Journal of Computer and System Sciences 81 (2015) 1194–1209

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

On multi-processor speed scaling with migration ☆

Susanne Albers [a,*,1], Antonios Antoniadis [b], Gero Greiner [a]

[a] Department of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
[b] Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany

ARTICLE INFO

Article history:
Received 8 March 2012
Accepted 31 January 2015
Available online 30 March 2015

Keywords:
Energy efficiency
Offline algorithm
Online algorithm
Flow computation
Competitive analysis

ABSTRACT

We investigate a very basic problem in dynamic speed scaling where a sequence of jobs, each specified by an arrival time, a deadline and a processing volume, has to be processed so as to minimize energy consumption. We study multi-processor environments with m parallel variable-speed processors assuming that job migration is allowed, i.e. whenever a job is preempted it may be moved to a different processor. We first study the offline problem and show that optimal schedules can be computed efficiently in polynomial time, given any convex non-decreasing power function. In contrast to a previously known strategy, our algorithm does not resort to linear programming. For the online problem, we extend two algorithms *Optimal Available* and *Average Rate* proposed by Yao et al. [15] for the single processor setting. Here we concentrate on power functions $P(s) = s^{\alpha}$, where s is the processor speed and $\alpha > 1$ is a constant.

1. Introduction

Algorithmic techniques for energy savings in computing devices have received considerable research interest recently. A relatively new, effective approach is *dynamic speed scaling*. It relies on the fact that many modern microprocessors can operate at variable speed. Both Intel and AMD offer a range of such variable-speed processors. Obviously, high speed implies high performance. However, the higher the speed, the higher the energy consumption. The goal of dynamic speed scaling is to utilize the full speed/frequency spectrum of a processor and to use low speeds whenever possible.

In a seminal paper, initiating the algorithmic study of dynamic speed scaling, Yao, Demers and Shenker [15] introduced the following problem. Consider a sequence $\sigma = J_1, \ldots, J_n$ of n jobs that have to be scheduled on a variable-speed processor. Each job J_i is specified by a release time r_i, a deadline d_i and a processing volume w_i, $1 \leq i \leq n$. In a feasible schedule, J_i must be processed within the time interval $[r_i, d_i)$. The processing volume w_i is the amount of work that must be finished to complete J_i and, intuitively, can be viewed as the number of required CPU cycles. The processing time of a job depends on the processor speed. If J_i is processed at speed s, then it takes w_i/s time units to complete the job. Preemption of jobs is allowed, i.e. the execution of a job may be stopped and resumed later. The well-known cube-root rule for CMOS devices states that the speed s of a processor is proportional to the cube-root of the power or, equivalently, that power is proportional to s^3. The algorithms literature considers generalizations of this rule. Early and in fact most of the previous

☆ A preliminary version of this paper has appeared in the *Proceedings of the 23th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 279–288, 2011.

* Corresponding author. Fax: +49 89 28917707.
E-mail addresses: albers@in.tum.de (S. Albers), aantonia@mpi-inf.mpg.de (A. Antoniadis), greiner@informatik.tu-muenchen.de (G. Greiner).

[1] Work supported by the German Research Foundation, AL 464/7-1.

http://dx.doi.org/10.1016/j.jcss.2015.03.001
0022-0000/

work assumes that if a processor runs at speed s, then the required power is $P(s) = s^\alpha$, where $\alpha > 1$ is a constant. More recent research even allows for general convex non-decreasing power functions $P(s)$. Obviously, energy is power integrated over time. The goal is to find a feasible schedule for the given job instance $\sigma = J_1, \ldots, J_n$ minimizing energy consumption.

The scheduling problem defined above has received the most research interest among algorithmic speed scaling problems. We will present the most important results below. Almost all of the previous studies assume that a single variable-speed processor is given. However, energy conservation and speed scaling techniques are equally interesting in multi-processor environments. Multi-core and many-core platforms will be the dominant processor architectures in the future. Nowadays many PCs and laptops are already equipped with dual-core and quad-core designs. Moreover, compute clusters and server farms, usually consisting of many high-speed processors, represent parallel multi-processor systems that have been used successfully in academia and enterprises for many years. Power dissipation has become a major concern in these environments. Finally, in research, multi-processor systems have always been investigated extensively, in particular as far as scheduling and resource management problems are concerned.

In this paper we study dynamic speed scaling in multi-processor environments. We adopt the framework by Yao et al. [15], as defined above, but assume that m parallel processors are given. Each processor can individually run at variable speed s; the associated power function is $P(s)$. We consider the specific family of functions $P(s) = s^\alpha$, where $\alpha > 1$, as well as general convex non-decreasing functions $P(s)$. *Job migration* is allowed, i.e. whenever a job is preempted it may be moved to a different processor. Hence, over time, a job may be executed on various processors as long as the respective processing intervals do not overlap. Executing a job simultaneously on two or more processors is not allowed. The goal is to construct a feasible schedule minimizing the total energy consumption incurred by all the processors.

Two scenarios are of interest. In the offline setting, all jobs and their characteristics are known in advance. We wish to construct optimal schedules minimizing energy consumption. In the online setting, jobs arrive over time. Whenever a new job J_i arrives at time r_i, its deadline d_i and processing volume w_i are known. However, future jobs J_k, with $k > i$, are unknown. We use competitive analysis to evaluate the performance of online strategies [14]. An online algorithm A is called c-competitive if, for any job sequence σ, the energy consumption of A is at most c times the consumption of an optimal offline schedule.

Previous work We review results on dynamic speed scaling, focusing on deadline-based scheduling introduced by Yao et al. [15]. As mentioned above, most of the previous work addresses single-processor environments and power functions $P(s) = s^\alpha$, where $\alpha > 1$. In [15] Yao et al. first studied the offline problem and presented a polynomial time algorithm for computing optimal schedules. Refinements of the algorithm were given by Li et al. [11,12]. Yao et al. also presented two elegant online algorithms called *Optimal Available* and *Average Rate*. They proved that *Average Rate* achieves a competitive ratio of $(2\alpha)^\alpha/2$. The analysis is essentially tight as Bansal et al. [2] showed a nearly matching lower bound of $((2-\delta)\alpha)^\alpha/2$, where δ goes to zero as α tends to infinity. Bansal et al. [5] analyzed *Optimal Available* and, using a clever potential function, proved a competitiveness of exactly α^α. They also proposed a new strategy that attains a competitive ratio of $2(\frac{\alpha}{\alpha-1})e^\alpha$. As for lower bounds, Bansal et al. [4] showed that the competitiveness of any deterministic strategy is at least $e^{\alpha-1}/\alpha$. Irani et al. [9] proved that the offline algorithm by Yao et al. [15] also constructs optimal schedules if any convex non-decreasing power function $P(s)$ is given. On the other hand, no online algorithm has been designed or analyzed for general convex power functions.

The framework by Yao et al. assumes that there is no upper bound on the allowed processor speed. Articles [3,7,13] study settings in which the processor has a maximum speed or only a finite set of discrete speed levels. Irani et al. [9] consider an extended problem where a variable-speed processor is equipped with an additional sleep state, assuming that even at speed zero a positive amount of energy is consumed.

The only previous work addressing deadline-based scheduling in multi-processor systems is [1,6,8,10]. Bingham and Greenstreet [6] show that, if job migration is allowed, the offline problem can be solved in polynomial time using linear programming. The result holds for general convex non-decreasing power functions. Lam et al. [10] study a setting with two speed-bounded processors. They show online algorithms that are constant competitive w.r.t. energy minimization and throughput maximization. Papers [1,8] assume that job migration is *not* allowed. In this case the offline problem is NP-hard, even if all jobs have the same processing volume [1]. A randomized B_α-approximation algorithm and a randomized $2(\frac{\alpha}{\alpha-1})e^\alpha B_\alpha$-competitive online algorithm are given in [8]. Here B_α is the α-th Bell number. All the latter results were developed for the family of power functions $P(s) = s^\alpha$.

Our contribution In this paper we investigate dynamic speed scaling in general multi-processor environments, assuming that job migration is allowed. Using migration, scheduling algorithms can take advantage of the parallelism given by a multi-processor system in an effective way. We present a comprehensive study addressing the offline and, for the first time, also the online scenario.

First in Section 2 we study the offline problem and develop an efficient polynomial time algorithm for computing optimal schedules. The algorithm works for general convex non-decreasing power functions $P(s)$. As mentioned above, Bingham and Greenstreet [6] showed that the offline problem can be solved using linear programming. However, the authors mention that the complexity of their algorithm is too high for most practical applications. Instead in this paper we develop a strongly combinatorial algorithm that relies on repeated maximum flow computations. The approach might be helpful to solve other problems in scheduling and speed scaling.

Our algorithm is different from the single-processor strategy by Yao et al. [15]. In a series of phases, the algorithm partitions the jobs $J_1, \ldots, J_n$ into job sets $\mathcal{J}_1, \ldots, \mathcal{J}_p$ such that all jobs $J_k \in \mathcal{J}_i$ are processed at the same uniform speed s_i, $1 \le i \le p$. Each such job set is computed using maximum flow calculations. In order to construct a flow network, we have to identify various properties of a specific class of optimal schedules. A key property is that, knowing $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$, one can exactly determine the number of processors to be allocated to $\mathcal{J}_i$. At the beginning of the phase computing $\mathcal{J}_i$, the algorithm conjectures that the set $\mathcal{J} = \{J_1, \ldots, J_n\} \setminus (\mathcal{J}_1, \ldots, \mathcal{J}_{i-1})$ of all remaining jobs forms the next set $\mathcal{J}_i$. If this turns out not to be the case, the algorithm repeatedly removes jobs $J_k \in \mathcal{J}$ that do not belong to $\mathcal{J}_i$. The crucial step in the correctness proof is to show that each of these job removals is indeed correct so that, when the process terminates, the true set $\mathcal{J}_i$ is computed.

In Section 3 we study the online problem and, as in the previous literature, focus on power functions $P(s) = s^\alpha$, where $\alpha > 1$. We adapt the two popular strategies *Optimal Available* and *Average Rate* to multi-processor environments. Algorithm *Optimal Available*, whenever a new job arrives, computes an optimal schedule for the remaining workload. This can be done using our new offline algorithm. We prove that *Optimal Available* is α^α-competitive, as in the single-processor setting. While the adaption of the algorithm is immediate, its competitive analysis becomes considerably more involved. We can extend the potential function analysis by Bansal et al. [5] but have to define a refined potential and have to prove several properties that specify how an optimal schedule changes in response to the arrival of a new job. As for the second algorithm *Average Rate*, we present a strategy that distributes load among the processors so that the densities $\delta_i = w_i/(d_i - r_i)$ of the active jobs are almost optimally balanced. We prove that *Average Rate* achieves a competitiveness of $(2\alpha)^\alpha/2 + 1$. Hence, compared to competitive ratio in the single-processor setting, the factor only increases by the additive constant of 1.

2. A combinatorial offline algorithm

We develop a strongly combinatorial algorithm for constructing optimal offline schedules in polynomial time. Let $\sigma = J_1, \ldots, J_n$ be any job sequence and $P(s)$ be an arbitrary convex non-decreasing power function. Lemma 1 below implies that there exist optimal schedules that use at most n different speeds, say speeds $s_1 > s_2 > \ldots > s_p$ where $p \le n$. Our algorithm constructs such an optimal schedule in p phases, starting from an initially empty schedule S_0. Let S_{i-1} be the schedule obtained at the end of phase $i-1$, $1 \le i \le p$. In phase i the algorithm identifies the set $\mathcal{J}_i$ of jobs that are processed at speed s_i, $1 \le i \le p$. Schedule S_{i-1} is then extended by the jobs of $\mathcal{J}_i$ to form a new schedule S_i. The job set $\mathcal{J}_i$ and the extension of the schedule are determined using repeated maximum flow computations. Finally S_p is an optimal feasible schedule.

We present three lemmas that we will also use when analyzing an extension of *Optimal Available* in Section 3. In that section we will consider power functions $P(s) = s^\alpha$, where $\alpha > 1$.

Lemma 1. *Any optimal schedule can be modified such that every job J_i, $1 \le i \le n$, is processed at a constant non-varying speed. During the modification the schedule remains feasible and the energy consumption does not increase.*

Proof. Consider any optimal schedule. While there exists a job J_i, $1 \le i \le n$, that is processed at non-constant speed, let s_i be the average speed at which the job is executed. In the intervals in which J_i is processed modify the schedule so that J_i is executed at constant speed s_i. The total processing volume w_i of J_i is still completed in these intervals. By the convexity of the power function $P(s)$ the modification does not increase the energy consumption of the schedule. □

By the above lemma we restrict ourselves to optimal schedules that process each job at a constant speed. Let $\mathcal{S}_{OPT}$ be the set of such optimal schedules. In any schedule $S_{OPT} \in \mathcal{S}_{OPT}$ the n jobs $J_1, \ldots, J_n$ can be partitioned into sets $\mathcal{J}_1, \ldots, \mathcal{J}_p$ such that all the jobs of $\mathcal{J}_i$ are processed at the same speed s_i, $1 \le i \le p$. Each job belongs to exactly one of these sets.

