gathered using EML Cabrera et al. (2014) utilizing the same RAPL interface for the CPU only. This methodology can be also applied with more complete measurements for the server as a whole.

Algorithm 7 Benchmark process

```
input: $H$ → Hardware specifications, $S$ → Software stack
output: $P$ → set of power settings

$D \leftarrow \varnothing$                                ▷ Hardware Power Behavior
for all $h \in H$ do
    $p_{min}, p_{max} = Determine\_min\_max\_power(h)$
    $P_{exp} \leftarrow experimental\_interval(p_{min}, p_{max})$      ▷ Experimental
Power Budgets
    for $p \in P_{exp}$ do
        for all $s \in S$ do
            $t, e \leftarrow measure\_execution(h, s, p)$
            $D \leftarrow D \bigcup \{[s, h, p, t, e]\}$
$P \leftarrow get\_power\_settings(D)$
return $P$
```

Figures 3 and 4 depict, as a heatmap, the behavior of the different benchmarks for performance and energy efficiency, respectively. The X axis represents each of the executed kernels from the NPB, while the Y axis represents the maximum power allowed to the CPU. The hardware allows us to limit the power below 3W and over 8W, but these ranges were discarded due to two reasons. Values over 8W did not make any sense, as the single-core execution was no longer limited by the power budget, but by the hardware itself. Limiting to 3W already illustrated decrements in both performance and efficiency, thus lowering the limit would not improve them. Each column of the heatmap is normalized, with 1.00 being the highest value for the benchmark. In both figures, dark blue represents the best configurations, while red represents bad configurations. Figure 3 illustrates the time required to reach the solution for each kernel. The most performant solution is to remove the power budget, independently of the problem. Figure 4 represents the number of operations performed for every watt spent in the execution, in mega-operations per watt (Mops/W). In this case, we observe two different trends, for every kernel excepting the *Data Cube* kernel, labeled as *DC*. This is caused by the *DC* having a high number of I/O operations that access secondary memory. These heatmaps allow us to extract different power configurations depending on the maximum allowed power budget, from best efficiency (4.5W) to best performance (8W) in every case. However, for *DC* the different power configurations would be from 4.0 to 8W.

5.2 Benchmarks of continuous applications and GPU accelerators

Our methodology requires slight modifications for measuring server side applications, such as web services. In the NPB, applications had a finite life cycle and data were obtained measuring at the beginning and at the end of the target software. In this case, the objective of server applications is to be ready to perform work on user requests, and they have to be ready to perform an undetermined amount of work. Despite these differences, the only modification for Algorithm 7 is in *measure_execution*, where measurements are taken for a fixed amount of time in a fully loaded environment. Our target application runs as an Anaconda server that gathers images from a large provided database and, using Tensorflow, trains an image classifier. Energy measurements are gathered using EML, again. However, this time they connect to an external sensor that returns an average power consumption for the whole microserver.

In this case, we limited the power consumption of both the CPU and the GPU to analyze the behavior of the system under different power budgets for its different components. These limits must be specified by the expert who is applying the benchmarking procedure. The CPU power limits can be set, similarly to the NPB case, by finding a lower limit where efficiency is lost, then increasing the power limit until maximum performance is obtained. The GPU version is slightly different, as the minimum power limit is highly dependent on the GPU model. Hence, the experimentation has to be set in

