Green Power Constrained Scheduling for Sequential Independent Tasks on Identical Parallel Machines

Ayham Kassab, Jean Nicod, Laurent Philippe, Veronika Sonigo

Green Power Constrained Scheduling for Sequential Independent Tasks on Identical Parallel Machines

Ayham Kassab, Jean-Marc Nicod, Laurent Philippe, Veronika Rehn-Sonigo
FEMTO-ST Institute, UBFC / ENSMM / CNRS
F-25000 Besançon, France
firstname.lastname@femto-st.fr

December 19, 2019

Abstract

Energy consumption is a major aspect to consider when designing large scale high performance computing HPC systems. Integrating renewable energy sources to the power supply of an HPC system is an efficient solution to lower its carbon footprint, but it is one faced by challenges since the power production of most renewable sources is variant while the power consumption of the system varies with the workload. Advanced workload management techniques in combination with powering down idle machines allow to increase the efficiency of this solution. We tackle here the problem of scheduling independent tasks on a multi-machine platform that is exclusively run with green energy. We propose different power constrained scheduling algorithms, and evaluate them through an experimental study on an HPC model that considers the possibility to switch machines on or off.

keywords: Green power, Constrained scheduling, Parallel computing

1 Introduction

The growing dependency on Information and Communication Technology (ICT) services is pushing service providers to build bigger and more energy consuming ICT structures in order to maintain the quality of their services. Studies estimate that by 2030 ICT will be responsible for about 20% of global electricity consumption [1] and can produce up to the quarter of the CO_2 emissions worldwide [AE15].

As a result, the energy efficiency of large scale ICT structures has been the focus of many research projects over the last years. Most of these efforts focus on reducing the system's energy consumption. However, even if the energy consumption is reduced, a system still has a carbon footprint as long as it relies on brown energy. Another "greener" approach consists of reducing the carbon footprint by integrating renewable (CO_2–free) energy sources, such as solar panels, wind turbines or fuel cells, in the system's power supply.

Renewable sources can be used either by signing green energy contracts with an electricity provider or by installing on-site renewable sources which, unlike a green contract, guarantees an exclusively green energy supply. Using on-site renewable energy sources however presents a challenge as both the system's workload and its power supply are intermittent and vary over time. This means that the availability of the power supply does not guarantee to allow the execution of all the submitted computational demands at a certain time. As a consequence the need for new workload and machine management solutions has arisen in order to optimally use the instantaneous available green power. The main challenge is to cope with the available power constraints. In [KNPRS17], we proved that scheduling independent tasks within a given power

[1] https://theshiftproject.org

envelope on one machine is already NP-hard. On multi-machine platforms, a machine may also be switched on or off to avoid that it consumes static power when idle.

This work tackles the problem of scheduling tasks on a multi-machine HPC platform where the computing power comes from renewable sources. Therefore we consider an available predicted power envelope that varies over time. The parallel computational platform consists of multiple multi-core machines that can be turned on or off separately. The workload is a set of independent sequential tasks. The contributions of the paper are the following:

- We propose and discuss different scheduling heuristics for HPC systems that take power constraints into consideration. We assess their performance in extensive simulations with different workload models.
- We show the impact of the on/off model for the machine management on the scheduling solutions.
- We provide a comparison of the heuristics to use depending on the parameters of the scheduling situation (available power envelope and task characteristics).

The rest of this paper is organized as follows. Studies on scheduling problems in parallel HPC platforms and in systems that run on renewable power are presented in Section 2. The models used in the paper are detailed in Section 3. Section 4 presents the power constraint aware heuristics. In Section 5 we discuss the experimental study and the obtained results. Finally section 6 presents our conclusion and future work.

2 Related Work

This work addresses HPC systems powered by renewable energy sources as a way to reduce ICT's carbon footprint. Its aim is to find an appropriate scheduling strategies to carry out the workload management process in such systems. Even if our contribution targets parallel architectures of HPC systems, this section focuses on research work that tackles scheduling problems in large scale ICT structures.

The three most common types of large scale ICT structures are data centers, HPC systems and clouds. These systems differ from one another in architecture, IT equipment and types of submitted jobs. Some research works about scheduling in data centers and cloud environments are based on task duplication [AK98] or virtualization techniques [BL09] to reduce energy consumption. Mathew et al. have compared and analyzed scheduling strategies used in data centers [MSJ14], some of which may have guarantees of robustness [GHBRK12] or optimality [AJ+14]. The addressed problem is multi-objective for which Iturriaga et al. have proposed in 2016 scheduling strategies for green-powered federation of data centers [INTD16]. In this context many scheduling criteria are studied as the number of dead line violations (SLA) [NPTP18] or the request flow [KSTI11].

In the HPC context, where tasks are compute intensive, the makespan is a decisive criterion. In some cases, as grids or cloud computing environments, HPC systems are considered as heterogeneous platforms. before the extensive use of graphical processor, researchers were already interested in scheduling tasks on heterogeneous parallel platforms [WSRM97, THW02]. In this context, theoretical results are proposed on the complexity of tasks scheduling algorithms [IT07]. Some works consider homogeneous platforms [HAR94, Wal10] since they are a more relevant model for HPC clusters. We thus also consider such homogeneous systems for the experimental part of our work that target HPC clusters.