In the set $\mathcal{S}_{OPT}$ we identify a subset $\mathcal{S}'_{OPT}$ of schedules having some favorable properties. If the power function is $P(s) = s^\alpha$, with $\alpha > 1$, then let $\mathcal{S}'_{OPT} = \mathcal{S}_{OPT}$. Otherwise, if $P(s)$ is a different power function, fix any $\alpha > 1$ and let $P_\alpha(s) = s^\alpha$ be an auxiliary power function. Let $\mathcal{S}'_{OPT} \subseteq \mathcal{S}_{OPT}$ be the subset of the schedules that, among schedules of $\mathcal{S}_{OPT}$, incur the smallest energy consumption if $P(s)$ is replaced by $P_\alpha(s)$. That is, among the schedules in $\mathcal{S}_{OPT}$, we consider those that would yield the smallest energy consumption if energy was accounted according to $P_\alpha(s)$. Since set $\mathcal{S}_{OPT}$ is non-empty, subset $\mathcal{S}'_{OPT}$ is also non-empty. Such an auxiliary power function was also used in [9] to break ties among optimal schedules.

The next lemma ensures that any schedule of $\mathcal{S}'_{OPT}$ can be modified such that the processor speeds change only at the release times and deadlines of jobs. Let $\mathcal{I} = \{r_i, d_i \mid 1 \le i \le n\}$ be the set of all release times and deadlines. We consider the elements of $\mathcal{I}$ in sorted order $\tau_1 < \ldots < \tau_{|\mathcal{I}|}$, where $|\mathcal{I}| \le 2n$. The time horizon in which jobs can be scheduled is $[\tau_1, \tau_{|\mathcal{I}|})$. We partition this time horizon along the job release times and deadlines into intervals $I_j = [\tau_j, \tau_{j+1})$, $1 \le j < |\mathcal{I}|$. Let $|I_j| = \tau_{j+1} - \tau_j$ be the length of I_j.

Lemma 2. *Given any optimal schedule $S_{OPT} \in \mathcal{S}'_{OPT}$, in each interval I_j we can rearrange the schedule such that every processor uses a constant, non-varying speed in I_j, $1 \le j < |\mathcal{I}|$. During the modification the schedule remains feasible and the energy consumption with respect to $P(s)$ and $P_\alpha(s)$ does not increase.*

Proof. Consider an arbitrary optimal schedule $S_{OPT} \in \mathcal{S}'_{OPT}$ and let I_j be any interval, $1 \le j < |\mathcal{I}|$. We show how to modify the schedule in I_j, without changing the energy consumption, so that the desired property holds for I_j. Modifying all the intervals in that way, we obtain the lemma.

For a given I_j, let $S_{OPT}(I_j)$ be the schedule of S_{OPT} restricted to I_j. Moreover, let $s_{j_1} > s_{j_2} > \ldots > s_{j_l}$ be the different speeds employed in I_j. Assume that s_{j_k} is used for t_k time units in I_j, considering all the m processors, and let $\mathcal{J}_{j_k}(I_j)$ be the set of jobs processed at speed s_{j_k} in I_j, $1 \le k \le l$. Since jobs are processed at constant speed the sets $\mathcal{J}_{j_k}(I_j)$, $1 \le k \le l$, are disjoint. For a job $J_i \in \mathcal{J}_{j_k}(I_j)$, let $t_{i,j}$ be the total time for which J_i is scheduled in I_j. We have $t_{i,j} \le |I_j|$ since otherwise $S_{OPT}(I_j)$ and hence S_{OPT} would not be feasible. Moreover, $\sum_{J_i \in \mathcal{J}_{j_k}(I_j)} t_{i,j} = t_k$.

We next construct a *working schedule* W of length $\sum_{k=1}^{l} t_k$ for the jobs of $\mathcal{J}_{j_1}(I_j), \ldots, \mathcal{J}_{j_l}(I_j)$ and then distribute it among the processors. More specifically, for any k with $1 \le k \le l$, we construct a schedule W_k of length t_k in which the jobs of $\mathcal{J}_{j_k}(I_j)$, using a speed of s_{j_k}, are scheduled as follows: First the jobs J_i with $t_{i,j} = |I_j|$ are processed, followed by the jobs J_i with $t_{i,j} < |I_j|$. Each J_i is assigned $t_{i,j}$ consecutive time units in W_k. We concatenate the W_k to form the working schedule $W = W_1 \circ \ldots \circ W_l$ of length $\sum_{k=1}^{l} t_k \le m|I_j|$. Next we distribute W among the m processors by assigning time window $[(\mu-1)|I_j|, \mu|I_j|)$ of W to processor μ, for $1 \le \mu \le \lceil \sum_{k=1}^{l} t_k/|I_j| \rceil$. Each window is scheduled from left to right on the corresponding processor, starting at the beginning of I_j. As we shall prove below, the total length of W is an integer multiple of $|I_j|$ so that exactly $\lceil \sum_{k=1}^{l} t_k/|I_j| \rceil$ processors are filled, without any idle period at the end. Let $S'_{OPT}(I_j)$ be the new schedule for I_j. Furthermore, let S'_{OPT} be the schedule obtained from S_{OPT} when replacing $S_{OPT}(I_j)$ by $S'_{OPT}(I_j)$. Schedule S_{OPT} is feasible because, for each job, the total execution time and employed speed have not changed. If in $S'_{OPT}(I_j)$ the execution of a job $J_i \in \mathcal{J}_{j_k}(I_j)$ is split among two processors μ and $\mu+1$, then J_i is processed at the end of I_j on processor μ and at the beginning of I_j on processor $\mu+1$. The two execution intervals do not overlap because $t_{i,j} \le |I_j|$. Furthermore, the energy consumption of S'_{OPT} is the same as that of S_{OPT} because the processing intervals of the jobs have only been rearranged. This holds true with respect to both $P(s)$ and $P_\alpha(s)$.

To establish the statement of the lemma for I_j we show that each t_k is an integer multiple of $|I_j|$, i.e. $t_k = m_k|I_j|$ for some positive integer m_k, $1 \le k \le l$. This implies that the new schedule never uses two or more speeds on any processor. So suppose that some t_k is not an integer multiple of $|I_j|$. Consider the smallest index k with this property. Then in $S'_{OPT}(I_j)$, on the corresponding processor, the processing of W_k ends at some time t strictly before the end of I_j, i.e. $t < \tau_{j+1}$. Furthermore, the job J_i handled last in W_k is processed for less than $|I_j|$ time units since otherwise all jobs in W_k would be executed for $|I_j|$ time units, contradicting the choice of k. So let $t_{i,j} = |I_j| - \delta_i$, for some $\delta_i > 0$. Set $\delta = \min\{t - \tau_j, \tau_{j+1} - t, \delta_i, t_{i,j}\}$. We now modify $S'_{OPT}(I_j)$ on the processor handling the end of J_i within I_j. During the last δ time units before t we decrease the speed by ϵ, and during the first δ time units after t we increase the speed by ϵ, where $\epsilon = (s_{j_k} - s_{j_{k+1}})/2$. If $k = l$, we set $s_{j_{k+1}} = 0$, assuming that the processing of W_l is followed by an idle period. Since $\delta \le \tau_{j+1} - t$ and $\delta \le t - \tau_j$, the modifications only affect the processor under consideration. Furthermore, the total work finished in $[t-\delta, t+\delta]$ remains unchanged. As the speed was lowered in $[t-\delta, t)$, the first time units after t are used to finish the work on J_i. This can be accomplished before time $t + \delta \le t + \delta_i$ so that J_i is not processed for more than $|I_j|$ time units and the resulting schedule is feasible.

Let S^m_{OPT} be the modified schedule obtained. Since $P(s)$ is convex, S^m_{OPT} does not consume more energy than S'_{OPT} with respect to $P(s)$. Hence S^m_{OPT} is an optimal schedule. However, since $\delta s_{j_k}^\alpha + \delta s_{j_{k+1}}^\alpha > 2\delta((s_{j_k} + s_{j_{k+1}})/2)^\alpha$ schedule S^m_{OPT} consumes strictly less energy than S'_{OPT} with respect to $P_\alpha(s)$. Finally, modify S^m_{OPT} so that each job is processed at constant speed. This can be done by processing job J_i and the job possibly scheduled after J_i in I_j using their respective average speeds. Since $P(s)$ and $P_\alpha(s)$ are convex, the respective energy consumptions do not increase. The resulting schedule belongs to set $\mathcal{S}_{OPT}$ and consumes strictly less energy than schedules S'_{OPT} and S_{OPT} with respect to $P_\alpha(s)$. We obtain a contradiction to the fact that $S_{OPT} \in \mathcal{S}'_{OPT}$. □

In the remainder of this section we consider optimal schedules $S_{OPT} \in \mathcal{S}'_{OPT}$ satisfying the property of Lemma 2. Since set $\mathcal{S}'_{OPT}$ is non-empty, these schedules always exist. As described above we will construct such a schedule S_{OPT} in phases, where phase i identifies the set $\mathcal{J}_i$ of jobs processed at speed s_i, $1 \le i \le p$. By Lemma 2, in each interval I_j, $1 \le j < |\mathcal{I}|$, the sets $\mathcal{J}_1, \ldots, \mathcal{J}_p$ occupy different processors, i.e. there is no processor handling jobs of two different sets. Hence, in phase i, when determining $\mathcal{J}_i$ and extending the previous schedule S_{i-1}, we only need to know the number of processors occupied by $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$ in any interval I_j, $1 \le j < |\mathcal{I}|$. The exact assignment of the corresponding jobs to the reserved processors is irrelevant though, as $\mathcal{J}_i$ does not share processors with the previous sets. In determining $\mathcal{J}_i$ a crucial question is, how many processors to allocate to the jobs of $\mathcal{J}_i$ in any interval I_j, $1 \le j < |\mathcal{I}|$. Here the following lemma is essential.

A job J_k, $1 \le k \le n$, is called *active* in I_j if $I_j \subseteq [r_k, d_k)$, i.e. the time period in which J_k can be scheduled includes I_j. Given schedule S_{OPT}, let n_{ij} be the number of jobs of $\mathcal{J}_i$ that are active in I_j. Furthermore, let m_{ij} denote the number of processors occupied by $\mathcal{J}_i$ in I_j.

Lemma 3. *Let $S_{OPT} \in \mathcal{S}'_{OPT}$ be any schedule satisfying the property of Lemma 2. Then the jobs of $\mathcal{J}_i$ occupy $m_{ij} = \min\{n_{ij}, m - \sum_{l=1}^{i-1} m_{lj}\}$ processors in I_j, where $1 \le i \le p$ and $1 \le j < |\mathcal{I}|$.*

Proof. Consider a set $\mathcal{J}_i$ and an interval I_j. Obviously, the n_{ij} jobs of $\mathcal{J}_i$ that are active in I_j cannot occupy more than n_{ij} processors using a positive speed s_i throughout I_j. Furthermore, the jobs can only occupy the $m - \sum_{l=1}^{i-1} m_{lj}$ processors not taken by $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$. We show that in fact $m_{ij} = \min\{n_{ij}, m - \sum_{l=1}^{i-1} m_{lj}\}$ processors are used by $\mathcal{J}_i$

So suppose that m'_{ij} processors, where $m'_{ij} < m_{ij}$, are used. Then, since $m'_{ij} < m - \sum_{l=1}^{i-1} m_{lj}$, there exists at least one processor P running at speed s with $s < s_i$ in I_j. Consider the schedule at the beginning of I_j. Choose a δ, $\delta > 0$, such that the m'_{ij} processors handling $\mathcal{J}_i$ and processor P do not preempt jobs in $[\tau_j, \tau_j + \delta)$, i.e. they each handle at most one job in that time window. In fact P might not handle a job if $s = 0$. As $m'_{ij} < n_{ij}$, there must exist one job $J_k \in \mathcal{J}_i$ that is active in I_j but not scheduled within $[\tau_j, \tau_j + \delta)$ on any of the m'_{ij} processors using speed s_i. This job is not scheduled on any other processor within $[\tau_j, \tau_j + \delta)$ either, because the other processors run at speeds higher or lower than s_i. Thus J_k is not executed on any processor within $[\tau_j, \tau_j + \delta)$. In the entire schedule, consider a processor and an associated time window W of length $\delta' \leq \delta$ handling J_k. We now modify the schedule by reducing the speed in W by ϵ, where $\epsilon = (s_i - s)/2$. At the same time we increase the speed on processor P in $[\tau_j, \tau_j + \delta')$ by ϵ and use the extra processing capacity to complete the missing portion of J_k not finished in W. Let S^m_{OPT} be the modified schedule. Since $P(s)$ is convex, S^m_{OPT} does not consume more energy than S_{OPT} with respect to $P(s)$. With respect to $P_\alpha(s)$, schedule S^m_{OPT} consumes less energy than S_{OPT} because $\delta' s_i^\alpha + \delta' s^\alpha > 2\delta'((s_i + s)/2)^\alpha$. Finally, modify S^m_{OPT} so that each job is processed at a constant speed. We obtain a schedule that belongs to set $\mathcal{S}_{OPT}$ and consumes strictly less energy than schedule S_{OPT} with respect to $P_\alpha(s)$. This contradicts the fact that $S_{OPT} \in \mathcal{S}'_{OPT}$. □

Lemma 3 has an interesting implication: Suppose that job sets $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$ along with the number of occupied processors $m_{1j}, \ldots, m_{i-1,j}$, for $1 \leq j < |\mathcal{I}|$, have been determined. Furthermore, suppose that the set $\mathcal{J}_i$ is known. Then, using Lemma 3, we can immediately determine the number m_{ij} of processors used by $\mathcal{J}_i$ in I_j, $1 \leq j < |\mathcal{I}|$. Moreover, we can compute the speed s_i by observing that, since $P(s)$ is non-decreasing, s_i can be set to the minimum average speed necessary to complete the jobs of $\mathcal{J}_i$ in the reserved processing intervals. More precisely, let $W_i = \sum_{J_k \in \mathcal{J}_i} w_k$ be the total processing volume of $\mathcal{J}_i$ and $P_i = \sum_{1 \leq j < |\mathcal{I}|} m_{ij}|I_j|$ be the total length of the reserved processing intervals. Then $s_i = W_i/P_i$. We will make use of this fact when identifying $\mathcal{J}_i$.