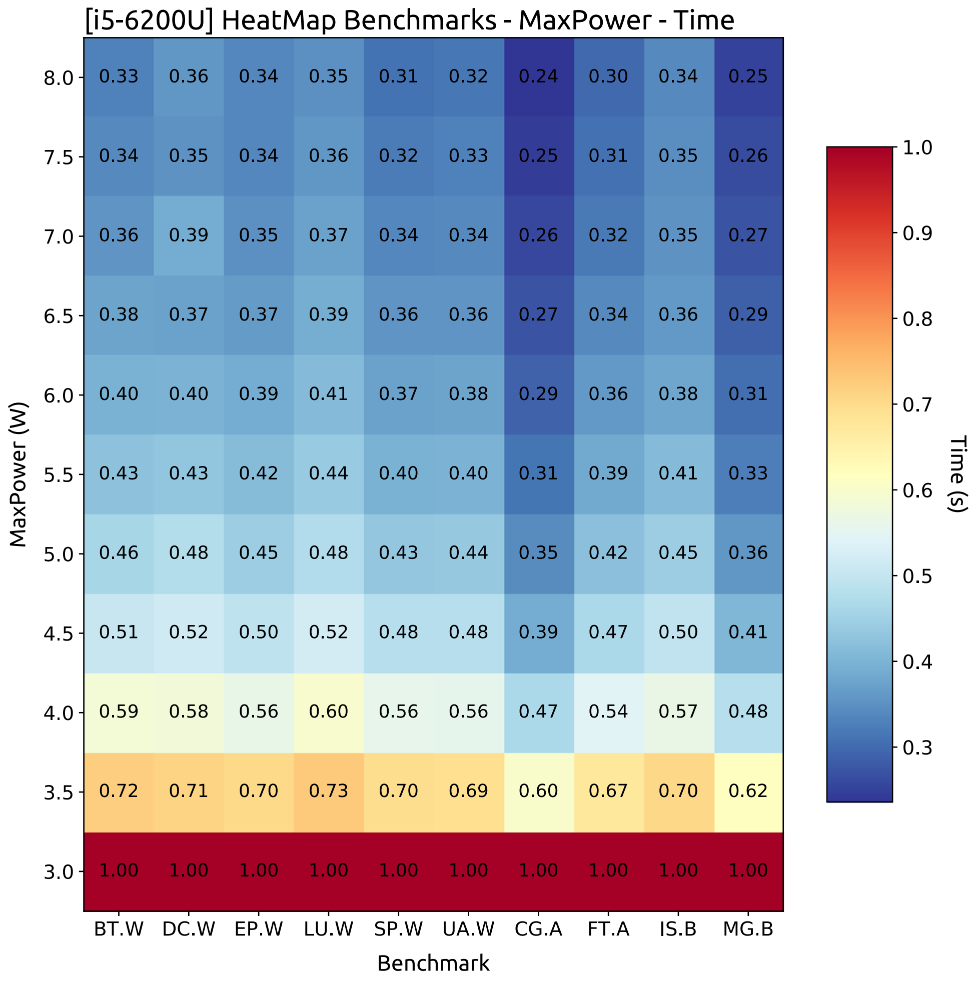


Fig. 3 Serial NPB Benchmarks performance while applying a power cap

between the minimum and maximum power limits allowed by the Nvidia driver. In a Linux environment, these limits are found and set through the *nvidia-smi* command. The maximum power limit is also affected by the performance of the GPU application. If the application is badly optimized, the target GPU consumes less power than the minimum power limit, and any applied power capping policy has no effect.

Figure 5 presents on two curves the performance and the efficiency of the systems. This metric is the equivalent of the mega-operations per watt utilized in the NPB. The X axis represents the maximum power allowed for the CPU, the left axis the number of operations per watt, represented using blue o, and the right axis the performance of the application in images per second, represented using orange $+$.

The efficiency range in these cases is:

1. Xeon D-1548, from best efficiency at 26W to best performance at 40W.
2. Xeon D-1577, from best efficiency at 22W to best performance at 38W.
3. Xeon E3-1505M, from best efficiency at 20W to best performance at 44W.

Finally, Fig. 6 shows the results of the experiment with power limitations used also for GPUs, in the same way as presented in the CPU case. In these cases, the same Xeon E3-1505M processor was tested. These charts present more information, as the power capping was performed for both the GPU and the CPU. The X axis represents the maximum power allowed for the GPU, while the CPU power is represented through different colors. Efficiency is differentiated from performance using different markers: $+$ represents performance, while o represents efficiency. The left axis, in combination with the $+$ markers, depicts the number of operations per watt, and the same is presented for the performance of the application using the right axis and the o markers, in images per second. For both metrics, different colors represent different power caps for the CPU.

Figure 6a, b presents the case where CPU power capping does not affect performance for the Nvidia P40 and the Nvidia V100, respectively. These GPUs offer a broad power capping range, and limits for efficiency and performance can be obtained similarly to how we determined ranges of efficiency for the NPB.

