Scheduling on power-heterogeneous processors

Susanne Albers, Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli, Richard Stotz

Scheduling on Power-Heterogeneous Processors*

Susanne Albers† Evripidis Bampis‡ Dimitrios Letsios§
Giorgio Lucarelli¶ Richard Stotz†

Abstract

We consider the problem of scheduling a set of jobs, each one specified by its release date, its deadline and its processing volume, on a set of heterogeneous speed-scalable processors, where the energy-consumption rate is processor-dependent. Our objective is to minimize the total energy consumption when both the preemption and the migration of jobs are allowed. We propose a new algorithm based on a compact linear programming formulation. Our method approaches the value of the optimal solution within any desired accuracy for a large set of continuous power functions. Furthermore, we develop a faster combinatorial algorithm based on flows for standard power functions and jobs whose density is lower bounded by a small constant. Finally, we extend and analyze the AVerage Rate (AVR) online algorithm in the heterogeneous setting.

1 Introduction

Nowadays energy consumption of computing devices is an important issue in both industry and academia. One of the main technological alternatives in order to take into account the energy consumed in modern computer systems is based on the use of speed-scalable processors where the speed of a processor may be dynamically changed over time. When a processor runs at speed s, then the rate with which the energy is consumed (i.e., the power) is $f(s)$ with f being a non-decreasing function of the speed. According to the well-known cube-root rule for CMOS devices, the speed of a device is proportional to the cube-root of the power and hence $f(s) = s^3$. However, the standard model that is usually studied in the literature considers that the power is $f(s) = s^\alpha$ with $\alpha > 1$ a constant. Other works consider that the power is an arbitrary convex function [7, 9].

The algorithmic study of this area started with the seminal paper of Yao et al. [20], where a set of jobs, each one specified by its work, its release date and its deadline, has to be scheduled preemptively on a single processor so that the energy consumption is minimized. In [20], an optimal polynomial-time algorithm has been proposed for this problem, while

*A preliminary version has been presented to LATIN 2016
†Fakultät für Informatik, Technische Universität München, Germany
‡Sorbonne Universités, UPMC Univ. Paris 06, UMR 7606, LIP6, Paris, France
§Université Côte d'Azur, Inria, CNRS, I3S, Sophia Antipolis, France
¶Université Grenoble-Alpes, INP, UMR 5217, LIG, France

Li et al. [19] proposed an optimal algorithm with lower running time. The *homogeneous multiprocessor* setting in which the *preemption* and the *migration* of the jobs are allowed has been also studied. Chen et al. [12] proposed a greedy algorithm if all jobs have common release dates and deadlines. Bingham and Greenstreet [11] presented a polynomial-time algorithm for the more general problem with arbitrary release dates and deadlines. Their algorithm is based on solving a series of linear programs. Since the complexity of this algorithm can be high for practical applications, Albers et al. [1] and Angel et al. [3], independently, have been interested in the design of a combinatorial algorithm. Both works are based on the computation of several maximum flows in appropriate networks.

Albers et al. [1] have also considered the online version of the multiprocessor problem and they studied two well-known algorithms, namely the *Optimal Available (OA)* and the *Average Rate (AVR)*, which have been proposed by Yao et al. in [20] for the single-processor setting. Specifically, they proved that OA is α^α-competitive and that AVR is $(2^{\alpha-1}\alpha^\alpha + 1)$-competitive. Note that, for the single-processor case, the competitive ratio of OA cannot be better than α^α [10], while the lower bound for AVR is $2^{\alpha-1}\alpha^\alpha$ [8].

When migrations of jobs are not allowed, the problem is strongly $\mathcal{NP}$-hard even in the special case with homogeneous processors and unit work jobs [2]. Furthermore, there exists a $(1+\epsilon)^\alpha \widetilde{B}_\alpha$-approximation algorithm, where α is the maximum power exponent and $\widetilde{B}_\alpha$ is the α-generalized Bell number [6]. When both migrations and preemptions are forbidden, the problem is strongly $\mathcal{NP}$-hard even in the case with a single processor [4]. In this setting, the approximability of the problem is an open question. For the special case with homogeneous processors, there exists a $O((w_{\max}/w_{\min})^\alpha)$-approximation algorithm [5] as well as a quasi-polynomial time approximation scheme (QPTAS) producing a $(1+\epsilon)$-approximate solution in $n^{O(\text{polylog}(n))}$ time [18].

In this paper, we consider the problem of scheduling a set of jobs on a set of *power-heterogeneous* processors when the preemption and the migration of the jobs are allowed. In our setting, each processor P_p is characterized by its own power function. This means that if a processor P_p runs at speed s, then its power is given by a non-decreasing function $f_p(s)$. The motivation to study power-aware scheduling problems is based on the need for more efficient computing. Indeed, parallel heterogeneous systems with multiple cores running at lower frequencies offer better performances than a single core. However, in order to exploit the opportunities offered by the heterogeneous systems, it is essential to focus on the design of new efficient power-aware algorithms taking into account the heterogeneity of these architectures. In this direction, some recent papers [6, 16, 17] have studied the impact of the introduction of the heterogeneity on the difficulty of some power-aware scheduling problems. Especially in [16], Gupta et al. show that well-known priority scheduling algorithms that are energy-efficient for homogeneous systems become energy inefficient for heterogeneous systems.

For the case where job migrations are allowed and the heterogeneous power functions are convex, an algorithm has been proposed in [6] that returns a solution within an additive factor of ϵ far from the optimal and runs in time polynomial to the size of the instance and to $1/\epsilon$. This result generalizes the results of [1, 3, 7, 11] from the homogeneous setting to the heterogeneous one. However, the algorithm proposed in [6] is based on solving a configuration linear program using the Ellipsoid method. Given that this method may not

be very efficient in practice, we focus on other approaches. We first propose a polynomial-time algorithm based on a compact linear programming formulation which solves the problem within any desired accuracy. Our algorithm does not need the use of the Ellipsoid method like in [6] and it applies for a large family of continuous non-decreasing power functions.

The above result leaves open a natural question: *is it possible to generalize the flow-based approach used in [1, 3] for the homogeneous multiprocessor problem to the power-heterogeneous case?* This question is interesting even for standard power functions of the form $f_p(s) = s^{\alpha_p}$. This last case is the goal of the second part of our paper. However, when power-heterogeneous processors are considered some structural properties of the optimal schedules of the homogeneous case are no longer valid. For instance, in the heterogeneous setting, in any optimal schedule, the speed of a job is not necessarily unique, but it may change when parts of the same job are executed on different processors. A second difficulty comes from the fact that, while in the homogeneous case the processor on which a job is executed at a given time has no influence on the energy consumption, this is a crucial decision when scheduling on heterogeneous multiprocessors. Here, we overcome these subtle difficulties and propose a max-flow based algorithm which is more complicated than its homogeneous counterpart (for example, the network formulation is more enhanced). In particular, we show that it produces a solution arbitrarily close to the optimal for jobs whose density is lower bounded by a small constant; this constant depends on the exponents of the power functions. The above assumption ensures that no job is processed with a speed less than one by any processor and allows us to solve the problem by performing maximum flow computations in a principled way. Note that this assumption is reasonable in practice because the speed of a processor is multiple CPU cycles per second.

The third part of our paper is devoted to the analysis of the well known online algorithm AVR. Our analysis simplifies the analysis in [1] for the homogeneous case and allows us to extend it in the power-heterogeneous setting. Specifically, we prove that Heterogeneous-AVR is $((1+\epsilon)(\rho+1))$-competitive algorithm for arbitrary power functions, where ρ is the worst competitive ratio of the single-processor AVR algorithm among all processors. This turns to be $((1+\epsilon)(\alpha^{\alpha}2^{\alpha-1}+1))$-competitive algorithm for standard power functions of the form $f_p(s) = s^{\alpha_p}$, where α is the maximum power exponent among all processors.

In the following section we formally define our problem and we give the notation that we use. In Section 3, we present our LP-based algorithm, while in Section 4 we describe a flow-based combinatorial algorithm. Finally, the Heterogeneous-AVR and its analysis are given in Section 5.

2 Problem Definition and Notations

An instance of the heterogeneous speed-scaling problem consists of a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be executed by a set of m parallel speed-scalable power-heterogeneous processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job J_j is specified by an amount of work w_j, a release time r_j and a deadline d_j. We say that J_j is *alive* during an interval of time I, if $I \subseteq [r_j, d_j)$. Moreover, we define the *density* of a job J_j as $\delta_j = \frac{w_j}{d_j - r_j}$.