The focus of the paper is also on power constraints when scheduling computational tasks. Previous work in this context has payed attention on reducing either power consumption [VAN08, Bun09] or energy consumption [ABRGR15, WKC+13] of an HPC platform. Minimizing the power or energy consumption in HPC systems is however complex since it is conflicting with the makespan criterion. It leads to multi-objective optimization problems [BTW13] that are even more difficult to solve than the previous ones and only provide a trade-off [LLTW12, PJZA07]. Moreover optimizing the power or energy consumption does not always mean a reduction of the ICT platform because

of the rebound effect. [Pie11] highlighted that reducing power consumption may lead to increasing the number of tasks to be processed. On the other hand, when considering the energy exclusively coming from renewable sources, the available power at any given time is a more important factor than the total available energy on a time range since the renewable energy is available anyway and theoretically free. In this case, the power production is limited and intermittent. The challenge is to optimize the use of an available power envelope when scheduling tasks as soon as possible and it can also been solved by minimizing the makespan [KNPRS17, KNPRS18].

GreenSlot [GHL+15] manages parallel batch jobs in a data center powered by both a green energy source and the electrical grid. The jobs are divided into critical and non-critical and the scheduler exploits the flexibility of non-critical jobs to produce a schedule matching with the renewable power production. The system is however connected to the grid to avoid deadline violations when the renewable power supply is not high enough. Caux et al. propose a genetic-based algorithm to minimize violations of task due dates in a cloud environment [CRS18], while respecting the renewable power envelope and the resource constraints [CRGRS18] and without considering the grid power. Lei et al. propose in [LWZ+16] a genetic algorithm for solving a multi-objective energy-efficient scheduling problem on a data center that is partially powered by renewable energy sources. The proposed algorithm addresses the minimization of both the makespan and the total energy consumption objectives. The work presented here tackles HPC systems solely powered by on-site renewable sources. The objective is not to reduce the energy consumption but rather to minimize the makespan under power availability constraints.

3 Model

We define here the elements of the studied model.

Power model: as the power provisioning of the platform solely comes from renewable sources, its production is not stable and varies over time. For technical reasons the power has to be maintained at a constant level (setpoint), at least during a time interval. The available power is thus modeled by X time intervals Δ_x of length δ_x with an available power of Φ_x. This power is shared by all the machines of the platform whose cumulative power consumption during a time interval Δ_x is constrained by the level of available power Φ_x.

Task model: we consider a set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of n tasks T_i. According to [SS09, GGT15], the power consumption varies from one task to another depending whether the task intensively computes or not. Each task T_i is defined by its processing time p_i and its power consumption φ_i.

Machine model: the platform is a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of m machines M_j, each with *nb_cores* cores. Since the total available power is variable and constrained, machines are powered off as soon as they become idle or when the available power is too low. Powering on a machine (Son_j) takes a delay Ton_j during which the power consumption of the machine equals to Pon_j. When running, a machine consumes at least its static power $Pstat_j$. Then, it takes a delay $Toff_j$, during which the power consumption of the machine equals to $Poff_j$, to switch off the machine ($Soff_j$). The cores of a machine are available only when this machine is powered on.

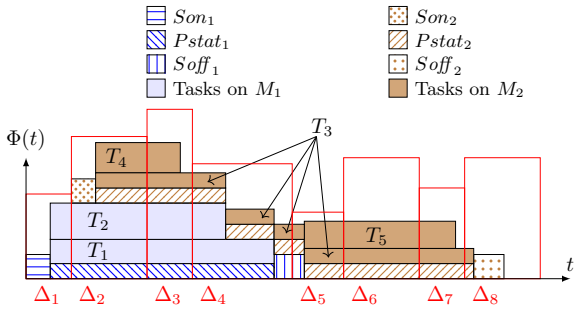

Figure 1: Task scheduling on machines under power constraints

Problem definition: the problem tackled here is to statically schedule the tasks on parallel machines under limited power constraints, without preemption, to minimize the total completion time $Cmax$. Note that, for simplicity reasons, we consider sequential independent tasks in order to have a simple task model without communications nor precedence relations and identical machines. One of the interests of this model is to consider switching machines on and off when the power supply is not high enough to turn on all the machines.

Figure 1 illustrates the case where five tasks are to be run on a platform with dual-core machines. In interval Δ_1 there is enough power to start machine M_1 and run tasks T_1 and T_2 but not to switch a second machine on. In interval Δ_2, as the available power increases, machine M_2 is started but only the tasks T_3 and T_4 can be run within the power envelope. Once machine M_1 is finished with its tasks, it is switched off since the available power does not allow to run tasks on it. Once it is off, the released power allows to run task T_5 on M_2.

Complexity: in [KNPRS17] we prove that finding the shortest makespan when scheduling a set of independent sequential tasks on a multi-core machine with power constraints is a NP-Hard problem. Since this problem is an instance of the problem studied here (special case where $Ton_j = Pstat_j = Toff_j = 0, \forall 1 \leq i \leq m$) then this problem is also NP-Hard.

