Speed Scaling on Parallel Processors with Migration*

Eric Angel[1], Evripidis Bampis[2], Fadi Kacem[1], and Dimitrios Letsios[1,2]

[1] IBISC, Université d' Évry, France
{eric.angel,fadi.kacem,dimitris.letsios}@ibisc.univ-evry.fr
[2] LIP6, Université Pierre et Marie Curie, France
{Evripidis.Bampis}@lip6.fr

Abstract. We study the problem of scheduling a set of jobs with release dates, deadlines and processing requirements (works), on parallel speed-scalable processors so as to minimize the total energy consumption. We consider that both preemption and migration of jobs are allowed. We formulate the problem as a convex program and we propose a polynomial-time combinatorial algorithm which is based on a reduction to the maximum flow problem. We extend our algorithm to the multiprocessor speed scaling problem with preemption and migration where the objective is the minimization of the maximum lateness under a budget of energy.

1 Introduction

Energy consumption is a major issue in our days. Great efforts are devoted to the reduction of energy dissipation in computing environments ranging from small portable devices to large data centers. From an algorithmic point of view, new challenging optimization problems are studied, in which the energy consumption is taken into account as a constraint or as the optimization goal itself (for recent reviews see [1], [2]). This later approach has been adopted in the seminal paper of Yao et al. [11], where a set of independent jobs with release dates and deadlines have to be scheduled on a single processor so that the total energy is minimized, under the so-called *speed-scaling* model where the processor may run at variable speeds. Under this model, if the speed of a processor is s then the power consumption is s^α, where $\alpha > 1$ is a constant, and the energy consumption is the power integrated over time.

Single Processor Case. Yao et al., in [11], proposed an optimal off-line algorithm, known as the YDS algorithm, for the preemptive problem, i.e., where the execution of a job may be interrupted and resumed later on. In the same work, they initiated the study of online algorithms for the problem, introducing the Average Rate (AVR) and the Optimal Available (OA) algorithms. Bansal et al.

* Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, and by GDR du CNRS, RO.

C. Kaklamanis et al. (Eds.): Euro-Par 2012, LNCS 7484, pp. 128–140, 2012.

[5] proposed a new online algorithm, the BKP algorithm, which improves the competitive ratio of OA for large values of α.

Multiprocessor Case. There are two variants of the model. The first variant allows the preemption of the jobs but not their migration. We call this variant, the *non-migratory* variant. This means that a job may be interrupted and resumed later on, on the same processor, but it is not allowed to continue its execution on a different processor. In the second variant, the *migratory* variant, both the preemption and the migration of the jobs are allowed.

In [4], Albers et al. considered the multiprocessor non-migratory problem of minimizing the total energy consumption of a set of jobs with release dates and deadlines. For unit-work jobs with *agreeable* deadlines, they proposed a polynomial time algorithm. When the release dates and deadlines of jobs are arbitrary, they proved that the problem becomes NP-hard even for unit-work jobs and proposed approximation algorithms with constant approximation ratios for the off-line version of the problem. A generic reduction is given by Greiner et al. (see [9]) transforming a β-approximation algorithm for the single-processor problem to a βB_α-approximation algorithm for the multi-processor non-migratory problem, where B_α is the α-th Bell number. Furthermore, they showed that a β-approximation for multiple processors with migration yields a deterministic βB_α-approximation algorithm for multiple processors without migration.

For the migratory variant, Chen et al., in [8], initiated the study of the energy minimization speed scaling problem on multiprocessors with migration and they proposed a efficient algorithm when the jobs have arbitrary works but a common release date and deadline. In [7], Bingham and Greenstreet proposed a polynomial-time algorithm for the general problem when the release dates and deadlines of jobs are arbitrary. Their algorithm makes use of the Ellipsoid method (see [10]). Since the complexity of the Ellipsoid algorithm is high for practical applications, it was interesting to define a faster combinatorial algorithm.

When preparing a previous version of this paper, it came to our knowledge that Albers et al. [3], independently of our work, considered the same problem and proposed an optimal $O(n^2 f(n))$-time combinatorial algorithm, where n is the number of jobs and $f(n)$ is the complexity of finding a maximum flow in a graph with $O(n)$ vertices. They, also, extended the analysis of the single processor OA and AVR online algorithms to the multiprocessor case with migration.

Our Contribution and Organization of the Paper. We consider the multiprocessor migratory scheduling problem with the objective of minimizing the energy consumption. In Section 3, we give a convex programming formulation of the problem and in Section 4, we apply, the well known KKT conditions to our convex program. In this way, we obtain a set of properties that are satisfied by any optimal schedule. Then in Section 5, we propose an optimal algorithm in the case where the jobs have release dates, deadlines and the power function is of the form s^α, where $\alpha > 2$. The time complexity of our algorithm, which we call BAL, is in $O(nf(n) \log U)$, where n is the number of jobs, U is the range of all possible values of processors' speed divided by the desired accuracy and $f(|V|)$

is the complexity of computing a maximum flow in a layered graph with $O(|V|)$ vertices. Notice that our algorithm is faster than the one of Albers et al. [3] only if moderate precision is required. If full accuracy is required, our algorithm is not faster. Finally, we extend our algorithm so as to obtain an optimal algorithm for the problem of maximum lateness minimization under a budget of energy.