The speed of a processor can be varied over time and it corresponds to the amount of work that the processor executes per unit of time. Furthermore, the power of processor P_p

(i.e. its instantaneous energy consumption) is assumed to be a function $f_p(s)$ of its speed. We consider two classes of functions:

1. *Arbitrary Power Functions*: The function $f_p(s)$ of each processor P_p is an arbitrary and continuous function of s. However, we require an oracle for computing $f_p(s)$ in polynomial time, for any value of s.

2. *Standard Power Functions*: Each processor P_p satisfies the power function $f_p(s) = s^{\alpha_p}$, where $\alpha_p > 1$ is a small constant. This is the usual assumption in the speed-scaling literature. Note that, we denote by α the maximum power exponent, i.e., $\alpha = \max_{p \in \mathcal{P}}\{\alpha_p\}$.

During an interval of time I, the energy consumption of P_p is $\int_I f_p(s_{p,t})dt$, where $s_{p,t}$ is the speed of P_p at $t \in I$. The objective is to find a minimum energy schedule such that every job J_j is executed during the interval $[r_j, d_j)$. Preemptions and migrations of jobs are allowed, which means that a job may be executed, suspended and resumed later from the point of suspension on the same or on a different processor. However, we do not allow parallel execution of a job, i.e., each job may be executed by at most one processor at each time.

We define the important times $t_1 < t_2 < \ldots < t_\ell < t_{\ell+1}$ which correspond to all the different possible release dates and deadlines of jobs, sorted in increasing order. Moreover, let $I_i = [t_i, t_{i+1})$, for $i = 1, 2, \ldots, \ell$ and $\mathcal{I}$ be the set of all I_i's. We denote by n_i the number of jobs which are alive during I_i. Then, for each interval $I_i \in \mathcal{I}$, we define $m_i = \min\{m, n_i\}$. Furthermore, we denote by $\mathcal{J}(t)$ and $\mathcal{J}(I)$ the set of the alive jobs at time t and during the interval I, respectively. At a given time t, we say that processor P_p is *occupied* if it executes some job, or we say that it is *idle*, otherwise. For a given schedule $\mathcal{S}$, we define by $E(\mathcal{S})$ the total energy consumption of $\mathcal{S}$. Finally, we denote by $\mathcal{S}^*$ an optimal schedule and by $OPT = E(\mathcal{S}^*)$ its energy consumption.

3 LP-based Algorithm for Generalized Power Functions

In this section we present a linear program (LP) for the heterogeneous speed-scaling problem for a wide family of continuous power functions. Our formulation is more compact than the configuration LP proposed in [6] which contains an exponential number of variables and requires the use of the Ellipsoid method. Moreover, the formulation in [6] is polynomially solvable only for convex functions.

In order to define our LP, we discretize the possible speed values. Let s_{LB} and s_{UB} be a lower and an upper bound, respectively, on the speed of any processor in an optimal schedule. For example, we could choose $s_{LB} = w_{min}/[m \sum_i |I_i|]$ and $s_{UB} = \sum_j w_j d_j / |I_{min}|$. Given a constant $\epsilon > 0$, we geometrically discretize the interval $[s_{LB}, s_{UB}]$ and we define the set of discrete speeds $D = \{s_{LB}, s_{LB}(1+\epsilon), s_{LB}(1+\epsilon)^2, \ldots, s_{LB}(1+\epsilon)^k\}$, where $k = \min\{i : s_{LB}(1+\epsilon)^i \geq s_{UB}\}$. The set D contains $O(\frac{1}{\epsilon}\log(\frac{s_{UB}}{s_{LB}}))$ different speeds.

We consider a wide class of continuous power functions satisfying the following invariant: for any speed value $s \in [s_{LB}, s_{UB}]$ and small constant $\epsilon > 0$, there exists a sufficiently

small constant $\epsilon' > 0$ such that $f((1+\epsilon)s) \leq (1+\epsilon')f(s)$. Note that ϵ' is a characteristic of the function f. For example, in the case of standard power functions, we have that $f((1+\epsilon)s) \leq (1+\epsilon')f(s)$ with $\epsilon' = (1+\epsilon)^\alpha - 1$. In the reminder of this section, we consider this kind of functions.

Lemma 1. *There exists a $(1+\epsilon')$-approximate schedule such that, at each time, the speed of every processor belongs to the discrete set D, where $|D| = O(\frac{1}{\epsilon}\log(\frac{s_{UB}}{s_{LB}}))$.*

Proof. Consider an optimal schedule $\mathcal{S}^*$. Starting from $\mathcal{S}^*$, we produce a new schedule $\mathcal{S}$ as follows. For each processor P_p and time t, P_p executes the same job at t in both schedules but its speed is rounded up to the closest speed in the discrete set D. Clearly, the new schedule is feasible because at least w_j units of work are executed for each job J_j. Let $s_{p,t}$ and $s^*_{p,t}$ be the speed of processor P_p at time t in schedule $\mathcal{S}$ and $\mathcal{S}^*$, respectively. For simplicity, assume that the earliest release time is at time 0 and that the latest deadline is equal to T. Then, for any small constant $\epsilon > 0$,

$$\begin{aligned} E(\mathcal{S}) &= \sum_{p=1}^{m}\int_0^T f_p(s_{p,t})dt \leq (1+\epsilon')\sum_{p=1}^{m}\int_0^T f_p\left(\frac{s_{p,t}}{1+\epsilon}\right)dt \\ &\leq (1+\epsilon')\sum_{p=1}^{m}\int_0^T f_p(s^*_{p,t})dt = (1+\epsilon')E(\mathcal{S}^*) \end{aligned}$$

where $\epsilon' > 0$ is a small constant. The first inequality comes from the fact that $f_p(s)$ satisfies our assumption and the second inequality holds because $f_p(s)$ is increasing. □

The feasibility of our LP formulation is based on the following lemma.

Lemma 2. *Consider a schedule $\mathcal{S}$ and let $t_{i,j,p,s}$ be the total amount of time that job J_j is processed during the interval I_i by the processor P_p with speed s. Then, $\mathcal{S}$ is feasible if and only if all the following hold.*
– *$\sum_{i,p,s} t_{i,j,p,s} \cdot s \geq w_j$, for each job J_j,*
– *$\sum_{p,s} t_{i,j,p,s} \leq |I_i|$, for each interval I_i and job J_j, and*
– *$\sum_{j,s} t_{i,j,p,s} \leq |I_i|$, for each interval I_i and processor P_p.*

Proof. If all the conditions of the statement are satisfied, there exists a feasible schedule which can be constructed by producing a partial schedule for each interval I_i as follows. During I_i, we assume that each job J_j consists of m operations, where the i-th operation has processing time $\sum_s t_{i,j,p,s}$ and it must be entirely executed by processor P_p. Given that the last two properties are satisfied, such a schedule can be constructed by the algorithm of Gonzalez and Sahni [14] for the well known preemptive open shop problem. The operation of job J_j on processor P_p during the interval I_i is executed with speed s for $t_{i,j,p,s}$ units of time, for every $s \in D$. By the first property, we get that each job is entirely executed.

To the other direction, assume that at least one of the properties is not true. If the first property is not true, then at least one job is not entirely executed. If the second property does not hold, then it is not possible to construct a schedule such that each job is executed by at most one processor at each time. Finally, if the third property is not true, then we cannot produce a schedule in which each processor executes at most one job per time. □

Let $E_{p,s} = f_p(s)$ be the power consumption of processor P_p if it runs with speed s. We introduce a variable $x_{i,j,p,s}$ which corresponds to the total amount of time that the job J_j is processed during the interval I_i by the processor P_p with speed s. Then, we obtain the following LP:

$$\begin{aligned}
\min \sum_{i,j,p,s} x_{i,j,p,s} \cdot E_{p,s} & \\
\sum_{i,p,s} x_{i,j,p,s} \cdot s \geq w_j & \qquad \forall j \\
\sum_{p,s} x_{i,j,s,p} \leq |I_i| & \qquad \forall i,j \\
\sum_{j,s} x_{i,j,p,s} \leq |I_i| & \qquad \forall i,p \\
x_{i,j,p,s} \geq 0 & \qquad \forall i,j,p,s
\end{aligned}$$

Given a solution of the above LP, we obtain an operation of job J_j on processor P_p with processing time $\sum_s x_{i,j,p,s}$ during each interval I_i. So, for each I_i, we obtain an instance of the preemptive open shop problem, which can be solved in polynomial time with the algorithm of Gonzalez and Sahni [14]. This observation implies an algorithm for our problem, and the following theorem holds.

Theorem 1. *There is an algorithm which produces an $(1+\epsilon')$-approximate schedule in $O(poly(n, m, \frac{1}{\epsilon}, \log \frac{s_{UB}}{s_{LB}}))$ time.*

4 Flow-based Algorithm for Standard Power Functions

In this section, we first characterize the structure of an optimal solution for the heterogeneous speed-scaling problem with power functions of the form $f_p(s) = s^{\alpha_p}$ and jobs whose density is lower bounded by a small constant, which is defined below. Then, we derive a combinatorial algorithm based on flow computations.

