Discrete Applied Mathematics 181 (2015) 11–20

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

From preemptive to non-preemptive speed-scaling scheduling☆

Evripidis Bampis [a], Alexander Kononov [b], Dimitrios Letsios [a,c], Giorgio Lucarelli [a,c,*], Ioannis Nemparis [a,d]

[a] LIP6, Université Pierre et Marie Curie, France
[b] Sobolev Institute of Mathematics, Novosibirsk, Russia
[c] IBISC, Université d'Évry, France
[d] Department of Informatics and Telecommunications, NKUA, Athens, Greece

ARTICLE INFO

Article history:
Received 9 November 2013
Received in revised form 23 September 2014
Accepted 9 October 2014
Available online 28 October 2014

Keywords:
Approximation algorithms
Speed-scaling
Non-preemptive scheduling

ABSTRACT

We are given a set of jobs, each one specified by its release date, its deadline and its processing volume (work), and a single (or a set of) speed-scalable processor(s). We adopt the standard model in speed-scaling in which if a processor runs at speed s then the energy consumption is s^α units of energy per time unit, where $\alpha > 1$ is a small constant. Our goal is to find a schedule respecting the release dates and the deadlines of the jobs so that the total energy consumption to be minimized. While most previous works have studied the preemptive case of the problem, where a job may be interrupted and resumed later, we focus on the non-preemptive case where once a job starts its execution, it has to continue until its completion without any interruption. As the preemptive case is known to be polynomially solvable for both the single-processor and the multiprocessor case, we explore the idea of transforming an optimal preemptive schedule to a non-preemptive one. We prove that the preemptive optimal solution does not preserve enough of the structure of the non-preemptive optimal solution, and more precisely that the ratio between the energy consumption of an optimal non-preemptive schedule and the energy consumption of an optimal preemptive schedule can be very large even for the single-processor case. Then, we focus on some interesting families of instances: (i) equal-work jobs on a single-processor, and (ii) agreeable instances in the multiprocessor case. In both cases, we propose constant factor approximation algorithms. In the latter case, our algorithm improves the best known algorithm of the literature. Finally, we propose a (non-constant factor) approximation algorithm for general instances in the multiprocessor case.

1. Introduction

One of the main mechanisms used for minimizing the energy consumption in computing systems and portable devices is the so called *speed-scaling mechanism* [1], where the speed of a processor may change dynamically. If the speed of the

☆ A preliminary version of this work has been published in the Proceedings of 19th International Computing and Combinatorics Conference (COCOON 2013).

* Corresponding author at: LIP6, Université Pierre et Marie Curie, France.
E-mail addresses: Evripidis.Bampis@lip6.fr (E. Bampis), alvenko@math.nsc.ru (A. Kononov), dimitris.letsios@ibisc.univ-evry.fr (D. Letsios), Giorgio.Lucarelli@lip6.fr (G. Lucarelli), sdi0700181@di.uoa.gr (I. Nemparis).

http://dx.doi.org/10.1016/j.dam.2014.10.007
0166-218X/

processor is $s(t)$ at a time t then its power is $s(t)^\alpha$, where $\alpha > 1$ is a small machine dependent constant. The energy consumption is the power integrated over time. In this setting, we consider the *non-preemptive speed scaling* scheduling problem: we are given a set $\mathcal{J}$ of n jobs, where each job $J_j \in \mathcal{J}$ is characterized by its processing volume (work) w_j, its release date r_j and its deadline d_j, and a single (or a set of identical) speed-scalable processor(s). We call a schedule for the jobs in $\mathcal{J}$ *feasible* if every job is executed between its release date and its deadline. Moreover, a schedule is called *non-preemptive*, if every job is executed without interruption, i.e. once a job starts its execution it has to continue until its completion. We seek for a feasible non-preemptive schedule of the jobs minimizing the overall energy consumption.

Recently, it has been proved that the non-preemptive speed scaling scheduling problem is $\mathcal{NP}$-hard even for the single-processor case [6]. Using the standard three-field notation we denote the single-processor (resp. multiprocessor) case as $1|r_j, d_j|E$ (resp. $P|r_j, d_j|E$). Our aim is to study the performance of one of the most standard approaches in scheduling for designing approximation algorithms: construct a non-preemptive schedule from an optimal preemptive one (see for example [18]). We prove that for general instances this approach cannot lead to a constant factor approximation algorithm since the ratio between the energy consumption of an optimal non-preemptive schedule and the energy consumption of an optimal preemptive one can be very large even for the single-processor case. Despite this negative result, we show that for some important families of instances, this approach leads to interesting results. Moreover, we show how to use this approach in order to obtain an (non-constant) approximation algorithm for the general case.

1.1. Related work

Different variants of the problem have been considered in the literature: with or without preemptions, with equal or arbitrary works, arbitrary release dates and deadlines or particular instances. The main families of instances, with respect to the release dates and the deadlines of the jobs, that have been studied are the following. In a *laminar instance*, for any two jobs J_j and $J_{j'}$ with $r_j \leq r_{j'}$ it holds that either $d_j \geq d_{j'}$ or $d_j \leq r_{j'}$. In fact, such instances arise when recursive calls in a program create new jobs. In an *agreeable instance*, for any two jobs J_j and $J_{j'}$ with $r_j \leq r_{j'}$ it holds that $d_j \leq d_{j'}$, i.e. latter released jobs have latter deadlines. Such instances may arise in situations where the goal is to maintain a fair service guarantee for the waiting time of jobs. Note that agreeable instances correspond to proper interval graphs. In a *pure-laminar instance*, for any two jobs J_j and $J_{j'}$ with $r_j \leq r_{j'}$ it holds that $d_j \geq d_{j'}$. Note that the family of pure-laminar instances is a special case of laminar instances. Finally, two other interesting special cases of all the above families of instances, studied by several works in scheduling, are those where all the jobs have either a common release date or a common deadline.

For the preemptive single-processor case ($1|r_j, d_j, pmtn|E$), Yao et al. [20] proposed an optimal algorithm for finding a feasible schedule with minimum energy consumption. The multiprocessor case, $P|r_j, d_j, pmtn|E$, where there are m available processors has been solved optimally in polynomial time when both the preemption and the migration of jobs are allowed [2,5,8,10]. A schedule is called migratory if a job may be interrupted and resumed on the same or on another processor. Note that the parallel execution of parts of the same job is not allowed.

Albers et al. [3] considered the multiprocessor problem where the preemption of the jobs is allowed but not their migration ($P|r_j, d_j, pmtn, no\text{-}mig|E$). They first studied the problem where each job has unit work. They proved that it is polynomial time solvable for instances with agreeable deadlines. For general instances with unit-work jobs, they proved that the problem becomes strongly $\mathcal{NP}$-hard and they proposed an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm. For the case where the jobs have arbitrary works, the problem was proved to be $\mathcal{NP}$-hard even for instances with common release dates and common deadlines. Albers et al. proposed a $2(2 - \frac{1}{m})^\alpha$-approximation algorithm for instances with common release dates, or common deadlines, and an $(\alpha^\alpha 2^{4\alpha})$-approximation algorithm for instances with agreeable deadlines. Greiner et al. [14] gave a generic reduction transforming an optimal schedule for the multiprocessor problem with migration, $P|r_j, d_j, pmtn|E$, to a $B_{\lceil\alpha\rceil}$-approximate solution for the multiprocessor problem with preemptions but without migration, $P|r_j, d_j, pmtn, no\text{-}mig|E$, where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number. This result holds only when $\alpha \leq m$.