2 Preliminaries

Let $\mathcal{J} = \{j_1, ..., j_n\}$ be a set of jobs. Each job j_i is specified by a work w_i, a release date r_i and a deadline d_i. We define the span of a job j_i to be $span_i = [r_i, d_i]$ and we say that j_i is *alive* at time t if $t \in span_i$. We also define the density of job j_i as $den_i = w_i/(d_i - r_i)$. We assume a set of m variable-speed processors in the sense that they can all, dynamically, change their speeds and have a common speed-to-power function $P(t) = s(t)^\alpha$, where $P(t)$ is the power consumption at time t, $s(t)$ is the speed at time t and $\alpha > 2$ is a constant. Consider any interval of time $[a, b]$ and a given processor. The amount of work processed by this processor and its energy consumption, during $[a, b]$, are $\int_a^b s(t)dt$ and $\int_a^b s(t)^\alpha dt$, respectively. Hence, if the processor runs at a constant speed s, during $[a, b]$, then $s \cdot (b - a)$ units of work are executed and $s^\alpha \cdot (b - a)$ units of energy are consumed, during $[a, b]$. In our setting, preemption and migration of jobs are allowed. That is, the processing of a job may be suspended and resumed later, on the same or on different processor. Nevertheless, we do not allow parallel execution of a job which means that a job cannot be run simultaneously on two or more processors. We also assume that a continuous spectrum of speeds is available and that there is no upper bound on the speed of any processor. Our objective is to find a feasible schedule that minimizes the total energy consumed by all processors.

We define $\mathcal{T} = \{t_0, \ldots, t_L\}$ to be the set of release dates and deadlines taken in a non-decreasing order and without duplication. Clearly, $t_0 = \min_{j_i \in \mathcal{J}}\{r_i\}$ and $t_L = \max_{j_i \in \mathcal{J}}\{d_i\}$. Let $I_j = [t_{j-1}, t_j]$, for $1 \leq j \leq L$, and $\mathcal{I} = \{I_1, \ldots, I_L\}$. We denote $|I_j|$ the length of the interval I_j. Also, let $A(j)$ be the set of jobs that are alive during I_j, i.e. all the jobs j_i with $I_j \subseteq span_i$, and $a_j = |A(j)|$ be the number of jobs in $A(j)$. Given any schedule, we denote $t_{i,j}$ the total units of time that job j_i is processed during the interval I_j. As already mentioned in many other works (see [11] for example), one can show, through a simple exchange argument, that, in any optimal schedule, every job j_i is executed at a constant speed s_i and this comes from the convexity of the power function.

Next, we state a problem which is a variation of our problem that we will need throughout our analysis, we call it the *Work Assignment Problem* (or WAP) and can be described as follows: Consider a set of n jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ and a set of intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_L\}$. Each job can be alive in one or more intervals in $\mathcal{I}$. During each interval I_j there are m_j available processors. Moreover, we are given a value v. Our objective is to find whether or not there is a feasible schedule that executes all jobs in $\mathcal{J}$ with constant speed v. Recall that a schedule is feasible if and only if each job is executed during its alive intervals and is executed by at most one processor at each time t. Preemption

and migration of jobs are allowed. Note that the WAP is almost the feasibility scheduling problem where, given a set of jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$, so that each job j_i has a processing time p_i, a release date r_i and a deadline d_i, we ask whether there exists a feasible preemptive and migratory schedule that executes each job between its release date and its deadline (according to the classical 3-field notation of Graham, this problem is denoted by $P|r_i, d_i, pmtn|-$). The $P|r_i, d_i, pmtn|-$ problem is almost the same with the WAP with the difference that, in WAP, not all intervals have the same number of available processors. Therefore, WAP is polynomially solvable by applying a variant of an algorithm for $P|r_i, d_i, pmtn|-$ (see [6]).

We also consider the problem of maximum lateness minimization given a fixed budget of energy. We are given a set of n jobs $\mathcal{J} = \{j_1, ..., j_n\}$, a set of m parallel homogeneous processors and a budget of energy E. Each job j_i is characterized by a release date r_i, a due date $\bar{d}_i$ and a work w_i. Given a schedule $\mathcal{S}$, the lateness of a job j_i in $\mathcal{S}$ is defined as $L_i(\mathcal{S}) = C_i(\mathcal{S}) - \bar{d}_i$, where $C_i(\mathcal{S})$ is the completion time of j_i is $\mathcal{S}$. The objective is to find a feasible schedule, where preemption and migration of jobs are allowed, with minimum $L_{max} = \max_i\{L_i\}$ whose energy consumption does not exceed a given budget E.