4.1 Structure of an Optimal Schedule

We elaborate on the structure of a specific optimal schedule and we derive a set of properties and lemmas which are always satisfied by this optimal schedule. Since we allow preemptions and migrations of jobs, more than one processors may execute part of one job J_j. Due to convexity of the power functions, in any minimum energy schedule, the part of job J_j assigned to processor P_p is executed (preemptively) with constant speed $s_{j,p}$. Of course, a job may be executed with different speeds by different processors. However, the following lemma shows that these speeds are related through the derivatives of the power functions.

Lemma 3. *For each job $J_j \in \mathcal{J}$ which is partially executed by the processors P_p and P_q with speeds $s_{j,p}$ and $s_{j,q}$, respectively, it holds that $f'_p(s_{j,p}) = f'_q(s_{j,q})$.*

Proof. Assume that J_j is executed by P_p and P_q during two disjoint intervals I and $\tilde{I}$, respectively. Let w be the amount of work of J_j executed during these intervals, i.e. $w = |I| \cdot s_{j,p} + |\tilde{I}| \cdot s_{j,q}$. Note that $0 < s_{j,p} < \frac{w}{|I|}$ and $0 < s_{j,q} < \frac{w}{|\tilde{I}|}$. Moreover, the energy consumption for the execution of w is equal to

$$|I| \cdot f_p(s_{j,p}) + |\tilde{I}| \cdot f_q(s_{j,q}) = |I| \cdot f_p(s_{j,p}) + |\tilde{I}| \cdot f_q\left(\frac{w - |I| \cdot s_{j,p}}{|\tilde{I}|}\right)$$

We will show that the above expression is minimized when $f'_p(s_{j,p}) = f'_q(s_{j,q})$. Consider the function $g(s) = |I| \cdot f_p(s) + |\tilde{I}| \cdot f_q\left(\frac{w-|I|\cdot s}{|\tilde{I}|}\right)$ with domain of definition the interval $\left[0, \frac{w}{|I|}\right]$. By the derivation of $g(s)$, we get $g'(s) = |I| \cdot \left[f'_p(s) - f'_q\left(\frac{w-|I|\cdot s}{|\tilde{I}|}\right)\right]$, for which it holds that $g'(0) < 0$ and $g'(\frac{w}{|I|}) > 0$. Since the functions $f_p(s)$ and $f_q(s)$ are convex, the functions $f'_p(s)$ and $f'_q(s)$ are increasing with s. That is, the function $g'(s)$ is also increasing with s in the interval $\left[0, \frac{w}{|I|}\right]$. Therefore, by equating $g'(s)$ with zero, we get that $g(s)$ is minimized for the unique value of s satisfying $f'_p(s) = f'_q\left(\frac{w-|I|\cdot s}{|\tilde{I}|}\right) = f'_q(s)$. □

The above lemma describes the relation of the speeds of a job on different processors. Based on this, we define the *hypopower* of a job $J_j \in \mathcal{J}$ as $Q_j = f'_p(s_{j,p})$, for every $P_p \in \mathcal{P}$. The following property is a corollary of Lemma 3.

Property 1. *Each job $J_j \in \mathcal{J}$ is executed with constant hypopower Q_j.*

The following property implies that the jobs which are executed at each time are the ones with the greatest hypopowers.

Property 2. *For each pair of jobs $J_j, J_k \in \mathcal{J}$ and time $t \in [r_j, d_j) \cap [r_k, d_k)$ such that J_j is executed at t and J_k is not executed at t, it holds that $Q_j \geq Q_k$.*

Proof. Assume for the sake of contradiction that the property does not hold. Then, there exists a pair of jobs J_j, J_k and an interval $I \subseteq [r_j, d_j) \cap [r_k, d_k)$ such that J_j is executed during I by some processor P_p, J_k is not executed during I and it holds that $Q_j < Q_k$. Let $\tilde{I} \subseteq [r_k, d_k)$ be an interval during which J_k is executed by some processor P_q. We modify the schedule as follows. We increase Q_j so that an idle period appears during I and we decrease Q_k by executing part of J_k during this idle period. A similar argument to the one for proving Lemma 3 implies that this modification results in a schedule of lower energy. □

In the following lemma we set the minimum job density such that all speeds in the optimal schedule are at least one.

Lemma 4. *Assume that $\delta_j \geq \max_{p,q}\{(\frac{\alpha_p}{\alpha_q})^{1/(\alpha_q-1)}\}$ for every $J_j \in \mathcal{J}$. For every pair of job $J_j \in \mathcal{J}$ and processor $P_p \in \mathcal{P}$, it holds that $s_{j,p} \geq 1$.*

Proof. Since we do not allow parallel execution of a job, in any feasible schedule, including the optimal one, there exists a processor P_q which executes job J_j with speed $s_{j,q} \geq \delta_j$. Given that the function $f_q(s)$ is convex, this means that $f'_q(s_{j,q}) \geq f'_q(\delta_j)$. By Lemma 3, we have that $f'_p(s_{j,p}) = f'_q(s_{j,q})$ for any $P_p \in \mathcal{P}$. Hence, $f'_p(s_{j,p}) \geq f'_q(\delta_j)$, that is $\alpha_p \cdot$

$s_{j,p}^{\alpha_p-1} \geq \alpha_q \cdot \delta_j^{\alpha_q-1}$. Thus, using our assumption about the minimum density we get that $\alpha_p \cdot s_{j,p}^{\alpha_p-1} \geq \alpha_q \left((\frac{\alpha_p}{\alpha_q})^{1/(\alpha_q-1)} \right)^{\alpha_q-1}$, and the lemma follows. □

By using Lemma 4, we can define an order $P_1, P_2, \ldots, P_m$ of the processors such that for any value of speed $s \geq 1$, we have that $f_1(s) \leq f_2(s) \leq \ldots \leq f_m(s)$. Observe that this order is obtained by sorting the processors in non-decreasing order of their power exponent, i.e., $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_m$. Furthermore, it is not hard to verify that, for every speed s of a job in the optimal schedule, it also holds that $f_1'(s) \leq f_2'(s) \leq \ldots \leq f_m'(s)$. Based on the above, we say that $P_p \in \mathcal{P}$ is *cheaper* than $P_q \in \mathcal{P}$ if $p < q$; similarly, P_q is *more expensive* than P_p. The following lemma implies that cheap processors run, in general, with greater speeds than expensive processors in the optimal schedule.

Lemma 5. *For an interval I and any pair of jobs $J_j, J_k \in \mathcal{J}$ executed by the processors $P_p, P_q \in \mathcal{P}$ during whole I, respectively, if $p < q$ then $s_{j,p} \geq s_{k,q}$.*

Proof. Assume for contradiction that in the optimal schedule it holds that $s_{j,p} < s_{k,q}$. Let $g(s) = |I| \cdot [f_q(s) - f_p(s)]$. Then, we obtain a new schedule by exchanging the speeds and the jobs executed by the two processors during I. By comparing the two schedules, we have that

$$E_{new} - E_{opt} = |I| \left(f_p(s_{k,q}) + f_q(s_{j,p}) - [f_p(s_{j,p}) + f_q(s_{k,q})] \right) = g(s_{j,p}) - g(s_{k,q})$$

Since $p < q$, it must be the case that $g'(s) = f_q'(s) - f_p'(s) \geq 0$ for every $s \geq 1$, which means that the function $g(s)$ is non-decreasing with s. Given our assumption that $s_{j,p} < s_{k,q}$, we have that $g(s_{j,p}) \leq g(s_{j',q})$. Therefore, $E_{new} \leq E_{opt}$ which means that the new schedule is also optimal. □

The next property implies that cheap processors execute, in general, jobs with greater hypopowers compared to expensive processors.

Property 3. *For an interval I and any pair of jobs $J_j, J_k \in \mathcal{J}$ executed by the processors $P_p, P_q \in \mathcal{P}$ during whole I, respectively, if $p < q$ then $Q_j \geq Q_k$.*

Proof. Let $w = |I| \cdot s_{j,p}$ and $\tilde{w} = |I| \cdot s_{k,q}$ be the amount of work executed for J_j and J_k, respectively, during I and assume for contradiction that $Q_j < Q_k$. We modify the schedule as follows. The work $w + \tilde{w}$ is executed by the processors P_p and P_q during the whole interval I with constant hypopower. That is, $w + \tilde{w}$ units of work are now executed during the whole interval I by the processors P_p and P_q with constant speeds s_p and s_q such that $f_p'(s_p) = f_q'(s_q)$ and $w + \tilde{w} = |I| \cdot (s_p + s_q)$. A similar argument with the one that we used in the proof of Lemma 3 implies the energy consumption of the new schedule is lower. In order to reach a contradiction, it remains to show that the new schedule is feasible, i.e. both J_j and J_k are executed for exactly $|I|$ units of time during I.