Note that for the family of agreeable instances, and hence for its special families of instances (instances with common release dates and/or common deadlines), the assumption of preemption and no migration is equivalent to the non-preemptive assumption that we consider throughout this paper. In fact, any preemptive schedule for agreeable instances can be transformed into a non-preemptive one of the same energy consumption, where the execution of each job $J_j \in \mathcal{J}$ starts after the completion of any other job which is released before J_j. The correctness of this transformation can be proved by induction on the order where the jobs are released. Hence, the results of [3,14] for agreeable deadlines hold for the non-preemptive case as well.

Finally, Antoniadis and Huang [6] proved that the problem is $\mathcal{NP}$-hard even for pure-laminar instances. They also presented a $2^{4\alpha-3}$-approximation algorithm for laminar instances and a $2^{5\alpha-4}$-approximation algorithm for general instances. Bampis et al. [7] have recently improved these results for small and moderate values of α by providing a $2^{\alpha-1}\tilde{B}_\alpha$-approximation algorithm, where $\tilde{B}_\alpha$ is the generalized Bell number and corresponds to the α-th (fractional) moment of Poisson random variable with parameter 1. Notice that the polynomial-time algorithm for finding an optimal preemptive schedule presented in [20] for the single-processor case returns a non-preemptive schedule when the input instance is agreeable.

In Table 1, we summarize the most related results of the literature. Several other results concerning scheduling problems in the speed-scaling setting have been presented, involving the optimization of some Quality of Service (QoS) criterion under

Table 1
Complexity and approximability results.

Problem	Complexity	Approximation ratio
$1\|r_j, d_j, pmtn\|E$	Polynomial [20]	–
$P\|r_j = 0, d_j = d, pmtn\|E$	Polynomial [13]	–
$P\|r_j, d_j, pmtn\|E$	Polynomial [2,5,8,10]	–
$P\|$agreeable, $w_j = 1$, $pmtn$, $no\text{-}mig\|E^*$	Polynomial [3]	–
$P\|r_j, d_j, w_j = 1, pmtn, no\text{-}mig\|E$	$\mathcal{NP}$-hard ($m \geq 2$) [3]	$B_{\lceil\alpha\rceil}$ [14]
$P\|r_j = 0, d_j = d, pmtn, no\text{-}mig\|E^*$	$\mathcal{NP}$-hard [3]	*PTAS* [15,3]
$P\|r_j = 0, d_j, pmtn, no\text{-}mig\|E^*$	$\mathcal{NP}$-hard	$\min\{2(2-\frac{1}{m})^\alpha, B_{\lceil\alpha\rceil}\}$ [3,14]
$P\|$agreeable, $pmtn$, $no\text{-}mig\|E^*$	$\mathcal{NP}$-hard	$B_{\lceil\alpha\rceil}$ [14]
$P\|r_j, d_j, pmtn, no\text{-}mig\|E$	$\mathcal{NP}$-hard	$B_{\lceil\alpha\rceil}$ [14]
$1\|r_j, d_j, w_j = 1\|E$	Polynomial [4,16]	–
$1\|$agreeable$\|E$	Polynomial [20,6]	–
$1\|$laminar$\|E$	$\mathcal{NP}$-hard [6]	$\min\{2^{4\alpha-3}, 2^{\alpha-1}\tilde{B}_\alpha\}$ [6,7]
$1\|r_j, d_j\|E$	$\mathcal{NP}$-hard	$\min\{2^{5\alpha-4}, 2^{\alpha-1}\tilde{B}_\alpha\}$ [6,7]

* The problem is equivalent with the corresponding non-preemptive problem.

a budget of energy, or the optimization of a linear combination of the energy consumption and some QoS criterion (see for example [9,11,19]). Moreover, two variants of the speed-scaling model considered in this paper have been studied in the literature, namely the *bounded speed model* in which the speeds of the processors are bounded above and below (see for example [12]), and the *discrete speed model* in which the speeds of the processors can be selected among a set of discrete speeds (see for example [17]). The interested reader can find more details in the recent survey [1].

1.2. Our contribution

In this paper, we are interested in designing approximation algorithms for the non-preemptive speed-scaling scheduling problem using a standard approach in scheduling: given an optimal preemptive solution, design an algorithm to convert it into a feasible non-preemptive solution with as small degradation as possible in the approximation ratio. For the single-processor case, we use the optimal preemptive solution obtained by the algorithm of Yao et al. [20], while for the multi-processor case, we use the preemptive migratory solution obtained by [2,5,8,10]. Unfortunately, the following proposition shows that for general instances the ratio between an optimal non-preemptive schedule to an optimal preemptive one can be very large even for the single-processor case.

Proposition 1. *The ratio of the energy consumption of an optimal non-preemptive schedule to the energy consumption of an optimal preemptive schedule of the single-processor speed-scaling problem can be* $\Omega(n^{\alpha-1})$.

Proof. Consider the instance consisting of a single processor, $n-1$ unit-work jobs, $J_1, J_2, \ldots, J_{n-1}$, and the job J_n of work n. Each job J_j, $1 \leq j \leq n-1$, has release date $r_j = 2j-1$ and deadline $d_j = 2j$, while $r_n = 0$ and $d_n = 2n-1$ (see Fig. 1).

The optimal preemptive schedule $\mathcal{S}_{pr}$ for this instance assigns to all jobs a speed equal to one. Each job J_j, $1 \leq j \leq n-1$, is executed during its whole active interval, while J_n is executed during the remaining n unit length intervals. The total energy consumption of this schedule is $E(\mathcal{S}_{pr}) = (n-1)\cdot 1^\alpha + n \cdot 1^\alpha = 2n-1$.

An optimal non-preemptive schedule $\mathcal{S}_{npr}$ for this instance assigns a speed $\frac{n+2}{3}$ to jobs J_1, J_n and J_2 and schedules them non-preemptively in this order between time 1 and 4. Moreover, in $\mathcal{S}_{npr}$ each job J_j, $3 \leq j \leq n-1$, is assigned a speed equal to one and it is executed during its whole active interval. The total energy consumption of this schedule is $E(\mathcal{S}_{npr}) = 3\cdot(\frac{n+2}{3})^\alpha + (n-3)\cdot 1^\alpha$.

Therefore, we have $\frac{E(\mathcal{S}_{npr})}{E(\mathcal{S}_{pr})} = \frac{3\cdot(\frac{n+2}{3})^\alpha + (n-3)\cdot 1^\alpha}{2n-1} = \Omega(n^{\alpha-1})$. □

In what follows we show that for some particular instances, this approach leads to interesting results. More specifically, in Section 3 we consider the single-processor case and we present an algorithm whose approximation ratio depends on the ratio of the maximum to the minimum work in the input instance. For equal-work jobs, our algorithm achieves an approximation ratio of 2^α. It has to be noticed here that after the conference version of this paper the complexity status of this special case has been settled. Angel et al. [4] and Huang and Ott [16] independently proposed an optimal polynomial-time algorithm based on dynamic programming. However, the time complexity of these algorithms compared to the complexity of our approximation algorithm is much higher.