3 Convex Programming Formulation

In order to derive a convex program for our problem, we introduce a variable s_i and a variable $t_{i,j}$, for each $j_i \in \mathcal{J}$ and for all I_j such that $j_i \in A(j)$, to be the speed of job j_i and the total execution time of job j_i during the interval I_j, respectively. So, we propose the following convex programming formulation:

$$\min \sum_{j_i \in \mathcal{J}} w_i s_i^{\alpha-1} \tag{1}$$

$$\frac{w_i}{s_i} - \sum_{I_j:\ j_i \in A(j)} t_{i,j} = 0 \qquad j_i \in \mathcal{J} \tag{2}$$

$$\sum_{j_i \in A(j)} t_{i,j} - m \cdot |I_j| \leq 0 \qquad 1 \leq j \leq L \tag{3}$$

$$\sum_{j_i \in A(j)} t_{i,j} - a_j \cdot |I_j| \leq 0 \qquad 1 \leq j \leq L \tag{4}$$

$$t_{i,j} - |I_j| \leq 0 \qquad 1 \leq j \leq L,\ j_i \in A(j) \tag{5}$$

$$-t_{i,j} \leq 0 \qquad 1 \leq j \leq L,\ j_i \in A(j) \tag{6}$$

$$-s_i \leq 0 \qquad j_i \in \mathcal{J} \tag{7}$$

Note that the total running time and the total energy consumption of each job j_i is $\frac{w_i}{s_i}$ and $w_i s_i^{a-1}$, respectively. Then, the term (1) is the total energy consumed by all jobs which is our objective function and the constraints (2) enforce that w_i units of work must be executed for each job j_i. The constraints (3) enforce that we can use at most m processors for $|I_j|$ units of time during any interval

I_j. Also, we can use at most a_j processors operating for $|I_j|$ units of time during any interval I_j, otherwise we would have parallel execution of a job and this is expressed by (4). The constraints (5) prevent any job j_i from being executed for more than $|I_j|$ units of time during any interval $I_j \subseteq span_i$, otherwise we would have parallel execution of a job. The constraints (6) and (7) insure the positivity of the variables $t_{i,j}$ and s_i, respectively.

The above mathematical program is indeed convex because the objective function and the first constraint are convex while all the other constraints are linear. Since our problem can be written as a convex program, it can be solved in polynomial time to arbitrary precision, by applying the Ellipsoid Algorithm [10]. Nevertheless, the Ellipsoid Algorithm is not used in practice and we would like to construct a faster and less complicated combinatorial algorithm.

At this point, notice that once the speeds of the jobs are computed, by solving the convex program, a further step is needed in order to construct a feasible schedule. This can be done by solving the feasibility problem $P|r_i, d_i, pmtn|-$.

4 Structure of the Optimal Schedule

We apply the KKT conditions to our convex program so as to obtain necessary conditions for optimality of a feasible schedule. We next show that these conditions are sufficient for optimality.

The following lemma is a direct consequence of the KKT conditions applied to the convex program of our problem combined with the fact that the power function with respect to the speed is convex.

Lemma 1. *There is always an optimal schedule for our problem that satisfies the following properties:*

1. *Each job j_i is executed at a constant speed s_i.*
2. *During any interval I_j, we have that $\sum_{j_i \in A(j)} t_{i,j} = \min\{a_j, m\}|I_j|$.*
3. *If $a_j \leq m$ during an interval I_j, then $t_{i,j} = |I_j|$, for every j_i with $I_j \subseteq span_i$.*
4. *If $a_j > m$ then*
   - i. *All jobs j_i that are alive during I_j, with $0 < t_{i,j} < |I_j|$, have equal speeds.*
   - ii. *If a job j_i is not executed during an interval $I_j \subset span_i$, i.e. $t_{i,j} = 0$, then $s_i \leq s_k$ for every job j_k with $I_j \subseteq span_k$ and $t_{k,j} > 0$.*
   - iii. *If a job j_i has $t_{i,j} = |I_j|$ in an interval I_j, then $s_i \geq s_k$ for any job j_k alive during I_j with $t_{k,j} < |I_j|$.*

Proof. The Properties 1, 2 and 3 can be easily proved by applying the definition of convexity and a simple exchange argument.

Next, we focus on proving the Property 4. For this, we will use the KKT conditions whose general form can be found in the full version. In order to apply the KKT conditions, we need to associate with each constraint a dual variable. Therefore, to each set of the constraints from (2) up to (7), we associate the dual variables β_i, γ_j, δ_j, $\epsilon_{i,j}$, $\zeta_{i,j}$ and η_i, respectively.