Obviously, $Q_j < Q < Q_k$. That is, in order to produce the new schedule, we decrease the hypopower for executing w and we increase the hypopower for executing w'. Now, a part of w is executed by P_q and a part of $\tilde{w}$ is executed by P_p. The remaining parts of w and $\tilde{w}$ are still executed by P_p and P_q, respectively. In order to show that the new schedule is feasible, we will prove that there exists a value $t \in [0, |I|]$ satisfying the following. During I,

the execution time of J_j is equal to t on P_p and $|I| - t$ on P_q. Furthermore, the execution time of J_k is t on P_p and $|I| - t$. So, it must be the case that $w = t \cdot s_p + (|I| - t) \cdot s_q$ and $\tilde{w} = (|I| - t) \cdot s_p + t \cdot s_q$. By solving these equations with respect to s_p and equating them, we get that $t = \frac{w - |I|s_q}{w + \tilde{w} - 2|I|s_q}|I|$. It remains to show that $t \in [0, |I|]$, or, equivalently,

$$0 \leq \frac{w - |I|s_q}{w + \tilde{w} - 2|I|s_q} \leq 1 \tag{1}$$

Given that $Q_j < Q < Q_k$, it holds that $s_{j,p} < s_p$ and $s_q < s_{k,q}$. Since the initial schedule is optimal, by Lemma 5, we have that $s_{j,p} \geq s_{k,q}$. As a result, we have that $s_p > s_q$. Moreover, the fact that $s_{j,p} \geq s_{k,q}$ implies that $w \geq \tilde{w}$. Overall, we get that $2|I|s_q < |I|(s_p + s_q) = w + \tilde{w} \leq 2w$. By the last expression, it holds that $w > |I|s_q$ and $w + \tilde{w} > 2|I|s_q$. Hence, the first inequality of expression (1) is true. Since $s_{k,q} > s_q$, we get that $\tilde{w} > |I|s_q$. Thus, the second inequality of expression (1) is also true. □

The next property specifies the set of occupied processors at each time; these are the m_i cheapest ones. The remaining processors are idle. This means that, in the optimal schedule, the total processing time of all jobs is equal to $\sum_i \{m_i \cdot |I_i|\}$, i.e., the maximum possible that any feasible schedule may have.

Property 4. *During an interval $I_i \in \mathcal{I}$, the processors in $\{P_1, P_2, \ldots, P_{m_i}\}$ are occupied, while the processors in $\{P_{m_i+1}, P_{m_i+2}, \ldots, P_m\}$ are idle.*

Proof. Initially, we claim that exactly m_i processors are occupied at any $t \in I_i$. In order to prove our claim, it suffices to show that, for every interval $I \subseteq I_i$, the total processing time of all jobs during I is exactly $m_i \cdot |I|$. In any feasible schedule, the total execution time of all jobs during I cannot exceed $m \cdot |I|$, since there are m processors available. Moreover, it cannot be more than $n_i \cdot |I|$ because there are exactly n_i alive jobs during I and a job is allowed to be executed by at most one processor at each time. On the other hand, in the optimal schedule, the total execution time during I cannot be less than $\min\{m, n_i\} \cdot |I|$. Otherwise, there would be a sub-interval $\tilde{I} \subseteq I$, a job J_j which is alive but not executed during $\tilde{I}$ and a processor which is idle during $\tilde{I}$. In this case, we could execute part of J_j during $\tilde{I}$ by decreasing the speeds of J_j during all the intervals in which it is executed in a way that constant hypopower is attained. A similar argument with the one for proving Lemma 3 implies that we could produce a schedule of lower energy consumption. So, our claim is true, which means that exactly m_i processors are occupied at t. Obviously, by Lemma 4, the cheapest option is to use processors $P_1, P_2, \ldots, P_{m_i}$. □

The following corollary, which follows directly from Properties (1)-(4) implies that if we know the hypopowers of the jobs in the optimal schedule, then we know the speed of each processor at each time.

Corollary 1. *Consider an interval $I_i \in \mathcal{I}$ and let J_{j_k} be the the alive job during I_i with the k-th greatest hypopower, breaking ties arbitrarily. Then, at each time $t \in I_i$, processors $P_1, P_2, \ldots, P_{m_i}$ run with hypopowers $Q_{j_1} \geq Q_{j_2} \geq \ldots \geq Q_{j_{m_i}}$, respectively. Moreover, processors $P_{m_i+1}, P_{m_i+2}, \ldots, P_m$ are idle.*

Theorem 2. *Properties (1)-(4) are necessary and sufficient for optimality.*

Proof. Up to this point, we have showed the existence of an optimal schedule satisfying Properties (1)-(4). It remains to prove that these properties are also sufficient for optimality. In particular, we show that all schedules satisfying the properties attain equal energy consumption.

For a given schedule $\mathcal{S}$ satisfying Properties (1)-(4), we denote by $E(\mathcal{S})$ its energy consumption and by $Q_j(\mathcal{S})$ the hypopower of job $J_j \in \mathcal{J}$. Assume for contradiction that there exist two different schedules A and B satisfying Properties (1)-(4) such that $E(A) \neq E(B)$. Clearly, there exists at least one job J_j such that $Q_j(A) \neq Q_j(B)$. Otherwise, due to Corollary 1, schedules A and B attain equal energy consumption. Without loss of generality, we assume that there is at least one job J_j with $Q_j(A) < Q_j(B)$. Let $L = \{J_j : Q_j(A) < Q_j(B)\}$ be the non-empty subset of jobs with lower hypopower in A than in B. We will show that the total amount of work executed for the jobs in L is strictly smaller in A compared to B. Then, we will have reached a contradiction on the fact that, in both schedules, exactly w_j units of work should be executed for each job J_j.

We fix a time t. Let $L(A) \subseteq L$ be the set consisting of all jobs in L which are executed at time t in A. We denote by J_{a_j}, $1 \leq j \leq |L(A)|$, the job executed with the j-th highest speed among the jobs in $L(A)$, at time t in schedule A, breaking ties arbitrarily. Analogously, we define the set $L(B)$ and the job J_{b_j}, for $1 \leq j \leq |L(B)|$, with respect to schedule B. By the definition of the set L and Property 2, it is not hard to verify that $|L(A)| \leq |L(B)|$. Moreover, consider any pair of jobs J_{a_j} and J_{b_j}, for some $1 \leq j \leq |L(A)|$. Assume that J_{a_j} and J_{b_j} are executed by processors P_{p_j} and P_{q_j} in schedules A and B, respectively. By the definition of the set L and Property 3, we get that $p_j \geq q_j$. So, by Lemma 5, job J_{a_j} is executed at t in A with strictly lower speed than the one of J_{b_j} at t in B. We have showed that there is a one-to-one correspondence of the jobs in $L(A)$ to jobs in $L(B)$ (by possibly ignoring some jobs in $L(B)$) such that, at time t, every job in $L(A)$ has strictly lower speed in schedule A than the speed that its corresponding job in $L(B)$ has in schedule B. □

4.2 Presentation and Analysis of the Algorithm

Given the optimal structure presented in the previous section, we are now ready to describe a polynomial-time algorithm which is based on maximum flow computations. Initially, we present the high-level idea of the algorithm and, then, we describe its main components, in more detail, together with its analysis.

4.2.1 High-Level Idea.

Initially, we define a slightly more general problem which is the one that the algorithm actually solves. An instance of this problem is specified by a triple $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$. Specifically, there is a set $\mathcal{J}$ of n jobs which have to be executed by a set $\mathcal{P}$ of m parallel processors during a set $\mathcal{I}$ of disjoint time intervals. During each interval $I_i \in \mathcal{I}$ there is a subset $\mathcal{J}(I_i) \subseteq \mathcal{J}$ of alive jobs and a subset $\mathcal{P}(I_i) \subseteq \mathcal{P}$ of available processors. Every job $J_j \in \mathcal{J}(I_i)$ (and processor $P_p \in \mathcal{P}(I_i)$) is alive (resp. available) during the whole I_i. We denote by $n_i = |\mathcal{J}(I_i)|$ (and $a_i = |\mathcal{P}(I_i)|$) the number of alive jobs (resp. available processors) during I_i. Our original problem is a special case of the above; we observe that J_j is alive during every interval $I_i \in [r_j, d_j)$ and all the m processors are available in each interval. Moreover, the optimal

structure of the previous section is extended in a straightforward way to this more general problem.