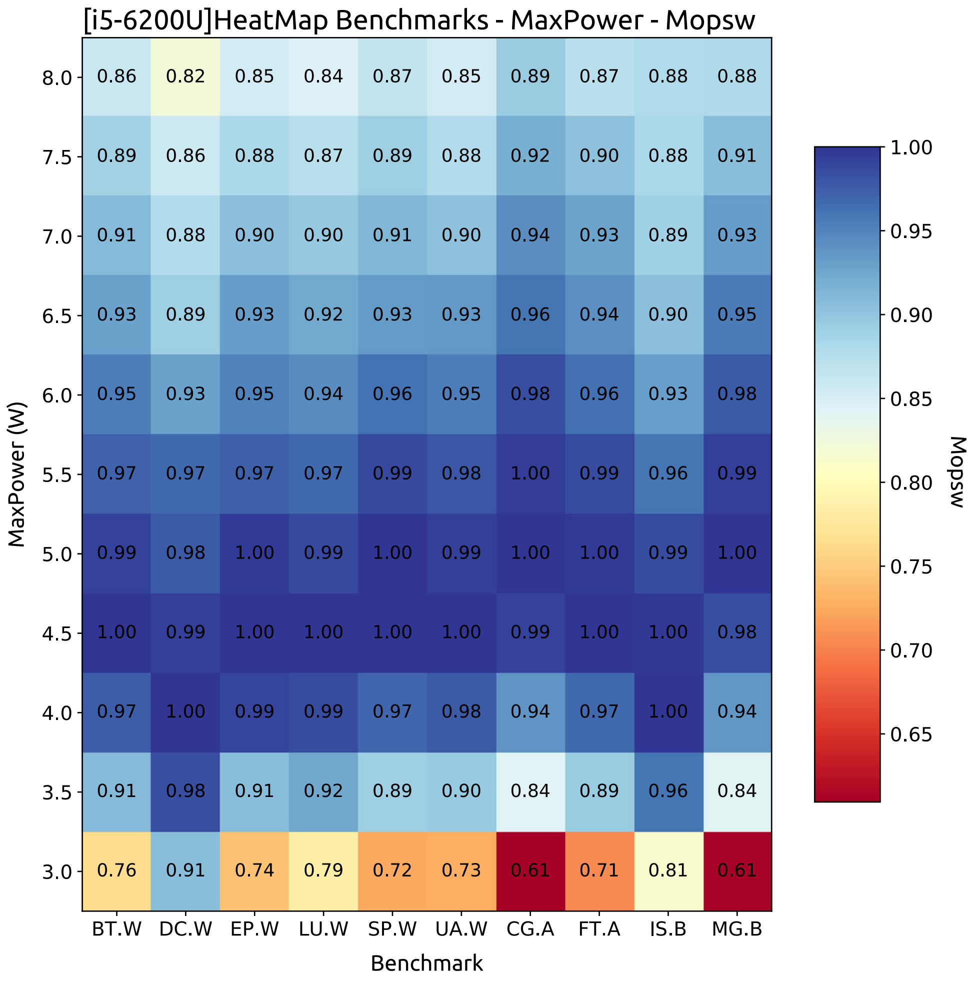


Fig. 4 Serial NPB Benchmarks efficiency while applying a power cap

For these two cases, the information that can be extracted from the data is:

1. The Nvidia P40 has the best efficiency at 125W, and the best performance starts at 175W.
2. The Nvidia V100 has the best efficiency at 115W, and the best performance starts at 165W.

It is important to remark that these two cases could be studied more carefully, as the CPU will affect performance and efficiency if the power cap is applied more aggressively.