By stationarity conditions, we have that

$$\nabla \sum_{j_i \in \mathcal{J}} w_i s_i^{\alpha-1} + \sum_{j_i \in \mathcal{J}} \beta_i \cdot \nabla \left(\frac{w_i}{s_i} - \sum_{I_j:\, j_i \in A(j)} t_{i,j} \right)$$
$$+ \sum_{j=1}^{L} \gamma_j \nabla \left(\sum_{j_i \in A(j)} t_{i,j} - m \cdot |I_j| \right) + \sum_{j=1}^{L} \delta_j \nabla \left(\sum_{j_i \in A(j)} t_{i,j} - a_j \cdot |I_j| \right)$$
$$+ \sum_{j=1}^{L} \sum_{j_i \in A(j)} \epsilon_{ij} \nabla (t_{i,j} - |I_j|) + \sum_{j=1}^{L} \sum_{j_i \in A(j)} \zeta_{ij} \nabla (-t_{i,j}) + \sum_{j_i \in \mathcal{J}} \eta_i \nabla (-s_i) = 0$$

The previous equation can be rewritten equivalently as

$$\sum_{j=1}^{L} \sum_{j_i \in A(j)} \left(-\beta_i + \gamma_j + \delta_j + \epsilon_{i,j} - \zeta_{i,j} \right) \nabla t_{i,j}$$
$$+ \sum_{j_i \in \mathcal{J}} \left((\alpha - 1) w_i s_i^{\alpha-2} - \frac{\beta_i w_i}{s_i^2} - \eta_i \right) \nabla s_i = 0 \tag{8}$$

Furthermore, complementary slackness conditions imply that

$$\gamma_j \cdot \left(\sum_{j_i \in A(j)} t_{i,j} - m \cdot |I_j| \right) = 0 \qquad 1 \le j \le L \tag{9}$$

$$\delta_j \cdot \left(\sum_{j_i \in A(j)} t_{i,j} - a_j \cdot |I_j| \right) = 0 \qquad 1 \le j \le L \tag{10}$$

$$\epsilon_{ij} \cdot (t_{i,j} - |I_j|) = 0 \qquad 1 \le j \le L,\ j_i \in A(j) \tag{11}$$

$$\zeta_{ij} \cdot (-t_{i,j}) = 0 \qquad 1 \le j \le L,\ j_i \in A(j) \tag{12}$$

$$\eta_i \cdot (-s_i) = 0 \qquad j_i \in \mathcal{J} \tag{13}$$

We can safely assume that there are no jobs with zero work because we may treat such jobs as if they did not exist. So, for any job j_i, it holds that $s_i > 0$ and $\sum_{I_j \subseteq span_i} t_{i,j} > 0$. Then, (13) implies that $\eta_i = 0$. We set the coefficients of the partial derivatives ∇s_i and $\nabla t_{i,j}$ equal to zero so as to satisfy the stationarity conditions. Thus, (8) gives that $\beta_i = (\alpha - 1) s_i^{\alpha}$ for each job $j_i \in \mathcal{J}$ and

$$(\alpha - 1) s_i^{\alpha} = \gamma_j + \delta_j + \epsilon_{i,j} - \zeta_{i,j} \tag{14}$$

for each $j_i \in \mathcal{J}$ and $I_j \subseteq span_i$. Now, for each interval I_j such that $a_j > m$, because of (10), we have that $\delta_j = 0$. Next, we consider the following cases for the execution time of any job $j_i \in A(j)$:

- $0 < t_{i,j} < |I_j|$
  Complementary slackness conditions (11), (12) imply that $\epsilon_{i,j} = \zeta_{i,j} = 0$. As a result, (14) can be written as

$$(\alpha - 1) s_i^{\alpha} = \gamma_j. \tag{15}$$

The variable γ_j is specific for each interval and thus, all such jobs have the same speed throughout the whole schedule and Property 4(i) is valid.

- $t_{i,j} = 0$
  This means, by (11), that $\epsilon_{i,j} = 0$ and (14) is expressed as $(\alpha - 1)s_i^\alpha = \gamma_j - \zeta_{i,j}$. Thus, since $\zeta_{i,j} \geq 0$, we get that
$$(\alpha - 1)s_i^\alpha \leq \gamma_j. \tag{16}$$
- $t_{i,j} = |I_j|$
  In this case, by (12), we get that $\zeta_{i,j} = 0$. So, (14) becomes $(\alpha - 1)s_i^\alpha = \gamma_j + \epsilon_{i,j}$. Because of dual feasibility conditions, $\epsilon_{i,j} \geq 0$. Hence, all jobs of this kind have
$$(\alpha - 1)s_i^\alpha \geq \gamma_j. \tag{17}$$

By Equations (15), (16) and (17), we get Properties 4(ii) and 4(iii). □

Given a solution of the convex program that satisfies the KKT conditions, we derived some relations between the primal variables. Based on them, we defined some structural properties of any optimal schedule. These properties are necessary for optimality and we show that they are also sufficient because all schedules that satisfy these properties attain equal energy consumptions.