Let $\mathcal{S}^*$ be an optimal schedule with the structure presented in the previous section and consider an interval $I_i \in \mathcal{I}$. By Property 4, the $m_i = \min\{a_i, n_i\}$ cheapest processors are used during the entire I_i while the remaining ones are always idle during I_i. So, the property specifies the exact amount of time, say t_p, that each processor $P_p \in \mathcal{P}$ is used in $\mathcal{S}^*$ as well as the corresponding intervals. A similar argument with the one for proving Property 1 implies that the most energy-efficient, though not necessarily feasible, way to schedule the jobs is by executing them with the same hypopower Q such that

$$\sum_{p=1}^{m} t_p \left(\frac{Q}{\alpha_p}\right)^{\frac{1}{\alpha_p - 1}} = \sum_{J_j \in \mathcal{J}} w_j \tag{2}$$

In what follows, we assume that we can compute a solution to the above equation with arbitrary precision (we explain later how to treat errors occurred due to limited precision). If there is a feasible schedule in which all jobs are executed with equal hypopower Q, then this schedule is optimal and we are done. Note that, as we explain in the next subsection, this feasibility problem can be answered with a maximum flow computation. If such a feasible schedule does not exist, then $\mathcal{J}$ can be partitioned into two disjoint and non-empty subsets $\mathcal{J}_{\geq Q}$ and $\mathcal{J}_{<Q}$ containing the jobs executed with hypopower at least Q and smaller than Q, respectively, in $\mathcal{S}^*$. In each interval $I_i \in \mathcal{I}$, Properties 2 and 3 specify the subsets of available processors $\mathcal{P}_{\geq Q}(I_i), \mathcal{P}_{<Q}(I_i) \subseteq \mathcal{P}(I_i)$ dedicated to the execution of $\mathcal{J}_{\geq Q}$ and $\mathcal{J}_{<Q}$, respectively, which are disjoint. Specifically, let $\mathcal{J}_{\geq Q}(I_i)$ and $\mathcal{J}_{<Q}(I_i)$ be the subsets of jobs of $\mathcal{J}_{\geq Q}$ and $\mathcal{J}_{<Q}$, respectively, which are alive during I_i. The jobs in $\mathcal{J}_{\geq Q}$ occupy the cheapest $\min\{a_i, |\mathcal{J}_{\geq Q}(I_i)|\}$ processors during I_i while the jobs in $\mathcal{J}_{<Q}$ use the remaining occupied processors. Then, the problem $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ can be decomposed into the two independent sub-problems $< \mathcal{J}_{\geq Q}, \mathcal{P}_{\geq Q}, \mathcal{I} >$ and $< \mathcal{J}_{<Q}, \mathcal{P}_{<Q}, \mathcal{I} >$. Therefore, $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ can be decomposed recursively as it is described in Algorithm 1.

Algorithm 1: OPT$(\mathcal{J}, \mathcal{P}, \mathcal{I})$

```
Compute the most energy-efficient hypopower Q for executing (J, P, I);
(J_{≥Q}, P_{≥Q}, I), (J_{<Q}, P_{<Q}, I) ← BISEPARATION(J, P, I, Q);
if J = J_{≥Q} then
    return CONSTANTHYPOPOWERSCHEDULE(J, P, I, Q);
else
    S_{≥Q} ← OPT(J_{≥Q}, P_{≥Q}, I);
    S_{<Q} ← OPT(J_{<Q}, P_{<Q}, I);
    return S_{≥Q} ∪ S_{<Q};
```

In order to complete the presentation of our algorithm, it remains to describe a way of answering the *feasibility* of the problem $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ in which all jobs are executed with constant hypopower Q (which has been computed according to Equation 2) and, in the case of infeasibility, the *biseparation* procedure.

4.2.2 Feasibility.

Consider an interval $I_i \in \mathcal{I}$ and a processor $P_p \in \mathcal{P}(I_i)$. Recall that, if processor P_p runs with hypopower Q during I_i, then its speed is $s_{i,p} = (\frac{Q}{\alpha_p})^{1/(\alpha_p - 1)}$. For simplicity, in what follows, we slightly abuse our notation: let $s_{i,p}$ be the speed of the p-th cheapest (and fastest) available processor during I_i, and $\mathcal{P}(I_i)$ be the set of the m_i cheapest available processors during I_i.

The feasibility of $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ w.r.t. the hypopower Q is based on a maximum flow computation in an appropriate network $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$ which is defined as follows (see Fig. 1). There is a source node u_0, a node u_j for each job $J_j \in \mathcal{J}$, a node $v_{i,p}$ for each pair of interval $I_i \in \mathcal{I}$ and processor $P_p \in \mathcal{P}(I_i)$, a node v_i for each interval $I_i \in \mathcal{I}$ and a destination node v_0. Moreover, the network contains the arc (u_0, u_j) with capacity w_j for each job $J_j \in \mathcal{J}$, the arc $(u_j, v_{i,p})$ with capacity $(s_{i,p} - s_{i,p+1})|I_i|$ for each interval I_i, job $J_j \in \mathcal{J}(I_i)$ and processor $P_p \in \mathcal{P}(I_i)$, the arc $(v_{i,p}, v_i)$ with capacity $p(s_{i,p} - s_{i,p+1})|I_i|$ for each interval $I_i \in \mathcal{I}$ and processor $P_p \in \mathcal{P}(I_i)$ as well as the arc (v_i, v_0) with infinite capacity for each $I_i \in \mathcal{I}$. By convention, let $s_{i,m_i+1} = 0$. This formulation was introduced by Federgruen and Groenevelt [13] and the following theorem is a corollary of [13].

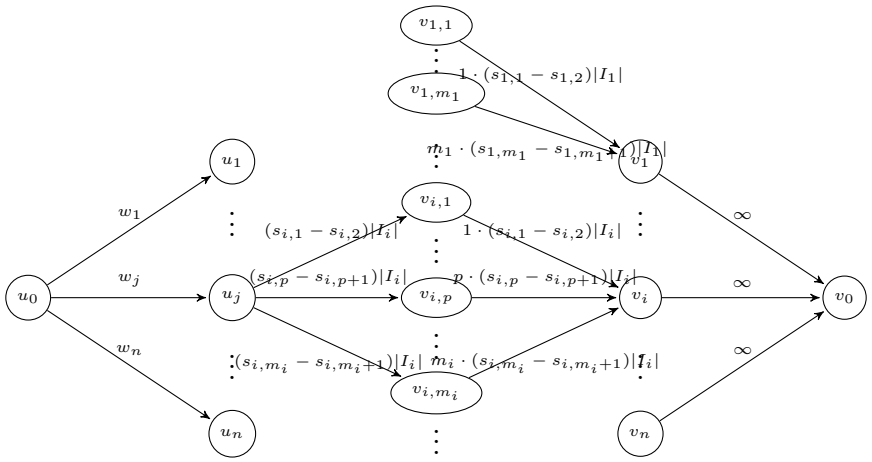

Figure 1: The flow network $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$

Theorem 3. *There exists a feasible schedule of $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ with constant hypopower Q iff there exists a feasible flow in $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$ of value $\sum_{J_j \in \mathcal{J}} w_j$.*

Theorem 3 implies that any feasible schedule for $< \mathcal{J}, \mathcal{P}, \mathcal{I}, Q >$ can be transformed to a feasible flow of value $\sum_{J_j \in \mathcal{J}} w_j$ in the network $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$ and vice versa. In particular, consider a feasible schedule, an interval $I_i \in \mathcal{I}$ and assume that $w_{i,j,p} = t \cdot s_{i,p}$ units of work of job $J_j \in \mathcal{J}(I_i)$ are processed by the p-th fastest processor in $\mathcal{P}(I_i)$. Then, it holds that

$$w_{i,j,p} = t \cdot s_{i,p} = t \cdot (s_{i,p} - s_{i,p+1}) + t \cdot (s_{i,p+1} - s_{i,p+2}) + \ldots + t \cdot (s_{i,m_i} - s_{m_i+1})$$

This observation shows the way for obtaining a feasible flow of value $\sum_{J_j \in \mathcal{J}} w_j$. Conversely, consider a feasible flow, an interval $I_i \in \mathcal{I}$, a job $J_j \in \mathcal{J}(I_i)$ and assume that $w_{i,j}$ units of

flow cross the network induced by the nodes u_j, v_i and $v_{i,p}$ for each $p = 1, 2, \ldots, m_i$. By applying the algorithm of Gonzalez and Sahni [15] for scheduling a set of jobs (where job J_j has work $w_{i,j}$) with common release dates and deadlines on related machines, we obtain a feasible schedule.

