Scheduling independent tasks in parallel under power constraints

Ayham Kassab, Jean-Marc Nicod, Laurent Philippe, Veronika Rehn-Sonigo
FEMTO-ST Institute, Université Bourgogne Franche-Comté / CNRS / ENSMM
F-25000 Besançon, France
[ayham.kassab|jean-marc.nicod|laurent.philippe|veronika.sonigo]@femto-st.fr

Abstract

Energy consumption has become a major concern in the recent years and Green computing has arisen as one of the challenges in order to reduce CO_2 emissions in the computing domain. Many efforts have been made to make hardware less energy consuming, reduce cooling energy of data and computing centers by relocating those facilities to cool regions and other. A novel approach to make the computing domain greener is to add renewable energy sources for the power supply. The challenge of this work is to consider computing facilities which are solely run by renewable energy sources such as solar panels and wind turbines. In this work we tackle the problem of scheduling independent tasks within a predicted power envelope that varies during the time. First we evaluate different instances of the problem from a theoretical point of view. Then we propose several heuristics for the case of multi-core architectures and we assess their performance on synthetic workloads and power envelopes.

Energy efficiency, computing center, scheduling, optimization, complexity

1 Introduction

As a solution to minimize global warming, improving energy efficiency is one of today's major concerns. In computer science, lots of research works tackle this problem as computing or data centers are known to be one of the big energy consumers. There are however several ways to reduce the energy footprint, either reducing the consumption or using energy sources that less impact the environment. In the case of computing resources lots of effort is put on reducing the energy consumption of the resources, from the processor to the cooling. Nevertheless, as low as their consumption will be, they will still consume power. An alternative solution to further reduce their impact is to use green sources such as solar panels, wind turbines, or fuel cells as these devices do not produce CO_2. The challenge of this work is to consider computing facilities which are solely powered by renewable energy sources.

Efficiently powering a computing or data center implies to deliver the requested power level. Renewable energy sources however provide a variable energy provisioning depending on solar and wind conditions so that they must be completed by batteries or other energy storage systems to guarantee a continuous work. In this problem, the most central part is taken by the computers. If they do not have enough power to run tasks then it is useless to supply energy to the rest of the center. On the other hand at some points of the power supplier life-cycle the energy storage components become full and there is no need to spare energy that should rather be consumed than lost. For these reasons, we concentrate in this work on processing tasks depending on the available power.

This article tackles the problem that we can formally solve in scheduling with green energy optimization. Our approach is different from traditional energy aware scheduling approaches in that it does not target energy minimization itself but it rather targets to better use available power. The optimization problem is thus rather to limit the energy waste, the produced energy that cannot be used, than finding ways to decrease the consumed energy. We tackle on the one hand computing center oriented problems where the optimization objective is the makespan to finish a set of jobs as soon as possible and, on the other hand, data center oriented problems where the optimization objective is the flowtime to reduce the mean waiting time. To concentrate on the scheduling problem, this first work considers scheduling a set of tasks on a shared memory machine as, in that case, we do not need to take machine power on/off into account for our optimizations.

The presented contributions are as follows:

- We show that most power constraint scheduling problems are complex. We provide formal complexity results on three out of four scheduling problems on one machine and extend them to more general parallel problems.
- We provide a performance study of several heuristics proposed to solve the problem. These simulations show that the best algorithm depends on the weight of the processing time compared to the power need of the tasks.

The paper is organized as follows: related work is presented in Section 2, then a system model is proposed in Section 3. We present formal results on the complexity of the related optimization problems in Section 4 and propose several heuristics to solve the considered problems in Section 5. Experiments to assess the heuristic performance are presented in Section 6 and we conclude in Section 7.

2 Related Work

Energy saving is a major concern in the computing domain and there exists a large variety of research that tackles the problem of reducing the energy consumption. Several surveys give an interesting overview of the research done in the field of green computing. For instance [19, 22, 23] give a wide survey on all the technologies and tools that can be used in data centers to lower the energy consumption.

One possible technique for energy reduction is the usage of Dynamic Voltage and Frequency Scaling (DVFS), which allows to run processors and servers at a lower speed at the price of increased execution times for tasks. Wu et al. [24] use a two step approach to allocate tasks to servers and then determine the voltage/frequency combination to reduce the energy consumed by servers in data centers. Garg et al. [11] include Dynamic Voltage Scaling (DVS) in their scheduling algorithm for HPC applications on Cloud-oriented data centers. Their solution allows to reduce up to 25% the energy usage in comparison to profit based scheduling policies with even higher profit. It seems however that the DVFS tuning tends to be more and more embedded directly inside the processor with less control actions left to the user as explained in [16].

Another approach to reduce the energy consumption is the consideration of a shutdown model, where processors have an additional state to on and off, the sleep state. In a one machine offline setting with jobs of unit processing times, release dates, and deadlines, a dynamic programming approach allows to minimize the number of idle time periods in polynomial time [4]. In follow up work of Baptiste et al. [5, 6] the result is extended to heterogeneous preemptive jobs. Albers and Antoniadis [3] combine speed scaling with the sleep state model. They prove NP-completeness of the energy minimization problem for heterogeneous tasks with release dates and deadlines.

As minimizing the energy consumption without another complementary objective simply leads to stop all executions, works on energy efficiency often use a bi-criteria objective. Beloglazov et al. [7] for example propose energy efficient solutions for Cloud data centers via virtual machine migration while respecting quality of service (QoS) requirements defined by service level agreements (SLA). Bi-objectives criteria are however complex to manage as they usually implies to balance between two opposite.

In [1] the authors also tackle the theoretical complexity of energy-efficient scheduling algorithms. They schedule independent tasks on parallel identical and uniform machines and give optimal solutions for divisible loads minimizing the makespan. They also show that the non-divisible case of tasks is NP-hard. The optimization criteria is the makespan and they optimize in one step the makespan and the consumed energy by considering the energy consumed by the machines when they are idle. They however have no constraint on power and can always use the available machines.

In recent years an other trend has arisen: data and computing centers integrate renewable energy sources as power supply. An early work on green energy utilization in data centers by Aksanli et al. [2] shows the importance of power prediction. They propose an adaptive data center job scheduler that reduces the number of aborted jobs while improving the green energy utilization. In [13, 14] the authors propose GreenSlot, a batch scheduler for parallel tasks, that aims to reduce the brown power consumption of a data center partially powered by solar panels. In GreenSlot the jobs have deadlines and the scheduler first reserves resources for the jobs with lower slack (distance from latest possible start time to current time). Based on weather forecasting and power prediction, GreenSlot schedules the tasks on time slots. However the authors do not try to optimize their schedules, they just reduce the consumption and costs while meeting as much deadlines as possible. Similarly [20] presents an holistic approach to optimize the energy cost with incomes from running batch jobs and outcomes to buy brown energy. The paper also provides a proposition for net zero scheduling batch jobs. It is however based on virtualization and is not bounded by the number of resources.

None of the aforementioned work deals with computing resources provisioned with 100% renewable energy that we consider in this work. In this context the available power is constrained by the power production and the scheduling algorithms have to cope with a variable power envelope. We thus tackle the classical mono criteria optimization objectives makespan and flowtime under power constraints.