Description of the algorithm We show how each phase i, that determines job set $\mathcal{J}_i$, works. Assume again that $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$ along with processor numbers $m_{1j}, \ldots, m_{i-1,j}$, for $1 \leq j < |\mathcal{I}|$, are given. In phase i the algorithm operates in a series of rounds, maintaining a job set $\mathcal{J}$ that represents the current estimate for the true $\mathcal{J}_i$. At any time the invariant $\mathcal{J}_i \subseteq \mathcal{J}$ holds. Initially, prior to the first round, the algorithm sets $\mathcal{J} := \{J_1, \ldots, J_n\} \setminus (\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i-1})$, which is the set of all remaining jobs, and the invariant is obviously satisfied. In each round the algorithm checks if the current $\mathcal{J}$ is the desired $\mathcal{J}_i$. This can be verified using a maximum flow computation. If $\mathcal{J}$ turns out to be $\mathcal{J}_i$, then phase i stops. Otherwise the algorithm determines a job $J_k \in \mathcal{J} \setminus \mathcal{J}_i$, removes that job from $\mathcal{J}$ and starts the next round in which the updated set $\mathcal{J}$ is checked.

In the following we describe the maximum flow computation invoked for a given $\mathcal{J}$. First the algorithm determines the number of processors to be allocated to $\mathcal{J}$ in each interval. For any I_j, $1 \leq j < |\mathcal{I}|$, let n_j be the number of jobs in $\mathcal{J}$ that are active in $|I_j|$. According to Lemma 3 the algorithm reserves $m_j = \min\{n_j, m - \sum_{l=1}^{i-1} m_{lj}\}$ processors in I_j. These numbers form a vector $\vec{m} = (m_j)_{1 \leq j < |\mathcal{I}|}$. The speed is set to $s = W/P$, where $W = \sum_{J_k \in \mathcal{J}} w_k$ is the total processing volume and $P = \sum_{1 \leq j < |\mathcal{I}|} m_j|I_j|$ is the reserved processing time.

Next we define the graph $G(\mathcal{J}, \vec{m}, s)$ of the maximum flow computation. For each job $J_k \in \mathcal{J}$ we introduce a vertex u_k, and for each interval I_j with $m_j > 0$ we introduce a vertex v_j. For any u_k and v_j such that J_k is active in I_j we add a directed edge (u_k, v_j) of capacity $|I_j|$. Hence each u_k is connected to exactly those v_j such that J_k can be scheduled in I_j. The edge capacity of (u_k, v_j) is equal to $|I_j|$, i.e. the maximum time for which a job can be processed in I_j. Furthermore, we introduce a source vertex u_0 that is connected to every u_k on a directed edge (u_0, u_k). The edge capacity is set to w_k/s, which is the total processing time needed for J_k using speed s. Finally we introduce a sink v_0 and connect each v_j to v_0 via a directed edge (v_j, v_0) of capacity $m_j|I_j|$, which is the total processing time available on the reserved processors in I_j. The global structure of $G(\mathcal{J}, \vec{m}, s)$ is depicted in Fig. 1. The value of a maximum flow in $G(\mathcal{J}, \vec{m}, s)$ is upper bounded by $F_G = \sum_{J_k \in \mathcal{J}} w_k/s$ because this is the total capacity of the edges leaving the source u_0. This is equal to the total capacity of the edges leading into the sink v_0 because that value is

$$\sum_{1 \leq j < |\mathcal{I}|} m_j|I_j| = P = W/s = \sum_{J_k \in \mathcal{J}} w_k/s = F_G.$$

We argue that job set $\mathcal{J}$ using a constant speed of s can be feasibly scheduled on the reserved processors represented by $\vec{m}$ if and only if there exists a maximum flow of value F_G in $G(\mathcal{J}, \vec{m}, s)$. First suppose that there exists a feasible schedule using speed s. Then each $J_k \in \mathcal{J}$ is processed for w_k/s time units and we send w_k/s units of flow along edge (u_0, u_k). The w_k/s processing units of J_k are scheduled in intervals I_j in which the job is active. If J_k is processed for t_{kj} time units in I_j, then we send t_{kj} units of flow along edge (u_k, v_j). The edge capacity is observed because $t_{kj} \leq |I_j|$. The total amount of flow entering and hence leaving any v_j is equal to the total processing time on the m_j reserved processors in I_j. This value is equal to $m_j|I_j|$ because, by the choice of s, all reserved processors are busy throughout the execution intervals. Hence the amount of flow sent along (v_j, v_0) is equal to the edge capacity.

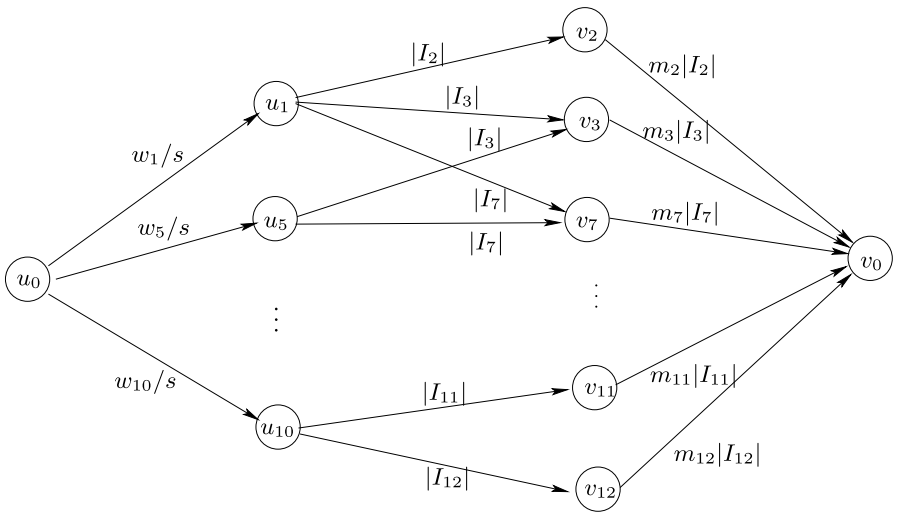

Fig. 1. The basic structure of $G(\mathcal{J}, \vec{m}, s)$, assuming that set $\mathcal{J} = \{J_1, J_5, \ldots, J_{10}\}$ can be scheduled in intervals $I_2, I_3, I_7, \ldots, I_{11}, I_{12}$.

Algorithm Optimal Schedule

1. $\mathcal{J} := \{J_1, \ldots, J_n\}$; $i := 0$; $S_0 :=$ empty schedule;
2. **while** $\mathcal{J} \neq \emptyset$ **do**
3. $\quad i := i + 1$; set_found := false;
4. $\quad$ **while** $\neg$ set_found **do**
5. $\quad\quad n_j :=$ number of jobs of $\mathcal{J}$ that are active in I_j, for $1 \leq j < |\mathcal{I}|$;
6. $\quad\quad m_j := \min\{n_j, \sum_{l=1}^{i-1} m_{lj}\}$; $\vec{m} := (m_j)_{1 \leq j < |\mathcal{I}|}$;
7. $\quad\quad W := \sum_{J_k \in \mathcal{J}} w_k$; $P := \sum_{1 \leq j < |\mathcal{I}|} m_j |I_j|$; $s := W/P$;
8. $\quad\quad$ Compute the value F of a maximum flow in $G(\mathcal{J}, \vec{m}, s)$;
9. $\quad\quad$ **if** $F = W/s$ **then** set_found := true;
10. $\quad\quad$ **else** Determine edge (v_j, v_0) carrying less than $m_j|I_j|$ units of flow and edge (u_k, v_j) carrying less than $|I_j|$ units of flow; Set $\mathcal{J} := \mathcal{J} \setminus \{J_k\}$;
11. $\quad \mathcal{J}_i := \mathcal{J}$; $s_i := s$; $m_{ij} := m_j$ for $1 \leq j < |\mathcal{I}|$;
12. $\quad$ Extend S_{i-1} by a feasible schedule for $\mathcal{J}_i$ using speed s_i and m_{ij} processors in I_j; Let S_i be the resulting schedule;
13. $\quad \mathcal{J} := \{J_1, \ldots, J_n\} \setminus (\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_i)$;

Fig. 2. The entire offline algorithm for computing an optimal schedule.

On the other hand, assume that a flow of value F_G is given. If t_{kj} units of flow are routed along (u_k, v_j), we process J_k for t_{kj} time units in I_j. This is possible because J_k is active in I_j and $t_{kj} \leq |I_j|$. The capacity constraints on the edge (v_j, v_0) ensure that not more than $m_j I_j$ time units are assigned to I_j. Moreover, since the flow value is F_G, each edge (u_0, u_k) is saturated and hence J_k is fully processed for w_k/s time units at speed s. Within each I_j we can easily construct a feasible schedule by concatenating the processing intervals of length t_{kj} associated with the jobs J_k active in I_j. The resulting sequential schedule, say S, of length $\sum_{J_k \in \mathcal{J}} t_{kj} = m_j |I_j|$ is split among the m_j reserved processors by assigning time range $[(\mu - 1)|I_j|, \mu|I_j|)$ of S to the μ-th reserved processor, $1 \leq \mu \leq m_j$.

We proceed to describe how the algorithm determines $\mathcal{J}_i$. In each round, for the current set $\mathcal{J}$, the algorithm invokes a maximum flow computation in the graph $G(\mathcal{J}, \vec{m}, s)$ defined in the previous paragraphs. If the flow value is equal to F_G, then the algorithm stops and sets $\mathcal{J}_i := \mathcal{J}$. We will prove below that in this case the current $\mathcal{J}$ is indeed equal to $\mathcal{J}_i$. If the flow value is smaller than F_G, then there must exist an edge (v_j, v_0) into the sink carrying less than $m_j|I_j|$ units of flow. Hence there must exist an edge (u_k, v_j) carrying less than $|I_j|$ units of flow because at least m_j jobs of $\mathcal{J}$ are active in I_j. The algorithm removes the corresponding job J_k, i.e. $\mathcal{J} := \mathcal{J} \setminus \{J_k\}$. We will also prove below that this removal is correct, i.e. $\mathcal{J}_i$ does not contain J_k. For the new set $\mathcal{J}$ the algorithm starts the next round, invokes a maximum flow computation in the updated graph $G(\mathcal{J}, \vec{m}, s)$, where $\vec{m}$ and s are updated too. The sequence of rounds ends when finally a flow of value F_G in the current graph $G(\mathcal{J}, \vec{m}, s)$ is found. A formal description of the entire algorithm is given in Fig. 2. The pseudo-code uses a boolean variable set_found that keeps track whether or not the desired $\mathcal{J}_i$ has been correctly determined.

Analysis of the algorithm It remains to prove correctness of our algorithm. We prove that in any phase i, $\mathcal{J}_i$ is correctly determined. Lemma 3 immediately implies that the number m_{ij} of reserved processors, for $1 \leq j < |\mathcal{I}|$, as specified in lines 6 and 11 of the algorithm is correct. A schedule for $\mathcal{J}_i$ on the reserved processors can be derived easily from the maximum flow computed in line 8. We described the construction of this schedule above when arguing that a flow of maximum value F_G in $G(\mathcal{J}, \vec{m}, s)$ implies a feasible schedule using speed s on the reserved processors $\vec{m}$. It remains to show that sets

$\mathcal{J}_1, \ldots, \mathcal{J}_p$ are correct. The proof is by induction on i. Consider an i, $1 \le i \le p$, and suppose that $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$ have been computed correctly. The inductive step and hence correctness of $\mathcal{J}_i$ follow from Lemmas 4 and 5.

Lemma 4. *In phase i, the set $\mathcal{J}$ maintained by the algorithm always satisfies the invariant $\mathcal{J}_i \subseteq \mathcal{J}$.*

Proof. At the beginning of the phase $\mathcal{J} = \{J_1, \ldots, J_n\} \setminus (\mathcal{J}_1, \ldots, \mathcal{J}_{i-1})$ is the set of all remaining jobs and the invariant is satisfied. Consider a round, represented by lines 5 to 10 of the algorithm, where in the beginning $\mathcal{J}_i \subseteq \mathcal{J}$ holds and a job J_{k_0} is removed in line 10 at the end of the round. We prove that J_{k_0} does not belong to $\mathcal{J}_i$.