4.2.3 Biseparation.

If there is not a feasible schedule for $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ with constant hypopower Q computed by Equation (2), we next show how to decompose the problem into two subproblems $(\mathcal{J}_{<Q}, \mathcal{P}_{<Q}, \mathcal{I})$ and $(\mathcal{J}_{\geq Q}, \mathcal{P}_{\geq Q}, \mathcal{I})$. Initially, we introduce some notation. Consider an optimal schedule $\mathcal{S}^*$ with the structure presented in Section 4.1. We refer to every job $J_j \in \mathcal{J}_{\geq Q}$, i.e., job executed with hypopower at least Q, as *critical*. By Corollary 1, during interval $I_i \in \mathcal{I}$, the critical jobs occupy the $c_i = \min\{m_i, |\mathcal{J}_{\geq Q}(I_i)|\}$ fastest processors in $\mathcal{S}^*$. In the network $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$, we denote by $U(x, y)$ the capacity of the arc (x, y). Moreover, given a feasible (u_0, v_0)-flow $\mathcal{F}$, let $\mathcal{F}(x, y)$ be the amount of flow crossing the arc (x, y). Our biseparation algorithm is based on the following lemma.

Lemma 6. *Let $\mathcal{J}' \subseteq \mathcal{J}_{<Q}$ be any subset of non-critical jobs. A job $J_j \in \mathcal{J} \setminus \mathcal{J}'$ is critical if and only if, in the network $\mathcal{N}(\mathcal{J} \setminus \mathcal{J}', \mathcal{P}, \mathcal{I}, Q)$, there exists a minimum (u_0, v_0)-cut which does not contain the arc (u_0, u_j).*

Proof. For simplicity, we prove the lemma for $\mathcal{J}' = \mathcal{J}_{<Q}$. The proof can be easily extended for any subset $\mathcal{J}'$.

Let $\mathcal{S}^*$ be an optimal schedule for $< \mathcal{J}, \mathcal{P}, \mathcal{I} >$ with the structure presented in the previous section. We denote by $s^*_{i,p}$ the speed of processor $P_p \in \mathcal{P}(I_i)$ during interval $I_i \in \mathcal{I}$ in $\mathcal{S}^*$. Moreover, let (i, j, p) be the entire piece of job $J_j \in \mathcal{J}(I_i)$ executed on processor $P_p \in \mathcal{P}(I_i)$ during interval $I_i \in \mathcal{I}$ in $\mathcal{S}^*$ and note that this piece might not be executed consecutively during I_i. Assume that $w^*_{i,j,p}$ and $t^*_{i,j,p}$ are the total amount of work and the total processing time of the piece (i, j, p). By definition, $w^*_{i,j,p} = s^*_{i,p} \cdot t^*_{i,j,p}$. For each interval $I_i \in \mathcal{I}$, we denote by $s_{i,p}$ the speed of processor $P_p \in \mathcal{P}(I_i)$ if it runs with hypopower Q, i.e. $s_{i,p} = (\frac{Q}{\alpha_p})^{1/(\alpha_p - 1)}$.

We modify $\mathcal{S}^*$ and obtain a new schedule $\mathcal{S}$ as follows. For each piece (i, j, p) such that J_j is critical (that is $s^*_{i,p} \geq s_{i,p}$), we decrease its speed $s^*_{i,p}$ down to $s_{i,p}$ without changing its processing time; this modification implies that less work is now executed for J_j, except if it was originally executed with speed exactly $s_{i,p}$ in $\mathcal{S}^*$. For each piece (i, j, p) such that J_j is non-critical (i.e. $s^*_{i,p} < s_{i,p}$), we increase its speed $s^*_{i,p}$ up to $s_{i,p}$ without changing the amount of work executed for it, which means that its processing time has become smaller. If we ignore the fact that there exist some critical jobs which are not entirely executed, we observe that the resulting schedule $\mathcal{S}$ is a valid schedule. That is, by applying the transformation of Federgruen and Groenevelt [13] it can be converted into a feasible flow $\mathcal{F}$ in the network $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$.

We show that $\mathcal{F}$ is a maximum (u_0, v_0)-flow in $\mathcal{N}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$ by presenting a (u_0, v_0)-cut $\mathcal{C}$ of the same value. This cut contains the arc $(v_{i,p}, v_i)$ for $I_i \in \mathcal{I}$ and $p = 1, \ldots, c_i$, the arc $(u_j, v_{i,p})$ for $I_i \in \mathcal{I}$, $p = c_i + 1, \ldots, a_i$ and $J_j \in \mathcal{J}_{\geq Q}(I_i)$, the arc (u_0, u_j), for $J_j \in \mathcal{J}_{<Q}$.

Moreover, its value is equal to

$$\begin{aligned}
&\sum_{I_i\in\mathcal{I}}\sum_{p=1}^{c_i} U(v_{i,p},v_i) + \sum_{J_j\in\mathcal{J}_{\geq Q}(I_i)}\sum_{I_i\in\mathcal{I}}\sum_{p=c_i+1}^{m_i} U(u_j,v_{i,p}) + \sum_{J_j\in\mathcal{J}_{<Q}} U(u_0,u_j)\\
=&\sum_{I_i\in\mathcal{I}}\sum_{p=1}^{c_i} p(s_{i,p}-s_{i,p+1})|I_i| + \sum_{J_j\in\mathcal{J}_{\geq Q}(I_i)}\sum_{I_i\in\mathcal{I}}\sum_{p=c_i+1}^{m_i}(s_{i,p}-s_{i,p+1})|I_i| + \sum_{J_j\in\mathcal{J}_{<Q}} w_j\\
=&\sum_{I_i\in\mathcal{I}}\left(|I_i|\sum_{p=1}^{c_i} s_{i,p} - |I_i|\cdot c_i\cdot s_{i,c_i+1}\right) + \sum_{I_i\in\mathcal{I}}\sum_{p=c_i+1}^{m_i} c_i(s_{i,p}-s_{i,p+1})|I_i| + \sum_{J_j\in\mathcal{J}_{<Q}} w_j\\
=&\sum_{I_i\in\mathcal{I}}\sum_{p=1}^{c_i} s_{i,p}|I_i| - \sum_{I_i\in\mathcal{I}} c_i\cdot s_{i,m_i+1} + \sum_{J_j\in\mathcal{J}_{<Q}} w_j\\
=&\sum_{I_i\in\mathcal{I}}\sum_{p=1}^{c_i} s_{i,p}|I_i| + \sum_{J_j\in\mathcal{J}_{<Q}} w_j
\end{aligned}$$

By Corollary 1, in $\mathcal{S}^*$, the c_i fastest processors are fully occupied by critical jobs during $I_i \in \mathcal{I}$. This also holds for the schedule $\mathcal{S}$ because the processing time of a critical job is not modified. Therefore, after the transformation of Federgruen and Groenevelt [13], we get that $\sum_{J_j\in\mathcal{J}_{\geq Q}} \mathcal{F}(u_0,u_j) = \sum_{I_i\in\mathcal{I}} |I_i| \sum_{i=1}^{c_i} s_{i,p}$. On the other hand, every non-critical job is entirely executed in $\mathcal{S}$ which means that $\sum_{J_j\in\mathcal{J}_{<Q}} \mathcal{F}(u_0,u_j) = \sum_{J_j\in\mathcal{J}_{<Q}} w_j$. Hence, the sizes of $\mathcal{F}$ and $\mathcal{C}$ are indeed equal. That is, $\mathcal{F}$ is a maximum (u_0,v_0)-flow and $\mathcal{C}$ is a minimum (u_0,v_0)-cut.

Now, we are ready to complete the proof of the lemma. For each critical job J_j, there exists the minimum (u_0,v_0)-cut $\mathcal{C}$ which does not contain the arc (u_0,u_j). On the other hand, if J_j is non-critical, by construction of the schedule $\mathcal{S}$, its processing time during each interval $I_i \in [r_j,d_j]$ is strictly smaller than $|I_i|$. That is, after the transformation of Federgruen and Groenevelt [13], there is at least one interval $I_i \subseteq [r_j,d_j]$ (any of the intervals during which J_j is executed in $\mathcal{S}$) such that neither the arc (u_j,v_{i,m_i}), nor the arc (v_{i,m_i},v_i) is saturated by $\mathcal{F}$ which means that none of these two arcs can belong to any minimum (u_0,v_0)-cut. Thus, (u_0,u_j) must belong to every minimum (u_0,v_0)-cut. □

We define the *residual network* $\mathcal{R}_{\mathcal{F}}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$ of $\mathcal{N}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$ with respect to $\mathcal{F}$ as the network which contains the same nodes with $\mathcal{N}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$, the arc (x,y) with capacity $U(x,y)-\mathcal{F}(x,y)$, if (x,y) is not saturated by $\mathcal{F}$ in $\mathcal{N}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$ and the arc (y,x) with capacity $\mathcal{F}(x,y)$, if there is a positive amount of flow $\mathcal{F}(x,y)>0$ crossing the arc (x,y) by $\mathcal{F}$ in $\mathcal{N}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$. Then, we define the *inverse residual network* $\widetilde{\mathcal{R}}_{\mathcal{F}}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$ which is the same as $\mathcal{R}_{\mathcal{F}}(\mathcal{J},\mathcal{P},\mathcal{I},Q)$ except that all arcs are reversed. Algorithm 2 formally describes the biseparation procedure.