3 Model

In this section we define the models of machines, power and tasks that we use in the addressed optimization problem.

The considered computing platform is parallel which means that several execution units are available to process the tasks. The platform thus consists of a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of m machines M_j that represent execution units.

As the power provisioning of the platform solely comes from green energy sources, its production is not stable and varies over time. The available power is represented at each time t by a curve $\Phi^{available}(t)$. To be able to optimize the usage of the power we assume that the available power $\Phi^{available}(t)$ is a constant value $\Phi_x^{available}$ over an interval of time Δ_x. For a given time horizon $\mathcal{H}$ the available power is thus modeled by X intervals Δ_x of length δ_x, such that $\sum_{x=1}^{X} \delta_x = \mathcal{H}$. This power is shared by all the machines of the platform.

Each machine M_j consumes a static power Φ_j^{stat} at any time t when it does not process any tasks. This static power has a constant value for the entire considered time horizon $\mathcal{H}$, and since it is useless to run a machine without processing tasks, we rather consider useful available power $\Phi_x = \Phi_x^{available} - \sum_{j=0}^{\mathcal{M}} \Phi_j^{stat}$ for the period of time Δ_x in the following. If the platform consists of only one machine, $\Phi(t) = \Phi_x$.

For the task model we consider a set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of n tasks T_i characterized by their processing time p_i. These tasks are sequential independent tasks. Running a task on one machine generates an extra power consumption [12] which varies over time depending whether the task intensively computes or not. It has to be approximated to be used in an optimization problem. We assume that each task T_i has a constant power demand, its largest power need φ_i over its lifetime. By taking the larger power consumption, we guarantee that the resulting schedule will fit in the power envelope.

To schedule a given task T_i we need to find a time slot, a group of intervals, where the available power is always higher than φ_i, the task need. We define the set $\mathcal{E}_j(\varphi_i) = \{\mathcal{E}_{1,i}, \mathcal{E}_{2,i}, \ldots, \mathcal{E}_{K_i,i}\}$ of K_i eligible time slots where task T_i can run. Let $b_{k,i}$ be the beginning of the slot $\mathcal{E}_{k,i}$ and $f_{k,i}$ be its finish time. Then, for $\mathcal{E}_{k,i} = [b_{k,i}, f_{k,i}[$, the available power must be greater than φ_i, with $b_{k,i} \leq t < f_{k,i}$ and $\Phi(t) \geq \varphi_i$. Formally, it exists two integer values x and s such that the kth time slot $\mathcal{E}_{k,i}$ is defined by $\mathcal{E}_{k,i} = \Delta_x \cup \Delta_{x+1} \cup \ldots \cup \Delta_{x+s}$ where at any time $t \in \mathcal{E}_{k,i}$, $\Phi(t) \geq \varphi_i$ and at any time

$t \in \Delta_{x-1}$ or $t \in \Delta_{x+s+1}$ $\Phi(t) < \varphi_i$ (see Figure 1). Finally, we consider an allocation function $A(i, j) = k$ that returns in which time interval $\mathcal{E}_{k,i}$ the task T_i is scheduled on machine M_j.

Table 1 summarizes the notations used in the remainder of the paper.

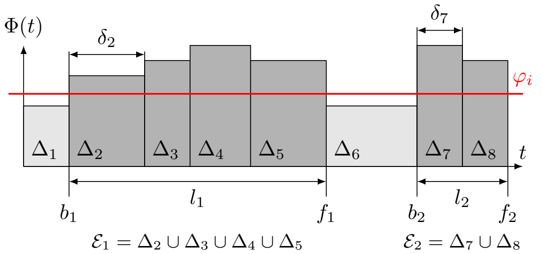


Figure 1: Illustrative example for intervals $(\Delta_1, \dots, \Delta_8)$, available power on time and time slots ($\mathcal{E}_2$, $\mathcal{E}_2$) in which one task T_i could be scheduled when its power need is φ_i onto a one machine platform.

4 Optimization Problems

Using the preceding model we consider static optimization problems where the number, the consumption, the duration of the tasks and the available power are known in advance. Static problems are sometimes far from real practical cases but tackling such problems is however necessary to formally prove the complexity of the optimization problems related to real cases.

4.1 Notations and objectives

Graham [15] defined the $\alpha|\beta|\gamma$ notation that characterizes a scheduling optimization problem. In this notation the α value gives the characteristics of the execution platform: 1 for one machine, P, Q or R for parallel machines respectively identical, uniform or unrelated. The β value gives the tasks characteristics and/or constraints: $p_i = p$ for tasks of the same size, *prec* for precedence between tasks, *pmtn* if tasks can be preempted, etc. The γ value gives the criteria to be optimized as, for instance the makespan $C_{\max}$ or the (total) flowtime [8], $\sum C_i$, where C_i is the completion time of task T_i.

To express the constraint of limited available power we propose to add $\varphi_i \leq \Phi_x$ for one machine problems and $\sum \varphi_i \leq \Phi_x$ for parallel machine problems to the Graham notation. This enforces that the power needed by one (φ_i) or several tasks ($\sum \varphi_i$) must be lower than the power provided (Φ_x) by the energy sources. For example the problem $1|\varphi_i \leq \Phi_x|C_{max}$ is a one machine problem where we target makespan optimization for independent tasks, available power is not a constant over the time horizon and each task has a power need different from each other. If the Φ_x variable is set to Φ, the available power is constant over the considered horizon and if the φ_i variable is set to φ, each task needs the same power to run.

Considering computing and data centers, two main criteria are usually considered for minimization. The makespan ($C_{\max}$) targets the minimization of the running time for a set of tasks and is thus relevant for computing centers where applications are composed of a set of tasks. In the case of several tasks launched by different users, as in data centers, then the flowtime ($\sum C_i$) is more relevant as minimizing this criterion leads to minimizing the mean finish time which enforces a fair share of the resources between users.

4.2 One Machine Problems

Here, we first tackle one machine problems as showing that these problems are NP-Hard proves that the more general parallel problems are NP-Hard as well. We consider these problems for both objectives makespan and flowtime and for the cases with or without preemption. These cases are simple without power constraint. We recall that all no delay schedules (i.e., schedules without delay between the tasks) are optimal solutions for the makespan objective and that the Shortest Processing Time (SPT) algorithm gives optimal solutions for the flowtime objective. We show here that, with power constraints, these problems are polynomial in the case of identical tasks (i.e., tasks with same p_i) and that the problems where tasks have different processing times are actually NP-Hard if preemption is not allowed.