In Section 4 we consider the multiprocessor case. First, in Section 4.1, we deal with agreeable instances for which we present a $(2-\frac{1}{m})^{\alpha-1}$-approximation algorithm. This ratio improves the best known approximation ratio of $B_{\lceil\alpha\rceil}$ given in [14] for any $\alpha > 1$. For $\alpha = 3$ our algorithm achieves a ratio of 4 while $B_3 = 5$, for $\alpha = 4$ our algorithm achieves a ratio of 8 while $B_4 = 15$, etc. Note that in general $B_{\lceil\alpha\rceil} = O(\alpha^\alpha)$. Finally, in Section 4.2 we present an approximation algorithm of ratio $m^\alpha(\sqrt[m]{n})^{\alpha-1}$ for general instances.

Before beginning, in the following section we present our notation and some preliminary known results that we use in our proofs.

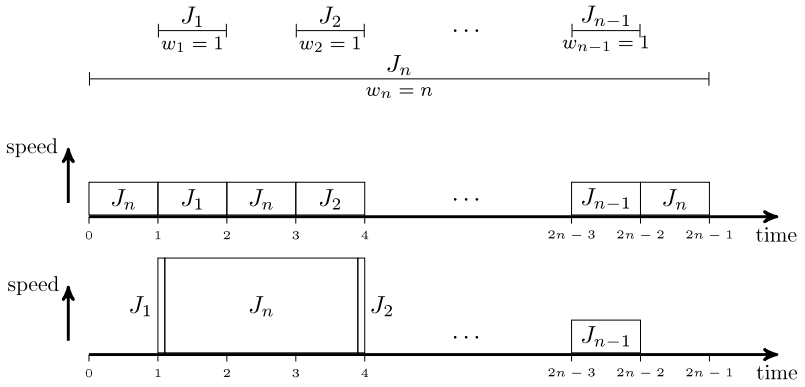

Fig. 1. An instance of $1|r_j, d_j|E$ for which the ratio of the energy consumption in an optimal non-preemptive schedule to the energy consumption in an optimal preemptive schedule is $\Omega(n^{\alpha-1})$.

2. Notation and preliminaries

For the problems that we study in this paper, it is easy to see that in any optimal schedule, any job $J_j \in \mathcal{J}$ runs at a constant speed s_j due to the convexity of the speed-to-power function. Given a schedule $\mathcal{S}$ and a job $J_j \in \mathcal{J}$, we denote by $E(\mathcal{S}, J_j) = w_j s_j^{\alpha-1}$ the energy consumed by the execution of J_j in $\mathcal{S}$ and by $E(\mathcal{S}) = \sum_{j=1}^{n} E(\mathcal{S}, J_j)$ the total energy consumed by $\mathcal{S}$. We denote by $\mathcal{S}^*$ an optimal non-preemptive schedule for the input instance $\mathcal{I}$.

The following proposition has been proved in [6] for $1|r_j, d_j|E$ but holds also for the corresponding problem on parallel processors.

Proposition 2 ([6]). *Suppose that the schedules $\mathcal{S}$ and $\mathcal{S}'$ process job J_j with speed s and s' respectively. Assume that $s \leq \gamma s'$ for some $\gamma \geq 1$. Then $E(\mathcal{S}, J_j) \leq \gamma^{\alpha-1} E(\mathcal{S}', J_j)$.*

The following proposition has been proved in [3] and gives the relation between the energy consumption of an optimal single-processor preemptive schedule and an optimal multiprocessor preemptive schedule without migrations.

Proposition 3 ([3]). *For a given set of jobs $\mathcal{J}$, let $\mathcal{S}$ be an optimal single-processor preemptive schedule and $\mathcal{S}'$ be an optimal multiprocessor preemptive schedule without migrations. Then $E(\mathcal{S}) \leq m^{\alpha-1} \cdot E(\mathcal{S}')$.*

3. Single-processor

In this section we consider the single-processor non-preemptive speed-scaling problem. We first prove some structural properties of the optimal preemptive schedule created by the algorithm in [20]. Then, we present an approximation algorithm for the non-preemptive case, using as lower bound the energy consumed by the optimal preemptive schedule. Our algorithm achieves a constant factor approximation ratio for equal-work jobs.

3.1. Properties of the optimal preemptive schedule

Let $\{t_0, t_1, \ldots, t_k\}$ be the set of all different release dates and deadlines in increasing order, i.e., $t_0 \leq t_1 \leq \cdots \leq t_k$. Then, we define the intervals $I_{p,q} = [t_p, t_q]$, for $0 \leq p < q \leq k$, and we denote by $|I_{p,q}| = t_q - t_p$ the length of $I_{p,q}$. We say that a job J_j is alive in a given interval $I_{p,q}$, if $[r_j, d_j] \subseteq I_{p,q}$. The set of alive jobs in interval $I_{p,q}$ is denoted by $A(I_{p,q})$. The density $d(I_{p,q})$ of an interval $I_{p,q}$ is the total work of its alive jobs over its length, i.e., $d(I_{p,q}) = \frac{\sum_{J_j \in A(I_{p,q})} w_j}{|I_{p,q}|}$.

In [20], Yao et al. proposed a polynomial-time algorithm for finding an optimal schedule for $1|r_j, d_j, pmtn|E$. This algorithm schedules the jobs in distinct phases. More specifically, in each phase, the algorithm searches for the interval of the highest density, denoted as $I_{p,q}$. All jobs in $A(I_{p,q})$ are assigned the same speed, which is equal to the density $d(I_{p,q})$, and they are scheduled in $I_{p,q}$ using the Earliest Deadline First (EDF) policy. We can assume, w.l.o.g., that in the case where two jobs have the same deadline, the algorithm schedules first the job of the smallest index. Then, the set of jobs $A(I_{p,q})$ and the interval $I_{p,q}$ are eliminated from the instance and the algorithm searches for the next interval of the highest density, and so on.

Given a preemptive schedule $\mathcal{S}$ and a job J_j, let $b_j(\mathcal{S})$ and $c_j(\mathcal{S})$ be the starting and the completion time, respectively, of J_j in $\mathcal{S}$. For simplicity, we will use b_j and c_j, if the corresponding schedule is clear from the context. Note that there are no jobs with the same starting times, and hence all b_j's are distinct. For the same reason, all c_j's are distinct.

The following lemma describes some structural properties of the optimal schedule created by the algorithm in [20].

Lemma 1. *Consider the optimal preemptive schedule $\mathcal{S}_{pr}$ created by the algorithm in* [20]. *For any two jobs J_j and $J_{j'}$ in $\mathcal{S}_{pr}$, it holds that:*

(i) *if $b_j < b_{j'}$ then either $c_j > c_{j'}$ or $c_j \leq b_{j'}$, and*
(ii) *if $b_j < b_{j'}$ and $c_j > c_{j'}$ then $s_j \leq s_{j'}$.*

Proof. (i) Assume for contradiction that there are two jobs J_j and $J_{j'}$ in $\mathcal{S}_{pr}$ with $b_j < b_{j'}$, $c_j < c_{j'}$ and $c_j > b_{j'}$.