Consider the flow computed in line 8 of the round. We call a vertex v_j *saturated* if the amount of flow routed along the outgoing edge (v_j, v_0) is equal to the capacity $m_j|I_j|$. Otherwise v_j is *unsaturated*. Since the computed flow has value $F < F_G = W/s = P = \sum_{1 \le j < |\mathcal{I}|} m_j|I_j|$, there exists at least one unsaturated vertex. We now modify the flow so as to increase the number of unsaturated vertices. For an edge e in the graph, let $f(e)$ be the amount of flow routed along it. The modification is as follows: While there exists a vertex u_k with flow $f(u_k, v_j) < |I_j|$ into an unsaturated vertex v_j and positive flow $f(u_k, v_{j'}) > 0$ into a saturated vertex $v_{j'}$, change the flow as follows. Along (u_k, v_j) and (v_j, v_0) we increase the flow by ϵ and along $(u_k, v_{j'})$ and $(v_{j'}, v_0)$ we reduce the flow by ϵ, where $\epsilon = \frac{1}{2}\min\{f(u_k, v_{j'}), |I_j| - f(u_k, v_j), m_j|I_j| - f(v_j, v_0)\}$. At u_k, v_j and $v_{j'}$ the flow conservation law is observed. By the choice of ϵ, the new flow along $(u_k, v_{j'})$ and $(v_{j'}, v_0)$ is positive. Also, by the choice of ϵ, the capacity constraints on (u_k, v_j) and (v_j, v_0) are not violated. Hence the new flow is feasible. Moreover, the total value F of the flow does not change. Vertex $v_{j'}$ is a new unsaturated vertex while v_j continues to be unsaturated because $\epsilon < m_j|I_j| - f(v_j, v_0)$ and hence the new flow along (v_j, v_0) is strictly smaller than the capacity. Also, since $\epsilon < |I_j| - f(u_k, v_j)$, the new flow along (u_k, v_j) remains strictly below $|I_j|$. The modifications stop when the required conditions are not satisfied anymore, which happens after less than $|\mathcal{I}|$ steps.

Given the modified flow, let $U = \{j \mid v_j \text{ is unsaturated}\}$ be the indices of the unsaturated vertices. Intuitively, these vertices correspond to the intervals in which less than $m_j|I_j|$ units of processing time would be scheduled. Let $\mathcal{J}'$, $\mathcal{J}' \subseteq \mathcal{J}$, be the set of jobs J_k such that $f(u_0, u_k) = w_k/s$ and the outgoing edges leaving u_k that carry positive flow all lead into unsaturated vertices. We argue that the job J_{k_0} removed from $\mathcal{J}$ in line 10 of the algorithm belongs to $\mathcal{J}'$: Prior to the flow modifications, there was an unsaturated vertex v_{j_0} having an incoming edge (u_{k_0}, v_{j_0}) with $f(u_{k_0}, v_{j_0}) < |I_{j_0}|$. Throughout the flow modifications, v_{j_0} remains an unsaturated vertex, and any flow update on (u_{k_0}, v_{j_0}) ensures that the amount of flow is strictly below $|I_{j_0}|$. Hence, if u_{k_0} had an outgoing edge with positive flow into a saturated vertex, the flow modifications would not have stopped. Moreover, if $f(u_0, u_{k_0}) < w_{k_0}/s$ held, we could increase the flow along edges (u_0, u_{k_0}), (u_{k_0}, v_{j_0}) and (v_{j_0}, v_0), thereby increasing the total value F of the flow.

Next let $\mathcal{J}''$, $\mathcal{J}'' \subseteq \mathcal{J} \setminus \mathcal{J}'$, be the set of jobs J_k such that u_k has at least one edge (u_k, v_j) into an unsaturated vertex v_j. Here we argue that for any such job, all the edges (u_k, v_j) into unsaturated vertices v_j carry a flow of exactly $|I_j|$ units: First observe that a job $J_k \in \mathcal{J}''$ does not belong to $\mathcal{J}'$ and hence (a) there is an edge (u_k, v_j) with positive flow into a saturated vertex v_j or (b) $f(u_0, u_k) < w_k/s$. In case (a), if there was an edge (u_k, v_j) carrying less than $|I_j|$ units of flow into an unsaturated vertex v_j, the flow modifications would not have stopped. In case (b), we could increase the flow along (u_0, u_k), (u_k, v_j) and (v_j, v_0), thereby raising the flow value F.

For any I_j, let $\mathcal{J}'(I_j)$, be the set of jobs of $\mathcal{J}'$ that are active in I_j. Set $\mathcal{J}''(I_j)$ is defined analogously. Given the modified flow, let t_{kj} be the amount of flow along (u_k, v_j). Note that for no job $J_k \in \mathcal{J} \setminus (\mathcal{J}' \cup \mathcal{J}'')$ there exists an edge (u_k, v_j) into an unsaturated vertex and hence J_k is not active in an I_j with $j \in U$. Hence, for any $j \in U$, since v_j is unsaturated, $\sum_{J_k \in \mathcal{J}'(I_j) \cup \mathcal{J}''(I_j)} t_{kj} < m_j|I_j|$. As $t_{kj} = |I_j|$, for $J_k \in \mathcal{J}''(I_j)$,

$$\sum_{J_k \in \mathcal{J}'(I_j)} t_{kj} < (m_j - |\mathcal{J}''(I_j)|)|I_j|. \tag{1}$$

We next prove that the job J_{k_0} removed at the end of the round in line 10 of the algorithm does not belong to $\mathcal{J}' \cap \mathcal{J}_i$ and hence does not belong to $\mathcal{J}_i$ because, as argued above, $J_{k_0} \in \mathcal{J}'$. So suppose that $J_{k_0} \in \mathcal{J}' \cap \mathcal{J}_i$. We show that in this case the total processing volume of $\mathcal{J}_i$ is not large enough to fill the reserved processing intervals using a continuous speed of s_i. We observe that s_i is at least as large as the speed s computed in the current round of the algorithm: If $s_i < s$, then in the optimal schedule, all jobs of $\mathcal{J}$ would be processed at a speed smaller than s. However, the total length of the available processing intervals for $\mathcal{J}$ is not more than $P = \sum_{1 \le j < |\mathcal{I}|} m_j|I_j|$ and total work to be completed is $W = \sum_{J_k \in \mathcal{J}} w_k$. Hence a speed of $s_i < s = W/P$ is not sufficient to finish the jobs in time.

Let m_{ij} be the number of processors used by $\mathcal{J}_i$ in interval I_j in an optimal solution. We prove that the jobs of $\mathcal{J}_i$ cannot fill all the reserved processors in intervals I_j with $j \in U$ using a speed of $s_i \ge s$. Recall that no job of $\mathcal{J} \setminus (\mathcal{J}' \cup \mathcal{J}'')$ is active in any interval I_j with $j \in U$. Only the jobs of $\mathcal{J}'(I_j) \cap \mathcal{J}_i$ and $\mathcal{J}''(I_j) \cap \mathcal{J}_i$ are active. The jobs of $\mathcal{J}'(I_j) \cap \mathcal{J}_i$ must fill at least $m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|$ processors over a time window of length $|I_j|$ because each job of $\mathcal{J}''(I_j) \cap \mathcal{J}_i$ can fill at best one processor. Thus, over all intervals I_j with $j \in U$, the jobs of $\mathcal{J}' \cap \mathcal{J}_i$ have to provide a work of at least

$$\sum_{j \in U} (m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|)|I_j|s_i. \tag{2}$$

We show that this is not the case. The total work of $\mathcal{J}' \cap \mathcal{J}_i$ is

$$\sum_{J_k \in \mathcal{J}' \cap \mathcal{J}_i} w_k = \sum_{J_k \in \mathcal{J}' \cap \mathcal{J}_i} f(u_0, u_k)s = \sum_{j \in U} \sum_{J_k \in \mathcal{J}' \cap \mathcal{J}_i} t_{kj}s, \tag{3}$$

where t_{kj} is again the amount of flow along (u_k, v_j) in the modified flow. The first and second equations hold because, for any $J_k \in \mathcal{J}'$, $f(u_0, u_k) = w_k/s$ and along edges (u_k, v_j) positive amounts of flow are routed only into unsaturated vertices v_j, $j \in U$. For any $j \in U$, we have

$$\sum_{J_k \in \mathcal{J}'(I_j) \cap \mathcal{J}_i} t_{kj} \leq |\mathcal{J}'(I_j) \cap \mathcal{J}_i||I_j|$$

because the edge capacity of (u_k, v_j) is $|I_j|$. For the unsaturated vertex v_{j_0} and the job J_{k_0}, determined in line 10 of the algorithm, the flow along (u_{k_0}, v_{j_0}) is strictly below $|I_{j_0}|$. Hence for $j_0 \in U$ the stronger relation

$$\sum_{J_k \in \mathcal{J}'(I_{j_0}) \cap \mathcal{J}_i} t_{kj_0} < |\mathcal{J}'(I_{j_0}) \cap \mathcal{J}_i||I_{j_0}|$$

holds because by assumption $J_{k_0} \in \mathcal{J}' \cap \mathcal{J}_i$. Taking into account (1), we obtain that expression (3) is

$$\sum_{J_k \in \mathcal{J}' \cap \mathcal{J}_i} w_k < \sum_{j \in U} \min\{|\mathcal{J}'(I_j) \cap \mathcal{J}_i|, m_j - |\mathcal{J}''(I_j)|\}|I_j|s.$$

We will show that for any $j \in U$,

$$\min\{|\mathcal{J}'(I_j) \cap \mathcal{J}_i|, m_j - |\mathcal{J}''(I_j)|\} \leq m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|. \tag{4}$$

Since $s_i \geq s$ this implies that the total work of $\mathcal{J}' \cap \mathcal{J}_i$ is strictly smaller than $\sum_{j \in U}(m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|)|I_j|s_i$, the expression in (2).

So suppose that (4) is violated for some $j \in U$. Then $|\mathcal{J}'(I_j) \cap \mathcal{J}_i| > m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|$ and the number of jobs of $\mathcal{J}_i$ that are active in I_j is strictly larger than m_{ij}. Hence m_{ij} is the total number of processors not yet occupied by $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i-1}$. As the number of jobs of $\mathcal{J}$ active in I_j is at least as large as the corresponding number of jobs of $\mathcal{J}_i$, we have $m_j = m_{ij}$. Since $|\mathcal{J}''(I_j)| \geq |\mathcal{J}''(I_j) \cap \mathcal{J}_i|$ we have $m_j - |\mathcal{J}''(I_j)| \leq m_{ij} - |\mathcal{J}''(I_j) \cap \mathcal{J}_i|$, which contradicts the assumption that (4) was violated. □

Lemma 5. *When phase i ends, $\mathcal{J} = \mathcal{J}_i$.*

Proof. When phase i ends, consider set $\mathcal{J}$ and the flow in the current graph $G = (\mathcal{J}, \vec{m}, s)$. Again, for any edge e let $f(e)$ be the amount of flow along e. Since the value of the computed flow is equal to F_G we have $f(u_0, u_k) = w_k/s$, for any $J_k \in \mathcal{J}$. By Lemma 4, $\mathcal{J}_i \subseteq \mathcal{J}$. So suppose $\mathcal{J}_i \neq \mathcal{J}$, which implies $\mathcal{J}_i \subset \mathcal{J}$. We prove that in this case the speed s_i used for $\mathcal{J}_i$ in an optimal schedule satisfies $s_i \leq s$ and then derive a contradiction.

In the graph $G = (\mathcal{J}, \vec{m}, s)$ we remove the flow associated with any $J_k \in \mathcal{J} \setminus \mathcal{J}_i$. More specifically, for any such J_k, we remove $f(u_k, v_j)$ units of flow along the path (u_0, u_k), (u_k, v_j) and (v_j, v_0) until the flow along (u_0, u_k) is zero. Let $f'(v_j, v_0)$ be the new amount of flow along (v_j, v_0) after the modifications. Then for any interval I_j, at least $\lceil f'(v_j, v_0)/|I_j| \rceil$ jobs of $\mathcal{J}_i$ are active in I_j because for each job at most $|I_j|$ units of flow are routed into v_j. Also, at least $\lceil f'(v_j, v_0)/|I_j| \rceil$ processors are available because $m_j = f(v_j, v_0)/|I_j|$ processors are available in I_j. Hence

$$s_i \leq \sum_{J_k \in \mathcal{J}_i} w_k / \sum_{1 \leq j < |\mathcal{I}|} \lceil f'(v_j, v_0)/|I_j| \rceil |I_j|.$$

The latter expression is upper bounded by s because

$$\sum_{J_k \in \mathcal{J}_i} w_k = s \sum_{J_k \in \mathcal{J}_i} w_k/s = s \sum_{J_k \in \mathcal{J}_i} f(u_0, u_k) = s \sum_{1 \leq j < |\mathcal{I}|} f'(v_j, v_0) \leq s \sum_{1 \leq j < |\mathcal{I}|} \lceil f'(v_j, v_0)/|I_j| \rceil |I_j|.$$

Hence $\mathcal{J}_i$ is processed at speed $s_i \leq s$ while jobs of $\mathcal{J} \setminus \mathcal{J}_i$ are processed at speed s' with $s' < s_i \leq s$. However, this is impossible because the minimum average speed to process the jobs of $\mathcal{J}$ is $s = \sum_{J_k \in \mathcal{J}} w_k / \sum_{1 \leq j < |\mathcal{I}|} m_j |I_j|$. □

We conclude with the following theorem.