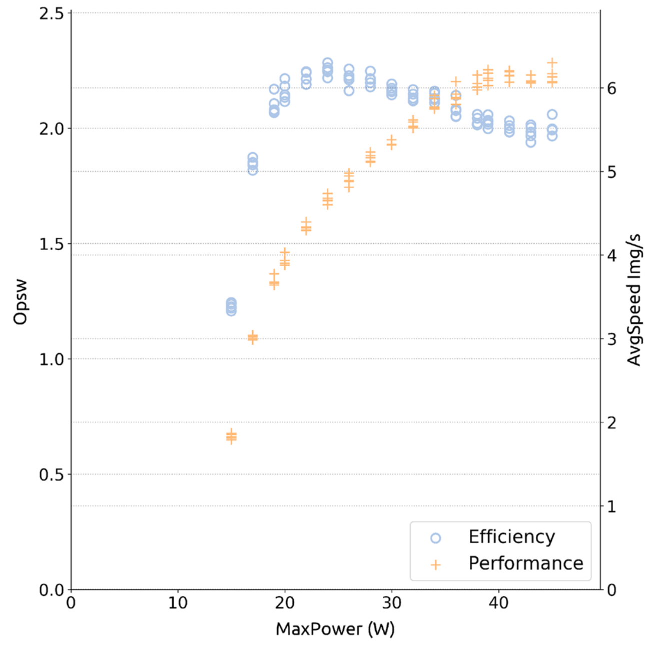
6 Power capping evaluation

In this section, we present a single run experiment conducted on the real infrastructure to evaluate our power capping methodology. We start with providing a description of hardware characteristics used in our studies, and the applications that were run on the system. Then, we provide a configuration and corresponding parameters of the performed experiments. Finally, the results of the evaluation are presented and discussed.

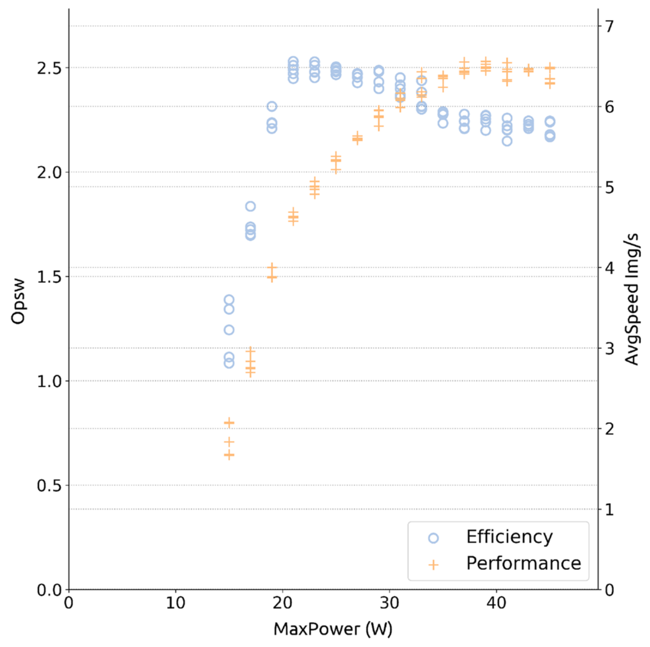
6.1 Resource characteristics

We tested our power capping approach on 11 server nodes. Their types and power-related characteristics are presented in Table 4.

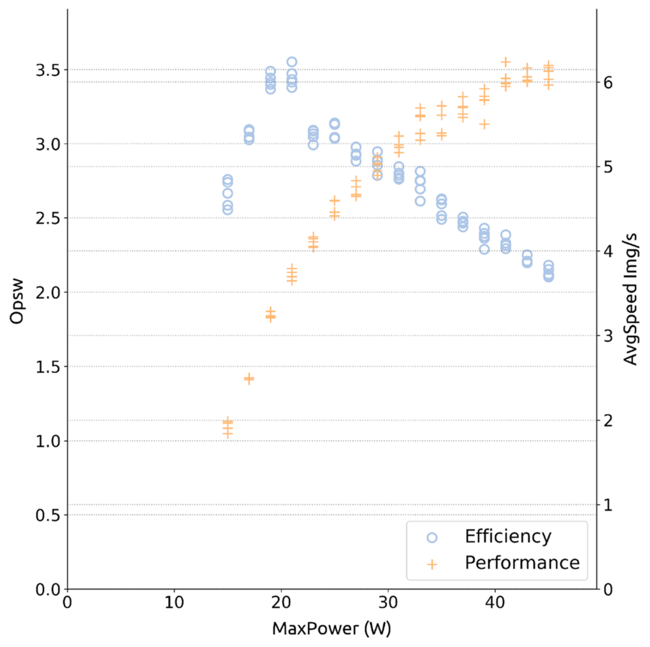
The overall power drawn by the fully loaded testbed is 1160W, and the power draw in the idle state is 550W. The maximum number of power settings is 11, and thus, the number of variables for the MILP is 121 (11 nodes). If the node had less than 11 available power setting, then the additional power settings had 0 efficiency and the power usage was set to a large value (much greater then the overall power limit) so the solver would not choose such state. There were 12 constraints—11 related to the power setting on each node (each node can have only one power setting) and one related to the overall power limit (power cap).



