# Speed-scaling with no Preemptions[*]

Evripidis Bampis[1], Dimitrios Letsios[2], and Giorgio Lucarelli[1,3]

[1] Sorbonne Universités, UPMC Univ. Paris 06, UMR 7606, LIP6, Paris, France.
`Evripidis.Bampis@lip6.fr`
[2] Institut für Informatik, Technische Universität München, Germany.
`letsios@in.tum.de`
[3] Université Grenoble-Alpes, INP, UMR 5217, LIG, France.
`giorgio.lucarelli@inria.fr`

**Abstract.** We revisit the non-preemptive speed-scaling problem, in which a set of jobs have to be executed on a single or a set of parallel speed-scalable processor(s) between their release dates and deadlines so that the energy consumption to be minimized. We adopt the speed-scaling mechanism first introduced in [Yao et al., FOCS 1995] according to which the power dissipated is a convex function of the processor's speed. Intuitively, the higher is the speed of a processor, the higher is the energy consumption. For the single-processor case, we improve the best known approximation algorithm by providing a $(1+\epsilon)^\alpha \tilde{B}_\alpha$-approximation algorithm, where $\tilde{B}_\alpha$ is a generalization of the Bell number. For the multiprocessor case, we present an approximation algorithm of ratio $\tilde{B}_\alpha((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^\alpha$ improving the best known result by a factor of $(\frac{5}{2})^{\alpha-1}(\frac{w_{\max}}{w_{\min}})^\alpha$. Notice that our result holds for the fully heterogeneous environment while the previous known result holds only in the more restricted case of parallel processors with identical power functions.

## 1 Introduction

Speed-scaling (or dynamic voltage scaling) is one of the main mechanisms to save energy in modern computing systems. According to this mechanism, the speed of each processor may dynamically change over time, while the energy consumed by the processor is proportional to a convex function of the speed. More precisely, if the speed of a processor is equal to $s(t)$ at a time instant $t$, then the power dissipated is $P(s(t)) = s(t)^\alpha$, where $\alpha > 1$ is a small constant. For example, the value of $\alpha$ is theoretically between two and three for CMOS devices, while some experimental studies showed that $\alpha$ is rather smaller: 1.11

for Intel PXA 270, 1.62 for Pentium M770 and 1.66 for a TCP offload engine [21]. Intuitively, higher speeds lead to higher energy consumption. The energy consumption is the integral of the power over time, i.e., $E = \int P(s(t))dt$.

In order to handle the energy consumption in a computing system with respect to the speed-scaling mechanism, we consider the following scheduling problem. We are given a set of jobs and a single processor or a set of parallel processors. Each job is characterized by a release date, a deadline and an amount of workload that has to be executed between the job's release date and deadline. The objective is to find a feasible schedule that minimizes the energy consumption. In order to describe such a feasible schedule, we have to determine not only the job that has to be executed on every processor at each time instant, but also the speed of each processor.

Speed-scaling scheduling problems have been extensively studied in the literature. Since the seminal paper by Yao et al. [22] in 1995 until very recently, all the energy minimization works considered the *preemptive* case in which the execution of a job may be interrupted and restarted later on the same or even on a different processor (*migratory* case). However, the last three years, there are some works studying the *non-preemptive* case. In this paper, we improve the best known approximation algorithms for the non-preemptive case for both the single-processor and the multiprocessor environments.

*Problem definition and notation.* We consider a set $\mathcal{J}$ of $n$ jobs, each one characterized by an amount of work $w_j$, a release date $r_j$ and a deadline $d_j$. We will consider both the single-processor and the multiprocessor cases. If the speed of a processor is equal to $s(t)$ at a time instant $t$, then the power dissipated is $P(s(t)) = s(t)^\alpha$, where $\alpha > 1$ is a small constant. In the multiprocessor environment, we denote by $\mathcal{P}$ the set of the $m$ available parallel processors. Moreover, we distinguish between the *homogeneous* and the *heterogeneous* multiprocessor cases. In the latter one, we assume that each processor $i \in \mathcal{P}$ has a different constant $\alpha_i$, capturing in this way the existence of processors with different energy consumption rate. For simplicity, we define $\alpha = \max_{i \in \mathcal{P}} \{\alpha_i\}$. Moreover, in the *fully heterogeneous* case we additionally assume that each job $j \in \mathcal{J}$ has a different work $w_{i,j}$, release date $r_{i,j}$ and deadline $d_{i,j}$ on each processor $i \in \mathcal{P}$. In all cases, the objective is to find a schedule that minimizes the energy consumption, $E = \int P(s(t))dt$, with respect to the speed-scaling mechanism, such that each job $j \in \mathcal{J}$ is executed during its *life* interval $[r_j, d_j]$. The results presented in this paper assume that the preemption of jobs is not allowed; and hence neither their migration in the multiprocessor environments.