4 Heuristics

In this section we propose heuristics to compute schedules for the studied problem. Two classes of heuristics are proposed, list based heuristics and power envelope partition based heuristics. Both kinds of heuristics use the same task to interval assignment algorithm that also manages the on/off switching of machines.

4.1 Task to interval and machine assignment

Algorithm 1: The $PlaceTask()$ function

```
1  PlaceTask (T_i)
   Data: Boolean placed, init to false
   Result: sTime, time when the task is started
2  begin
3      x ← 0
4      while ¬ placed ∧ x ∈ X do
5          machine ← findAvailableMachine(M, Δ_x)
6          if machine ≠ nil then
7              placed ← checkPowerAvaibility(Δ_x, p_i, φ_i)
8          else
9              machine ← newMachine(M)
10             placed ← checkPowerAvaibility(Δ_x, Ton_j, Pon_j) ∧ checkPowerAvaibility(Δ_x + Ton_j, p_i,
                φ_i + Pstat_j)
11         if ¬ placed then x ← x + 1
12     if machine.state = Soff then
13         machine.state ← on
14         sTime ← Δ_x
15     if machine.state = Soff then
16         machine.state ← on
17         sTime ← Δ_x + Toff_j
18     return sTime
```

At the beginning of the run time, all machines are switched off. The decision to switch a machine on or off is based on two factors: the computational demand and the power supply constraints. The task to interval assignment is done by the $PlaceTask$ function. This function is responsible for task to machine allocation, in addition to task to interval assignment, assuming that a machine can be in one of 4 states: *on*, *off*, Son (switching on) or $Soff$ (switching off). The

function takes one task at a time and schedules it as early as possible on the first suitable machine under the following rules (see Algorithm 1):

- No preemption nor job migration are allowed, therefore if a task starts executing on a machine, it should finish the execution on the same machine. The *checkPowerAvaibility* function checks the availability of cores and power during enough time ahead to execute the task.
- If no core is available, a machine must be started, if possible. The function first checks that the power envelope is enough to start the machine then if it is possible to run the task after this boot time with enough power for both the machine static power consumption $Pstat_j$ and the task power consumption φ_i.

After placing a task, the scheduler looks for possible optimizations in the start and stop times. Leaving a machine on reduces the delays and energy losses caused by its booting and shutdown while keeping a machine on for too long might lead to unnecessary static power waste. The machine should stay turned on just enough time to finish the execution of all tasks, the issue here is thus how to determine in advance how long that period is before executing the tasks.

- If a task is assigned to a machine in $Soff$ state, performing a shutdown, during the first interval of its execution Δ_{start}, the shutdown is canceled, and the execution of the task then starts as soon as possible.
- If a task is assigned to a machine that is switching on or is in an "off" state at Δ_{start}, then a delay is applied while the machine boots, and the execution of that task starts at $\Delta_{start} + Ton_j$.
- To avoid unnecessarily reboots of machines, before switching a machine M_j to off, the $PlaceTask$ function looks ahead in $Toff_j$ for a machine switch on, if yes, the machine is kept on. Symmetrically, when switching a machine on, the function looks back for a switch off in Ton, if yes, then the machine is kept on.

4.2 List based heuristics

List scheduling, one of the most common algorithms in scheduling, can give good solutions within low time complexity for many scheduling problems. A list algorithm consists of two steps: sorting the tasks based on a priority value then allocating the tasks, according to this order, to a resource and schedule it for execution as early as possible. The second step is carried out by the $PlaceTask$ function.

Our problem can be described as an optimization problem whose objective (makespan minimization) lay on a time axis, while the scheduling constraints (the power availability versus the power consumption) lay on a power axis. It is thus a two dimensional problem (time and power dimensions). For this reason we test priorities that consider the time axis, priorities that consider the power axis and priorities that consider both axis simultaneously. The list based heuristics that use those priorities are the following: **LPT** for Largest Processing Time (p_i) first, **LPN** for Largest Power Need (φ_i) first, **LPTPN**, for Largest Processing Time times Power Need first ($p_i \times \varphi_i$). The **TwoQs** heuristic, or Two Queues, is an adaptation of a list heuristic that alternates between two queues, namely LPTPN's queue and LPT's queue. This heuristic exploits the advantages of two priority assignments at the same time, privileging tasks that would have low priority in one queue but might have high priority in the other. The **LPP** heuristics, for Least Possible Places first, calculates the number of time slots in which it is possible to execute each task, then, it sorts the tasks in the queue starting with the task with the Least Possible Places first. This prioritizes tasks that are harder to schedule regardless of whether it is because they have long processing times p_i or high power consumption φ_i or both. We finally test the case where the task list is not ordered with the **Random** heuristic.

The list heuristics use a criterion to sort a task. We also propose a genetic algorithm that tries to find the best list order without any criterion. An individual is the order of the task list

and the algorithm just evaluates the list orders and mixes them. The genetic algorithm proposed in [KNPRS18] repeatedly applies genetic operators to modify a set of individuals. The initial population is composed of individuals generated with the list heuristics plus random individuals. First, a selection process is performed to chose individuals from the current population. The fitness of an individual is the $Cmax$ resulting from using the task list order in the schedule. Then, the task list order of chosen individuals are shuffled by applying mutation and crossover operators. Between two generations a selection operator is applied. Several selection, mutation and cross-over operators are evaluated but we only use Order Crossover(OX) with wheel selection here.