In the one machine problems the static power Φ_j^{static} consumed by the machine $M_j \in \mathcal{M}$ is constant over time. Then the machine cannot run at any time t

Table 1: Summary of the notations.

var.	definition	var.	definition
$\mathcal{T}$	set of tasks	$\mathcal{M}$	set of machines
T_i	task i	M_j	jth machine of $\mathcal{M}$
n	number of tasks	m	number of machines
p_i	processing time of T_i	Φ_x	useful power on M_j
φ_i	power needed by T_i		at interval Δ_x
Δ_x	interval with constant power	δ_x	length of Δ_x
X	number of intervals Δ_x		
$\mathcal{E}_j(\varphi_i)$	time slots defined with φ_i	$\mathcal{E}_{k,i}$	kth term of $\mathcal{E}_j(\varphi_i)$
$b_{k,i}$	beginning of $\mathcal{E}_{k,i}$	$f_{k,i}$	finish time of $\mathcal{E}_{k,i}$
K	cardinal of $\mathcal{E}_j(\varphi_i)$	$l_{k,i}$	length of $\mathcal{E}_{k,i}$

where the power provided is lower than this value. For that reason we assume that the static power needed by the machine is at least available in each interval and, in the remainder of the paper, we thus only consider $\Phi_x = \Phi_x^{available} - \Phi_j^{static}$, the useful power to run tasks.

We now consider different cases for the task processing time and the objective function.

4.2.1 Problems without preemption

In computing centers the nodes are usually dedicated to the users and no preemption is applied to the tasks. We assess here the complexity of the scheduling problem in this context.

Identical tasks $p = p_i$ and $\varphi_i \leq \Phi_x$ The most simple problems are the cases where every task has the same computing time $p_i = p$ and the available power is constant. To optimize our objective, we just have to put as many tasks as possible in each time slot, starting with the tasks with the largest power need. If the interval length is not a multiple of the task size then the remaining time of an interval can be used if the next tasks can be shifted of less than p. Obviously this solution is optimal for the makespan objective, as every task is interchangeable with another, changing the order will not give a better solution and we do not leave empty places where a task can be put. For the flowtime, as every task has the same processing time, none of them has a larger weight in the final sum and there is no need to put the tasks in a specific order.

Non-identical tasks The non-identical task problems, denoted respectively $1|\varphi_i \leq \Phi_x|C_{\max}$ and $1|\varphi_i \leq \Phi_x|\sum C_i$, are NP-Hard. In the following, these complexity results are proven in a row.

Theorem 1. *Minimizing the makespan of the schedule of a set of tasks ($1|\varphi_i \leq \Phi_x|C_{\max}$) to run in a set of intervals is NP-Hard in the strong sense if the tasks have different processing times p_i.*

Proof. First note that, in the case where all tasks need the same power to run $\varphi_i = \varphi$, a time interval Δ_x either provides enough power to run a task or not. The real amount of power provided during this interval is not important as it is just a binary question of enough power or not. The NP-Hardness of the makespan minimization problem will be demonstrated by proving first the problem where each task needs a power φ ($1|\varphi_i = \varphi_x \leq \Phi|C_{\max}$) to be executed and where the set $\mathcal{E}(\varphi)$ defines time slots in which it is possible to schedule tasks.

Let us consider the following decision problem: given a time Z, is there a schedule where the last task is completed before Z? We assume that the allocation respects the constraints of the problem, i.e., every task allocated to one time slot has enough time to be completed before the end of the slot and the power available into this time slot is greater or equal to the sum of power needed by the tasks scheduled in the time slot.

The problem is in NP: given a schedule it is easy to check in polynomial time whether it is valid or not before the time Z. The NP-Completeness is obtained by reduction from 3-PARTITION [10] which is NP-Complete in the strong sense.

Let us consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3H$ positive integers $a_1, a_2, \ldots, a_{3H}$ such that for all $i \in \{1, \ldots, 3H\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{3H} a_i = HB$, does it exist a partition $I_1, \ldots, I_H$ of $\{1, \ldots, 3H\}$ such that for all $h \in \{1, \ldots, H\}$, $|I_h| = 3$ and $\sum_{i \in I_h} a_i = B$?

We build the following instance $\mathcal{I}_2$ of our problem with $\mathcal{E}(\varphi) = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_H\}$ the set of qualified time slots $\mathcal{E}_h$ to run tasks (i.e., available power greater than φ) and whose length are all equals to $f_h - b_h = l_h = l = B$. There are $3H$ tasks $T_i \in \mathcal{T}$ such that each T_i needs a power of φ to be executed and its processing time is $p_i = a_i$ for all $1 \leq i \leq 3H = n$. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. For $1 \leq h \leq H$, task T_i is assigned to time slot $\mathcal{E}_h = [b_h, f_h[$ with $i \in I_h$ within the period and $p_i = a_i$. Then, we have $\sum_{i|A(i)=h} p_i = l = \sum_{i \in I_h} a_i = B$ and therefore the constraint on the processing time is respected for the H slots. We have a solution to $\mathcal{I}_2$.

Suppose that $\mathcal{I}_2$ has a solution. Let $\mathcal{T}_h$ be the set of tasks allocated to the slot $\mathcal{E}_h$ (We recall that if $T_i \in \mathcal{T}_h$, $A(i) = h$) such that for all tasks $T_i \in \mathcal{T}_h$ with $i \in I_h$, $\sum_{i \in I_h} p_i = l = B$. Because of $p_i = a_i$, $|\mathcal{T}_h| = |I_h| = 3$. The length of the time slot l in which the available power is φ has to be fully filled for all H periods to be sure to complete the last task within the slot $\mathcal{E}_H = [b_H, f_H[$ at time $t = f_H = Z$. Otherwise, an other slot has to be used to complete unprocessed tasks. Thus the solution is a 3-PARTITION and we have proven that the addressed decision problem is NP-Complete and thus minimizing the makespan $C_{\max}$ of a set of tasks with different processing times and the same power need to run on one machine is NP-Hard in the strong sense.

Since this problem $1|\varphi_i = \varphi \leq \Phi_x|C_{\max}$ is a special case of $1|\varphi_i \leq \Phi_x|C_{\max}$ it proves that this problem is also NP-Hard. This concludes the proof. □

Theorem 2. *Optimizing the flowtime of the schedule of a set of tasks ($1|\varphi_i \leq \Phi_x|\sum C_i$) to run in a set of intervals is NP-Hard in the strong sense if the tasks have different processing times p_i.*

Proof. This proof relies on the 3-PARTITION problem. The idea is to take the same intervals and tasks that match the preceding 3-PARTITION problem, plus to add one $K + 1$th interval that is sufficiently far from the other ($N \times f_K$ for instance) to generate flowtime that will be higher than any solution that do not use this interval. Due to lack of space, we do not develop that proof here. The proof of this theorem can be found in the companion research report [18]. □

4.2.2 Problems with preemption

In the case of data centers that process requests preemption is allowed. We thus consider the impact of preemption on the scheduling problems complexity.

The $1|\varphi \leq \Phi_x, pmtn|C_{\max}$ problem, all tasks need the same power to run, accepts a polynomial solution. Remember that without power constraints non delay schedules are optimal. With power constraints it is however not possible to always have non delay schedules as some of the intervals Δ_x may not provide enough power Φ_x to schedule a task. The general idea is to avoid leaving intervals empty when there are still unscheduled tasks. For this purpose we schedule tasks with the following policy: At the beginning of a new interval or when a task is finished, we schedule the task (or the remaining part of a task) which wastes the less power ($\min(\Phi_x - \varphi_i)$). If another task than the current running task is selected, the running task is preempted and rescheduled later. We call this algorithm Less Wasting Remaining Task (LWRT).