Lemma 2. *The properties of Lemma 1 are also sufficient for optimality.*

Proof. Assume for the sake of contradiction that there is a schedule A, that satisfies the properties of Lemma 1, which is not optimal and let B be an optimal schedule that also satisfies the properties (by Lemma 1 we know that the schedule B always exists). We denote E^X, s_i^X and $t_{i,j}^X$ the energy consumption, the speed of job j_i and the total execution time of job j_i during the interval I_j in schedule X, respectively. Because of our assumption, $E^A > E^B$. Let S be the set of jobs j_i with $s_i^A > s_i^B$. Clearly, there is at least one job j_k such that $s_k^A > s_k^B$, otherwise A would not consume more energy than B. So, $S \neq \emptyset$. By definition of S,

$$\sum_{j_i \in S} \sum_{I_j : j_i \in A(j)} t_{i,j}^A < \sum_{j_i \in S} \sum_{I_j : j_i \in A(j)} t_{i,j}^B.$$

Hence, there is at least one interval I_p such that

$$\sum_{j_i \in S} t_{i,p}^A < \sum_{j_i \in S} t_{i,p}^B.$$

If $a_p \leq m$, then there is at least one job j_q such that $t_{q,j}^A < t_{q,j}^B$. Due to the property 3 of Lemma 1, it should hold that $t_{q,j}^A = t_{q,j}^B = |I_j|$ which is a contradiction. So, assume that $a_p > m$. Then, the last equation gives that $t_{k,p}^A < t_{k,p}^B$ for some job $j_k \in S$. Thus, $t_{k,p}^A < |I_p|$ and $t_{k,p}^B > 0$. Both schedules have equal sum of processing times $\sum_{j_i \in I_j} t_{i,j}$ during any interval I_j. So, there must be a job $j_\ell \notin S$ such that $t_{\ell,p}^A > t_{\ell,p}^B$. Therefore, $t_{\ell,p}^A > 0$ and $t_{\ell,p}^B < |I_p|$. We conclude that $s_\ell^A \geq s_k^A > s_k^B \geq s_\ell^B$, which contradicts the fact that $j_\ell \notin S$. □

Notice that the properties of Lemma 1 do not explain how to find an optimal schedule. The basic reason is that they do not determine the exact speed value of each job. Moreover, they do not specify exactly the structure of the optimal schedule. That is, they do not specify which job is executed by each processor at each time.

5 An Optimal Combinatorial Algorithm

In this section, we propose an optimal combinatorial algorithm for our problem. Our algorithm always constructs a schedule satisfying the properties of Lemma 1 which, as we have already showed, are necessary and sufficient for optimality.

Our algorithm is based on the notion of *critical jobs* defined below. Initially, the algorithm conjectures that all jobs are executed at the same speed and it assigns to all of them a speed which is an upper bound on the maximum speed that a job has in any optimal schedule. The key idea is to continuously decrease the speeds of jobs step by step. At each step, it assigns a speed to the critical jobs that we ignore in the subsequent steps and goes on with the remaining subset of jobs. At the end of the last step, every job has been assigned a speed. Critical jobs are recognized by finding a minimum (s,t)-cut in an (s,t)-network as we describe in the following.

Now, for each instance of the WAP, we define a graph so as to reduce our original problem to the maximum flow problem. Given an instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP, consider the graph $G = (V, E)$ that contains one node x_i for each job j_i, one node y_j for each interval I_j, a source node s and a destination node t. We introduce an edge (s, x_i) for each $j_i \in \mathcal{J}$ with capacity $\frac{w_i}{v}$, an edge (x_i, y_j) with capacity $|I_j|$ for each couple of j_i and I_j such that $j_i \in A(j)$ and an edge (y_j, t) with capacity $m_j|I_j|$ for each interval $I_j \in \mathcal{I}$. We say that this is the *corresponding graph* of $< \mathcal{J}, \mathcal{I}, v >$.

We are ready, now, to introduce the notion of criticality. Given a feasible instance for the WAP, we say that job j_c is *critical* iff for any feasible schedule and for each $I_j \subseteq span_c$, either $t_{c,j} = |I_j|$ or $\sum_{j_i \in A(j)} t_{i,j} = m_j|I_j|$. Moreover, we say that an instance $< \mathcal{J}, \mathcal{I}, v >$ of the WAP is critical iff v is the minimum speed so that the set of jobs $\mathcal{J}$ can be feasibly executed over the intervals in $\mathcal{I}$ and we refer to the speed v as the critical speed of $\mathcal{J}$ and $\mathcal{I}$.

5.1 Properties of the Work Assignment Problem

Next, we will prove some lemmas that will guide us to an optimal algorithm. Our algorithm will be based on a reduction of our problem to the maximum flow problem which is a consequence of the following theorem whose proof is omitted.

Theorem 1. *[6] There exists a feasible schedule for the work assignment problem iff the corresponding graph has maximum (s,t)-flow equal to $\sum_{i=1}^{n} \frac{w_i}{v}$.*

Based on the above theorem, we can extend the notion of criticality. Specifically, with respect to graph G that corresponds to a feasible instance of the WAP, a job j_c is critical iff, for any maximum flow, either the edge (x_c, y_j) or the edge

(y_j, t) is saturated for each path x_c, y_j, t. Recall that an edge is saturated by a flow $\mathcal{F}$ if the flow that passes through the edge according to $\mathcal{F}$ is equal to the capacity of the edge. Moreover, we say that a path is saturated if at least one of its edges is saturated.