4.3 Power envelope partition based algorithms

List algorithms place tasks one after another and do not consider grouping tasks to better use the two dimensions of the problem. We present here algorithms that generate power envelope partitions and group tasks in these partitions.

The first proposed approach, called **Binary search**, tries to power on the least machines as possible to spare energy consumed by Son_j and $Soff_j$. It is based on a binary search to find the best schedule. The starting points of the search are time 0 and a schedule that uses only one machine (time horizon). The algorithm then tries to find a solution by reducing the time horizon using a binary search scheme. In a time horizon the algorithm powers a first machine on, schedules as much tasks as possible on it, then powers a second machine on, schedules tasks and so on until either all the tasks are scheduled or a task cannot be scheduled. If all the tasks are scheduled the algorithm reduces the time horizon, else it increases it, using the binary search scheme. Binary search thus provides an acceptably time costly way to determine the point at which a machine is switched off, which corresponds to the minimum time horizon under which all tasks can be executed.

The second approach, called **Stripe**, tries to group tasks vertically. It first switches a machine on to schedule one task. It starts with the longest task T_i to guarantee that the rest of the tasks can fit (on the time axis) in this range as well. Then it schedules as much tasks as possible in the time interval $[\Delta_{start}, \Delta_{start}, +p_i]$, where Δ_{start} is the time where the machine is ready to run tasks, powering on new machines when necessary and possible.

To evaluate the effect of task list orders on the last two approaches **BSLPT** (Binary Search - LPT), **BSLPN**, **BSLPTPN**, **BSLPP**, **BS2Qs**, **stripeLPT**, **stripeLPTPN**, **stripe2Qs** and **stripeLPP** are deployed.

5 Experiments

In this section we present an experimental study of the proposed heuristics. We have developed a python simulator [2] presented in [KNPRS17] that offers a wide range of experimental setups on the parallel computational platform. The simulator takes as input a set of tasks and the green power production, expressed by a set of intervals of time with the available power level at each interval.

5.1 Experimental settings

The platform consists of several identical multi-core machines. We assume that the number of machines available in the platform equals to the length of the task list. This way the availability of machines is guaranteed which exempts the machine as a resource constraint, and we focus on variable green power constraints. Other experiments can be done in the future to consider both constraints at the same time. The number of cores in each machine is $nb\_cores = 4$. Based on the values measured in [DGG+17], the static power consumption of each machine $Pstat$ is set to 95 W, the time to switch a machine on $Ton_j = 150$ s and the time to switch a machine off $Toff_j = 6$ s. The power consumption during the start up and the switch off are set to $Pon_j = 125$ W and $Poff_j = 100$ W since they are usually higher than the static power consumption.

[2] Source code available at `http://github.com/laurentphilippe/greenpower`

To carry out our simulations we must produce the data that represents the two main input of the scheduling problem, namely the task list and the power envelope.

A realistic model is used to generate the power envelope that represents the various renewable power supply based on real data. This model [HNP17, HNVP19] provides hourly Φ_x values according to a number of wind turbines and an area of solar panels. In order to examine the effect of the power availability on the heuristics' performance, we conduct several tests with increasing maximum power Φ_x in each generated set of power envelopes. We start by setting the power generation parameters as follows: solar panels area = $2000\,m^2$, $pvEfficiency = 0.163$, number of wind turbines = 1, and $turbinePowerNominal = 350$, where $pvEfficiency$ and $turbinePowerNominal$ are the efficiency factors of the solar panels and wind turbines respectively. These settings yielded power envelopes with a maximum power Φ of about 350 W, we then multiply the values of these settings by 2, 5 and 10 times to yield a maximum Φ_x of around 700, 1750 and 3500 W respectively. The data used in this realistic model is collected every hour, therefore, we partition the power envelope into unified intervals Δ_x with length of $\delta = 1$ time unit = 1/10 hour. Hence each 10 consecutive intervals have the same available power level Φ_x. Each power envelope is a list of 10000 intervals.

We use two kind of task data sets for the task processing time p_i. The firsts, the hyper-gamma sets, use the model proposed in [LF03]. Based on real workloads, the processing time distribution of HPC tasks is modeled using a hyper-gamma law with ($\alpha_1 = 4.2,\ \beta_1 = 0.94$) and ($\alpha_2 = 312,\ \beta_2 = 0.03$). Although realistic, this model is based on workload logs collected from only three sites which limits the experiments to the range of values of these samples. Therefore, to explore a wider set of parameters, we also use an exponential distribution law to generate p_i, between $p_{\min}$ and $p_{\max}$ time units, to produce the synthetic workloads for the exponential sets. Since we did not find any model nor real data for the task power consumption values, we use random generation of φ_i, between $\varphi_{\min}$ and $\varphi_{\max}$, with a uniform law, in both sets.