Theorem 3. *Algorithm LWRT gives an optimal solution for the $1|\varphi_i \leq \Phi_x, pmtn|C_{\max}$ problem.*

Proof. The optimally of the LWRT algorithm is demonstrated by contradiction.

We consider that an optimal schedule S^* does not always run LWRT at each interval, starting from $t = 0$.

We assume that interval Δ_x is the first interval such that it includes task T_i ($S^*(T_i) = t$) which is not the LWRT task and such that T'_i, the LWRT task, runs later ($S^*(T'_i) = t', t' > t$). As T_i is not the LWRT task then we have $\Phi_x - \varphi_i > \Phi_x - \varphi'_i$ and $\varphi_i < \varphi'_i \leq \Phi_x$. Since the power consumed by T'_i is higher than the power consumed by T_i and since T'_i fits in interval Δ_x because it is the LWRT tasks for this interval then we can swap T_i and T'_i (or at least part of them). Moreover, since T_i needs less power than T'_i it could be scheduled before t' in an interval that was not exploited by T'_i with more power. After this step the resulting schedule is at least the same but it could also have been improved by moving T_i. This result is a contradiction with the assumption that S^* is optimal and given any schedule we can do better if we respect the LWRT order. Thus the LWRT algorithm gives an optimal schedule which concludes the proof. □

Figures 2 and 3 illustrate the case where the LWRT task is or is not scheduled at each interval change or when a task is completed. On Figure 2 task T_2 is not preempted at the end of interval Δ_2. As a result task T_4 is scheduled later because of its large power need and interval Δ_5 is not used. On Figure 3 task T_2 is preempted at the end of interval Δ_2 and Task T_4 is executed instead. As Task T_2 needs less power to run it can be executed in interval Δ_5 which improves the makespan.

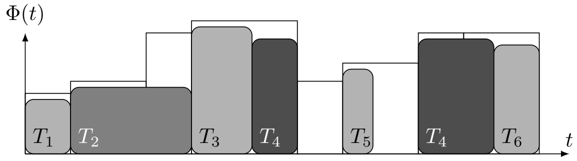


Figure 2: Illustrating example for the LWRT algorithm, T_2 is not the LWRT task for interval Δ_3, T_4 must be run here.

The complexity of the problem $1|\varphi_i \leq \Phi_x, pmtn|\sum C_i$ is still open. We have counter examples that SPT (Shortest Processing Time) does not always give the optimal result as due to power constraints it can be necessary to schedule longer tasks before short ones. Even if the complexity of this case remains an open problem, we suspect it to be NP-Hard.

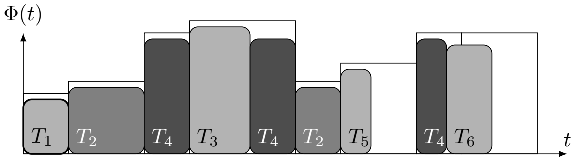


Figure 3: Illustrating example for the LWRT algorithm, part of T_4 has been swapped with T_2 which can be executed sooner than T_5, the makespan is optimal.

4.3 Parallel Problems

We consider here the problem of scheduling a set of tasks on a set of machines. Several sub-problems can be identified from the general parallel problem aside from the classical P, Q, R cases. Shared memory problems are indeed different from distributed memory problems. In the shared memory problems only one machine is used and the tasks are processed by the different cores of the machine. As just one machine must be powered on, we do not need to take static power into consideration and, as for the one machine problems, we take the task power consumption into account. In this case the machines are cores and we can limit the study to identical machines (P in the Graham notation).

From the previous complexity results we can deduce that $P|\sum \varphi_i \leq \Phi_x|C_{\max}$ and $P|\sum \varphi_i \leq \Phi_x|\sum C_i$ problems are NP-Hard as parallel problems are generalizations of one machine problems. Problems with preemption must however be investigated. For the $P|\sum \varphi_i \leq \Phi_x, pmtn|C_{\max}$ problem we have to schedule several tasks at the same time such that the sum of their power needs $\sum \varphi_i$ is lower than the available power Φ_x in each interval.

If the power needed by the tasks is the same ($P|\sum \varphi_i \leq \Phi_x, \varphi_i = \varphi, pmtn|C_{\max}$), then the problem is simple: in a given interval we execute as many tasks as possible in parallel provided that the power Φ_x and the constraint on the number of cores P are respected. Then, at the end of a task, we schedule another one and, at the end of the interval, we either stop tasks if there is less available power than before or start additional tasks if idle cores remain.

If the power needed by each task is different ($P|\sum\varphi_i \leq \Phi_x, pmtn|C_{\max}$), the problem is NP-Hard.

Theorem 4. *Minimizing the makespan of the schedule of a set of power heterogeneous preemptive tasks to run in a set of intervals ($P|\sum\varphi_i \leq \Phi_x, pmtn|C_{\max}$) is NP-Hard in the strong sense.*

Proof. The NP-Hardness of this problem will be demonstrated by proving that the simpler problem where the processing time of each task is one unit of time (ut) is NP-hard in the strong sense. The remainder of the proof is build on a similar pattern than used within the proof of the theorem 1.

Let us consider the following decision problem: given a horizon of K intervals of time Δ_k ($1 \leq k \leq K$) where their length δ_k is equal to one unit of time and where the available power is $\Phi(t) = \Phi_k = \Phi$ ($1 \leq k \leq K$) and given a processor with 3 cores that share the available power, is there a schedule that allocates tasks over time such that the power needed by the cores never exceeds Φ for every time interval Δ_k ($1 \leq k \leq K$)? In other words, if $\mathcal{T}_k \subset \mathcal{T}$ is the set of tasks that are scheduled within the time interval Δ_k, $\forall k \leq K$, is $\sum_{i|T_i \in \mathcal{T}_k} \varphi_i \leq \Phi_k = \Phi$? The problem is in NP: given a schedule of K time intervals, it is easy to check in polynomial time whether this schedule is valid or not. The NP-Completeness is obtained by reduction from 3-PARTITION [10] which is NP-Complete in the strong sense.

Let us consider an instance $\mathcal{I}_1$ of 3-PARTITION: given an integer B and $3K$ positive integers $a_1, a_2, \ldots, a_{3K}$ such that for all $i \in \{1, \ldots, 3K\}$, $B/4 < a_i < B/2$ and with $\sum_{i=1}^{K} a_i = KB$, does there exist a partition $I_1, \ldots, I_K$ of $\{1, \ldots, 3K\}$ such that for all $k \in \{1, \ldots, K\}$, $|I_k| = 3$ and $\sum_{i \in I_k} a_i = B$?

We build the following instance $\mathcal{I}_2$ of our problem with K time intervals, each interval Δ_k having a length of time $\delta_k = 1$ and with an available power $\Phi_k = \Phi = B$ for $1 \leq k \leq K$. There are $3K$ tasks T_i in $\mathcal{T}$ with $p_i = 1\,ut$ and $\varphi_i = a_i$ for all $1 \leq i \leq 3K = m$. Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does.