(a) Xeon D-1548



(b) Xeon D-1577



(c) Xeon E3-1505M

Fig. 5 Image classification benchmark CPU results

Table 4 Testbed configuration

Type of node	Total number of nodes	Number of cores / Number of threads	Max frequency (GHz)	TDP (for CPU)
Xeon E5-2640	4	16/32	3	95 W
Xeon D-1548	2	8/16	2.6	45 W
Xeon D-1577	3	16/32	2.1	45 W
i5 4400E	2	2/4	3.3	37 W

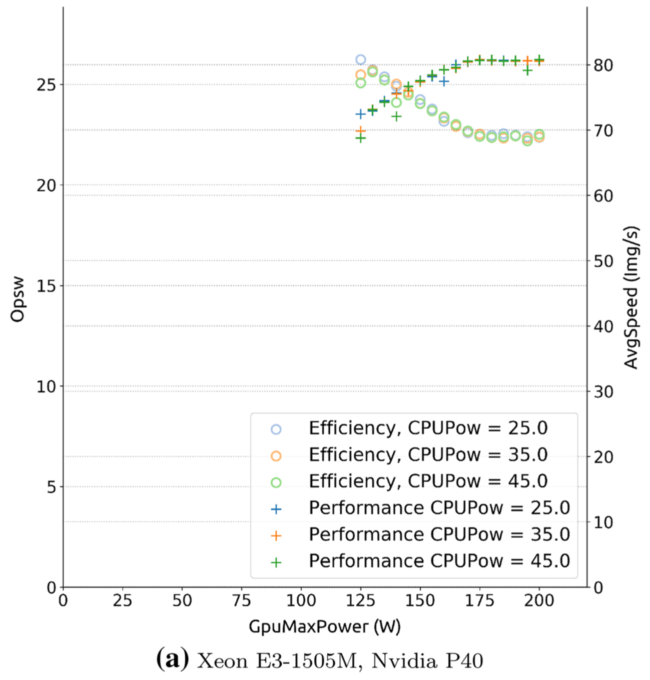


Fig. 6 Image classification benchmark GPU results

6.2 Applications

In order to evaluate the power capping efficiency for various types of applications, we used the NPB benchmarks. Table 5 presents the characteristics of a subset of benchmarks applied during our experiments. *Benchmark Id* is composed of the benchmark specifications and benchmark class identifiers as specified in https://www.nas.nasa.gov/. *Time* represents the amount of time necessary to finish the job on the Xeon E5-2640 node with maximum performance (no power restrictions).

The workload used for experiments consisted of the set of benchmarks submitted to the SLURM queuing system at the same time. As the number of jobs was greater than the available resources, they were scheduled in a queue on a first-come, first-served basis.

Table 5 NPB benchmarks characteristics

Benchmark Id	Time	Number of benchmark jobs
dc.B	400	5
sp.C	180	20
sp.B	60	20
cg.C	60	10
ua.B	30	10
mg.B	2	20

6.3 Experiments

In order to show the capabilities of our approach, we conducted experiments corresponding to the four algorithms presented in Sect. 4 and the case without power capping. During each experiment, we executed the aforementioned set of applications on the testbed and applied the appropriate power capping strategy with the power limit set to 750W. Taking into account the minimum (550 W when idle) and maximum (1160W for the fully loaded system) power usage of the system, choosing 750W as a power limit seemed to be a reasonable trade-off that represented realistic power capping thresholds for the evaluated system. Table 6 summarizes the considered test cases. The α parameter was equal to 1.0 (maximum performance). The priority of each node was equal, and therefore, the priority could be skipped in the evaluation process.