5.2 Evaluation metric

To compare the results of simulations over different data sets with different characteristics we need a metric. Raw makespan values cannot be compared as they depend on the considered set of tasks and intervals. A set of long tasks always gives a longer makespan than a set of shorter ones. Normalizing the makespan values with the processing times is however not sufficient since the schedule also depends on the available power. Therefore, we propose a metric, called *NM* (normalized metric), that provides a fair comparison regardless of the power envelope. The idea is to compare the distance of the makespan to the best makespan obtained with the same data sets and to normalize this difference to the power envelope size.

$$NM = \sum out \div \sum total$$

Where $\sum out$ is the sum of the $p_i \times \varphi_i$ area of all the tasks that are scheduled after the best $Cmax$, and $\sum total$ is the total $p_i \times \varphi_i$ area of the task list. Good schedules have low the NM values and the best have a value of 0. This metric is fair because all heuristics are compared to each other on the same data sets and the result is normalized on both the processing time and the power consumption. We take the average of this metric value for each heuristics over 150 executions.

5.3 Results

We conduct two experiments that combines the two kinds of data sets with realistic power envelopes.

5.3.1 Hyper-gamma tasks and realistic power envelopes

In this experiment we use a hyper-Gamma distribution law to generate each task processing time p_i that are close to real data. Each task list contains 50 tasks with a range of processing times between 1 and 500 time units ($1\,t.u. = 6\,min$). In the absence of actual data for task power consumption, each φ_i for each task is randomly chosen with a uniform distribution law between 1 power unit and $\varphi_{\max}$. In this experiment $\varphi_{\max}$ ranges between 15 and 150 power units, by steps of 15.

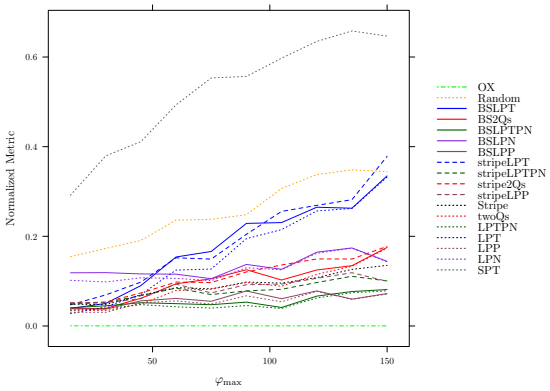

Figure 2: Average NM, $\Phi_{\max} = 350$

Figure 2 presents the average value of NM for each heuristic with increasing value of maximum task power consumption $\varphi_{\max}$ on the horizontal axis. The maximum available power Φ is 350 W. We notice that with the increase in $\varphi_{\max}$, the performance of most heuristics decreases, especially the heuristics that use LPT to sort the task list (blue lines). We also point out that heuristics that take the task power consumption into consideration when sorting the task list such as twoQs (red lines), LPTPN (green lines) and LPP (brown lines) outperform the LPT based heuristics when the power constraints are strict, which indicates that the power constraints are the most important criterion in this case. The reason is that the available power is not enough to execute all tasks simultaneously, which makes the decision of which task to execute first very important.

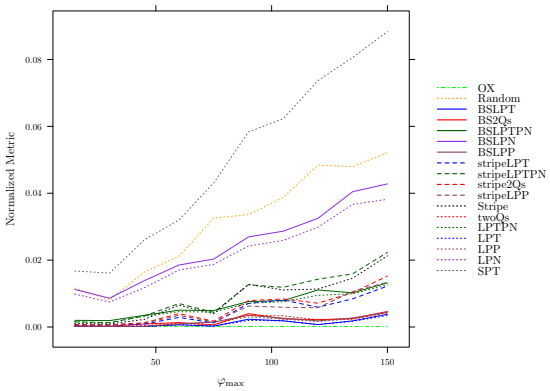

Figure 3: Average NM, $\Phi_{\max} = 3500$

Figure 3 presents the case where the maximum available power Φ is 3500 W. In this case, the tasks are less concurrent for power due to high level of available power. This problem is thus closer to the classical scheduling problem without power constraints, in which case LPT provides good performance regarding minimizing the makespan. We notice a degradation in the performance of heuristics that take the task power consumption into account such as LPN (purple lines) and LPTPN (green lines).

We point out that the light green dash-dotted line in Figures 2 and 3 represents the NM of the tested GA, which uses the Order Crossover (OX) with wheel selection. We notice that this line always has the value of $NM = 0$. Since the NM value of each heuristics during one simulation is calculated in relation to the best $Cmax$ produced in for simulation, this means that GA always produces the shortest makespan. The difference in performance between GA and the other heuristics increases as power constraints become stricter. On the other hand, when the power envelope provides plenty of power for task execution, all the heuristics that use LPTPN,

twoQs, LPT and LPP orders provide a very close performance level to GA for tasks with low power consumption. For tasks with higher power consumption, the performance of stripe based heuristics degrades in comparison to BS and list based heuristics as we notice that LPT, BSLPT, twoQs, BS2Qs and LPP keep the closest distance to GA's performance line. In addition, even list and BS heuristics, that use LPTPN for sorting the task list, provide worse performance when the task power consumption gets higher than $50\,t.u.$ We can then draw the conclusion that in the least strict power constraint case, the GA does not significantly improve solutions provided by LPT, twoQs and LPP for tasks that does not require a lot of power. Indeed the task processing time is a more relevant prioritizing criteria than the task power consumption even for tasks with high power consumption. We finally point out that LPN which only prioritizes the task power consumption does not perform well for the makespan minimization objective, neither does SPT which is designed for flowtime minimization. In fact, other than SPT, all the tested heuristics improve the produced schedule which is found when keeping the task list in its original random order, which proves the necessity for smart workload management in this problem.