Lemma 7. *Algorithm 2 correctly identifies $\mathcal{J}_{<Q}$ and $\mathcal{J}_{\geq Q}$.*

Proof. Before proving the correctness of the biseparation algorithm, we need to make an observation. Consider the maximum (u_0,v_0)-flow computed by the algorithm in the network

Algorithm 2: BISEPARATION$(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$

1: $\mathcal{J}_{\geq Q} = \mathcal{J}$ and $\mathcal{J}_{<Q} = \emptyset$;
2: Find a maximum flow $\mathcal{F}$ in the network $\mathcal{N}(\mathcal{J}_{\geq Q}, \mathcal{P}, \mathcal{I}, Q)$;
3: **while** there is a path from v_0 to some job node u_j in $\widetilde{\mathcal{R}}_{\mathcal{F}}(\mathcal{J}_{\geq Q}, \mathcal{P}, \mathcal{I}, Q)$ **do**
4: $\quad \mathcal{J}_{\geq Q} = \mathcal{J}_{\geq Q} \setminus \{J_j\}$ and $\mathcal{J}_{<Q} = \mathcal{J}_{<Q} \cup \{J_j\}$;
5: $\quad$ Remove the flow passing through u_j from $\mathcal{F}$ and u_j from $\mathcal{N}(\mathcal{J}_{\geq Q}, \mathcal{P}, \mathcal{I}, Q)$;
6: Compute $\mathcal{P}_{\geq Q}$ and $\mathcal{P}_{<Q}$ based on Corollary 1;
7: **return** $(\mathcal{J}_{\geq Q}, \mathcal{P}_{\geq Q}, \mathcal{I}), (\mathcal{J}_{<Q}, \mathcal{P}_{<Q}, \mathcal{I})$;

$\mathcal{N}(\mathcal{J}_{\geq Q}, \mathcal{P}, \mathcal{I}, Q)$ and let $\mathcal{J}' \subseteq \mathcal{J}_{<Q}$ be a subset of non-critical jobs. We observe that, if we remove the flow crossing the node u_j for each job $J_j \in \mathcal{J}'$, then the resulting flow is a maximum (u_0, v_0)-flow in the network $\mathcal{N}(\mathcal{J}_{\geq Q} \setminus \mathcal{J}', \mathcal{P}, \mathcal{I}, Q)$. This is true because the value of the new flow is equal to the value of the (u_0, v_0)-cut defined in a similar way with the cut $\mathcal{C}$ in the proof of Lemma 6.

Let us now prove the lemma. We claim that it is not possible to remove a critical job in any iteration of Algorithm 2. Assume for contradiction that we remove such a job in some iteration. We consider the first iteration that this happens. Let $\mathcal{J}'$ and J_j be the remaining set of jobs when the iteration starts and a removed critical job, respectively. By Lemma 6, there exists a minimum cut $\mathcal{C}$ in $\mathcal{N}(\mathcal{J}', \mathcal{P}, \mathcal{I}, Q)$ which does not contain the edge (u_0, u_j). Clearly, the cut $\mathcal{C}$ contains either the arc $(u_j, v_{i,p})$, or the arc $(v_{i,p}, v_i)$ for each $I_i \in \mathcal{I}$ and $p = 1, 2, \ldots, a_i$, and this arc is necessarily saturated by the maximum flow $\mathcal{F}$. Hence, the node u_j cannot be reached from v_0 in $\widetilde{\mathcal{R}}_{\mathcal{F}}(\mathcal{J}, \mathcal{P}, \mathcal{I}, Q)$.

Finally, we claim that, in each iteration, if there are remaining non-critical jobs, at least one of them is identified. Consider an iteration in which $\mathcal{J}'$ is the remaining set of jobs when the iteration starts and suppose that $\mathcal{J}_{\geq Q} \subset \mathcal{J}'$. Assume for contradiction that the algorithm removes no job. Then, every arc $(u_j, v_{i,p})$ and every arc $(v_{i,p}, v_i)$ of the network $\mathcal{N}(\mathcal{J}', \mathcal{P}, \mathcal{I}, Q)$ are saturated which means that there exists a min cut not containing the arc (u_0, u_j) for every remaining non-critical job. This is a contradiction to Lemma 6. □

4.2.4 Correctness and Running Time.

The correctness of the algorithm follows from the fact that it produces a schedule satisfying Properties (1)-(4). Assume that by solving Equation (2), we get a solution $Q + \epsilon$ instead of Q, where $\epsilon > 0$ is a small constant. If all jobs are executed with hypopower Q in the optimal schedule, then the algorithm will construct a feasible schedule in which all jobs are executed with hypopower $Q + \epsilon$. On the other hand, if there does not exist a feasible schedule of all jobs w.r.t. Q, then the algorithm will perform a biseparation w.r.t. $Q + \epsilon$. Even though this biseparation is performed w.r.t. $Q + \epsilon$, it is correct in the sense that a job is characterized as critical if and only if it is executed with hypopower at least $Q + \epsilon$ in the optimal schedule. Therefore, the algorithm will produce a $(1 + \epsilon')$-approximate schedule in which, at each time, the hypopower of a processor is at most an additive factor of ϵ more than its hypopower in the optimal schedule, where $\epsilon' > 0$ is a small constant.

Concerning its running time, it makes $O(n)$ recursive calls because there are $O(n)$ distinct

values of hypopower in the optimal schedule. In every such call, it computes a hypopower value by solving Equation (2) in $O(f(n, \frac{1}{\epsilon}))$ time, where ϵ is the desired accuracy, it computes a maximum flow in a graph with $O(nm)$ vertices in $O(g(nm))$ time and it performs $O(n)$ Breadth-First Searches in a graph with $O(n^2m)$ arcs in $O(n^3m)$ time.

Theorem 4. *Algorithm 1 produces a $(1+\epsilon')$-approximate schedule with running time $O(nf(n, \frac{1}{\epsilon})+ ng(nm) + n^4m)$.*

5 Online Scheduling with Heterogeneous AVR

For the single-processor case, Yao et al. [20] proposed the *AVerage Rate algorithm* (or simply AVR) and they showed that it is $\alpha^\alpha \cdot 2^{\alpha-1}$-competitive for standard power functions of the form $f(s) = s^\alpha$. AVR sets the processor's speed at each time t equal to the total density of the alive jobs at t, i.e., $\sum_{J_j \in \mathcal{J}(t)} \delta_j$. Then, it schedules the jobs according to the Earliest Deadline First (EDF) policy.

In order to generalize AVR to the multiprocessor setting, we consider a variation of the single-processor AVR algorithm which assigns exactly the same speed to the processor at each time t but it follows a different job selection policy. Without loss of generality, we assume that all release dates and deadlines are integers. Assume also that $r_{\min} = \min\{r_j : J_j \in \mathcal{J}\} = 0$ and let $d_{\max} = \max\{d_j : J_j \in \mathcal{J}\} = T$ be the maximum deadline among the released jobs. We can partition the time horizon into unit-size intervals of the form $I_t = [t, t+1)$, $0 \leq t < T$. In particular, for each job $J_j \in \mathcal{J}(I_t)$, the algorithm assigns $\delta_j = \frac{w_j}{d_j - r_j}$ work of J_j to the interval I_t, and then it produces an arbitrary schedule of the total work assigned to I_t using constant speed $\sum_{J_j \in \mathcal{J}(I_t)} \delta_j$ during the whole I_t. The above variation achieves the same competitive ratio as the original AVR algorithm proposed in [20], since they both follow the same speed assignment rule and hence they have the same energy consumption.

We now turn our attention to the case of multiple heterogeneous processors. Based on the previous variation, we say that a schedule $\mathcal{S}$ is an *AVR-schedule* if for every job $J_j \in \mathcal{J}$ and interval $I_t \subseteq [r_j, d_j)$ the total amount of work of J_j executed during I_t on all processors in $\mathcal{S}$ is exactly equal to δ_j. The following lemma provides a lower bound on the optimal offline solution.