The following lemmas that involve the notions of *critical job* and *critical instance* are important ingredients for the analysis of our algorithm. The following lemma links the concept of a critical instance with the concept of a critical job and it is omitted due to space constraints.

Lemma 3. *If* $< \mathcal{J}, \mathcal{I}, v >$ *is a critical instance of WAP, then there is at least one critical job* $j_i \in \mathcal{J}$.

Note that the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ is not feasible if $< \mathcal{J}, \mathcal{I}, v >$ is critical. Up to now, the notion of a critical job has been defined only in the context of feasible instances. We extend this notion for infeasible instances as follows: in an infeasible instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$, a job j_i is called critical if every path x_i, y_j, t is saturated by any maximum (s, t)-flow in the corresponding graph G'.

Let $< \mathcal{J}, \mathcal{I}, v >$ be a critical instance of the WAP and let G be its corresponding graph. Next, we propose a way for identifying the critical jobs of $< \mathcal{J}, \mathcal{I}, v >$ using the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{I}, v - \epsilon >$, for some sufficiently small constant $\epsilon > 0$ based on Lemmas 4 and 5 below. The value of ϵ is such that the two instances have exactly the same set of critical jobs. Moreover, the critical jobs of $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ can be found by computing a minimum (s, t)-cut in the graph that corresponds to $< \mathcal{J}, \mathcal{I}, v - \epsilon >$. The proofs of Lemmas 4 and 5 can be found in the full version of the paper.

Lemma 4. *Given a critical instance* $< \mathcal{J}, \mathcal{I}, v >$ *of the WAP, there exists a constant* $\epsilon > 0$ *such that the unfeasible instance* $< \mathcal{J}, \mathcal{I}, v - \epsilon >$ *and the critical one have exactly the same critical jobs. The same holds for any other value* ϵ' *such that* $0 < \epsilon' \leq \epsilon$.

Lemma 5. *Assume that* $< \mathcal{J}, \mathcal{I}, v >$ *is a critical instance for the WAP and let* G' *be the graph that corresponds to the instance* $< \mathcal{J}, \mathcal{I}, v - \epsilon >$, *for any sufficiently small constant* $\epsilon > 0$ *in accordance with the Lemma 4. Then, any minimum* (s, t)*-cut* $\mathcal{C}'$ *of* G' *contains exactly:*

i. at least one edge of every path x_i, y_j, t *for any critical job* j_i,
ii. the edge (s, x_i) *for each non-critical job* j_i.

5.2 The BAL Algorithm

We are now ready to give a high level description of our algorithm. Initially, we will assume that the optimal schedule consumes an unbounded amount of energy and we assume that all jobs are executed with the same speed s_{UB}. This speed is such that there exists a feasible schedule that executes all jobs with the same speed. Then, we decrease the speed of all jobs up to a point where no further reduction is possible so as to obtain a feasible schedule. At this point,

all jobs are assumed to be executed with the same speed, which is critical, and there is at least one job that cannot be executed with speed less than this, in any feasible schedule. The jobs that cannot be executed with speed less than the critical one form the current set of critical jobs. So, the critical job(s) is (are) assigned the critical speed and is (are) ignored after this point. That is, in what follows, the algorithm considers the subproblem in which some jobs are omitted (critical jobs), because they are already assigned the lowest speed possible (critical speed) so that they can be feasibly executed, and there are less than m processors during some intervals because these processors are dedicated to the omitted jobs.

In detail, the algorithm consists of a number of steps, where at each step a binary search is performed, in order to determine the minimum speed so as to obtain a feasible schedule for the remaining jobs, i.e. the critical speed. We denote s_{crit} the critical speed and $\mathcal{J}_{crit}$ the set of critical jobs at a given step. In order to determine s_{crit} and $\mathcal{J}_{crit}$, we perform a binary search assuming that all the remaining jobs are executed with the same speed. We know that each job will be executed with speed not less than its density. Therefore, given a set of jobs $\mathcal{J}$, we know that there does not exist a feasible schedule that executes all jobs with a speed $s < \max_{j_i \in \mathcal{J}}\{den_i\}$. Also, observe that if all jobs have speed $s = \max_j\{\frac{\sum_{j_i \in A(j)} w_i}{|I_j|}\}$, then we can construct a feasible schedule. These bounds define the search space of the binary search performed in the initial step. In the next step the critical speed of the previous step is an upper bound on the speed of all remaining jobs and a lower bound is the maximum density among them. We use these updated bounds to perform the binary search of the current step and we go on like that. A high level pseudo-code of our algorithm follows.