Table 1: Computation time $\Phi_{\max} = 350$

Algo	OX	BSLPT	stripeLPT	LPT
Time (s)	24674.80	87.08	67	3.39
Algo	LPP	BS2Qs	stripe2Qs	2Qs
Time (s)	35.68	89.83	65.34	3.59

Table 2: Computation time $\Phi_{\max} = 3500$

Algo	OX	BSLPT	stripeLPT	LPT
Time (s)	6157.18	49.04	0.71	0.32
Algo	LPP	BS2Qs	stripe2Qs	2Qs
Time (s)	34.59	49.35	0.81	0.35

Tables 1 and 2 present the average computation time of heuristics in both cases where GA is tested. We chose LPT and twoQs orders as a representation of each heuristics class to compare to the tested GA and to LPP. We notice that GA takes much higher computation time to find a schedule. We also point out that the computation time of GA in the case where it performs much better than the other heuristics ($\Phi_{\max} = 350\,\mathrm{W}$) is up to 5 times higher than its computation time in the case where its performance is closer to other heuristics ($\Phi_{\max} = 3500\,\mathrm{W}$). This means that when the scheduling problem is easier (less strict power constraints), the GA reaches its stopping conditions faster (25 iterations without improvement). In general, even fast list based heuristics need ten times the computation times when the power constraints are strict to place all tasks in comparison to their computation time in the ($\Phi_{\max} = 3500\,\mathrm{W}$) case. We finally point out that LPP is the only heuristics that does not show significant difference in computation time between the two cases of power envelope. However, it still takes 10 times longer as any other list based heuristics, due to the process of calculating in how many time slots each task can fit, before sort the task list. This process increases the time complexity of LPP which is however indifferent of the data set characteristics. In general, we notice that the computation time is rational to the length of the schedule. This is logical because the longer the schedule, the more time intervals there are to explored to find it. This is coherent with the observation that both LPP's performance and computation time are consistent with the power envelope variability. Finally we show that list based heuristics can find a schedule 60 % faster than stripe heuristics and up to 99 % faster than binary search heuristics. The tested GA on the other hand is around 18000 times slower than list heuristics while it offers a 0.05 % improvement over the performance of the best list based heuristics.

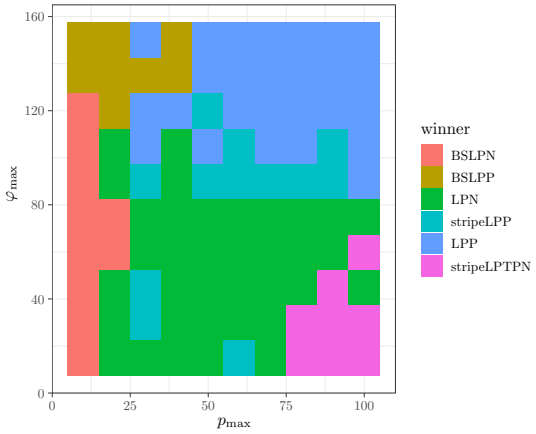

Figure 4: Best average *NM*, $\Phi_{\max} = 350$.

Table 3: Number of on/off, $\Phi_{\max} = 3500$

Algo	OX	Random	BS2Qs	LPT
$\varphi_{\max} = 15$	12.5	12.7	12.1	11.9
$\varphi_{\max} = 50$	12.1	12.3	11.5	11.45
$\varphi_{\max} = 100$	11.2	11.5	10.25	10.2
$\varphi_{\max} = 150$	10.2	10.5	2.55	2.5

Table 3 presents how many times each heuristics switches a machine on and off on average when $\Phi_{\max} = 3500\,\mathrm{W}$. We notice that when the available power is much higher than the average of task power consumption, the number of on/off operations increases. However, when $\Phi_{\max} = 350\,\mathrm{W}$ the available power level is not enough to run more machines even when the task power consumption is at its lowest. In comparison to the average *NM* results, we notice that the heuristics that perform less on/off operations provide a better value for *NM*.

5.3.2 Exponential tasks and realistic power envelope

In this experiment we use an exponential distribution law to generate p_i values between $p_{\min}$ and $p_{\max} \in [10, 100]$ with a step of $10\,t.u.$ This variation in $p_{\max}$ provides more test cases than the simulation using hyper-gamma distribution laws. As for the maximum power consumption we set $\varphi_{\max} \in [15, 150]$ with a step of 15 power units.

Figure 4 plots the heuristics with the best average *NM* when the maximum available power Φ is set to 350 W, with increasing value of maximum task power consumption $\varphi_{\max}$ on the vertical axis and increasing value of maximum task processing time $p_{\max}$ on the horizontal axis.