We prove, first, that J_j and $J_{j'}$ cannot be scheduled in a different phase of the algorithm. W.l.o.g., assume for contradiction that J_j is scheduled in a phase before $J_{j'}$ and that $I_{p,q}$ is the interval of the highest density in this phase. As $b_j < b_{j'} < c_j$, there is a subinterval $I \subseteq [b_{j'}, c_j] \subset [b_j, c_j] \subseteq I_{p,q}$ during which $J_{j'}$ is executed in $\mathcal{S}_{pr}$. By construction, each job is scheduled in a single phase and since $I \subset I_{p,q}$, it holds that $[b_{j'}, c_{j'}] \subset I_{p,q}$. Hence, J_j and $J_{j'}$ are scheduled in the same phase.

The algorithm schedules J_j and $J_{j'}$ using the EDF policy. Since the EDF policy schedules $J_{j'}$ at time $b_{j'}$ and $b_{j'} < c_j$, it holds that $d_{j'} \leq d_j$. In a similar way, since the EDF policy schedules J_j at time c_j and $c_j < c_{j'}$, it holds that $d_j \leq d_{j'}$. Hence, $d_j = d_{j'}$. However, since there is a tie, in both times $b_{j'}$ and c_j the algorithm should have selected the same job, i.e., the job of the smallest index. Therefore, there is a contradiction on the way that the algorithm works.

(ii) Assume for contradiction that there are two jobs J_j and $J_{j'}$ in $\mathcal{S}_{pr}$ with $b_j < b_{j'}$, $c_j > c_{j'}$ and $s_j > s_{j'}$. As $s_j > s_{j'}$, J_j is scheduled in a phase before $J_{j'}$; let $I_{p,q}$ be the interval of the highest density in this phase. However, it holds that $[b_{j'}, c_{j'}] \subset [b_j, c_j] \subseteq I_{p,q}$, and hence $J_{j'}$ should have been scheduled in the same phase as J_j, which is a contradiction. □

The above lemma implies that given an optimal preemptive schedule $\mathcal{S}_{pr}$ obtained by the algorithm in [20], the interval graph, in which for each job J_j there is an interval $[b_j, c_j]$, has a laminar structure. Therefore, we can create a tree-representation of $\mathcal{S}_{pr}$ as follows. For each job J_j we create a vertex. For each pair of jobs J_j and $J_{j'}$ with $[b_{j'}, c_{j'}] \subset [b_j, c_j]$, we create an arc $(J_j, J_{j'})$ if and only if there is not a job $J_{j''}$ with $[b_{j'}, c_{j'}] \subset [b_{j''}, c_{j''}] \subset [b_j, c_j]$. Note that, the created graph $T = (V, E)$ is, in general, a forest. Moreover, using Lemma 1 we have that for each arc $(J_j, J_{j'})$ it holds that $s_j \leq s_{j'}$ in $\mathcal{S}_{pr}$. In other words, the speed of a job is at most equal to the speed of its children in T.

In what follows, we denote by $T(J_j)$ the subtree of T rooted at vertex $J_j \in V$. Moreover, let n_j be the number of children of J_j in T.

Lemma 2. *Consider an optimal preemptive schedule $\mathcal{S}_{pr}$ created by the algorithm in* [20] *and its corresponding graph $T = (V, E)$. Each job J_j is preempted at most n_j times in $\mathcal{S}_{pr}$.*

Proof. We will prove the lemma by induction on the tree.

Assume for contradiction that the root job J_r is preempted more than n_r times in $\mathcal{S}_{pr}$, that is the execution of J_r is partitioned into more than $n_r + 1$ different maximal intervals. Thus, there is a child J_j of J_r and an interval $I \subset [b_j, c_j]$ such that J_r is executed during I. Observe first that J_r and J_j should be scheduled in the same phase by the algorithm in [20]. Hence, the EDF policy is used and using similar arguments as in the proof of Lemma 1(i) we have a contradiction.

For the induction step, assume for contradiction that the job J_j is preempted more than n_j times in $\mathcal{S}_{pr}$. Hence, either there is a child $J_{j'}$ of J_j and an interval $I \subset [b_{j'}, c_{j'}]$ such that J_j is executed during I, or there are two consecutive children $J_{j'}$ and $J_{j''}$ of J_j and two disjoint maximal intervals I and I', with $I, I' \subset [c_{j'}, b_{j''}]$, such that J_j is executed during both I and I'. In the first case, we have a contradiction using similar arguments as for the base of the induction. In the second case, we get a contradiction using the inductive hypothesis. □

3.2. An approximation algorithm

In this section we present an approximation algorithm, whose ratio depends on $w_{\max}$ and $w_{\min}$. In the case where all jobs have equal work to execute, this algorithm achieves a 2^{α}-approximation ratio. The main idea in Algorithm 1 is to transform the optimal preemptive schedule $\mathcal{S}_{pr}$ created by the algorithm in [20] into a non-preemptive schedule $\mathcal{S}_{npr}$, based on the corresponding graph $T = (V, E)$ of $\mathcal{S}_{pr}$. More specifically, the jobs are scheduled in three phases depending on the number (one, at least two or zero) of their children in T.

Algorithm 1

1: Create an optimal preemptive schedule $\mathcal{S}_{pr}$ using the algorithm in [20];
2: Create the corresponding graph $T = (V, E)$ of $\mathcal{S}_{pr}$;
3: Create the non-preemptive schedule $\mathcal{S}_{npr}$ as follows:
4: **for** each job J_j with $n_j = 1$ **do**
5: Schedule non-preemptively the whole work of J_j in the biggest interval where a part of J_j is executed in S_{pr};
6: **for** each remaining non-leaf job J_j **do**
7: Find an unlabeled leaf job $J_{j'} \in T(J_j)$; Label $J_{j'}$;
8: Schedule non-preemptively J_j and $J_{j'}$ with the same speed in the interval where $J_{j'}$ is executed in $\mathcal{S}_{pr}$;
9: Schedule the remaining leaf jobs as in $\mathcal{S}_{pr}$;
10: **return** $\mathcal{S}_{npr}$;

Theorem 1. *Algorithm* 1 *achieves an approximation ratio of* $(1 + \frac{w_{\max}}{w_{\min}})^{\alpha}$ *for* $1|r_j, d_j|E$.

Proof. Consider first the jobs with exactly one child in T. By Lemma 2, each such job J_j is preempted at most once in $\mathcal{S}_{pr}$, and hence it is executed in at most two intervals in $\mathcal{S}_{pr}$. In $\mathcal{S}_{npr}$ the whole work of J_j is scheduled in the largest of these two intervals. Thus, the speed of J_j in $\mathcal{S}_{npr}$ is at most twice the speed of J_j in $\mathcal{S}_{pr}$. Therefore, using Proposition 2, for any job J_j with $n_j = 1$ it holds that $E(\mathcal{S}_{npr}, J_j) \leq 2^{\alpha-1} \cdot E(\mathcal{S}_{pr}, J_j)$.