6.4 Results

Power usage results obtained from experiments A, B, C, D, and E are presented in Table 7, and job time-related statistics for the corresponding experiments are presented in Table 8. The evaluation of the results which compares all four algorithms (experiments B-E) to the approach without any power budget distribution (experiment A) is presented in Table 9. *Energy consumed* is the energy consumed by nodes between the start of the experiment and the completion of the last job executed on any node. *Total energy consumed* is the energy consumed by nodes between the start of the

Table 6 Evaluated power capping strategies

Experiment Id	Power capping method	Comments
A	No power distribution algorithm	Reference approach
B	Random power distribution algorithm	Reference approach
C	Simple power distribution algorithm	Reference approach
D	Greedy power distribution algorithm	Proposed heuristic used in power management procedures
E	MILP power distribution algorithm	Proposed exact approach used in power management procedures

Table 7 Experiment energy consumption

Experiment Id	Energy consumed (Wh)	Total energy consumed (Wh)
A	283	346
B	280	325
C	314	380
D	271	304
E	234	315

Table 8 Experiment time statistics

Experiment Id	Job average execution time (s)	Mean flow time (s)	Makespan (s)
A	141	496	1464
B	201	769	1839
C	203	744	2079
D	172	640	1639
E	151	525	1714

experiment and the completion of the last job executed in the queue. *Average performance* and *Average efficiency* refer to defined performance definition (Eq. 17) and efficiency ratio (Eq. 32). Figure 7 illustrates the total power usage within the experiments. One should see that test cases D and E significantly outperform other experiments (A, B, C) on the energy-based criteria (both energy consumed and total energy consumed). Using the proposed heuristic-based approaches for power capping allows reducing the power usage of the system.

As expected, using the power capping mechanism (experiments B, C, D, E) leads to an increase in time-based criteria. Reducing the processor efficiency causes an increase in job execution times, flow times, and makespan. However, again the heuristic-based solutions (D and E) get the better of the random (B) and simple (C) approaches.

Average performance and average efficiency metrics summarize the considerations above. One should note that applying heuristic power capping allows significant energy improvements with considerably low performance losses. Corresponding flow time increase is also acceptable (especially for scenario E)

7 Conclusions

This paper presents a new power capping method for heterogeneous servers. The method takes into account models of specific nodes (along with their priorities) and profiles of applications run in the system. We show that, using this information, application performance losses can be reduced and the energy efficiency improved. We achieved up to 17%

Table 9 Experiment results evaluation

Experiment Id	Average performance	Average efficiency	Mean flow time change (%)
A	1.00	1.00	0
B	0.70	1.01	55
C	0.70	0.90	50
D	0.82	1.04	29
E	0.94	1.20	6