Lemma 8. *There exists a feasible AVR-schedule $\mathcal{S}_{\mathcal{AVR}}$ for the heterogeneous speed-scaling problem with arbitrary power functions such that $E(\mathcal{S}_{\mathcal{AVR}}) \leq (\max_{P_p \in \mathcal{P}}\{\rho_p\}+1)OPT$, where ρ_p is the competitive ratio of the single-processor AVR algorithm when it is applied to the processor P_p with power function $f_p(s)$.*

Proof. Let $\mathcal{S}^*$ be an optimal offline schedule. We denote by $\mathcal{S}^*_p$ the part of $\mathcal{S}^*$ which corresponds to the processor P_p. In other words, $\mathcal{S}^*$ is the concatenation of $\mathcal{S}^*_p$'s. Let $w^*_{j,p}$ and $s^*_{j,p}$ be the amount of work and the corresponding speed of job J_j on processor P_p, respectively, in $\mathcal{S}^*$. For each $P_p \in \mathcal{P}$, we modify $\mathcal{S}^*_p$ to $\mathcal{S}_p$ by applying the variation of the single-processor AVR algorithm; the work executed for each $J_j \in \mathcal{J}$ in $\mathcal{S}_p$ is equal to $w^*_{j,p}$. Let $\mathcal{S}$ be the resulting schedule, i.e., the concatenation of $\mathcal{S}_p$'s. Moreover, let $w_{j,p,t}$ and $s_{j,p,t}$ be the amount of work and the corresponding speed of job J_j on processor P_p during I_t, respectively, in $\mathcal{S}$.

Finally, we modify $\mathcal{S}$ by setting the speed of the piece of J_j executed by P_p during I_t equal to $\max\{s^*_{j,p}, s_{j,p,t}\}$ and we denote the obtained schedule by $\mathcal{S}_{AVR}$.

The total amount of work executed for J_j during I_t in $\mathcal{S}_{AVR}$ is equal to

$$\sum_{P_p \in \mathcal{P}} w_{j,p,t} = \sum_{P_p \in \mathcal{P}} \frac{w^*_{j,p}}{d_j - r_j} = \frac{w_j}{d_j - r_j} = \delta_j$$

Thus, $\mathcal{S}_{AVR}$ is an AVR-schedule.

The total processing time of all the pieces of J_j during I_t in $\mathcal{S}_{AVR}$ is equal to

$$\sum_{P_p \in \mathcal{P}} \frac{\frac{w^*_{j,p}}{d_j - r_j}}{\max\{s^*_{j,p}, s_{j,p,t}\}} \leq \frac{1}{d_j - r_j} \sum_{P_p \in \mathcal{P}} \frac{w^*_{j,p}}{s^*_{j,p}} \leq 1 = |I_t|$$

where the last inequality follows because $\mathcal{S}^*$ is feasible. By Lemma 2, we conclude that $\mathcal{S}_{AVR}$ can be constructed to be feasible.

In $\mathcal{S}_{AVR}$, the speed of the piece of J_j executed by P_p during I_t is equal either to the speed that the piece has in $\mathcal{S}^*$ or to the speed that it has in S. Therefore, the energy consumption of $\mathcal{S}_{AVR}$ is

$$\begin{aligned} E(\mathcal{S}_{AVR}) &= \sum_{P_p \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \sum_{t=0}^{T-1} \int_{I_t} \max\{f_p(s^*_{j,p}), f_p(s_{j,p,t})\} \\ &\leq \sum_{P_p \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \sum_{t=0}^{T-1} \int_{I_t} f_p(s^*_{j,p}) + \sum_{P_p \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \sum_{t=0}^{T-1} \int_{I_t} f_p(s_{j,p,t}) \\ &= E(\mathcal{S}^*) + \sum_{P_p \in \mathcal{P}} E(\mathcal{S}_p) \end{aligned}$$

For each $P_p \in \mathcal{P}$, let $\tilde{\mathcal{S}}_p$ be an optimal offline schedule for P_p in which for each job $J_j \in \mathcal{J}$ an amount of work $w^*_{j,p}$ is executed. Therefore, given that the single-processor AVR algorithm is ρ_p-competitive when it is applied to the processor P_p with power function $f_p(s)$, we have that

$$\begin{aligned} E(\mathcal{S}_{AVR}) &\leq E(\mathcal{S}^*) + \sum_{P_p \in \mathcal{P}} \rho_p E(\tilde{\mathcal{S}}_p) \leq E(\mathcal{S}^*) + \max_{P_p \in \mathcal{P}}\{\rho_p\} \sum_{P_p \in \mathcal{P}} E(\tilde{\mathcal{S}}_p) \\ &\leq E(\mathcal{S}^*) + \max_{P_p \in \mathcal{P}}\{\rho_p\} \sum_{P_p \in \mathcal{P}} E(\mathcal{S}^*_p) = E(\mathcal{S}^*) + \max_{P_p \in \mathcal{P}}\{\rho_p\} E(\mathcal{S}^*) \\ &= (\max_{P_p \in \mathcal{P}}\{\rho_p\} + 1) E(\mathcal{S}^*) \end{aligned}$$

where the last inequality follows by the optimality of $\tilde{\mathcal{S}}_p$ and the fact that the amount of work of each job $J_j \in \mathcal{J}$ is the same on both $\tilde{\mathcal{S}}_p$ and $\mathcal{S}^*_p$. □

We are now ready to describe our algorithm. The high level idea is that we create a $(1+\epsilon)$-approximate AVR-schedule, by using the algorithm proposed in Section 3. More

specifically, given the assignment of work into intervals implied by the definition of the AVR-schedules, for each interval $I_t = [t, t+1)$ we create an offline $(1+\epsilon)$-approximate schedule for this subinstance of the heterogeneous speed-scaling problem. We call this algorithm *Heterogeneous-AVR* (or simply H-AVR). Note that, if the time $t+1$ does not correspond to a release date or a deadline then the schedules for the intervals I_t and I_{t+1} are the same, and hence we have to compute it only once. The following theorem follows.

Theorem 5. *H-AVR is a $((1+\epsilon)(\max_{P_p \in \mathcal{P}}\{\rho_p\}+1))$-competitive algorithm for the heterogeneous speed-scaling problem, where ρ_p is the competitive ratio of the single-processor AVR algorithm when it is applied to the processor P_p with power function $f_p(s)$.*

For the case of standard power functions of the form $f(s) = s^\alpha$, the single-processor AVR algorithm is $\alpha^\alpha 2^{\alpha-1}$-competitive [20]. Therefore, the following corollary holds.

Corollary 2. *H-AVR is a $((1+\epsilon)(\alpha^\alpha 2^{\alpha-1}+1))$-competitive algorithm for the heterogeneous speed-scaling problem for standard power functions of the form $f_p(s) = s^{\alpha_p}$, where $\alpha = \max_{P_p \in \mathcal{P}}\{\alpha_p\}$.*

References

[1] S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration. *J. Comput. Syst. Sci.*, 81(7):1194–1209, 2015.

[2] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. *Algorithmica*, 68(2):404–425, 2014.

[3] E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Euro-Par*, volume 7484 of *LNCS*, pages 128–140. Springer, 2012.

[4] A. Antoniadis, and C.-C. Huang. Non-preemptive speed scaling. *J. Scheduling*, 16(4):385–394, 2013.

[5] E. Bampis, A. V. Kononov, D. Letsios, G. Lucarelli, and I. Nemparis. From preemptive to non-preemptive speed-scaling scheduling. *Discrete Applied Mathematics*, 181:11–20, 2015.

[6] E. Bampis, A. V. Kononov, D. Letsios, G. Lucarelli, and M. Sviridenko. Energy efficient scheduling and routing via randomized rounding. In *FSTTCS*, volume 24 of *LIPIcs*, pages 449–460. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[7] E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. *Theor. Comput. Sci.*, 579:126–136, 2015.

[8] N. Bansal, D. P. Bunde, H.-L. Chan, and K. Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.

[9] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2):18, 2013.

[10] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

[11] B. D. Bingham and M. R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *ISPA*, pages 153–161, 2008.

[12] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *ECRTS*, pages 101–108. IEEE Computer Society, 2004.

[13] A. Federgruen and H. Groenevelt. Preemptive scheduling of uniform machines by ordinary network flow techniques. *Management Science*, 32(3):341–349, 1986.

[14] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *J. ACM*, 23(4):665–679, 1976.

[15] T. Gonzalez and S. Sahni. Preemptive scheduling of uniform processor systems. *J. ACM*, 25:92–101, 1978.

[16] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pp. 1242–1253, 2012.

[17] A. Gupta, R. Krishnaswamy, and K. Pruhs. Scalably scheduling power-heterogeneous processors. In *ICALP*, volume 6198, pages 312–323. Springer, 2010.

[18] S. Im, and M. Shadloo. Brief announcement: A QPTAS for non-preemptive speed-scaling. In *SPAA*, pp. 207–209, 2016.

[19] M. Li, A. C. Yao, and F. F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *PNAS*, 103(11):3983-3987, 2006.

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.