Consider now the remaining non-leaf jobs. As for each such job J_j it holds that $n_j \geq 2$, in the subtree $T(J_j)$ the number of non-leaf jobs with $n_j \geq 2$ is smaller than the number of leaf jobs. Hence, we can create an one-to-one assignment of the non-leaf jobs with $n_j \geq 2$ to leaf jobs such that each non-leaf job J_j is assigned to an unlabeled leaf job $J_{j'} \in T(J_j)$.

Consider a non-leaf job J_j with $n_j \geq 2$ and its assigned leaf job $J_{j'} \in T(J_j)$. Recall that leaf jobs are executed non-preemptively in $\mathcal{S}_{pr}$. Let I be the interval in which $J_{j'}$ is executed in $\mathcal{S}_{pr}$. The speed of $J_{j'}$ in $\mathcal{S}_{pr}$ is $s_{j'} = \frac{w_{j'}}{|I|}$ and its energy consumption is $E(\mathcal{S}_{pr}, J_{j'}) = w_{j'} s_{j'}^{\alpha-1}$. In $\mathcal{S}_{npr}$ both J_j and $J_{j'}$ are executed during I with speed $s = \frac{w_j + w_{j'}}{|I|}$. The energy consumed by J_j and $J_{j'}$ in $\mathcal{S}_{npr}$ is

$$\begin{aligned} E(\mathcal{S}_{npr}, J_j) + E(\mathcal{S}_{npr}, J_{j'}) &= (w_j + w_{j'}) s^{\alpha-1} = (w_j + w_{j'}) \left(\frac{w_j + w_{j'}}{|I|} \right)^{\alpha-1} \\ &= (w_j + w_{j'})^{\alpha} \left(\frac{s_{j'}}{w_{j'}} \right)^{\alpha-1} = \left(\frac{w_j + w_{j'}}{w_{j'}} \right)^{\alpha} \cdot w_{j'} s_{j'}^{\alpha-1} \\ &= \left(\frac{w_j + w_{j'}}{w_{j'}} \right)^{\alpha} \cdot E(\mathcal{S}_{pr}, J_{j'}) \\ &< \left(\frac{w_{\max} + w_{\min}}{w_{\min}} \right)^{\alpha} \cdot (E(\mathcal{S}_{pr}, J_j) + E(\mathcal{S}_{pr}, J_{j'})). \end{aligned}$$

Moreover, note that J_j is alive during I and hence $\mathcal{S}_{npr}$ is a feasible schedule.

Finally, for each remaining leaf job J_j, it holds that $E(\mathcal{S}_{npr}, J_j) = E(\mathcal{S}_{pr}, J_j)$, concluding the proof of the theorem. □

When all jobs have equal work to execute, the following corollary holds.

Corollary 1. *Algorithm* 1 *achieves an approximation ratio of* 2^{α} *for* $1|w_j = w, r_j, d_j|E$.

4. Parallel processors

In this section, we show how to use the optimal preemptive schedule to achieve approximation algorithms for the multiprocessor case. We first present a constant factor approximation algorithm for instances with agreeable deadlines. Then, we consider general instances. As by Proposition 1 we know that the energy consumption of an optimal preemptive schedule can be $\Omega(n^{\alpha-1})$ far from the energy consumption of an optimal non-preemptive schedule, we give an algorithm for the latter case that uses as a lower bound the optimal preemptive schedule and achieves an approximation factor that depends on n and m.

4.1. Agreeable instances

The problem $P|\text{agreeable}|E$ is known to be $\mathcal{NP}$-hard [3] and $B_{\lceil\alpha\rceil}$-approximable [14]. In this section we present an approximation algorithm of ratio $(2 - \frac{1}{m})^{\alpha-1}$, which is better than $B_{\lceil\alpha\rceil}$ for any $\alpha > 1$.

Our algorithm creates first an optimal preemptive schedule, using one of the algorithms in [2,5,8,10]. The total execution time e_j of each job $J_j \in \mathcal{J}$ in this preemptive schedule is used to define an appropriate processing time p_j for J_j. Then, the algorithm schedules non-preemptively the jobs using these processing times according to the Earliest Deadline First policy, i.e., at every time that a processor becomes idle, the non-scheduled job with the minimum deadline is scheduled on it. The choice of the values of the p_j's has been made in such a way that the algorithm completes all the jobs before their deadlines.

Algorithm 2

1: Create an optimal multiprocessor preemptive schedule $\mathcal{S}_{pr}$;
2: Let e_j be the total execution time of the job $J_j \in \mathcal{J}$, in $\mathcal{S}_{pr}$;
3: Schedule non-preemptively the jobs with the Earliest Deadline First (EDF) policy, using the appropriate speed such that the processing time of the job $J_j \in \mathcal{J}$, is equal to $p_j = e_j/(2 - \frac{1}{m})$, obtaining the non-preemptive schedule $\mathcal{S}_{npr}$;
4: **return** $\mathcal{S}_{npr}$;

Theorem 2. *Algorithm* 2 *achieves an approximation ratio of* $(2 - \frac{1}{m})^{\alpha-1}$ *for* $P|\text{agreeable}|E$.

Proof. The principal difficulty of the current proof is to show that Algorithm 2 produces a feasible schedule, i.e., a schedule in which each job completes before its deadline. Towards this goal, a key ingredient is a technical claim whose proof involves a quite stiff case analysis. For ease of presentation, the claim's proof is provided exactly after the current proof. The key idea

behind this claim is the fact that, at each time, the remaining processing time of the jobs in the algorithm's schedule is not too much compared with the optimal preemptive schedule. For this reason, the algorithm is able to complete all the jobs before their deadlines. It has to be noticed that the aforementioned claim lies heavily on the fact the instances are agreeable.

We consider the jobs indexed in non-decreasing order of their release dates/deadlines. In what follows, we denote by b_j the starting time of the job $J_j \in \mathcal{J}$ in $\mathcal{S}_{npr}$. Hence, the completion time c_j of J_j in $\mathcal{S}_{npr}$ is $c_j = b_j + p_j$. We first show that the $\mathcal{S}_{npr}$ is a feasible schedule. In other words, we will prove that for the completion time of the job $J_j \in \mathcal{J}$, it holds that $c_j \leq d_j$. Before that we introduce some additional notation.

Note that at each time either all processors execute some job or there is at least one processor which is idle. Based on this observation, we partition $\mathcal{S}_{npr}$ into maximal intervals: the "full" and the "non-full" intervals. At each time during a "full" interval, every processor executes some job. At each time during a "non-full" interval, there is at least one processor which is idle. Let ℓ be the number of the "non-full" intervals. Let $[\tau_i, t_i]$, $1 \leq i \leq \ell$, be the ith "non-full" interval. Hence, $[t_{i-1}, \tau_i]$, $1 \leq i \leq \ell + 1$, is a "full" interval. For convenience, $t_0 = 0$ and $\tau_{\ell+1} = \max_{J_j \in \mathcal{J}}\{c_j\}$. Note that the schedule can start at a "non-full" interval, i.e., $t_0 = \tau_1$, or can end with a "non-full" interval, i.e., $t_\ell = \tau_{\ell+1}$.

Consider first a job $J_j \in \mathcal{J}$ that is released during a "non-full" interval $[\tau_i, t_i]$. Since the jobs are scheduled according to the EDF policy, J_j starts its execution at its release date, i.e., $b_j = r_j$. Given that J_j has smaller processing time in $\mathcal{S}_{npr}$ than in $\mathcal{S}_{pr}$ and as $\mathcal{S}_{pr}$ is a feasible schedule, it holds that $c_j \leq d_j$.