Suppose first that $\mathcal{I}_1$ has a solution. For $1 \leq k \leq K$, task T_i is assigned to $\mathcal{T}_k$ within the period k with $i \in I_k$ and $\varphi_i = a_i$. Then, we have $\sum_{i|T_i \in \mathcal{T}_k} \varphi_i = \phi_k = \sum_{i \in I_k} a_i = B$ and therefore the constraint on the demand is respected for the K time intervals. We have a solution to $\mathcal{I}_2$.

Suppose that $\mathcal{I}_2$ has a solution. Let $\mathcal{T}_k$ be the set of machines allocated to the period k such that for all tasks $T_i \in \mathcal{T}_k$ with $i \in I_k$, $\sum_{i \in I_k} \varphi_i = \Phi_k = \Phi = B$. Because of φ_i, $|\mathcal{T}_{\|}| = |I_k| = 3$. Since the available power Φ has to be consumed for the K time intervals to process the scheduled tasks, the solution is a 3-PARTITION.

We have proven that the problem where $\Phi_k = \Phi$ for every time interval Δ_k ($1 \leq k \leq K$) and $p_i = 1$ for every task $T_i \in \mathcal{T}$ ($1 \leq i \leq n$) is NP-Complete in the strong sense. Since this problem is a special case of the more general problem where the available power Φ_k during each of the time intervals Δ_k is different from each other and where the processing time p_i of each task T_i is also different from each other, it is sufficient to prove the NP-Hardness of this associated general optimization problem. This concludes the proof. □

Note that the proof highlights that the problem $P|\sum\varphi_i \leq \Phi, p_i = p, pmtn|C_{\max}$ is NP-Hard when the tasks have the same size ($p_i = p$).

For the flowtime objective, the $P|\sum\varphi_i \leq \Phi_x, pmtn|\sum C_i$ problem, we can differentiate the particular case where tasks have the same power need $\varphi_i = \varphi$ which is simple from the more general case where tasks have different power needs. In the $\varphi_i = \varphi$ case the SPT algorithm, completed to take both the available power Φ_x and the number of cores P constraints into account, gives an optimal solution even if the tasks have different sizes. Then the problem where the tasks have different power needs is NP-Hard as the problem $P|\sum\varphi_i \leq \Phi_x, p_i = p, pmtn|\sum C_i$ is equivalent to $P|\sum\varphi_i \leq \Phi_x, pmtn|C_{\max}$ since the tasks do not need to be ordered as they are of the same size. This implies that the more general case $P|\sum\varphi_i \leq \Phi_x, pmtn|\sum C_i$ is NP-Hard too.

Algorithm 1: $Place\_Task(p_i, \varphi_i, x)$

```
   Data:
      $\Phi_x$: useful power in interval $\Delta_x$ whose length is $\delta_x$
      $b_x, f_x$: resp. beginning and finish times of $\Delta_x$
      $nc_x$: number of available cores in $\Delta_x$
      $found$: boolean, init to $false$
   Result:
      boolean true if the task can be scheduled in the interval
1  if $(\varphi_i \leqslant \Phi_x) \wedge (nc_x > 0)$ then
2      $found \leftarrow true$
3      if $p_i \leqslant \delta_x$ then
4          Remove $\Delta_x$ from the interval list
5          Add new intervals in the interval list:
6          $\Delta_{x'} = \{[b_x, b_x + p_i[, \Phi_x - \varphi_i, nc_x\text{-}1\}$
7          $\Delta_{x''} = \{[b_x + p_i, f_x[, \Phi_x, nc_x\}$
8      else
9          repeat
10             Take next $\Delta_x$
11             if $(\Phi_x < \varphi_i) \wedge (nc_x > 0)$ then
12                 $found \leftarrow false$
13         until $\sum \delta_x \geq p_i$
14         if $found$ then
15             for Intervals $\Delta_x$ in the time slot except the
                last one do
16                 $\Delta_x = \{[b_x, end_x[, \Phi_x - \varphi_i, nc_x\text{-}1\}$
17             Remove the last interval $\Delta_y$ from interval list
18             Add new intervals in the interval list:
19             $\Delta_{y'} = \{[s_y, s_y + p_i - \sum_{z=x}^{y-1} \delta_z[, \Phi_x - \varphi_i\, nc_x\text{-}1\}$
20             $\Delta_{y''} = \{[s_y + p_i - \sum_{z=x}^{y-1} \delta_z, e_y[, \Phi_x, nc_x\}$
21 return $found$
```

5 Heuristics

In this section we propose heuristic algorithms to solve the general problems $P|\varphi_i \leq \Phi_x|C_{\max}$ and $P|\varphi_i \leq \Phi_x|\sum C_i$ when we have one machine with several cores. Most of the heuristics are adapted from classical scheduling algorithms to introduce the power constraints and time slots. Remember that time slots are sets of consecutive intervals. A time slot is considered available for scheduling a task if, in all the intervals that compose this time slot, the available power is higher than the task's power need and there is at least one available core. In all the algorithms the task to time slot assignment is done by the $Place\_Task()$ function, illustrated in Algorithm 1.

The $Place\_Task()$ function takes a task, its computing time p_i and its power need φ_i, and a starting interval x. It first checks that the proposed interval still has enough available power and cores (line 1). Then, if the task fits in the interval, the interval is split in two new intervals (lines 2-7), one for the part where the task is scheduled and one for the remaining part, to keep the count of the available power and cores. If the task length exceeds the interval the algorithm checks the available resources (cores and power) in the following intervals until the end of the task (lines 9-15). If such a time slot is found then the corresponding power and one free core are subtracted (lines 16-19). The last interval is split in two as previously (lines 20 to 24). Note that splitting the intervals allows that a task can always be scheduled at the beginning of an interval. The function returns true if the task is placed, false otherwise.

5.1 List Algorithms

We designed a first family of heuristics for both objectives, makespan and flowtime, based on list algorithms. Given a list of tasks, the list algorithm greedily schedules tasks in the earliest available intervals, seeking to minimize their completion time. Then by sorting the task list the algorithm may foster one or another task type. *Random* does not sort tasks. This naive approach is used to compare the other algorithms with a non smart solution. *LPT* (Largest Processing Time) sorts tasks by decreasing processing times. This solution fosters long tasks which are more difficult to place and usually gives good results for the makespan minimization on parallel identical machines when the number of tasks exceeds 50, as shown in [9]. *SPT* (Shortest Processing Time) sorts tasks by increasing processing times p_i, as for flowtime minimization an increasing p_i order must be preferred. *LPN* (Largest Power Need) sorts tasks by decreasing power need φ_i. Tasks with large power need are difficult to place in the time slots and scheduling them first may avoid using later slots. *LPTPN* (Largest Processing Time Power Need) sorts tasks by decreasing values of $p_i \times \varphi_i$. As both the processing time p_i and the power need φ_i are important values for scheduling the tasks taking them independently only fosters one and ignores the second, we combine both values.

Note that the two last algorithms (LPT and

LPTPN) are rather makespan oriented and they do not produce appropriate solutions for the flowtime minimization.