Algorithm 1. BAL

1: $s_{UB} = \max_j\{\frac{\sum_{j_i \in A(j)} w_i}{|I_j|}\}$, $s_{LB} = \max_{j_i \in \mathcal{J}}\{den_i\}$
2: **while** $\mathcal{J} \neq \emptyset$ **do**
3: Find the minimum speed s_{crit} so that the instance $< \mathcal{J}, \mathcal{I}, s_{crit} >$ of the WAP problem is feasible, using binary search in the interval $[s_{LB}, s_{UB}]$, through repeated maximum flow computations.
4: Pick a sufficiently small $\epsilon > 0$.
5: Determine the set of critical jobs $\mathcal{J}_{crit}$ by computing a minimum (s,t)-cut in the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{I}, s_{crit} - \epsilon >$.
6: Assign to the critical jobs speed s_{crit} and set $\mathcal{J} = \mathcal{J} \backslash \mathcal{J}_{crit}$.
7: Update the number of available processors m_j for each interval I_j.
8: $s_{UB} = s_{crit}$, $s_{LB} = \max_{j_i \in \mathcal{J}}\{den_i\}$
9: Apply an optimal algorithm for $P|r_i, d_i, pmtn|-$ to schedule each job j_i with processing time w_i/s_i.

Algorithm BAL produces an optimal schedule, and this holds because any schedule constructed by the algorithm satisfies the properties of Lemma 1.

Theorem 2. *Algorithm BAL produces an optimal schedule.*

Proof. First of all, it is obvious that the algorithm assigns a constant speed to every job because each job is assigned exactly one speed in one step and the Property 1 of Lemma 1 is true.

Recall that at each step of the algorithm, a set of jobs is assigned a speed and some processors during some intervals are dedicated to these jobs. Consider the k-th step. At the beginning of the step, the remaining jobs $\mathcal{J}^{(k)}$ and available intervals $\mathcal{I}^{(k)}$ form the new instance of the WAP for which the critical speed and jobs are determined. We denote $G^{(k)}$ the graph that corresponds to the instance $< \mathcal{J}^{(k)}, \mathcal{I}^{(k)}, v >$ of the WAP, where the speed v varies between $s_{UB}^{(k)}$ and $s_{LB}^{(k)}$, i.e. the bounds of the step.

Assume that the Property 2 is not true. Then, there must be an interval I_j during which $\sum_{j_i \in A(j)} t_{i,j} < \min\{a_j, m\}|I_j|$, i.e. we can decrease the speed of some job and still get a feasible schedule. Note that it cannot be the case that $\sum_{j_i \in A(j)} t_{i,j} > \min\{a_j, m\}|I_j|$ because BAL assigns speeds only if there exists a feasible schedule with respect to these speeds. So, there must be a job $j_c \in A(j)$ such that $t_{c,j} < |I_j|$ and there is an idle period during I_j such that j_c is not executed. Suppose that j_c became critical during the k-th step. Then, in the graph $G^{(k)}$, since j_c is a critical job, either the edge (x_c, y_j) or the edge (y_j, t) belongs to a minimum (s,t)-cut and as a result, for any maximum flow in $G^{(k)}$, either $f(x_c, y_j) = |I_j|$ or $f(y_j, t) = m_j^{(k)}|I_j|$ where $m_j^{(k)}$ is the number of available processors during I_j at the beginning of the k-th step. Hence, we have a contradiction on the fact that $\sum_{j_i \in A(j)} t_{i,j} < \min\{a_j, m\}|I_j|$ and $t_{c,j} < |I_j|$.

For the Property 3, we claim that during the interval I_j with $a_j \leq m$, if a job j_c becomes critical, the edge (x_c, y_j) becomes saturated by any maximum (s,t)-flow in $G^{(k)}$ (given that j_c becomes critical at the k-th step). If this was not the case, then there would be a maximum (s,t)-flow $\mathcal{F}$ in $G^{(k)}$ such that $f(x_c, y_j) < |I_j|$. Also, in $\mathcal{F}$ it holds that $f(x_i, y_j) \leq |I_j|$ for any other $j_i \in A(j)$. Hence, $f(y_j, t) < a_j|I_j| \leq m|I_j|$. So, neither the edge (x_c, y_j) nor the edge (y_j, t) becomes saturated by $\mathcal{F}$, contradicting the criticality of j_c. Therefore, the total execution time of j_c during I_j is $|I_j|$.

Next we prove the Property 4. Initially, consider two jobs j_i and j_ℓ, alive during an interval I_j, such that $0 < t_{i,j} < |I_j|$ and $0 < t_{\ell,j} < |I_j|$. We will show that the jobs are assigned equal speeds by the algorithm. For this, it suffices to show that they are assigned a speed at the end of the same step. So, assume that j_i becomes critical at the end of the k-th step. Then, either the edge (x_i, y_j) or the edge (y_j, t) belongs to a minimum (s,t)-cut $\mathcal{C}$ in $G^{(k)}$. Since $0 < t_{i,j} < |I_j|$, we know that there exists a maximum (s,t)-flow in $G^{(k)}$ such that $0 < f(x_i, y_j) < |I_j|$. So, it is the edge (y_j, t) that belongs in $\mathcal{C}$. Therefore, in $G^{(k)}$, the edge (y_j, t) is saturated by any maximum (s,t)-flow, and as a result, all the processors during the interval I_j are dedicated to the execution of some tasks at the end of the k-th step. Hence, j_ℓ cannot be assigned a speed at a step strictly greater than k. Similarly, j_i is not assigned a speed later than j_ℓ. Hence, the two jobs are assigned a speed at the same step. That is, *4(i)* is true.