Consider now a job $J_j \in \mathcal{J}$ that is released during a "full" interval $[t_i, \tau_{i+1}]$. We denote by $\mathcal{J}_i = \{J_j \in \mathcal{J} : r_j < t_i\}$ the set of jobs which are released before t_i. Let $P_{npr,i}(t)$ be the amount of time that the jobs in $\mathcal{J}_i$ are executed after t for every $t \geq t_i$ in $\mathcal{S}_{npr}$ and $E_{pr,i}(t)$ be the amount of time that the jobs in $\mathcal{J}_i$ are executed after t for every $t \geq t_i$ in $\mathcal{S}_{pr}$. If $t = t_i$ we use $P_{npr,i}$ and $E_{pr,i}$, correspondingly. Given these definitions, we are now ready to state the claim needed for proving that J_j completes before its deadline, as we discussed in the beginning of the current proof. Recall that the claim's proof is provided exactly after the theorem's proof.

Claim 1. *For each i, $0 \leq i \leq \ell$, it holds that $P_{npr,i}(t) \leq \frac{E_{pr,i}(t)}{(2-\frac{1}{m})}$.*

Based on the above claim, we show that J_j is feasibly executed in the algorithm's schedule as follows. Let J_q be the first job which is released after t_i, i.e., $r_q = t_i$. For J_j we have

$$c_j \leq t_i + \frac{P_{npr,i} + \sum_{k=q}^{j-1} p_k}{m} + p_j \leq t_i + \frac{\frac{E_{pr,i} + \sum_{k=q}^{j-1} e_k}{m} + e_j}{\left(2 - \frac{1}{m}\right)}.$$

As $\mathcal{S}_{pr}$ is a feasible schedule and the instance is agreeable, all jobs $J_q, \ldots, J_j$ are executed inside the interval $[t_i, d_j]$ in $\mathcal{S}_{pr}$ and at least $E_{pr,i}$ amount of time of the jobs in $\mathcal{J}_i$ is also executed during the same time interval. Hence, it holds that $E_{pr,i} + \sum_{k=q}^{j} e_k \leq m(d_j - t_i)$ and $e_j \leq d_j - t_i$. Therefore, we obtain that $c_j \leq t_i + (2 - \frac{1}{m})\frac{d_j - t_i}{(2-\frac{1}{m})} = d_j$.

Finally, we have to prove the approximation ratio of our algorithm. When dividing the execution time of all jobs by $(2 - \frac{1}{m})$, at the same time the speed of each job is multiplied by the same factor. Using Proposition 2 we have that

$$E(\mathcal{S}_{npr}) \leq \left(2 - \frac{1}{m}\right)^{\alpha-1} E(\mathcal{S}_{pr}) \leq \left(2 - \frac{1}{m}\right)^{\alpha-1} E(\mathcal{S}^*)$$

since the energy consumed by the optimal preemptive schedule $\mathcal{S}_{pr}$ is a lower bound to the energy consumed by an optimal non-preemptive schedule $\mathcal{S}^*$ for the input instance $\mathcal{I}$. □

Next, we present the proof of Claim 1 which is needed for the proof of Theorem 2.

Proof of Claim 1. We prove the claim by induction to i.

For the induction basis, we have two cases. If $t_0 \neq \tau_1$, then $P_{npr,0} = E_{pr,0} = 0$. If $t_0 = \tau_1$, then the schedule begins with a "non-full" interval. Since the jobs are scheduled according to the EDF policy in $\mathcal{S}_{npr}$, every job $J_j \in \mathcal{J}_1$ starts at its release date, i.e., $b_j = r_j$. Given that the processing time of J_j in $\mathcal{S}_{npr}$ is $(2 - \frac{1}{m})$ times smaller than in $\mathcal{S}_{pr}$ and as $\mathcal{S}_{pr}$ is a feasible schedule, the claim holds.

Assume that the claim is true for i. We will show that $P_{npr,i+1}(t) \leq \frac{E_{pr,i+1}(t)}{(2-\frac{1}{m})}$. Recall that $P_{npr,i+1}(t)$ and $E_{pr,i+1}(t)$ are the amounts of time that the jobs in $\mathcal{J}_{i+1}$ are executed after t for every $t \geq t_{i+1}$ in $\mathcal{S}_{npr}$ and $\mathcal{S}_{pr}$, respectively. Recall also that $\mathcal{J}_{i+1}$ is the set of jobs with $r_j < t_{i+1}$ in $\mathcal{S}_{npr}$. In order to establish the induction step, we partition the set of jobs $\mathcal{J}_{i+1}$ with $c_j > t$ into the following 3 subsets:

- the set of jobs A with $b_j < t_i$,
- the set of jobs B with $t_i \leq b_j < \tau_{i+1}$, and
- the set of jobs C with $\tau_{i+1} \leq b_j < t_{i+1}$.

The reason of this partitioning is that we argue quite differently for each of these subsets. Let us, now, proceed with the proof. We consider two cases depending of the relevant values of t_i, τ_{i+1}, t.

Case 1: Let $\tau_{i+1} \geq \frac{(1-\frac{1}{m})t+t_i}{(2-\frac{1}{m})}$.

Let $P_A \leq P_{npr,i}$ be the total amount of time that the jobs in A are executed after t_i. Since the schedule $\mathcal{S}_{npr}$ is non-preemptive, each job $J_j \in A$ accomplishes $t - t_i$ amount of its processing time during $[t_i, t]$. Thus, we have that

$$P_{npr,i+1}(t) = (P_A - |A|(t - t_i)) + \sum_{J_j \in B}(b_j + p_j - t) + \sum_{J_j \in C}(b_j + p_j - t).$$

Moreover, we have that

$$\begin{aligned} E_{pr,i+1}(t) &\geq E_{pr,i} + \sum_{J_j \in \mathcal{J}_{i+1} \setminus \mathcal{J}_i} e_j - m \cdot (t - t_i) \\ &= \left(2 - \frac{1}{m}\right)\left(P_{npr,i} + \sum_{J_j \in \mathcal{J}_{i+1} \setminus \mathcal{J}_i} p_j\right) - m \cdot (t - t_i). \end{aligned}$$

Note that the amount of time $P_{npr,i} + \sum_{J_j \in \mathcal{J}_{i+1} \setminus \mathcal{J}_i} p_j$ is the total amount of time during which the jobs in $\mathcal{J}_{i+1}$ are executed after t_i. By definition, for the jobs in A this amount is P_A. Note that these jobs have $b_j < t_i$ and $c_j > t$ and hence $|A|$ processors are dedicated to them during the interval $[t_i, t]$. Consider the set of jobs not in A which are released before τ_{i+1} and are completed after t_i. These jobs contribute to $P_{npr,i} + \sum_{J_j \in \mathcal{J}_{i+1} \setminus \mathcal{J}_i} p_j$ with at least $(m - |A| - |B|)(\tau_{i+1} - t_i) + \sum_{J_j \in B}(b_j + p_j - t_i)$ amount of time, since there is no idle period in the interval $[t_i, \tau_{i+1}]$. Finally, for the jobs in C this contribution is $\sum_{J_j \in C} p_j$. Hence,

$$E_{pr,i+1}(t) \geq \left(2 - \frac{1}{m}\right)\left(P_A + (m - |A| - |B|)(\tau_{i+1} - t_i) + \sum_{J_j \in B}(b_j + p_j - t_i) + \sum_{J_j \in C} p_j\right) - m \cdot (t - t_i).$$