Theorem 1. *An optimal schedule can be computed in polynomial time using combinatorial techniques.*

3. Online algorithms

We present adaptions and extensions of the single processor algorithms *Optimal Available* and *Average Rate* [15]. Throughout this section, we consider power functions of the form $P(s) = s^\alpha$, where $\alpha > 1$.

3.1. Algorithm Optimal Available

For single processor environments, *Optimal Available (OA)* works as follows: Whenever a new job arrives, *OA* computes an optimal schedule for the currently available, unfinished jobs. While it is straightforward to extend this strategy to parallel processing environments, the corresponding competitive analysis becomes considerably more involved. We have to prove a series of properties of the online algorithm's schedule and analyze how a schedule changes in response to the arrival of a new job. Our algorithm for parallel processors works as follows.

Algorithm *OA*(*m*). Whenever a new job arrives, compute an optimal schedule for the currently available unfinished jobs. This can be done using the algorithm of Section 2.

At any given time t_0, let $S_{OA(m)}$ denote the schedule of $OA(m)$ for the currently available unfinished jobs. Let $J_1, \ldots, J_n$ be the corresponding jobs, which have arrived by time t_0 but are still unfinished. Let w_i be the remaining processing volume of J_i, $1 \le i \le n$. Since $J_1, \ldots, J_n$ are available at time t_0 we can ignore release times and only have to consider the deadlines $d_1, \ldots, d_n$. Let $\mathcal{I} = \{t_0\} \cup \{d_i \mid 1 \le i \le n\}$ be the set of relevant time points and $\tau_1 < \ldots < \tau_{|\mathcal{I}|}$ be the sorted order of the elements of $\mathcal{I}$, where $|\mathcal{I}| \le n+1$. In the time horizon $[\tau_1, \tau_{|\mathcal{I}|})$, the j-th interval is $I_j = [\tau_j, \tau_{j+1})$, where $1 \le j < |\mathcal{I}|$. Schedule $S_{OA(m)}$ is an optimal schedule for the current jobs $J_1, \ldots, J_n$; otherwise when last computing a schedule, $OA(m)$ would have obtained a better solution for time horizon $[\tau_1, \tau_{|\mathcal{I}|})$. By Lemma 1 we always assume that in $S_{OA(m)}$ each job is processed at a constant speed.

For the given time t_0, let $\mathcal{S}_{OPT}$ denote the set of optimal schedules for the unfinished jobs $J_1, \ldots, J_n$. We have $S_{OA(m)} \in \mathcal{S}_{OPT}$. Since $P(s) = s^\alpha$, there holds $\mathcal{S}_{OPT} = \mathcal{S}'_{OPT}$. Recall that the latter set was introduced in the beginning of Section 2 as the subset of optimal schedules that minimize energy consumption according to the power function $P(s) = s^\alpha$. Hence $S_{OA(m)} \in \mathcal{S}'_{OPT}$ and, by Lemma 2, in each interval I_j of $S_{OA(m)}$ we can rearrange the schedule such that each processor uses a fixed, non-varying speed in I_j, $1 \le j < |\mathcal{I}|$. In the following we restrict ourselves to schedules $S_{OA(m)}$ satisfying this property. Such schedules are feasible and incur a minimum energy. Let $s_{l,j}$ be the speed used by processor l in I_j, where $1 \le l \le m$ and $1 \le j < |\mathcal{I}|$. Also note that Lemma 3 holds.

The next Lemma 6 states that we can modify $S_{OA(m)}$ even further so that on each processor the speed levels used form a non-increasing sequence. The modification is obtained by simply permuting within each interval I_j the schedules on the m processors, $1 \le j < |\mathcal{I}|$. Hence feasibility and energy consumption of the overall schedule are not affected. Throughout the analysis we always consider schedules $S_{OA(m)}$ that also fulfill this additional property. This will be essential in the proofs of the subsequent Lemmas 7–9.

Lemma 6. *Given $S_{OA(m)}$, in each interval I_j, $1 \le j < |\mathcal{I}|$, we can permute the schedules on the m processors such that the following property holds. For any processor l, $1 \le l \le m$, inequality $s_{l,j} \ge s_{l,j+1}$ is satisfied for $j = 1, \ldots, |\mathcal{I}| - 2$.*

Proof. In each interval I_j, $1 \le j < |\mathcal{I}|$, we simply sort the schedules on the m processors in order of non-increasing speeds such that $s_{1,j} \ge \ldots \ge s_{m,j}$. Consider an arbitrary j, $1 \le j \le |\mathcal{I}| - 2$, and suppose that the desired property does not hold for all l, $1 \le l \le m$. Then let l_0 be the smallest index such that $s_{l_0,j} < s_{l_0,j+1}$. In I_{j+1} the first l_0 processors, running at speeds $s_{1,j+1} \ge \ldots \ge s_{l_0,j+1}$, execute a set $\mathcal{J}'(I_{j+1})$ of at least l_0 different jobs in the interval because a job cannot be executed in parallel on two or more processors. The jobs of $\mathcal{J}'(I_{j+1})$ can only be processed on the first $l_0 - 1$ processors in I_j because in $S_{OA(m)}$ each job is processed at a constant speed and the last $m - l_0 + 1$ processors in I_j use speeds which are strictly smaller than $s_{1,j+1}, \ldots, s_{l_0,j+1}$. Consider a time window $W = [\tau_j, \tau_j + \delta)$ with $\delta > 0$ in I_j such that each of the first l_0 processors does not change the job it executes in W. Among the at least l_0 jobs of $\mathcal{J}'(I_{j+1})$ there must exist a J_i which is not processed in W because jobs of $\mathcal{J}'(I_{j+1})$ are not executed on the $m - l_0 + 1$ last processors in I_j and the first $l_0 - 1$ processors handle at most $l_0 - 1$ different jobs of $\mathcal{J}'(I_{j+1})$ in W. Suppose that J_i is processed for δ_i time units in I_{j+1} using speed $s(J_i)$. Set $\delta' = \min\{\delta, \delta_i\}$ and $\epsilon = (s_{l_0,j+1} - s_{l_0,j})/2$. We now modify the schedule as follows. In I_{j+1} for δ' time units where J_i is processed we reduce the speed by ϵ. In I_j, in the time window $W' = [\tau_j, \tau_j + \delta')$ we increase the speed of processor l_0 by ϵ and use the extra processing capacity of $\delta'\epsilon$ units to process $\delta'\epsilon$ units of J_i. We obtain a feasible schedule. Since $s(J_i) - \epsilon \ge s_{j+1,l_0} - \epsilon = s_{j,l_0} + \epsilon$, by the convexity of the power function $P(s) = s^\alpha$ the modified schedule has a strictly smaller energy consumption. We obtain a contradiction to the fact that the original schedule was optimal. □

We investigate the event that a new job J_{n+1} arrives at time t_0. As usual d_{n+1} and w_{n+1} are the deadline and the processing volume of J_{n+1}. Let $S_{OA(m)}$ be the schedule of $OA(m)$ immediately before the arrival of J_{n+1}, and let $S'_{OA(m)}$ be the schedule immediately after the arrival when $OA(m)$ has just computed a new schedule. We present two lemmas that relate the two schedules. Loosely speaking we prove that processor speeds can only increase. The speed at which a

job J_k, $1 \le k \le n$, is processed in $S'_{OA(m)}$ is at least as high as the corresponding speed in $S_{OA(m)}$. Furthermore, at any time $t \ge t_0$, the minimum processor speed in $S'_{OA(m)}$ is at least as high as that in $S_{OA(m)}$.

Schedules $S_{OA(m)}$ and $S'_{OA(m)}$ are optimal schedules for the respective workloads. Recall that they process jobs at constant speeds. Let $s(J_k)$ be the speed at which J_k is processed in $S_{OA(m)}$, $1 \le k \le n$. Let $s'(J_k)$ be the speed used for J_k in $S'_{OA(m)}$, $1 \le k \le n+1$.

Lemma 7. *There holds $s'(J_k) \ge s(J_k)$, for any $1 \le k \le n$.*

Proof. In $S_{OA(m)}$ let $s_1 > s_2 > \ldots > s_p$ be the speeds used in the schedule and let $\mathcal{J}_1, \ldots, \mathcal{J}_p$ be the associated job sets, i.e. the jobs of $\mathcal{J}_i$ are processed at speed s_i, $1 \le i \le p$. In the following we assume that the statement of the lemma does not hold and derive a contradiction. So let i be the smallest index such that $\mathcal{J}_i$ contains a job J_k with $s'(J_k) < s(J_k) = s_i$. We partition $\mathcal{J}_i$ into two sets $\mathcal{J}_{i1}$ and $\mathcal{J}_{i2}$ such that all $J_k \in \mathcal{J}_{i1}$ satisfy $s'(J_k) \ge s_i$, as desired. For each job $J_k \in \mathcal{J}_{i2}$ we have $s'(J_k) < s_i$. By the choice of i, any job $J_k \in \mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i-1}$ also fulfills the desired property $s'(J_k) \ge s(J_k)$.

Let t_1 be the first point of time when the processing of any job of $\mathcal{J}_{i2}$ starts in $S'_{OA(m)}$. Let P be a processor executing such a job at time t_1. We first argue that any $J_k \in \mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i-1} \cup J_{i1}$ that is active at time $t \ge t_1$, i.e. that satisfies $d_k > t_1$, is scheduled at all times throughout $[t_1, d_k)$: So suppose J_k were not executed on any of the processors in a time window $W \subseteq [t_1, d_k)$ and let $\delta > 0$ be the length of W. Determine a δ' with $0 < \delta' \le \delta$ such that during the first δ' time units of W processor P does not change the job it executes. We set $\delta_k = \min\{\delta', w_k/s'(J_k)\}$. Here $w_k/s'(J_k)$ is the total time for which J_k is scheduled in $S'_{OA(m)}$. Let $s(P, W)$ be the speed used by processor P within the first δ' time units of W and set $\epsilon = (s'(J_k) - s(P, W))/2$. The value of ϵ is strictly positive because $s'(J_k) \ge s_i > s(P, W)$. The last inequality holds because during the first δ' time units of W processor P executes a job of $\mathcal{J}_{i2}$ at a speed that is strictly smaller than s_i. We now modify $S'_{OA(m)}$ as follows. During any δ_k time units where J_k is processed we reduce the speed by ϵ. During the first δ_k time units of W we increase the speed by ϵ and use the extra processing capacity of $\delta_k\epsilon$ units to execute $\delta_k\epsilon$ units of J_k. Hence we obtain a new feasible schedule. Since $s'(J_k) - \epsilon = s(P, W) + \epsilon$, by the convexity of the power function $P(s) = s^\alpha$, the schedule has a strictly smaller energy consumption, contradicting the fact that $S'_{OA(m)}$ is an optimal schedule for the given workload.

Next consider any fixed time $t \ge t_1$. In $S_{OA(m)}$ let m_{i2} be the number of processors that execute jobs of $\mathcal{J}_{i2}$ at time t. Similarly, in $S'_{OA(m)}$ let m'_{i2} be the number of processors that execute jobs of $\mathcal{J}_{i2}$ at time t. We will prove $m_{i2} \ge m'_{i2}$. Let n_l, where $1 \le l \le i-1$, be the number of jobs J_k of $\mathcal{J}_l$ that are active at time t, i.e. that satisfy $d_k > t$. Furthermore, let n_{i1} and n_{i2} be the number of jobs of $\mathcal{J}_{i1}$ and $\mathcal{J}_{i2}$, respectively, that are active at time t. As shown in the last paragraph, all jobs of $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i-1} \cup J_{i1}$ that are active at time t are also processed at this time in $S'_{OA(m)}$. Hence

$$m'_{i2} \le m - \sum_{l=1}^{i-1} n_l - n_{i1}. \tag{5}$$

Strict inequality holds, for instance, if the new job J_{n+1} or a job of $\mathcal{J}_{i+1} \cup \ldots \cup \mathcal{J}_p$ is scheduled at time t. Let m_l be the number of processors executing jobs of $\mathcal{J}_l$ in $S_{OA(m)}$, where $1 \le l \le i$. By Lemma 3, $m_i = \min\{n_i, m - \sum_{l=1}^{i-1} m_l\}$. We remark that Lemma 3 is formulated with respect to an interval I_j. We can apply the statement simply by considering the interval containing t. Since $m_l \le n_l$, for $1 \le l \le i-1$, we obtain

$$m_i \ge \min\{n_i, m - \sum_{l=1}^{i-1} n_l\}. \tag{6}$$

If the latter minimum is given by the term $m - \sum_{l=1}^{i-1} n_l$, then in (5) we obtain $m'_{i2} \le m - \sum_{l=1}^{i-1} n_l - n_{i1} \le m_i - n_{i1} \le m_{i2}$. The last inequality holds because m_{i2} cannot be smaller than m_i, the number of machines used by jobs of $\mathcal{J}_i$, minus the number of jobs of $\mathcal{J}_{i1}$ active at time t. Hence $m'_{i2} \le m_{i2}$, as desired. If the minimum in (6) is given by n_i, then since $m_{i2} \ge m_i - n_{i1}$, we have $m_{i2} \ge n_i - n_{i1}$. As $n_i - n_{i1} = n_{i2} \ge m'_{i2}$ we conclude again $m_{i2} \ge m'_{i2}$.