In what follows, we denote by $w_{\max}$ and $w_{\min}$ the maximum and the minimum work, respectively, among all jobs. Moreover, we call an instance *agreeable* if earlier released jobs have earlier deadlines, i.e., for each $j$ and $j'$ with $r_j \leq r_{j'}$ then $d_j \leq d_{j'}$. Finally, given a schedule $\mathcal{S}$ we denote by $E(\mathcal{S})$ its energy consumption.

*Related work.* In [22], a polynomial-time algorithm has been presented that finds an optimal preemptive schedule when a single processor is available. In

2

the case where the preemption and also the migration of jobs are allowed, several polynomial-time algorithms have been proposed when a set of homogeneous parallel processors is available [3, 6, 10, 12], while in the fully heterogeneous environment an $OPT + \epsilon$ algorithm with complexity polynomial to $\frac{1}{\epsilon}$ has been presented in [9]. In the case where the preemption of jobs is allowed but not their migration, the problem becomes strongly NP-hard even if all jobs have equal release dates and equal deadlines [4]. For this special case, the authors in [4] observed that a PTAS can be derived from [17]. For arbitrary release dates and deadlines, a $B_{\lceil \alpha \rceil}$-approximation algorithm is known [16], where $B_{\lceil \alpha \rceil}$ is the $\lceil \alpha \rceil$-th Bell number. This result has been extended in [9] for the fully heterogeneous environment, where an approximation algorithm of ratio $(1 + \epsilon)^\alpha \tilde{B}_\alpha$ has been presented, where $\tilde{B}_\alpha = \sum_{k=0}^{\infty} \frac{k^\alpha e^{-1}}{k!}$ is a generalization of the Bell number that is also valid for fractional values of $\alpha$.

When preemptions are not allowed, Antoniadis and Huang [7] proved that the single-processor case is strongly NP-hard, while they have also presented a $2^{5\alpha-4}$-approximation algorithm. In [9], an approximation algorithm of ratio $2^{\alpha-1}(1 + \epsilon)^\alpha \tilde{B}_\alpha$ has been proposed, improving the ratio given in [7] for any $\alpha < 114$. Recently, an approximation algorithm of ratio $(12(1 + \epsilon))^{\alpha-1}$ is given in [15], improving the approximation ratio for any $\alpha > 25$. Moreover, the relation between preemptive and non-preemptive schedules in the energy-minimization setting has been studied in [8]. The authors show that starting from the optimal preemptive solution created by the algorithm in [22], it is possible to obtain a non-preemptive solution which guarantees an approximation ratio of $(1+\frac{w_{\max}}{w_{\min}})^\alpha$. In the special case where all jobs have equal work this leads to a constant factor approximation of $2^\alpha$. Recently, for this special case, Angel et al. [5] and Huang and Ott [18], independently, proposed an optimal polynomial-time algorithm based on dynamic programming. Note also that for agreeable instances the single-processor non-preemptive speed-scaling problem can be solved to optimality in polynomial time, as the algorithm proposed by Yao et al. [22] for the preemptive case returns a non-preemptive schedule for agreeable instances.

For homogeneous multiprocessors when preemptions are not allowed, an approximation algorithm with ratio $m^\alpha(\sqrt[m]{n})^{\alpha-1}$ has been presented in [8]. More recently, Cohen-Addad et al. [15] proposed an algorithm of ratio $(\frac{5}{2})^{\alpha-1}\tilde{B}_\alpha((1+\epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$, transforming the problem to the fully heterogeneous preemptive non-migratory case and using the approximation algorithm proposed in [9]. This algorithm leads to an approximation ratio of $2(1+\epsilon)^\alpha 5^{\alpha-1}\tilde{B}_\alpha$ for the case where all jobs have equal work. The authors in [15] observe also that their algorithm can be used when each job $j \in \mathcal{J}$ has a different work $w_{i,j}$ on each processor $i \in \mathcal{P}$, by loosing an additional factor of $(\frac{w_{\max}}{w_{\min}})^\alpha$.

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 a budget of energy, or the optimization of a linear combination of the energy consumption and some QoS criterion (see for example [11, 13, 20]). Moreover, two other energy minimization variants of the speed-scaling model 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 [14]), and the *discrete speed model* in which the speeds of the processors can be selected among a set of discrete speeds (see for example [19]). The interested reader can find more details in the surveys [1, 2].