Thus, we have

$$\begin{aligned} \frac{E_{pr,i+1}(t)}{(2 - \frac{1}{m})} - P_{npr,i+1}(t) &\geq (m - |A| - |B|)(\tau_{i+1} - t_i) - \sum_{J_j \in B} t_i - \frac{m}{2 - \frac{1}{m}}(t - t_i) + |A|(t - t_i) + \sum_{J_j \in B} t - \sum_{J_j \in C}(b_j - t) \\ &= \tau_{i+1}(m - |A| - |B|) - m\left(t_i + \frac{t - t_i}{2 - \frac{1}{m}}\right) + (|A| + |B|)t + \sum_{J_j \in C}(t - b_j) \\ &\geq \left(\frac{(1 - \frac{1}{m})t + t_i}{2 - \frac{1}{m}}\right)(m - |A| - |B|) - m\left(t_i + \frac{t - t_i}{2 - \frac{1}{m}}\right) + (|A| + |B|)t \end{aligned}$$

where the last inequality follows from the fact that $t \geq b_j$ for each job in C and using our assumption. Note that $|A| + |B| \geq 1$ as otherwise $P_{npr,i+1} = 0$ and the claim holds. Therefore,

$$\begin{aligned} \frac{E_{pr,i+1}(t)}{(2 - \frac{1}{m})} - P_{npr,i+1}(t) &\geq m\left(\frac{(1 - \frac{1}{m})t + t_i}{2 - \frac{1}{m}} - t_i - \frac{t - t_i}{2 - \frac{1}{m}}\right) + (|A| + |B|)\left(t - \frac{(1 - \frac{1}{m})t + t_i}{2 - \frac{1}{m}}\right) \\ &\geq \frac{t_i - t}{2 - \frac{1}{m}} + \frac{t - t_i}{2 - \frac{1}{m}} \geq 0. \end{aligned}$$

Case 2: Let $\tau_{i+1} < \frac{(1-\frac{1}{m})t+t_i}{(2-\frac{1}{m})}$. Consider first a job $J_j \in A \cup B$. For this job it holds that $b_j \leq \tau_{i+1}$ and $c_j > t$ in $\mathcal{S}_{npr}$. Let $\delta_{npr,j}(t) = b_j + p_j - t$ be the processing time of J_j after time t in $\mathcal{S}_{npr}$.

We will first show that $e_j > t - t_i$. Assume for contradiction that $e_j \leq t - t_i$. Hence, we have

$$\begin{aligned} \delta_{npr,j}(t) &= b_j + p_j - t = b_j + \frac{e_j}{\left(2 - \frac{1}{m}\right)} - t \leq \tau_{i+1} + \frac{e_j}{\left(2 - \frac{1}{m}\right)} - t \\ &< \frac{\left(1 - \frac{1}{m}\right)t + t_i}{\left(2 - \frac{1}{m}\right)} + \frac{t - t_i}{\left(2 - \frac{1}{m}\right)} - t = 0 \end{aligned}$$

which is a contradiction as by definition it holds that $\delta_{npr,j}(t) > 0$. Thus, we consider that $e_j > t - t_i$.

Only jobs of A start before time t_i and finish after time t. Hence, by induction, we have

$$\sum_{J_j \in A} \delta_{npr,j}(t) = P_{npr,i}(t) \leq \frac{E_{pr,i}(t)}{\left(2 - \frac{1}{m}\right)}.$$

Consider now a job $J_j \in B$. Let $\delta_{pr,j}(t) \geq e_j + t_i - t$ be the execution time of J_j after time t in $\mathcal{S}_{pr}$. We have that

$$\begin{aligned}\frac{\delta_{pr,j}(t)}{\left(2-\frac{1}{m}\right)} - \delta_{npr,j}(t) &\geq \frac{e_j + t_i - t}{\left(2-\frac{1}{m}\right)} - (b_j + p_j - t)\\ &= p_j + \frac{t_i - t}{\left(2-\frac{1}{m}\right)} - b_j - p_j + t\\ &\geq \frac{t_i - t}{\left(2-\frac{1}{m}\right)} - \tau_{i+1} + t\\ &= \frac{\left(1-\frac{1}{m}\right)t + t_i}{\left(2-\frac{1}{m}\right)} - \tau_{i+1} > 0\end{aligned}$$

as $b_j \leq \tau_{i+1}$ and $\tau_{i+1} < \frac{(1-\frac{1}{m})t+t_i}{(2-\frac{1}{m})}$.

Consider now a job $J_j \in C$. This job starts its execution at its release date, i.e., $b_j = r_j$. Given that J_j has smaller processing time in $\mathcal{S}_{npr}$ than in $\mathcal{S}_{pr}$ and as $\mathcal{S}_{pr}$ is a feasible schedule, it holds that $\delta_{pr,j}(t) - \delta_{npr,j}(t) > 0$.

Summing up for all jobs in $A \cup B \cup C$, we get $P_{npr,i+1}(t) \leq \frac{E_{pr,i+1}(t)}{(2-\frac{1}{m})}$, and the claim follows. □

4.2. General instances

In this section we present an approximation algorithm for the multiprocessor non-preemptive speed scaling problem $P|r_j, d_j|E$. The main idea of our algorithm is to create an optimal single-processor preemptive schedule for the set of jobs $\mathcal{J}$. The jobs which are preempted at most $n^{\frac{1}{m}}$ times in this schedule, are scheduled non-preemptively on processor 1. For the remaining jobs we create again an optimal single-processor preemptive schedule, we use processor 2 for the jobs which are preempted at most $n^{\frac{1}{m}}$ times, and we continue this procedure until all jobs are assigned to a processor.

Algorithm 3

1: $i = 1$; $\mathcal{J}_i = \mathcal{J}$;
2: **repeat**
3: Run the algorithm in [20] for the problem $1|r_j, d_j, pmtn|E$ with input the set of jobs in $\mathcal{J}_i$ and get the preemptive schedule $\mathcal{S}_{pr,i}$;
4: Create the tree-representation T_i of $\mathcal{S}_{pr,i}$;
5: Let $\mathcal{J}_{i+1}$ be the set of jobs (vertices) with at least $n^{\frac{1}{m}}$ children in T_i;
6: For each job $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$, schedule J_j on the processor i in its largest interval in $\mathcal{S}_{pr,i}$ and get the non-preemptive schedule $\mathcal{S}_{npr,i}$ for the processor i; $i = i + 1$;
7: **until** $\mathcal{J}_i \neq \emptyset$
8: **return** $\mathcal{S}_{npr}$ which is the union of $\mathcal{S}_{npr,i}$'s;

Theorem 3. *Algorithm* 3 *achieves an approximation ratio of* $m^{\alpha}(\sqrt[m]{n})^{\alpha-1}$ *for* $P|r_j, d_j|E$.