Finally let T' be the total processing time used for jobs of $\mathcal{J}_{i2}$ in $S'_{OA(m)}$. This value can be determined as follows. The time horizon of $S'_{OA(m)}$ is partitioned into maximal intervals such that the number of processors handling jobs of $\mathcal{J}_{i2}$ does not change within an interval. For each such interval, the length of the interval is multiplied by the number of processors executing jobs of $\mathcal{J}_{i2}$. Value T' is simply the sum of all these products. Similarly, let T be the total processing time used for jobs of $\mathcal{J}_{i2}$ in $S_{OA(m)}$. At any time t we have $m_{i2} \ge m'_{i2}$. This was shown above for times $t \ge t_1$. It trivially holds for times t with $t_0 \le t < t_1$ because no jobs of $\mathcal{J}_{i2}$ are processed in $S'_{OA(m)}$ at that time. Hence the number of processors handling jobs of $\mathcal{J}_{i2}$ in $S_{OA(m)}$ is always at least as high as the corresponding number in $S_{OA(m)}$. This implies $T \ge T'$. The total processing volume $\sum_{J_k \in \mathcal{J}_{i2}} w_k$ of jobs in $\mathcal{J}_{i2}$ fills T' time units using a speed or speeds of $s < s_i$ in $S'_{OA(m)}$. Thus using a speed of s_i the processing volume is not sufficient to fill $T > T'$ time units in $S_{OA(m)}$. We obtain a contradiction to the fact that jobs of $\mathcal{J}_i$ are processed at constant speed s_i in $S_{OA(m)}$. □

Lemma 8. *At any time t, where $t \geq t_0$, the minimum speed used by the m processors at time t in $S'_{OA(m)}$ is at least as high as the minimum speed used on the m processors at time t in $S_{OA(m)}$.*

Proof. Consider an arbitrary time $t \geq t_0$. Let $s_{\min}$ be the minimum speed on the m processors in $S_{OA(m)}$ at time t. Let $s'_{\min}$ be the corresponding minimum speed in $S'_{OA(m)}$ at time t. If $s_{\min} = 0$, then the lemma trivially holds. So suppose $s_{\min} > 0$ and assume that the statement of the lemma does not hold, i.e. $s'_{\min} < s_{\min}$. Since $s_{\min} > 0$, m different jobs are processed at time t in $S_{OA(m)}$. By Lemma 7, for any job, the speed at which it is processed in $S'_{OA(m)}$ is at least as high as the corresponding speed in $S_{OA(m)}$. Hence among the m jobs processed in $S_{OA(m)}$ at time t there must exist at least one job J_k that is not executed in $S'_{OA(m)}$ at time t. For this job we have $d_k > t$ because J_k is available for processing and hence active at time t. Determine a $\delta_k > 0$ with $\delta_k \leq \min\{d_k - t, w_k/s'(J_k)\}$ such that J_k is not processed in the interval $[t, t+\delta_k)$ in $S'_{OA(m)}$. Choose a δ with $0 < \delta \leq \delta_k$ such that in $S'_{OA(m)}$ the last processor m, running at minimum speed $s'_{\min}$ at time t, does not change the job it executes within time window $W = [t, t+\delta)$. Throughout W a speed of $s'_{\min}$ is used on processor m. Set $\epsilon = (s_{\min} - s'_{\min})/2$, which is a strictly positive value.

We now modify $S'_{OA(m)}$. During δ time units in which J_k is processed we reduce the speed by ϵ. These time units exist because $\delta \leq w_k/s'(J_k)$. In the time window W we increase the speed on the last processor m by ϵ and use the extra processing capacity of $\delta\epsilon$ unit to process $\delta\epsilon$ units of J_k. Since $\delta \leq d_k - t$ job J_k finishes by its deadline, and we obtain a feasible schedule. Throughout W the newly set speed on the last processor m is $s'_{\min} + \epsilon$. Finally notice $s'_{\min} + \epsilon = s_{\min} - \epsilon \leq s(J_k) - \epsilon \leq s'(J_k) - \epsilon$. The last inequality follows from Lemma 7. Hence $s'_{\min} + \epsilon \leq s'(J_k) - \epsilon$, and by the convexity of the power function $P(s) = s^\alpha$, we obtain a schedule with a strictly smaller power consumption. This is a contradiction to the fact that $S'_{OA(m)}$ is optimal. □

We finally need a lemma that specifies the minimum processor speed for a time window if a job finishes at some time strictly before its deadline.

Lemma 9. *If in $S_{OA(m)}$ a job J_k finishes at some time t with $t < d_k$, then throughout $[t, d_k)$ the minimum speed on the m processors is at least $s(J_k)$.*

Proof. Consider a job J_k that finishes at time $t < d_k$. Suppose that the lemma does not hold and let t' with $t \leq t' < d_k$ be a time such that the minimum speed on the m processors is strictly smaller than $s(J_k)$. Recall that we consider schedules satisfying the property of Lemma 6, i.e. on any processor the speed levels form a non-increasing sequence. Hence within $W = [t', d_k)$ there exists a processor P whose speed throughout W is upper bounded by some $s < s(J_k)$. Choose a positive δ with $\delta \leq \min\{d_k - t', w_k/s(J_k)\}$ such that processor P does not switch jobs in $W' = [t', t'+\delta)$. Set $\epsilon = (s(J_k) - s)/2$. We modify the schedule as follows. During δ time units where J_k is processed we reduce the speed by ϵ. In W' we increase the speed of processor P by ϵ and use the extra processing capacity to finish the missing $\delta\epsilon$ processing units of J_k. The resulting schedule is feasible and incurs a strictly smaller energy consumption. Hence we obtain a contradiction to the fact that $S_{OA(m)}$ is an optimal schedule. □

We next turn to the competitive analysis of $OA(m)$. We use a potential function that is inspired by a potential used by Bansal et al. [5]. We first define our modified potential function. At any time t consider the schedule of $S_{OA(m)}$. Let $s_1 > \ldots > s_p$ be the speeds used in the schedule and let $\mathcal{J}_1, \ldots, \mathcal{J}_p$ be the associated job sets, i.e. jobs of $\mathcal{J}_i$ are processed at speed s_i. Let $W_{OA(m)}(i)$ be the total remaining processing volume of jobs $J_k \in \mathcal{J}_i$ in $OA(m)$'s schedule $1 \leq i \leq p$. Similarly, let $W_{OPT}(i)$ be the total remaining processing volume that *OPT* has to finish for jobs $J_k \in \mathcal{J}_i$, $1 \leq i \leq p$. We remark that, for a job $J_k \in \mathcal{J}_i$, *OPT*'s remaining work might be larger than that of $OA(m)$ if *OPT* has completed less work on J_k so far. Sets $\mathcal{J}_1, \ldots, \mathcal{J}_p$ represent the jobs that $OA(m)$ has not yet finished. Let $\mathcal{J}'$ be the set of jobs that are finished by $OA(m)$ but still unfinished by *OPT*. Obviously, these jobs have a deadline after the current time t. If $\mathcal{J}' \neq \emptyset$, then for any job of $\mathcal{J}'$ consider the speed $OA(m)$ used when last processing the job. Partition $\mathcal{J}'$ according to these speeds, i.e. $\mathcal{J}'_i$ consists of those jobs that $OA(m)$ executed last at speed s'_i, where $1 \leq i \leq p'$ for some positive p'. Let $W'_{OPT}(i)$ be the total remaining processing volume *OPT* has to finish on jobs of $\mathcal{J}'_i$, $1 \leq i \leq p'$. If $\mathcal{J}' = \emptyset$, we simply set $s'_1 = 0$ and $\mathcal{J}'_1 = \emptyset$. The potential Φ is defined as follows.

$$\Phi = \alpha \sum_{i \geq 1} (s_i)^{\alpha-1} (W_{OA(m)}(i) - \alpha W_{OPT}(i)) - \alpha^2 \sum_{i \geq 1} (s'_i)^{\alpha-1} W'_{OPT}(i)$$

Compared to the potential function used to analyze *OA* in the single processor setting, our function here consists of a second term representing *OPT*'s unfinished work. This second term is essential to establish the competitiveness of α^α and, in particular, is crucially used in the analysis of the working case (see below) where we have to consider m processor pairs.

At any time t let $s_{OA(m),1}(t) \geq \ldots \geq s_{OA(m),m}(t)$ and $s_{OPT,1}(t) \geq \ldots \geq s_{OPT,m}(t)$ be the speeds used by $OA(m)$ and *OPT* on the m processors, respectively. In the remainder of this section we will prove the following properties.

(a) Whenever a new job arrives or a job is finished by $OA(m)$ or OPT, the potential does not increase.
(b) At all other times

$$\sum_{l=1}^{m}(s_{OA(m),l}(t))^{\alpha}-\alpha^{\alpha}\sum_{l=1}^{m}(s_{OPT,l}(t))^{\alpha}+\frac{d\Phi(t)}{dt}\leq 0. \tag{7}$$

Integrating over all times we obtain that $OA(m)$ is α^{α}-competitive.

Working case We prove property (b). So let t be any time when the algorithms are working on jobs. In a first step we match the processors of $OA(m)$ to those of OPT. This matching is constructed as follows. First, processors handling the same jobs are paired. More specifically, if a job J_k is executed on a processor l in $OA(m)$'s schedule and on processor l' in OPT's schedule, then we match the two processors l and l'. All remaining processors of $OA(m)$ and OPT, handling disjoint job sets, are matched in an arbitrary way.

For each matched processor pair l, l', we consider its contribution in (7) and prove that the contribution is non-positive. Summing over all pairs l, l', this establishes (7). If the speed of processor l' in OPT's schedule is 0, then the analysis is simple: If the speed on processor l in $OA(m)$ schedule is also 0, then the contribution of the processor pair l, l' in (7) is 0. If the speed on processor l is positive, then $OA(m)$ executes a job on the processor. Suppose that the job is contained in set $\mathcal{J}_i$. Then in Φ the value of $W_{OA(m)}(i)$ decreases at rate s_i and the contribution of the processor pair l, l' in (7) is $s_i^{\alpha}-\alpha(s_i)^{\alpha-1}s_i<0$.

Hence in the following we assume that the speed of processor l' in OPT's schedule is positive. We show that $OA(m)$ also executes a job J_k on its processor l and that the contribution of the processor pair l, l' is upper bounded by

$$(1-\alpha)s_i^{\alpha}+\alpha^2 s_i^{\alpha-1}s_{OPT,l'}(t)-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}, \tag{8}$$

where i is the index such that $J_k\in\mathcal{J}_i$. As in [5] one can then show the non-positivity of this expression. First consider the case that both processors l and l' handle the same job J_k. Let $\mathcal{J}_i$ be the set containing J_k. Then $W_{OA(m)}(i)$ decreases at rate s_i and $W_{OPT}(i)$ decreases at rate $s_{OPT,l'}(t)$. Hence the contribution of the processor pair in (7) is $s_i^{\alpha}-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}+(-\alpha(s_i)^{\alpha-1}s_i+\alpha^2(s_i)^{\alpha-1}s_{OPT,l'}(t))=(1-\alpha)s_i^{\alpha}+\alpha^2(s_i)^{\alpha-1}s_{OPT,l'}(t)-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}$, as stated in (8).

Next consider the case that the job $J_{k'}$ executed on processor l' in OPT's schedule is not executed by $OA(m)$ on processor l and hence is not executed on any of its processors at time t. If $J_{k'}$ is still unfinished by $OA(m)$, then let $\mathcal{J}_{i'}$ be the set containing $J_{k'}$. Recall that $OA(m)$'s schedule is optimal. Hence, by Lemma 3, it always processes the jobs from the smallest indexed sets $\mathcal{J}_i$, which in turn use the highest speeds s_i. Thus, if $J_{k'}$ is not executed by $OA(m)$, then the schedule currently processes m jobs, each of which is contained in $\mathcal{J}_1\cup\ldots\cup\mathcal{J}_{i'-1}$ or contained in $\mathcal{J}_{i'}$ but is different from $J_{k'}$. Let J_k be the job executed by $OA(m)$ on processor l and let $\mathcal{J}_i$ be the job set containing J_k. Then $i\leq i'$ and $s_i\geq s_{i'}$. In Φ the term $W_{OA(m)}(i)$ decreases at rate s_i while the term $W_{OPT}(i')$ decreases at rate $s_{OPT,l'}(t)$. Hence the contribution of the processor pair l, l' in (7) is $s_i^{\alpha}-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}+(-\alpha(s_i)^{\alpha-1}s_i+\alpha^2(s_{i'})^{\alpha-1}s_{OPT,l'}(t))\leq(1-\alpha)s_i^{\alpha}+\alpha^2(s_i)^{\alpha-1}s_{OPT,l'}(t)-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}$, which is the bound of (8).