5.2 Binary Search Algorithms

The list based solutions fail to take into account how much extra power there is in a time slot compared to the task need. In fact it can only determine if the time slot is long (for the time) or large (for the power) enough for executing a given task or not. The use of time slots with high power levels to execute tasks that do not need a lot of power, might cause power waste. The next logical step was hence to reduce the power waste by placing each task in the time slot that has the closest available power level to its power demand, rather than placing it in the earliest one possible.

A key problem here is how to avoid scheduling a task towards the end of the runtime, because a time slot over there produces less power waste than many earlier ones. In other words, how to find the best fit for each task, without decreasing the quality of our solution (increasing $C_{\max}$).

The Binary Search family of heuristics is based on the Dual Approximation technique of Hochbaum and Shmoy [17]. This approach uses a time horizon to limit the search area. It is possible to schedule the tasks everywhere before the time horizon which allows to place tasks in the time slot where the power waste is the lowest. This solution has the advantage to take into consideration both power and performance constraints at the same time.

BSPW - Binary Search Power Waste At the beginning of the algorithm two time limits are set: the lower one, which must be lower or equal to the shortest possible schedule, and the higher one which is usually chosen such that we are sure that every algorithm will fit in the given time horizon. Here the lower value is set to $(\sum_{i=0}^{N} p_i)/NB$, where NB is the number of cores, as it is a lower bound. The higher value is set to $\mathcal{H}$, the time horizon. Then the algorithm attempts to reduce the time horizon on a binary search manner trying to fit all tasks in the new shorter time horizon. If they all fit, it re-reduces the searching area, if not, it increases it.

BSPW also works with a task list whose order impacts the algorithm performance. We use the same flavors of task ordering as with the list algorithms, which leads to the following variants of BSPW: BSPW-R (random), BSPW-LPT, BSPW-SPT, BSPW-LPN and, BSPW-LPTPN. Note that this method is assessed for both objectives to be minimized. In its design it however better supports the makespan objective as it targets the reduction of the time horizon and does not directly take the time ordering into account.

6 Experiments

In this part, we present the experiments conducted in order to assess the performance of the algorithms mentioned above. The presented experiments were realized using simulation rather than a real platform as running lots of real life experiments is costly and hence does not allow to explore a wide range of parameters. Furthermore we solely focus on synthetic workloads because real world traces like the parallel archive of Feitelson[1] do not include the power need for the tasks.

6.1 Simulation environment

We have implemented the different heuristics in Python[2]. Our code reads simple CSV files as input, these files contain the configuration values desired for tasks and intervals. Then, using Python, it randomly generates the necessary task lists and time slot lists before running the algorithms to compute the schedules. Finally, results are presented in different diagrams using R, in order to provide a clear comparison between algorithms.

6.2 Settings

The input of the heuristics is a list of tasks. In our simulations we did not use real data as input but we tried to use data sets that are as close as possible to real data collected in other experiments. For instance, p_i

[1] http://www.cs.huji.ac.il/labs/parallel/workload

[2] This source code is available on GitHub at `http://github.com/laurentphilippe/greenpower`

values of the generated task lists are randomly chosen using the hyper-gamma law suggested by Lublin and Feitelson in [21]. We also use random values following an exponential law for p_i in parts of the experiments to simplify them. When we use the exponential law for task generation, we define an upper bound $pmax$ and we set the mean value of this law to half of the $pmax$. In the experiments $pmax$ ranges from 10 to 100, by steps of 10.

The power consumption of the tasks is given in power units. As we were lacking values for the power consumption of the tasks we choose to use a random generation of φ_i with a uniform law between 0.1 and φmax. φmax ranges between 4 and 40 power units, by steps of 4.

For the experiments we choose intervals of equal length, 10 time units. To explore solar panel like power generation, we generate sets of intervals with a bell shape. For the interval generation we define 5 levels and we randomly generate the available power for each interval inside the level. To generate the bell shape we give a higher probability to increase (resp. decrease) the level when we are in an increasing (resp. decreasing) phase. The used random law in the levels is uniform, the maximum power that can be provided by the sources is 80 and each level has a height of 16.

For the experiments where the exponential law based task generation is used, we generate 250 intervals per set and we generate 600 intervals per set for the experiments that use the hyper-gamma law based task generation. Note that there is no guarantee, when we use a set of intervals, that a schedule can be found. For that reason we use large numbers of intervals.

For the experiments we generate 100 different sets of intervals but with the same parameters and for each couple of $pmax$ and φmax values, we generate 100 different sets of tasks. 10 000 experiments where thus performed, 100 for each $(pmax, \varphi max)$ couple, each with a different task set. The same interval set is used for each $(pmax, \varphi max)$ couple.

The number of cores is set to 8 for all the experiments, which means that up to 8 tasks can be scheduled in the same interval. To assess the impact of the available power on the algorithm performance we use two values for the available power, 40 and 80. As the φmax value ranges from 4 to 40, this means that the tasks may require up to 320 power units to run without constraint in the case where $\varphi max = 40$.

To compare the results we need normalized metrics. Raw makespan or flowtime values cannot be compared as they depend on the considered set of tasks and intervals. A set of larger tasks always gives a longer makespan than a set of shorter ones. Therefore we define the following metrics to compare the schedules: we define the makespan performance $PERMAK$ as $(makespan - useless)/\sum p_i$, where $makespan$ is obtained by the schedule and $useless$ is the sum of the length of the intervals, between 0 and the end of the schedule, where no task can be scheduled because of too low available power. In a same way we define $PERFLOW$ as the flowtime performance as $(\sum(C_i - useless_i))/\sum p_i$, where C_i is the completion time of task T_i and $useless_i$ is the sum of the lengths of the intervals, between 0 and C_i, where no task can be scheduled because of too low available power.

6.3 Results

In this section we present the results of our experiments. The figures are generated using R statistical environment.

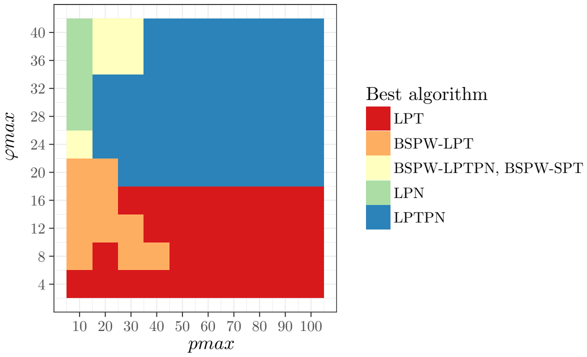


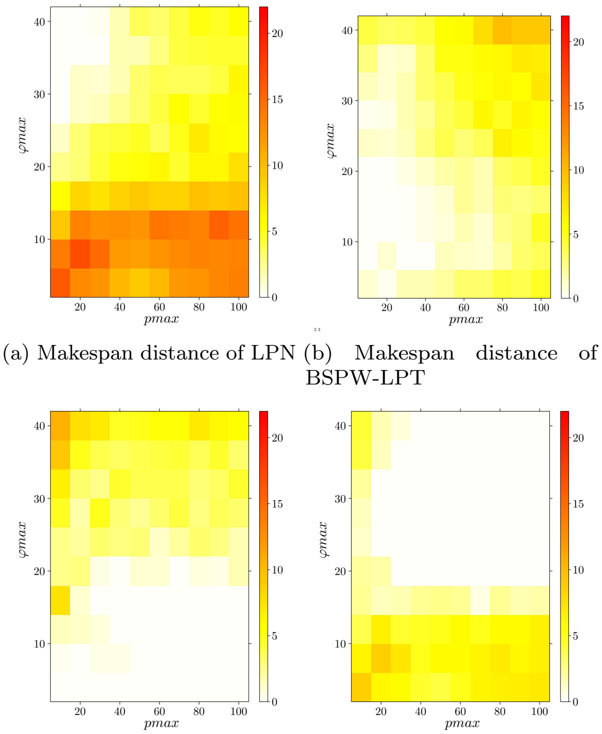
Figure 4: Heat map of the best average makespan performance $PERMAK$ for values of $pmax$ ranging from 10 to 100 and values of φmax ranging from 4 to 40.