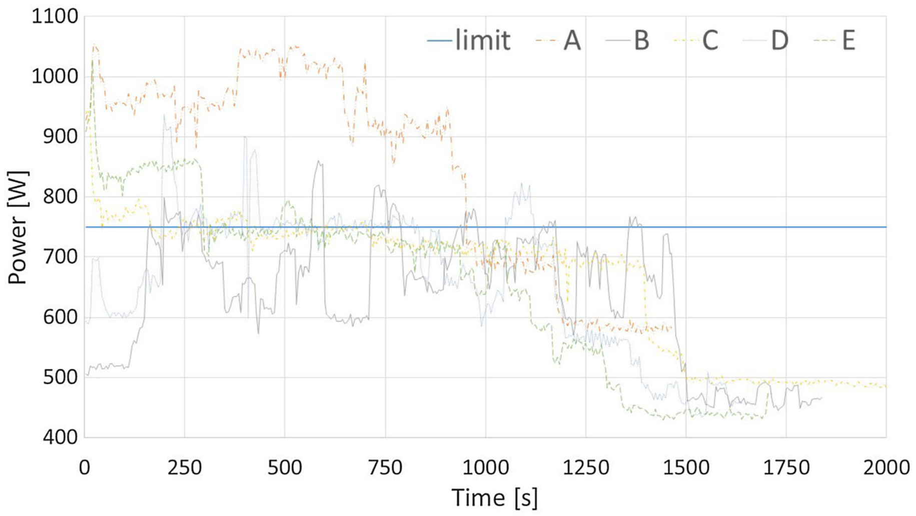


Fig. 7 Power usage within experiments

reduction of energy consumed compared to the solution without power capping or using the random approach, and 25% compared with the simple approach. Importantly, the energy consumed was also lower than in the case when no power capping was applied. The performance losses (mean job flow time) were reduced by 42–47% in the case of greedy heuristics and 88–89% in the case of the exact optimization method, compared to the random and simple approaches. It is worth noting that these results were obtained for a set of jobs with ready time equal to zero. In the case of workloads with higher idle times and lower utilization, the impact of power capping on deterioration of mean flow time should be even lower. That makes power capping not only a way to avoid exceeding certain power constraints caused by shortage of cooling capacity or limits of electrical infrastructure; power capping becomes a powerful tool for improvement of energy efficiency of heterogeneous and dynamic systems. Heterogeneity is supported by applying power capping to a variety of x86 CPUs and to hardware accelerators such as GPUs. To make this solution practical, we combined a greedy heuristic, allowing us to quickly react to changes in power usage or power caps, with an optimization procedure finding exact solutions. The latter was run in the background, improving power settings of servers once the power cap was not exceeded. In our case, optimization for a single chassis with multiple nodes was feasible (execution of the optimization procedure for 200 servers with 10 power states per each server took approximately 30s). Application to a bigger cluster requires adoption of heuristic approaches. Future work will include tests with other applications and real workloads (with lower utilization), experiments with larger clusters and support for other heterogeneous resources, such as FPGA boards or other x86 CPUs.

Acknowledgements The results presented in this paper are partially funded from European Union's Horizon 2020 research and innovation program under Grant agreement No. 688201 (M2DC). This work has been supported by the Spanish Ministry of Science, Innovation and Universities through the TIN2016-78919-R project, the Government of the Canary Islands, with the project ProID2017010130 and the Grant TESIS2017010134, which is cofinanced 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%), the Spanish network CAPAP-H, and the European COST Action CHIPSET. The research has also been supported by the Polish National Science Centre under Project No. 2013/11/B/ST6/00970.

References

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, *28*(5), 755–768. https://doi.org/10.1016/j.future.2011.04.017. Special Section: Energy efficiency in large-scale distributed systems.

Bhattacharya, A.A., Culler, D., Kansal, A., Govindan, S., & Sankar, S. (2012) The need for speed and stability in data center power capping. In 2012*International green computing conference* (IGCC) (pp. 1–10). https://doi.org/10.1109/IGCC.2012.6322253

Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., & Benini, L.(2016). Predictive modeling for job power consumption in hpc systems.In Kunkel, J. M., Balaji, P., & Dongarra, J. (Eds.) *High* *performance computing* (pp. 181–199). Cham: Springer International Publishing.

Cabrera, A., Acosta, A., Almeida, F., & Blanco, V.(2017) Energy efficient dynamic load balancing over multigpu heterogeneous systems. In *Parallel processing and applied mathematics-12th international conference*, PPAM 2017, Lublin, Poland, September 10–13, 2017, Revised Selected Papers, Part II (pp. 123–132). https://doi.org/10.1007/978-3-319-78054-2_12

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

Da Costa, G., Oleksiak, A., Piatek, W., Salom, J., & Sisó, L.(2015) Minimization of costs and energy consumption in a data center by a workload-based capacity management. In Klingert, S., Chinnici, M., & ReyPorto M. (eds.) *Energy efficient data centers* (pp. 102–119). Cham: Springer International Publishing.

Fan, X., Weber, W. D., & Barroso, L. A. (2007). Power provisioning for a warehouse-sized computer. *SIGARCH Computer Architecture News*, *35*(2), 13–23. https://doi.org/10.1145/1273440.1250665.

Fukazawa, K., Ueda, M., Aoyagi, M., Tsuhata, T., Yoshida, K.,Uehara, A., Kuze, M., Inadomi, Y., & Inoue, K. (2014) Power consumption evaluation of an mhd simulation with cpu power capping. In 2014 14th *IEEE/ACM international symposium on cluster, cloud and grid computing* (pp. 612–617). https://doi.org/10.1109/CCGrid.2014.47

Guzek, M., Bouvry, P., & Talbi, E. G. (2015). A survey of evolutionary computation for resource management of processing in cloud computing. *IEEE Computational Intelligence Magazine*, *10*(2), 53–67.

Guzek, M., Kliazovich, D., & Bouvry, P. (2015) HEROS: energy-efficient load balancing for heterogeneous data centers. In Pu, C., Mohindra A., (eds.) 8th *IEEE International conference on cloud computing, CLOUD* 2015, New York City, NY, USA, June 27-July 2, 2015 (pp. 742–749). IEEE. https://doi.org/10.1109/CLOUD.2015.103

Haidar, A., Jagode, H., Vaccaro, P., YarKhan, A., Tomov, S., & Dongarra, J. (2019). Investigating power capping toward energy-efficient scientific applications. *Concurrency and Computation: Practice and Experience*, *31*(6), e4485. https://doi.org/10.1002/cpe.4485.