It remains to consider the case that $J_{k'}$ is already finished by $OA(m)$. Then $J_{k'}$ is contained in the set $\mathcal{J}'$ of jobs that are finished by $OA(m)$ but unfinished by OPT. Let $\mathcal{J}'_{i'}$ be the set containing $J_{k'}$. In Φ the term $W'_{OPT}(i')$ decreases at rate $s_{OPT,l'}(t)$. When $OA(m)$ finished $J_{k'}$ in its schedule, it used speed $s'_{i'}$. By Lemma 9, in the former schedule of $OA(m)$, the minimum processor speed throughout the time window until $d_{k'}$ was always at least $s'_{i'}$. This time window contains the current time, i.e. $t<d_{k'}$. Since the completion of $J_{k'}$ $OA(m)$ might have computed new schedules in response to the arrival of new jobs but, by Lemma 8, the minimum processor speed at any time can only increase. Hence in the current schedule of $OA(m)$ and at the current time t the minimum processor speed is at least $s'_{i'}$. Thus on its processor l $OA(m)$ uses a speed of at least $s'_{i'}>0$. Hence $OA(m)$ executes a job J_k that belongs to, say, set $\mathcal{J}_i$. We have $s_i\geq s'_{i'}$. In Φ the term $W_{OA(m)}(i)$ decreases at rate s_i. We conclude again that the contribution of the processor pair l, l' is $s_i^{\alpha}-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}+(-\alpha(s_i)^{\alpha}+\alpha^2(s'_{i'})^{\alpha-1}s_{OPT,l'}(t))\leq(1-\alpha)s_i^{\alpha}+\alpha^2(s_i)^{\alpha-1}s_{OPT,l'}(t)-\alpha^{\alpha}(s_{OPT,l'}(t))^{\alpha}$, which is again the bound of (8).

Completion/arrival case When OPT finishes a job J_k the potential does not change because in the terms $W_{OPT}(i)$ or $W_{OPT}(i')$ the remaining processing volume of J_k simply goes to 0. Similarly if $OA(m)$ completes a job already finished by OPT, the potential does not change. If $OA(m)$ completes a job J_k not yet finished by OPT, then let $\mathcal{J}_i$ be the set containing J_k immediately before completion. In the first term of Φ, $W_{OPT}(i)$ increases by $\alpha^2(s_i)^{\alpha-1}w_k$, where w_k is the remaining work of J_k to be done by OPT. However, at the same time, the second term of Φ decreases by exactly $\alpha^2(s_i)^{\alpha-1}w_k$ because J_k opens or joins a set $\mathcal{J}'_{i'}$ with $s'_{i'}=s_i$.

When a new job J_{n+1} arrives, the structure of the analysis is similar to that in the single processor case, see [5]. We imagine that a job J_{n+1} of processing volume 0 arrives. Then we increase the processing volume to its true size. However, in the multi-processor setting, we have to study how the schedule of $OA(m)$ changes if the processing volume of J_{n+1} increases from a value w_{n+1} to $w'_{n+1}=w_{n+1}+\epsilon$, for some $\epsilon>0$. Let $S_{OA(m)}$ and $S'_{OA(m)}$ be the schedules of $OA(m)$ before and after the increase, respectively. For any job J_k, $1\leq k\leq n+1$, let $s(J_k)$ and $s'(J_k)$ be the speeds used for J_k in $S_{OA(m)}$ and $S'_{OA(m)}$, respectively. The next lemma states that the speeds at which jobs are processed does not decrease. The proof is identical to that of Lemma 7.

Lemma 10. *There holds $s'(J_k) \geq s(J_k)$, for any $1 \leq k \leq n+1$.*

In $S_{OA(m)}$, let $\mathcal{J}_{i_0}$ be the job set containing J_{n+1}, i.e. $J_{n+1} \in \mathcal{J}_{i_0}$, where $1 \leq i_0 \leq p$. The following lemma implies that the speeds of jobs $J_k \in \mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ do not change when increasing the processing volume of J_{n+1}.

Lemma 11. *There holds $s'(J_k) = s(J_k)$, for any $J_k \in \mathcal{J}_{i_0+1} \cup \ldots \cup J_p$.*

Proof. We first analyze $S_{OA(m)}$ and $S'_{OA(m)}$ with respect to the times when jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ are processed. Consider any time $t \geq t_0$ in the scheduling horizon, where t_0 is again the current time. Let m_1 be the number of processors executing jobs of $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i_0}$ in $S_{OA(m)}$ at time t, and let m_2 be the number of processors executing jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup J_p$ in $S_{OA(m)}$ at time t. Values m'_1 and m'_2 are defined analogously with respect to $S'_{OA(m)}$. In the following we will prove $m'_2 \geq m_2$.

If $m_2 = 0$, there is nothing to show. So suppose $m_2 > 0$. By Lemma 3 exactly m_1 jobs of $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i_0}$ are active at time t, i.e. have a deadline after time t; if there were more active jobs, $OA(m)$ would have assigned more processors to the jobs of $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i_0}$ because they are executed at higher speeds than the jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$. We next assume $m'_2 < m_2$ and derive a contradiction. We distinguish two cases depending on whether or not $S'_{OA(m)}$ has an idle processor at time t.

If there is no idle processor in $S'_{OA(m)}$ at time t, then $m'_1 = m - m'_2 > m - m_2 \geq m_1$ jobs of $\mathcal{J}_1 \cup \ldots \cup \mathcal{J}_{i_0}$ must be executed in $S'_{OA(m)}$ at time t. However, this is impossible because only m_1 jobs of this union of sets are active and hence available for processing at time t. On the other hand, if there is a processor P that is idle in $S'_{OA(m)}$ at time t, then there must exist a time window W containing time t such that P is idle throughout W. Moreover, since $m'_2 < m_2$, there must exist a job $J_k \in \mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ that is active but not processed at time t. Hence, within W we can select a time window $W' \subseteq W$ of length δ, where $\delta \leq w(J_k)/s'(J_k)$, such that J_k is not executed on any processor in $S'_{OA(m)}$ throughout W'. We now modify $S'_{OA(m)}$ as follows. For any δ time units where J_k is scheduled, we reduce the speed to $s'(J_k)/2$. On processor P we execute J_k at speed $s'(J_k)/2$ for δ time units in W'. By convexity of the power function, we obtain a better schedule with a strictly smaller energy consumption, contradicting the fact that $S'_{OA(m)}$ is an optimal schedule.

Let T be the total time used to process jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ in $S_{OA(m)}$. As in the proof of Lemma 7, this time is computed as follows. We partition the scheduling horizon of $S_{OA(m)}$ into maximal intervals such that within each interval the number of processors executing jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ does not change. The length of each interval is multiplied by the number of processors handling jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$. Then T is simply the sum of these products. Analogously, let T' be the total time used to process jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ in $S'_{OA(m)}$. In the last paragraph we showed that at any time t, $m'_2 \geq m_2$. Hence $T' \geq T$.

We are now ready to finish the proof of our lemma. By Lemma 10 we have $s'(J_k) \geq s(J_k)$ for any $J_k \in \mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$. Suppose that in the latter set there exists a job J_{k_0} with $s'(J_{k_0}) > s(J_{k_0})$. In $S_{OA(m)}$ the jobs of $\mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ are processed for T time units. The total processing volume of these jobs cannot fill $T' \geq T$ time units if J_{k_0} is processed at a strictly higher speed $s'(J_{k_0}) > s(J_{k_0})$ while all other jobs $J_k \in \mathcal{J}_{i_0+1} \cup \ldots \cup \mathcal{J}_p$ are processed at speed $s'(J_k) \geq s(J_k)$. We obtain a contradiction. □

In the remainder of this section we describe the change in potential arising in response to the arrival of a new job J_{n+1}. If the deadline d_{n+1} does not coincide with the deadline of a previous job J_i, $1 \leq i \leq n$, we have to update the set of intervals I_j, $1 \leq j < |\mathcal{I}|$. We add d_{n+1} to $\mathcal{I}$. If d_{n+1} is the last deadline of any of the jobs, we introduce a new interval $[\tau_{\max}, d_{n+1})$, where $\tau_{\max} = \max_{1 \leq i \leq n} d_i$; otherwise we split the interval $[\tau_j, \tau_{j+1})$ containing d_{n+1} into two intervals $[\tau_j, d_{n+1})$ and $[d_{n+1}, \tau_{j+1})$.

In a next step we add J_{n+1} to the current schedule $S_{OA(m)}$ assuming that the processing volume of J_{n+1} is zero. Let I_j be the interval ending at time d_{n+1}. If $S_{OA(m)}$ has an idle processor in I_j, then we open a new job set $\mathcal{J}_{p+1}$ with associated speed $s_{p+1} = 0$. Otherwise let s_i be the lowest speed used by any of the processors in I_j. We add J_{n+1} to $\mathcal{J}_i$. In the following we increase the processing volume of J_{n+1} from zero to its final size w_{n+1}. The increase proceeds in a series of phases such that, during a phase, the job sets $\mathcal{J}_i$, $i \geq 1$, do not change. However, at the beginning of a phase the job set $\mathcal{J}_{i_0}$ containing J_{n+1} might split into two sets. Moreover, at the end of a phase the set $\mathcal{J}_{i_0}$ might merge with the next set $\mathcal{J}_{i_0-1}$ executed at the next higher speed level. So suppose that during a phase the processing volume of J_{n+1} increases by ϵ. By Lemma 11, the speeds of the jobs in sets $\mathcal{J}_i$, $i > i_0$, do not change. Moreover, by Lemma 10, the speeds of the other jobs can only increase. Hence, by convexity of the power function, an optimal solution will increase the speed at the lowest possible level, which is s_{i_0}. In the current schedule of $OA(m)$, let T_{i_0} be the total time for which jobs of $\mathcal{J}_{i_0}$ are processed. The speed at which jobs of $\mathcal{J}_{i_0}$ are processed increases from $W_{OA(m)}(i_0)/T_{i_0}$ to $(W_{OA(m)}(i_0) + \epsilon)/T_{i_0}$. Hence the potential change is given by

$$\Delta\Phi = \alpha\left(\frac{W_{OA(m)}(i_0) + \epsilon}{T_{i_0}}\right)(W_{OA(m)}(i_0) + \epsilon - \alpha(W_{OPT}(i_0) + \epsilon)) - \alpha\left(\frac{W_{OA(m)}(i_0)}{T_{i_0}}\right)(W_{OA(m)}(i_0) - \alpha(W_{OPT}(i_0)).$$

Algorithm *AVR*(m): In each interval I_t, $0 \le t < T$, execute the following.

1. $\mathcal{J}'_t := \mathcal{J}_t$; M is the set of all processors in the system;
2. **while** $\mathcal{J}'_t \neq \emptyset$ **do**
3. **while** $\max_{J_i \in \mathcal{J}'_t} \delta_i > \Delta'_t/|M|$ **do**
4. Pick a processor $l \in M$; Schedule $J_i := \arg\max_{J_i \in \mathcal{J}'_t} \delta_i$ at a speed of δ_i on processor l;
5. $\mathcal{J}'_t := \mathcal{J}'_t \setminus \{J_i\}$; $M := M \setminus \{l\}$;
6. $s_\Delta := \Delta'_t/|M|$; Feasibly schedule every job $J_i \in \mathcal{J}'_t$ at a speed of s_Δ.

Fig. 3. The algorithm *AVR*(m).

As in [5] we can show the non-positivity of the last expression.

At the beginning of a phase $\mathcal{J}_{i_0}$ might split into two sets $\mathcal{J}'_{i_0}$ and $\mathcal{J}''_{i_0}$. Set $\mathcal{J}'_{i_0}$ contains those jobs that, in an optimal schedule, cannot be executed at an increased speed when raising the processing volume of J_{n+1}. Set $\mathcal{J}''_{i_0}$ contains the jobs for which the speed can be increased. This split into two sets does not change the value of the potential function. For completeness we mention that a job of $\mathcal{J}_{i_0}$ whose execution starts, say, in interval $I_j = [\tau_j, \tau_{j+1})$ belongs to $\mathcal{J}'_{i_0}$ iff in the current schedule (a) no job $J_k \in \mathcal{J}_{i_0}$ with $d_k > \tau_j$ is processed before τ_j or (b) any job $J_k \in \mathcal{J}_{i_0}$ with $d_k > \tau_j$ that is also processed before τ_j is scheduled continuously without interruption throughout $[\tau_j, d_k)$. When the processing volume of J_{n+1} increases as described above, the jobs of $\mathcal{J}'_{i_0}$ remain at their speed level s_{i_0} while the speed of the jobs in $\mathcal{J}''_{i_0}$ increases.