We assess all the algorithms regarding both the makespan performance $PERMAK$ and the flowtime performance *PERFLOW*. Figure 4 presents the best algorithm for each value of p_i and φ_i. The best algorithm is defined as the algorithm that has the best mean *PERMAK* on the 100 simulation runs for a couple of values $(pmax, \varphi max)$. As we can see on Figure 4, the best algorithm depends on both the values p_i and φ_i. Unsurprisingly when the power need is low, the power is not a constraint and the LPT algorithm that fosters long jobs, gives the best results. We are, in this case, close to the classical $P||C_{\max}$ problem which is efficiently solved using LPT. However when φ_i is higher, algorithms that take power need into consideration achieve better results. Moreover, many algorithms get the best results depending on the power need. When the processing time is small, the LPN algorithm is the best but when it increases, LPTPN, which takes both processing time and power need into account, is better. For the case of a medium power need and small processing times, the BSPW family of algorithms finds good schedules.

We also compare the makespan performances of our heuristics to each other, more precisely we compute the makespan distance of each algorithm to the best solution. BSPW-LPN, BSPW-LPT, LPT and LPTP give more often the best performance and we present their distances in Figure 5. From Figures 5b and 5d we can see that the LPTPN and the BSPW-LPT algorithms generate schedules with makespan never farther than 6 % from the best one. This makes them good candidates for a global solution. Between them the LPTPN algorithm gives more often the best makespan. Unsurprisingly, the LPT algorithm which gives the best makespan when the power need is low, generates its worst schedules when the power need is high and the processing time of tasks are small. It is however never worse than 11 %. On the other hand Figure 5a shows that the LPN algorithm gives its best solutions when the power is limited and tasks are small.

Figure 6 gives the standard deviation of the makespan performance *PERMAK* for (*pmax*, φmax). As can be seen, the variation is low, ranging between 0.3 and 0.38. Other realized measures show that the variation may increase up to 0.8 but is always low.

We experimentally assessed the performance of our algorithms regarding flowtime *PERFLOW*. We tested all the algorithms for *pmax* ranging from 10 to 100 and φmax ranging from 4 to 40. SPT has always produced the lower flowtime.



(a) Makespan distance of LPN

(b) Makespan distance of BSPW-LPT

(c) Makespan distance of LPT

(d) Makespan distance of LPTPN

Figure 5: Distance of the mean makespan performance *PERMAK* of the LPN, BSPW-LPT, LPT, LPTPN algorithms to the best makespan performance.

Similarly to the makespan performance assessment, we analyze the distance between the flowtime performance *PERFLOW* of each algorithm from the best performance. The distances for BSPW-SPT, BSPW-R and Random, LPN are presented on Figure 7. From this figure we can note that a list algorithm with random task order has better probability to produce a good flowtime than BSPW algorithms or list algorithms with power need based task list order. These poor performance shows that the task ordering criterion largely impacts the flowtime. We can also note that the BSPW based algorithms have poorer

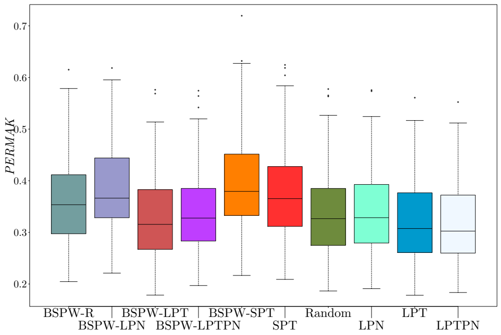


Figure 6: Standard deviation of the makespan performance *PERMAK* for $pmax = 100$ and $\varphi max = 20$.

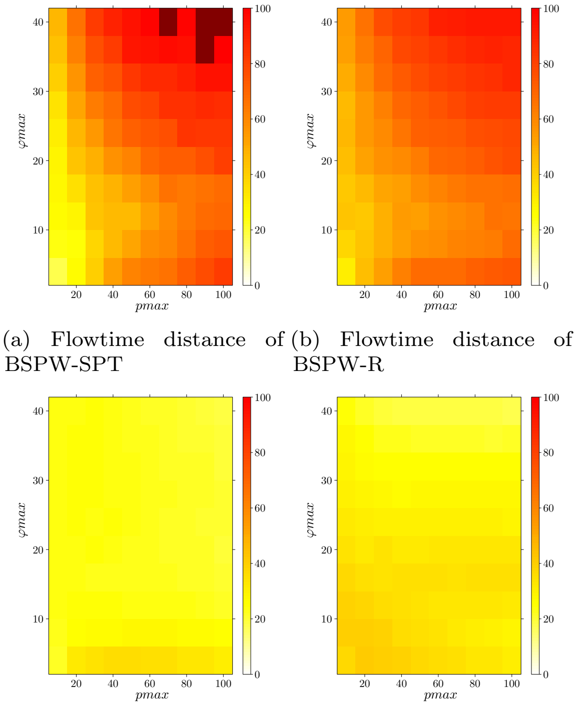
performance. This is because the BSPW algorithms first searchs for the shortest time horizon in which all the tasks can be scheduled, thus favouring the makespan, before taking the task order into account. These results highlights that the task order should be considered first to lower the flowtime.

Finally, we assess the algorithm performance regarding their computation time. Figure 9 shows a clear gap between the computation time of list and BSPW algorithms, and the gap expands for bigger tasks. This indicates that increasing p_i has a more negative effect on BSPW algorithms than on list algorithms regarding the computation time. The complexity of BSPW is an explanation. Our observations show that the complexity of both BSPW and list algorithms increases with the number of intervals.

Figure 8 gives the standard deviation for the flowtime for all the algorithms used in the experiments. The SPT algorithm gives the lowest deviation, lower than 10 %. As for the *PERFLOW* performance measure, the Random algorithm is ranked second with less than 12%. Globally the list based algorithms give lower variations than Binary Search algorithms. It means that the proposed algorithms are inefficient in this case.