Proof. Let $n_i = |\mathcal{J}_i|$ be the number of jobs in the ith iteration. We will first show that in iteration i there are at most $n_i^{1-\frac{1}{m}}$ vertices with at least $n^{\frac{1}{m}}$ children. Assume that there were at least $n_i^{1-\frac{1}{m}}$ vertices with at least $n^{\frac{1}{m}}$ children. Then the number of children in the tree T_i is at least $n_i^{1-\frac{1}{m}} \cdot n^{\frac{1}{m}} \geq n_i$, which is a contradiction. Let k be the number of the iterations of the algorithm. By the previous observation, we have that $k \leq m$.

Consider the ith iteration. Each job $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$ has strictly less than $n^{\frac{1}{m}}$ children in T_i, and hence by Lemma 2 it is preempted strictly less than $n^{\frac{1}{m}}$ times in $\mathcal{S}_{pr,i}$. Our algorithm schedules J_j in $\mathcal{S}_{npr,i}$ during its largest interval in $\mathcal{S}_{pr,i}$. Thus the speed of J_j in $\mathcal{S}_{npr,i}$ is at most $n^{\frac{1}{m}}$ times the speed of J_j in $\mathcal{S}_{pr,i}$. Therefore, using Proposition 2, for any job $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$ it holds that $E(\mathcal{S}_{npr,i}, J_j) \leq (\sqrt[m]{n})^{\alpha-1} \cdot E(\mathcal{S}_{pr,i}, J_j)$. For the energy consumed by $\mathcal{S}_{npr}$ we have

$$E(\mathcal{S}_{npr}) = \sum_{i=1}^{k} \sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{npr,i}, J_j) \leq (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{i=1}^{k} \sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{pr,i}, J_j).$$

Note that the schedule $\mathcal{S}_{pr,i}$ is the optimal preemptive schedule for the jobs in $\mathcal{J}_i$, while the schedule $\mathcal{S}_{pr,1}$ is the optimal preemptive schedule for the jobs in $\mathcal{J}_1$. As $\mathcal{J}_i \subset \mathcal{J}_1 = \mathcal{J}$ it holds that

$$\sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{pr,i}, J_j) \leq \sum_{J_j \in \mathcal{J}_i} E(\mathcal{S}_{pr,i}, J_j) \leq \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j).$$

Therefore, we get that

$$E(\mathcal{S}_{npr}) \leq (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{i=1}^{k} \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j) \leq m \cdot (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j).$$

Note that $\sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j)$ is the optimal energy consumption if all jobs are executed preemptively on a single processor. By Proposition 3 and since the energy consumption of an optimal multiprocessor preemptive schedule without migrations is a lower bound to the energy consumption of an optimal multiprocessor non-preemptive schedule (without migrations) $\mathcal{S}^*$, we have that

$$E(\mathcal{S}_{npr}) \leq m^{\alpha} \cdot (\sqrt[m]{n})^{\alpha-1} \cdot E(\mathcal{S}^*)$$

and the theorem follows. □

5. Concluding remarks

We have investigated the idea of transforming an optimal preemptive schedule to a non-preemptive one showing that for some interesting families of instances it leads to good approximation ratios. However, for the parallel processors case, we need to go beyond this simple idea in order to obtain an algorithm with constant approximation ratio. The existence of such an algorithm is a challenging open question for future research.

Acknowledgments

E. Bampis, D. Letsios and G. Lucarelli are partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, by the project ALGONOW, co-financed by the European Union (European Social Fund—ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES, and by a French-Chinese Cai Yuanpei project.

A. Kononov is partially supported by the RFBR Grant No. 12-01-00184 and the RFH Grant No. 13-22-10002.

References

[1] S. Albers, Energy-efficient algorithms, Commun. ACM 53 (2010) 86–96.
[2] S. Albers, A. Antoniadis, G. Greiner, On multi-processor speed scaling with migration: extended abstract, in: 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, ACM, 2011, pp. 279–288.
[3] S. Albers, F. Müller, S. Schmelzer, Speed scaling on parallel processors, in: 19th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2007, ACM, 2007, pp. 289–298.
[4] E. Angel, E. Bampis, V. Chau, Throughput maximization in the speed-scaling setting, 2013. CoRR, abs/1309.1732.
[5] E. Angel, E. Bampis, F. Kacem, D. Letsios, Speed scaling on parallel processors with migration, in: 18th International European Conference on Parallel and Distributed Computing, Euro-Par 2012, in: LNCS, vol. 7484, Springer, 2012, pp. 128–140.
[6] A. Antoniadis, C.-C. Huang, Non-preemptive speed scaling, in: 13th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2012, in: LNCS, vol. 7357, Springer, 2012, pp. 249–260.
[7] E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, M. Sviridenko, Energy efficient scheduling and routing via randomized rounding, in: 33rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, in: LIPIcs, Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2013.
[8] E. Bampis, D. Letsios, G. Lucarelli, Green scheduling, flows and matchings, in: 23rd International Symposium on Algorithms and Computation, ISAAC 2012, in: LNCS, vol. 7676, Springer, 2012, pp. 106–115.
[9] E. Bampis, D. Letsios, I. Milis, G. Zois, Speed scaling for maximum lateness, in: 18th Annual International Computing and Combinatorics Conference, COCOON 2012, in: LNCS, vol. 7434, Springer, 2012, pp. 25–36.
[10] B.D. Bingham, M.R. Greenstreet, Energy optimal scheduling on multiprocessors with migration, in: International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, IEEE, 2008, pp. 153–161.
[11] D.P. Bunde, Power-aware scheduling for makespan and flow, in: 18th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2006, ACM, 2006, pp. 190–196.
[12] H.-L. Chan, W.-T. Chan, T.W. Lam, L.-K. Lee, K.-S. Mak, P.W.H. Wong, Energy efficient online deadline scheduling, in: 18th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2007, 2007, pp. 795–804.
[13] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, T.-W. Kuo, Multiprocessor energy-efficient scheduling with task migration considerations, in: 16th Euromicro Conference of Real-Time Systems, ECTRS 2004, 2004, pp. 101–108.
[14] G. Greiner, T. Nonner, A. Souza, The bell is ringing in speed-scaled multiprocessor scheduling, in: 21st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2009, ACM, 2009, pp. 11–18.
[15] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results, J. ACM 34 (1987) 144–162.
[16] C.-C. Huang, S. Ott, New results for non-preemptive speed scaling, Research Report MPI-I-2013-1-001, Max-Planck-Institut für Informatik, 2013.
[17] M. Li, F.F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM J. Comput. 35 (2006) 658–671.
[18] C. Phillips, C. Stein, J. Wein, Scheduling jobs that arrive over time, in: S.G. Akl, F. Dehne, J.-R. Sack, N. Santoro (Eds.), Algorithms and Data Structures, in: Lecture Notes in Computer Science, vol. 955, Springer, Berlin, Heidelberg, 1995, pp. 86–97.
[19] K. Pruhs, R. van Stee, P. Uthaisombut, Speed scaling of tasks with precedence constraints, Theory Comput. Syst. 43 (2008) 67–80.
[20] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: 36th Annual Symposium on Foundations of Computer Science, FOCS 1995, 1995, pp. 374–382.