We can distinguish two main areas on this heat-map figure. When the task power consumption $\varphi_{\max} \leq 75\,W$, LPN heuristics give the best performance in most cases. Unlike its poor performance with hyper-gamma tasks, LPN is able to provide good results in this experiment because of the shorter processing times, in comparison to the average processing time of around $500\,t.u.$ in Experiment 1. The task power consumption becomes more important in this case. The right lower pink corner represents the case where the processing time increases enough to give the stripe heuristics with LPTPN order the edge. The stripe heuristics hence outperforms BS and list heuristics for task sets with long processing times and low power consumption. On the other hand, for tasks with short processing times, BS heuristics outperform both list and stripe heuristics (the left orange column). The second main area is defined by tasks with a power consumption $\varphi_{\max} > 75\,W$. In this case the power constraints become too strict and only the heuristics using LPP ordering are able to find the best makespan. Once again, for shorter tasks, BS heuristics outperforms list and stripe heuristics.

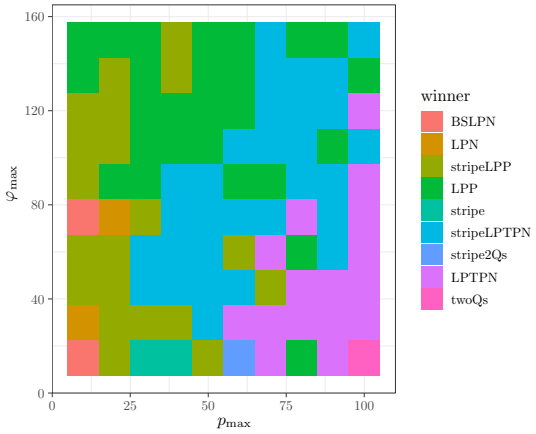

Figure 5: Best average *NM*, $\Phi_{\max} = 700$.

Figure 5 presents the heuristics with the best average *NM* when the maximum available power Φ is set to 700 W. We notice that the area where LPTPN based heuristics provide the best performance expands in comparison to the little pink corner from Figure 4. The area covers now the diagonal right half of the heat-map, while the performance of the LPN sorting policy drastically decreases as under less strict power constraints, the task power consumption becomes less relevant against the task processing time. The stripe heuristics using LPTPN order outperforms the list based LPTPN when the task power consumption is closer to their processing time. We conclude that the stripe heuristics performs better with square shaped tasks. The upper left half of the heat-map where tasks have high power consumption and processing times $p_i \leq 60\,t.u.$ is mostly dominated by LPP. When the processing time is below $25\,t.u.$, stripeLPP gives the best performance in most of the test cases. We notice a degradation in the performance of BS heuristics in comparison to the case of $\Phi = 350\,W$. Furthermore we observe that when power constraints are less strict, lower complexity heuristics such as list algorithms and stripe algorithms provide good solutions. However, in the hard cases of the problem, only more complex algorithms can give the best solution in some test cases. This once more asserts the need for advanced workload management solutions in an HPC system powered solely by renewable power. However, the time complexity of the solution should be taken into account when finding the schedule in these systems. The computation times of the different heuristics in this experiment follow the same ranking given in Tables 1 and 2 of Experiment 1.

We point out that in both previous experiments, the tested GA provides the best average *NM* in all the test cases. Therefore, the results presented in Figures 4 and 5 exclude GA from the comparison in order to compare the performance of the rest of the heuristics. The mean values of the *NM* for the other heuristics on the whole experiments range between a maximum degradation of 60 % (LPT) and a lower one of 6 % (LPN) in the low power case. Surprisingly the Random algorithm did not get the worst mean value with only 24 %. In the high power case, the mean values range between 19 % (BSLPT) and 3.6 % (LPP). These results clearly show that the used heuritics must be adapted to the avalaible power.

5.4 Summary

Our results show that choosing the right order to sort the task list depends on the characteristics of both the submitted task list (how much power does the execution of tasks consume ?) and the power envelope (how strict are power constraints ?). When the available power level is not high enough to run all the machines of a platform simultaneously, deploying a good on/off mechanism does not only save static energy consumption, but can also lead to better makespan due to less rebooting delays.

6 Conclusion

In this paper we present and test several scheduling heuristics to solve the optimization problem of task scheduling on HPC platforms powered solely by renewable energy sources. We develop a scheduler to control the process of switching machines on and off, taking account both time and power costs. We showed the importance of the initial order of the task list to schedule and the necessity for smart workload management in order to optimize the performance of the platform under power constraints.

For future work we plan to use actual HPC workload traces integrating the energy consumption of the tasks to test our approach. We also aim at investigating the use of energy storage into the power management process. This will allow to store unused power and to use it at some later moment to cope with insufficient power production from the intermittent energy sources.

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

[ABRGR15] Guillaume Aupy, Anne Benoit, Paul Renaud-Goud, and Yves Robert. Energy-aware algorithms for task graph scheduling, replica placement and checkpoint strategies. In *Handbook on Data Centers*, pages 37–80. Springer, 2015.

[AE15] A. SG Andrae and T. Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[AJ+14] Dr Agarwal, S. Jain, et al. Efficient optimal algorithm of task scheduling in cloud computing environment. *arXiv preprint arXiv:1404.2076*, 2014.