Next, for the Property *4(ii)*, consider the case where $t_{i,j} = 0$ for a job j_i during an interval $I_j \subseteq span_i$ and assume that j_i becomes critical at the k-th

step. Then, either y_j does not appear in $G^{(k)}$, that is no processors are available during I_j, or (y_j, t) belongs to a minimum (s,t)-cut of $G^{(k)}$. If none of these was true, then (y_j, t) would appear in $G^{(k)}$ and it would not belong to a minimum (s,t)-cut. Then, all the edges (x_ℓ, y_j) would belong to a minimum (s,t)-cut, for all j_ℓ alive during I_j that appear in $G^{(k)}$. So, (x_i, y_j) would be saturated by any maximum (s,t)-flow and we have a contradiction, since the fact that $t_{i,j} = 0$ implies that there exists a maximum (s,t)-flow with $f(x_i, y_j) = 0$. In both cases, that is if y_j does not appear in $G^{(k)}$ or (y_j, t) belongs to a minimum (s,t)-cut of $G^{(k)}$, no job executed during I_j will be assigned a speed after the k-th step. Hence, all jobs j_ℓ with $t_{\ell,j} > 0$ do not have lower speed than j_i.

Next, let j_i be a job with $t_{i,j} = |I_j|$ and assume that it is assigned a speed at the k-th step. As we have already shown, this cannot happen after a step where a job j_ℓ with $0 < t_{\ell,j} < |I_j|$ is assigned a speed because after such a step, the interval I_j is no longer considered. Also, as we showed in the previous paragraph, j_i becomes critical not after a job j_ℓ with $t_{\ell,j} = 0$. The Property *4(iii)* follows.

Finally, because of Lemmas 4 and 5, BAL correctly identifies the critical jobs at each step of the algorithm. The theorem follows. □

We turn, now, our attention to the complexity of the algorithm. Because of Lemma 3 at least one job (all critical ones) is scheduled at each step of the algorithm. Therefore, there will be at most n steps. Assume that U is the range of all possible values of speeds divided by our desired accuracy. Then, the binary search needs to check $O(\log U)$ values of speed to determine the next critical speed at one step. That is, BAL performs $O(\log U)$ maximum flow calculations at each step. Thus, the overall complexity of our algorithm is $O(nf(n)\log U)$ where $f(|V|)$ is the complexity of computing a maximum flow in a graph with $|V|$ vertices.

6 Maximum Lateness with a Budget of Energy

In order to solve the problem of minimizing the maximum lateness under a budget of energy, it is sufficient to determine an upper and a lower bound on the maximum lateness of the optimal schedule and then perform a binary search within this interval. The algorithm and its optimality are given in the full version of the paper.

Theorem 3. *The multiprocessor speed scaling problem of minimizing the maximum lateness of a set of jobs under a budget of energy can be solved in polynomial time when preemption and migration are allowed.*

Acknowledgments. We thank Alexander Kononov for helpful discussions on this work.

References

1. Albers, S.: Energy Efficient Algorithms. Communications of the ACM 53(5), 86–96 (2010)

2. Albers, S.: Algorithms for Dynamic Speed Scaling. In: STACS. LIPIcs, vol. 9, pp. 1–11 (2011)
3. Albers, S., Antoniadis, A., Greiner, G.: On Multi-Processor Speed Scaling with Migration. In: SPAA, pp. 279–288 (2011)
4. Albers, S., Muller, F., Schmelzer, S.: Speed Scaling on Parallel Processors. In: SPAA, pp. 289–298 (2007)
5. Bansal, N., Kimbrel, T., Pruhs, K.: Speed Scaling to Manage Energy and Temperature. Journal of the ACM 54(1), 3 (2007)
6. Baptiste, P., Néron, E., Sourd, F.: Modèles et Algorithmes en Ordonnancement. Ellipses (2004)
7. Bingham, B., Greenstreet, M.: Energy Optimal Scheduling on Multiprocessors with Migration. In: ISPA, pp. 153–161 (2008)
8. Chen, J.J., Hsu, H.R., Chuang, K.H., Yang, C.L., Pang, A.C., Kuo, T.W.: Multiprocessor Energy Efficient Scheduling with Task Migration Considerations. In: ECRTS, pp. 101–108 (2004)
9. Greiner, G., Nonner, T., Souza, A.: The Bell is Ringing in Speed Scaled Multiprocessor Scheduling. In: SPAA, pp. 11–18 (2009)
10. Nemirovski, A., Nesterov, I., Nesterov, Y.: Interior Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics (1994)
11. Yao, F., Demers, A., Shenker, S.: A Scheduling Model for Reduced CPU Energy. In: FOCS, pp. 374–382 (1995)