Jeyarani, R., Nagaveni, N., & Ram, R. V. (2012). Design and implementation of adaptive power-aware virtual machine provisioner (apa-vmp) using swarm intelligence. *Future Generation Computer Systems*, *28*(5), 811–821.

Krzywaniak, A., Proficz, J., & Czarnul, P. (2018a) Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors. In 2018 *Federated conference on computer science and information systems* (FedCSIS). (pp. 339–346)

Krzywaniak, A., Proficz, J., & Czarnul, P. (2018b) Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors. In *Proceedings of the 2018 federated conference on computer science and information systems* (pp. 339–346). https://doi.org/10.15439/2018f177. https://doi.org/10.15439/2018f177

Krzywda, J., Ali-Eldin, A., Wadbro, E., Östberg, P., & Elmroth, E. (2018) Alpaca: Application performance aware server power capping. In 2018 IEEE *International conference on autonomic computing* (ICAC) (pp. 41–50). https://doi.org/10.1109/ICAC.2018.00014

Liu, Y., Cox, G., Deng, Q., Draper, S.C., & Bianchini, R. (2016) Fastcap: An efficient and fair algorithm for power capping in many-core systems. In 2016 *IEEE International symposium on performance analysis of systems and software* (ISPASS) (pp. 57–68). https://doi.org/10.1109/ISPASS.2016.7482074

NASA: Nas Parallel Benchmarks. (2019). https://www.nas.nasa.gov/publications/npb.html. Accessed April 2019.

oj! Algorithms.http://ojalgo.org/

Oleksiak, A., Ciesielczyk, T., Kierzynka, M., & Piatek, W. (2018) Minimising energy costs of data centers using high dense heterogeneous systems and intelligent resource management. In *Proceedings of the ninth international conference on future energy systems, e-Energy* '18. (pp. 499–505). ACM, New York, NY, USA. https://doi.org/10.1145/3208903.3213777. https://doi.org/10.1145/3208903.3213777

Oleksiak, A., Kierzynka, M., Piatek, W., Agosta, G., Barenghi, A., Brandolese, C., Fornaciari, W., Pelosi, G., Cecowski, M., Plestenjak, R., inkelj, J., Porrmann, M., Hagemeyer, J., Griessl, R., Lachmair, J., Peykanu, M., Tigges, ., Berge, M.v.d., Christmann, W., Krupop, S., Carbon, A., Cudennec, L., Goubier, T., Philippe, J.M., Rosinger, S., Schlitt, D., Pieper, C., Adeniyi-Jones, C., Setoain, J., Ceva, L., & Janssen, U. (2017) M2dc modular microserver datacentre with heterogeneous hardware. microprocessors and microsystems **52**(C), 117–130. https://doi.org/10.1016/j.micpro.2017.05.019. https://doi.org/10.1016/j.micpro.2017.05.019

Rountree, B., Ahn, D., De Supinski, B., Lowenthal, D., & Schulz, M. (2013) Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *Proceedings of the 2012 IEEE 26th international parallel and distributed processing symposium workshops* IPDPSW 2012. (pp. 947–953). https://doi.org/10.1109/IPDPSW.2012.116

Takouna, I., Rojas-Cessa, R., Sachs, K., & Meinel, C. (2013) Communication-aware and energy-efficient scheduling for parallel applications in virtualized data centers. In IEEE/ACM 6th *International conference on utility and cloud computing*, UCC 2013, Dresden, Germany, December 9–12, 2013 (pp. 251–255). IEEE. https://doi.org/10.1109/UCC.2013.50

Tiwari, A., Schulz, M., Carrington, L. (2015) Predicting optimal power allocation for cpu and dram domains. In 2015 *IEEE International parallel and distributed processing symposium workshop*. (pp. 951–959). https://doi.org/10.1109/IPDPSW.2015.146

Zhang, H., & Hoffmann, H. (2016). Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *SIGARCH Computer Architecture News*, *44*(2), 545–559. https://doi.org/10.1145/2980024.2872375.