[AK98] I. Ahmad and Y.-K. Kwok. On exploiting task duplication in parallel program scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 9(9):872–892, 1998.

[BL09] Tom Brey and Larry Lamers. Using virtualization to improve data center efficiency. *the green grid, whitepaper*, 19, 2009.

[BTW13] P. Balaprakash, A. Tiwari, and S. M. Wild. Multi objective optimization of hpc kernels for performance, power, and energy. In *PMBS'13 Int. workshop*, pages 239–260. Springer, 2013.

[Bun09] David P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, Oct 2009.

[CRGRS18] S. Caux, P. Renaud-Goud, G. Rostirolla, and P. Stolf. It optimization for datacenters under renewable power constraint. In *Euro-Par 2018*, pages 339–351, Cham, 2018. Springer.

[CRS18] S. Caux, G Rostirolla, and P. Stolf. Smart Datacenter Electrical Load Model for Renewable Sources Management. In *ICREPQ'18*, volume 16, 2018.

[DGG+17] P.-F. Dutot, Y. Georgiou, D. Glesser, L. Lefevre, M. Poquet, and I. Rais. Towards energy budget control in hpc. In *17th IEEE/ACM CCGrid'17*, pages 381–390, 2017.

[GGT15] Y. Georgiou, D. Glesser, and D. Trystram. Adaptive resource and job management for limited power consumption. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 863–870, may 2015.

[GHBRK12] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012.

[GHL+15] Íñigo 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*, 25:520–534, 2015.

[HAR94] Edwin SH Hou, Nirwan Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, 5(2):113–120, 1994.

[HNP17] M Haddad, J.-M. Nicod, and M.-C. Péra. Hydrogen infrastructure: data-center supply-refueling station synergy. In *14th IEEE VPPC 2017*, page 6, Belfort, France, dec 2017.

[HNVP19] M. Haddad, J/-M. Nicod, C. Varnier, and M.-C. Péra. Mixed integer linear programming approach to optimize the hybrid renewable energy system management for supplying a stand-alone data center. In *IEEE IGSC'19*, USA, oct 2019.

[INTD16] S. Iturriaga, S. Nesmachnow, A. Tchernykh, and B. Dorronsoro. Multiobjective workflow scheduling in a federation of heterogeneous green-powered data centers. In *ACM/IEEE CCGrid 2016*, pages 596–599, 2016.

[IT07] E Ilavarasan and P Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer sciences*, 3(2):94–103, 2007.

[KNPRS17] A. Kassab, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo. Scheduling independent tasks in parallel under power constraints. In *46th ICPP'17*, pages 543–552, jul 2017.

[KNPRS18] A. Kassab, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo. Assessing the use of genetic algorithms to schedule independent tasks under power constraints. In *IEEE HPCS'18*, pages 252–259, 2018.

[KSTI11] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis. Schedule Optimization for Data Processing Flows on the Cloud. In *ACM SIGMOD'11 Management of Data*, 2011.

[LF03] 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.

[LLTW12] Tak-Wah Lam, Lap-Kei Lee, Isaac K. K. To, and Prudence W. H. Wong. Improved multi-processor scheduling for flow time and energy. *Journal of Scheduling*, 15(1):105–116, Feb 2012.

[LWZ+16] H. Lei, R. Wang, T. Zhang, Y. Liu, and Y. Zha. A multi-objective co-evolutionary algorithm for energy-efficient scheduling on a green data center. *Comp. & Op. Research*, 75:103–117, 2016.

[MSJ14] T. Mathew, K. C. Sekaran, and J. Jose. Study and analysis of various task scheduling algorithms in the cloud computing environment. In *ICACCI 2014*, pages 658–664. IEEE, 2014.

[NPTP18] S. C. Nayak, S. Parida, C. Tripathy, and P. K. Pattnaik. An enhanced deadline constraint based task scheduling mechanism for cloud environment. *Journal of King Saud University-Computer and Information Sciences*, 2018.

[Pie11] Jean-Marc Pierson. Green task allocation: taking into account the ecological impact of task allocation in clusters and clouds. *Journal of Green Engineering*, 1(2):129–144, 2011.

[PJZA07] Soyeon Park, Weihang Jiang, Yuanyuan Zhou, and Sarita Adve. Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):169–180, 2007.

[SS09] C. Stewart and K. Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *W. on Power Aware Comp. and Sys.*, pages 15–19. IEEE, 2009.

[THW02] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS*, 13(3):260–274, 2002.

[VAN08] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of hpc applications. In *ISC'08*, pages 175–184, 2008.

[Wal10] R. Walter. *Analyzing Various Aspects of Scheduling Independent Jobs on Identical Machines*. Logos Verlag Berlin GmbH, 2010.

[WKC+13] Lizhe Wang, Samee U Khan, Dan Chen, Joanna KołOdziej, Rajiv Ranjan, Cheng-Zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661–1670, 2013.

[WSRM97] L. Wang, H. Ja. Siegel, V. P Roychowdhury, and A. A Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of parallel and distributed computing*, 47(1):8–22, 1997.