Figure 9 gives the mean computation time for the 100 runs done with one given value of the power need φmax (20). Note that the LPTPN algorithm is barely visible on the plot as it gets the same running times as SPT and the curves overlap. If we except the BSPW-R algorithm, the power need value only slightly impacts the computation time that slightly decreases when the φ_i value increases. On the contrary when p_i increases, the computation time increases too. It is also clear that the BSPW family of algorithms has a larger computation time than the list based family. Indeed as they iterate on the horizon value and, at each iteration, they apply the same type of algorithm as the list based algorithms. From Figure 9, we can see that BSPW-R generates the longest execution times, 10 times more than the fastest algorithm, i.e., LPT.

It is worthwhile noting that discretizing the time in intervals increases the complexity of the algorithms. The worst case complexity of the *Place_Task*() function depends on X, the number of intervals. As the



(a) Flowtime distance of BSPW-SPT
(b) Flowtime distance of BSPW-R
(c) Flowtime distance of Random
(d) Flowtime distance of LPN

Figure 7: Distance of the mean flowtime performance *PERFLOW* of the Random LPN, BSPW-LPN and BSPW-SPT algorithms to the best makespan performance.

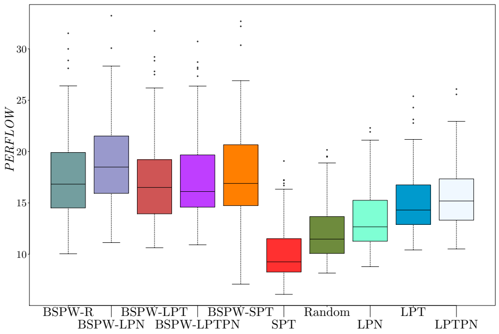


Figure 8: standard deviation of the flowtime performance $PERFLOW$ for $pmax = 100$ and $\varphi max = 20$.

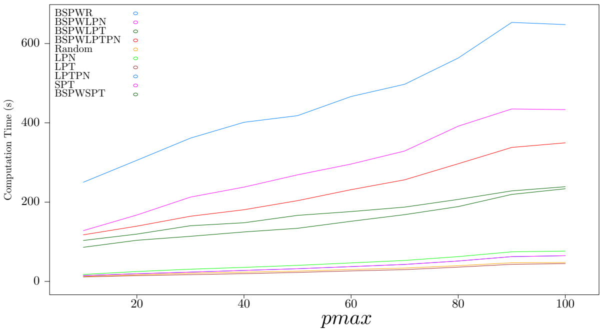


Figure 9: Computation time of the algorithms for $\varphi max = 20$.

algorithms iterate on the intervals to schedule a task then their complexity depends on X^2. Considering that the complexity of a list based algorithm depends on the list ordering ($O(n \log(n))$, n the number of tasks), the complexity of the list based algorithms with power constraints becomes $O(X^2 n \log(n))$. It turns out that the complexity of placing tasks in intervals heavily weighs on the computation times: running the whole set of experiments to compute the heatmaps based on 250 intervals takes 2 days and more than one week when based on 500 intervals.

As a synthesis of this performance assessment, LPTPN generates the best performance for the makespan objective on many values and it generates schedules never worse than 5 % of the best ones while keeping reasonable computation times. This approach is a good candidate in the general case. On the other hand, a multi-policy algorithm that differently orders tasks within the list depending on the p_i and φ_i could also be implemented to improve the performance.

7 Conclusion

In this paper we have tackled the problem of scheduling independent tasks under power constraints. For the cases where parallel tasks are independent, we have proven that the static problems of minimizing the makespan and the flowtime without preemption are NP-Hard. On the other hand, considering preemption, we have proven that the one machine problem with makespan minimization is polynomial while parallel problems are NP-Hard. To complete the study we assess heuristics in the case of a parallel shared memory machine model. Comprehensive simulations show that algorithms that take both task processing time and power constraints into account globally provide the best results.

The first simulation results show that simple algorithms provide rather good solutions compared to more sophisticated ones such as binary search based algorithms for this complex problem. For future work we plan to integrate meta-heuristics such as genetic algorithms and design smarter algorithms. We also plan to create semi-synthetic workloads by using workload traces from parallel archives and adding synthetic power needs.

The current results are limited to shared machine problems. For distributed machines static power consumption and on/off powering of machines must be taken into consideration. This will also be explored in the follow up of our work.

Acknowledgments

This work was supported in part by the ANR DATAZERO (contract “ANR-15-CE25-0012”) project and by the Labex ACTION project (contract “ANR-11-LA BX-01-01”). Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté – Besançon.

References

[1] Pragati Agrawal and Shrisha Rao. Energy-efficient scheduling: Classification, bounds, and algorithms. *CoRR*, abs/1609.06430, 2016.

[2] Baris Aksanli, Jagannathan Venkatesh, Liuyi Zhang, and Tajana Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11. ACM, 2011.

[3] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2), 2014.

[4] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 364–367. Society for Industrial and Applied Mathematics, 2006.

[5] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. *Polynomial Time Algorithms for Minimum Energy Scheduling*, pages 136–150. Springer Heidelberg, 2007.

[6] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Trans. Algo.*, 8(3), 2012.

[7] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10. IEEE Computer Society, 2010.

[8] Peter Brucker. *Scheduling Algorithms*. Springer Heidelberg, 2007.

[9] Stéphane Chrétien, Jean-Marc Nicod, Laurent Philippe, Veronika Rehn-Sonigo, and Lamiel Toch. Using a sparse promoting method in linear programming approximations to schedule parallel jobs. *Concurrency and Computation: Practice and Experience*, 27(14):3561–3586, 2015.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979.

[11] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732 – 749, 2011. Special Issue on Cloud Computing.

[12] Y. Georgiou, D. Glesser, and D. Trystram. Adaptive resource and job management for limited power consumption. In *IEEE IPDPS Workshop*, May 2015.

[13] Inigo Goiri, Md E. Haque, Kien Le, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Matching renewable energy supply and demand in green datacenters. *Ad Hoc Networks*, 25, Part B, 2015.

[14] Inigo Goiri, Kien Le, Md E. Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenslot: Scheduling energy consumption in green datacenters. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, 00, 2012.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5(2), 1979.

[16] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the intel haswell processor. In *IEEE IPDPS Workshop*, 2015.

[17] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling

problems theoretical and practical results. *J. ACM*, 34(1):144–162, January 1987.

[18] Ayham Kassab, Jean-Marc Nicod, Laurent Philippe, and Veronika Rehn-Sonigo. Scheduling independent tasks in parallel under power constraints. Technical Report 8426, Institut FEMTO-ST, 2017.

[19] Tarandeep Kaur and Inderveer Chana. Energy efficiency techniques in cloud computing: A survey and taxonomy. *ACM Comput. Surv.*, 48(2), 2015.

[20] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. *SIGMETRICS Perform. Eval. Rev.*, 40(1):175–186, June 2012.

[21] Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63(11):1105–1122, 2003.

[22] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, 2014.

[23] Eduard Oró, Victor Depoorter, Albert Garcia, and Jaume Salom. Energy efficiency and renewable energy integration in data centres. strategies and modelling review. *Renewable and Sustainable Energy Reviews*, 42, 2015.

[24] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan. A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters. *Future Generation Computer Systems*, 37:141 – 147, 2014.