At the end of a phase $\mathcal{J}_{i_0}$ might be merged with the next set $\mathcal{J}_{i_0-1}$. This event occurs if speed value s_{i_0} reaches the next higher level s_{i_0-1}. We always choose ϵ such that $(W_{OA(m)}(i_0) + \epsilon)/T_{i_0}$ does not exceed s_{i_0-1}. Again the merge of two job set does not change the value of the potential function.

Theorem 2. *Algorithm OA(m) is α^α-competitive.*

3.2. Algorithm Average Rate

The original *Average Rate* algorithm for a single processor considers job densities, where the density δ_i of a job J_i is defined as $\delta_i = w_i/(d_i - r_i)$. At time t the processor speed s_t is set to the sum of the densities of the jobs active at time t. Using these speeds s_t, *AVR* always processes the job having the earliest deadline among the active unfinished jobs.

In the following we describe our algorithm, called *AVR*(m), for m parallel processors. We assume without loss of generality that all release times and deadlines are integers. Moreover, we assume that the earliest release time $\min_{1 \le i \le n} r_i$ is equal to 0. Let $T = \max_{1 \le i \le n} d_i$ be the last deadline. For any time interval $I_t = [t, t+1)$ $0 \le t < T$, let $\mathcal{J}_t$ be the set of jobs J_i that are active in I_t, i.e. that satisfy $I_t \subseteq [r_i, d_i)$. Algorithm *AVR*(m) makes scheduling decisions for the intervals I_t over time, $0 \le t < T$. In each I_t, the algorithm schedules δ_i units of w_i, for each job $J_i \in \mathcal{J}_t$. The schedule is computed iteratively. In each iteration there is a set $\mathcal{J}'_t \subseteq \mathcal{J}_t$ of jobs to be scheduled on a subset M of the processors in interval I_t. Initially $\mathcal{J}'_t = \mathcal{J}_t$, and M contains all the processors of the system. Furthermore, let $\Delta_t = \sum_{J_i \in \mathcal{J}_t} \delta_i$ be the total density of jobs active at time t, and $\Delta'_t = \sum_{J_i \in \mathcal{J}'_t} \delta_i$ be the total density of the jobs in $\mathcal{J}'_t$. The average load of the jobs in $\mathcal{J}_t$ and $\mathcal{J}'_t$ is Δ_t/m and $\Delta'_t/|M|$, respectively.

In each iteration, if $\max_{J_i \in \mathcal{J}'_t} \delta_i > \Delta'_t/|M|$, then a job J_i of maximum density among the jobs in $\mathcal{J}'_t$ is scheduled on a processor $l \in M$. Job J_i is processed during the whole interval I_t at a speed of $s_{t,l} = \delta_i$. Then $\mathcal{J}'_t$ and M are updated to $\mathcal{J}'_t \setminus \{J_i\}$ and $M \setminus \{l\}$, respectively. On the other hand if $\max_{J_i \in \mathcal{J}'_t} \delta_i \le \Delta'_t/|M|$, the complete set $\mathcal{J}'_t$ is scheduled at a uniform speed of $s_\Delta = \Delta'_t/|M|$ on the processors of M as follows. First a sequential schedule S of length $|M|$ is constructed by concatenating the execution intervals of all jobs in $\mathcal{J}'_t$. Again, each job in $\mathcal{J}'_t$ is executed at a speed of s_Δ and therefore has an execution interval of length δ_i/s_Δ. Then S is split among the processors of M by assigning time range $[(\mu - 1), \mu)$ of S from left to right to the μ-th processor of M, $1 \le \mu \le |M|$. A summary of *AVR*(m) is given in Fig. 3.

It is easy to see that *AVR*(m) computes feasible schedules. In any interval I_t where a job J_i is active, δ_i processing units of J_i are finished. Summing over all intervals where J_i is active, we obtain that exactly $\delta_i(d_i - r_i) = w_i$ processing units, i.e. the total work of J_i, are completed. Furthermore, at no point in time is a job scheduled simultaneously on two different processors: In any interval I_t, a job $J_i \in \mathcal{J}_t$ is either scheduled alone on just one processor or it is scheduled at a speed of s_Δ. In the second case, since J_i has a density of $\delta_i \le s_\Delta$ its execution interval in S has a length of at most one time unit. Therefore, even if J_i is split among two processors μ and $\mu + 1$, $1 \le \mu < |M|$, it is processed at the end of processor μ and at the beginning of $\mu + 1$ so that these two execution intervals do not overlap (recall that $|I_t| = 1$).

Theorem 3. *AVR*(m) *achieves a competitive ratio of* $(2\alpha)^\alpha/2 + 1$.

Proof. Let $\sigma = J_1, \ldots, J_n$ be an arbitrary job sequence. The energy incurred by *AVR*(m) is

$$E_{AVR(m)}(\sigma) = \sum_{t=0}^{T-1} \sum_{l=1}^{m} (s_{t,l})^\alpha |I_t|.$$

The length $|I_t|$ of each interval I_t is equal to 1 time unit, but we need to incorporate $|I_t|$ in the above expression in order to properly evaluate the consumed energy.

For each I_t, we partition the m processors into two sets depending on whether or not the actual speeds are higher than Δ_t/m. More precisely, let $M_{t,1}$ be the set of all l, $1 \le l \le m$, such that the speed of processor l is $s_{t,l} \le \frac{\Delta_t}{m}$. Let $M_{t,2} = \{1, \ldots, m\} \setminus M_{t,1}$. We have

$$E_{AVR(m)}(\sigma) \le \sum_{t=0}^{T-1} \sum_{l \in M_{t,1}} \left(\frac{\Delta_t}{m}\right)^{\alpha} |I_t| + \sum_{t=0}^{T-1} \sum_{l \in M_{t,2}} (s_{t,l})^{\alpha} |I_t|$$
$$\le \sum_{t=0}^{T-1} m^{1-\alpha} (\Delta_t)^{\alpha} |I_t| + \sum_{t=0}^{T-1} \sum_{l \in M_{t,2}} (s_{t,l})^{\alpha} |I_t|.$$

The last inequality holds because $|M_{t,1}| \le m$. Next, for any fixed I_t, consider the processors of $M_{t,2}$. By the definition of $AVR(m)$ any such processor is assigned exactly one job. Hence, for any processor l with $l \in M_{t,2}$, the speed $s_{t,l}$ is the density of the job assigned to it. Hence $\sum_{l \in M_{t,2}} (s_{t,l})^{\alpha} \le \sum_{J_i \in \mathcal{J}_t} (\delta_i)^{\alpha}$ and we obtain

$$E_{AVR(m)}(\sigma) \le m^{1-\alpha} \sum_{t=0}^{T-1} (\Delta_t)^{\alpha} |I_t| + \sum_{t=0}^{T-1} \sum_{J_i \in \mathcal{J}_t} (\delta_i)^{\alpha} |I_t|. \tag{9}$$

In the above expression $\sum_{t=0}^{T-1} \sum_{J_i \in \mathcal{J}_t} (\delta_i)^{\alpha} |I_t|$ each job contributes an energy of $(\delta_i)^{\alpha}(d_i - r_i)$ because $J_i \in \mathcal{J}_t$ for any $t \in [r_i, d_i)$. This amount is the minimum energy required to process J_i if no other jobs were present. Hence $\sum_{t=0}^{T-1} \sum_{J_i \in \mathcal{J}_t} (\delta_i)^{\alpha} |I_t| \le E_{OPT}(\sigma)$, where $E_{OPT}(\sigma)$ is the energy consumption of an optimal schedule for σ on the m processors.

As for the first term in (9), $\sum_{t=0}^{T-1} (\Delta_t)^{\alpha} |I_t|$ is the energy consumed by the single processor algorithm *AVR* when executing σ on one processor. Recall that *AVR* sets the speed at time t to the accumulated density of jobs active at time t. Since *AVR* achieves a competitive ratio of $(2\alpha)^{\alpha}/2$, see [15], we obtain $\sum_{t=0}^{T-1} (\Delta_t)^{\alpha} |I_t| \le \frac{1}{2}(2\alpha)^{\alpha} E^1_{OPT}(\sigma)$, where $E^1_{OPT}(\sigma)$ is the energy of an optimal single processor schedule for σ. Thus

$$E_{AVR(m)}(\sigma) \le m^{1-\alpha} \tfrac{1}{2}(2\alpha)^{\alpha} E^1_{OPT}(\sigma) + E_{OPT}(\sigma).$$

To establish the desired competitive ratio, we finally prove $m^{1-\alpha} E^1_{OPT}(\sigma) \le E_{OPT}(\sigma)$.

Consider an optimal schedule S_{OPT} for σ on m processors and let $s^o_{t,l}$ be the speed on processor l in I_t, where $1 \le l \le m$ and $0 \le t < T - 1$. Let $s^o_t = \sum_{l=1}^{m} s^o_{t,l}$ be the sum of the speeds in I_t. By the convexity of the power function

$$E_{OPT}(\sigma) = \sum_{t=0}^{T-1} \sum_{l=1}^{m} (s^o_{t,l})^{\alpha} |I_t| \ge \sum_{t=0}^{T-1} m (s^o_t/m)^{\alpha} |I_t| = m^{1-\alpha} \sum_{t=0}^{T-1} (s^o_t)^{\alpha} |I_t|. \tag{10}$$

Finally, consider the single processor schedule S^1 that, in any interval I_t, uses speed s^o_t and accomplishes the same work as S_{OPT} in I_t. More precisely, if S_{OPT} finishes $w_{t,i}$ processing units of J_i in I_t, where $0 \le w_{t,i} \le w_i$, then S^1 processes the same amount of J_i in I_t. Using speed s^o_t, this is indeed possible. Hence S^1 is a feasible schedule whose energy consumption is $\sum_{t=0}^{T-1} (s^o_t)^{\alpha} |I_t|$, and this expression is at least $E^1_{OPT}(\sigma)$. Combining with (10) we obtain $E_{OPT}(\sigma) \ge m^{1-\alpha} \sum_{t=0}^{T-1} (s^o_t)^{\alpha} |I_t| \ge m^{1-\alpha} E^1_{OPT}(\sigma)$. □

4. Conclusion and open problems

In this paper we have investigated speed scaling in multi-processor environments. We considered a classical scheduling problem where jobs have associated deadlines and assumed that job migration is allowed. We have developed a combinatorial algorithm for computing optimal offline schedules and extended the single processor algorithms *Optimal Available* and *Average Rate*. Bansal et al. [5] gave an online algorithm for a single processor that, for large α, achieves a smaller competitiveness than *Optimal Available*. An open problem is if this strategy can also be extended to multi-processor systems.

Another working direction is to devise and analyze online algorithms for general convex power functions. Even for a single processor, no competitive strategy is known. Moreover, Irani et al. [9] investigated a problem setting where a single variable-speed processor is equipped with an additional sleep state into which the processor can be transitioned when idle. The framework is interesting because modern microprocessors, even at speed zero, consume a significant amount of static energy. It would be worthwhile to investigate combined speed scaling and power-down mechanisms in multi-processor environments.

References

[1] S. Albers, F. Müller, S. Schmelzer, Speed scaling on parallel processors, Algorithmica 68 (2) (2014) 404–425.
[2] N. Bansal, D.P. Bunde, H.-L. Chan, K. Pruhs, Average rate speed scaling, Algorithmica 60 (4) (2011) 877–889.
[3] N. Bansal, H.-L. Chan, T.-W. Lam, L.-K. Lee, Scheduling for speed bounded processors, in: Proc. 35th International Colloqium on Automata, Languages and Programming, ICALP, in: LNCS, vol. 5125, Springer, 2008, pp. 409–420.
[4] N. Bansal, H.-L. Chan, K. Pruhs, D. Katz, Improved bounds for speed scaling in devices obeying the cube-root rule, Theory Comput. 8 (1) (2012) 209–229.
[5] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, J. ACM 54 (2007).
[6] B.D. Bingham, M.R. Greenstreet, Energy optimal scheduling on multiprocessors with migration, in: Proc. IEEE International Symp. on Parallel and Distributed Processing with Applications, 2008, pp. 143–152.
[7] H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, P.W.H. Wong, Optimizing throughput and energy in online deadline scheduling, ACM Trans. Algorithms 6 (1) (2009).
[8] G. Greiner, T. Nonner, A. Souza, The bell is ringing in speed-scaled multiprocessor scheduling, Theory Comput. Syst. 54 (1) (2014) 24–44.
[9] S. Irani, S.K. Shukla, R. Gupta, Algorithms for power savings, ACM Trans. Algorithms 3 (2007).
[10] T.-W. Lam, L.-K. Lee, I.K.-K. To, P.W.H. Wong, Energy efficient deadline scheduling in two processor systems, in: Proc. 18th International Symposium on Algorithms and Computation, in: LNCS, vol. 4835, Springer, 2007, pp. 476–487.
[11] M. Li, B.J. Liu, F.F. Yao, Min-energy voltage allocation for tree-structured tasks, J. Comb. Optim. 11 (2006) 305–319.
[12] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, Proc. Natl. Acad. Sci. USA 103 (2006) 3983–3987.
[13] M. Li, F.F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM J. Comput. 35 (2005) 658–671.
[14] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Commun. ACM 28 (1985) 202–208.
[15] F.F. Yao, A.J. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proc. 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 374–382.