*Our contribution.* In Section 2 we revisit the single-processor non-preemptive speed-scaling problem, and we present an approximation algorithm of ratio $(1 + \varepsilon)^{\alpha-1}\tilde{B}_\alpha$ which becomes the best algorithm for any $\alpha \leq 77$. Recall that in practice $\alpha$ is a small constant and usually $\alpha \in (1, 3]$. In [8], where the relation between preemptive and non-preemptive schedules has been explored, an example has been proposed which shows that the ratio of the energy consumption of the optimal non-preemptive schedule over the energy consumption of the optimal preemptive schedule can be $\Omega(n^{\alpha-1})$. A similar example was used in [15] to show that the standard configuration linear programming formulation has the same integrality gap. In both cases, $w_{\max} = n$ and $w_{\min} = 1$ and the worst-case ratio of the energy consumption of the optimal non-preemptive schedule over the energy consumption of the optimal preemptive one can be seen as $\Omega((\frac{w_{\max}}{w_{\min}})^{\alpha-1})$. In this direction, a $(1 + \frac{w_{\max}}{w_{min}})^{\alpha-1}$-approximation algorithm for the single-processor case has been presented in [8]. To overcome the above lower bound, all known constant-factor approximation algorithms for the single-processor problem [7, 9, 15] consider an initial partition of the time horizon into some specific intervals defined by the so-called *landmarks*. These intervals are defined in such a way that there is not a job whose life interval is included in one of them. Intuitively, this partition is used in order to improve the lower bound by focusing on special preemptive schedules that can be transformed to a feasible non-preemptive schedule without loosing a lot in terms of approximation. Here, we are able to avoid the use of this partition improving in this way the result of [9] by a factor of $2^{\alpha-1}$. In order to do that, we modify the *configuration linear program* proposed in [9] by including an additional structural property that is valid for any feasible non-preemptive schedule. This property helps us to obtain a "good" preemptive schedule after a randomized rounding procedure. We transform this "good" preemptive schedule to a new instance of the energy-minimization single-processor problem that is *agreeable* by choosing in an appropriate way new release dates and deadlines for the jobs. In this way, it is then sufficient to apply the algorithm proposed in [22] in order to get a non-preemptive schedule of energy consumption at most the energy consumption of the preemptive one.

In Section 3 we consider the fully heterogeneous non-preemptive speed-scaling problem, and we improve the approximation ratio of $(\frac{w_{\max}}{w_{\min}})^\alpha(\frac{5}{2})^{\alpha-1}\tilde{B}_\alpha((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$ given in [15] to $\tilde{B}_\alpha((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$. Consecutively, our result generalizes and improves the approximation ratio for the equal-works case from $2(1 + \epsilon)^\alpha 5^{\alpha-1}\tilde{B}_\alpha$ to $(2(1 + \epsilon))^\alpha\tilde{B}_\alpha$. Note also that we generalize the machine environment and we pass from the homogeneous with different $w_{i,j}$'s to the fully heterogeneous one. Our algorithm combines two basic ingredients: the $\tilde{B}_\alpha(1 + \epsilon)^\alpha$-approximation algorithm of [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem and the $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$-approximation

4

algorithm of [8] for the single-processor non-preemptive speed-scaling problem. The first algorithm is used in order to assign the jobs to the processors, while the second one to get a non-preemptive schedule for each processor independently. The key observation here is that the algorithm for the single-processor non-preemptive case presented in [8] transforms the optimal preemptive schedule obtained by the algorithm in [22] into a non-preemptive one. In this way, its approximation ratio is computed with respect to the energy consumption of the optimal preemptive schedule, which is a lower to the energy consumption of the optimal non-preemptive schedule.

We summarize our results with respect to the existing bibliography in Table 1.

| Machine environment | Previous known result | Our results |
|---|---|---|
| single-processor | $2^{\alpha-1}(1+\epsilon)^{\alpha}\tilde{B}_{\alpha}$ [9] $(12(1+\epsilon))^{\alpha-1}$ [15] | $(1+\epsilon)^{\alpha}\tilde{B}_{\alpha}$ |
| homogeneous | $(\frac{5}{2})^{\alpha-1}\tilde{B}_{\alpha}((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^{\alpha}$ [15] | |
| homogeneous with $w_{i,j}$'s | $(\frac{5}{2})^{\alpha-1}\tilde{B}_{\alpha}((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}})\frac{w_{\max}}{w_{\min}})^{\alpha}$ [15] | |
| fully heterogeneous | | $\tilde{B}_{\alpha}((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^{\alpha}$ |

**Table 1.** Comparison of the approximation ratios obtained in this paper with the previously best known approximation ratios.

## 2  Single-processor

In this section we consider the single-processor non-preemptive case and we present an approximation algorithm of ratio $(1+\varepsilon)\tilde{B}_{\alpha}$, improving upon the previous known results [7, 9, 15] for any $\alpha \leq 77$. Our algorithm is based on a linear programming formulation combining ideas from [9, 15] and the randomized rounding proposed in [9].

Before formulating the problem as a linear program we need to discretize the time into slots. Consider the set of all different release dates and deadlines of jobs in increasing order, i.e., $t_1 < t_2 < \ldots < t_k$. For each $\ell$ , $1 \leq \ell \leq k - 1$, we split the time between $t_\ell$ and $t_{\ell+1}$ into $n^2(1 + \frac{1}{\epsilon})$ equal length slots as proposed in [18]. Let $\mathcal{T}$ be the set of all created slots. Henceforth, we will consider only solutions in which each slot can be occupied by at most one job which uses the whole slot. Huang and Ott [18] proved that this can be done by loosing a factor of $(1 + \epsilon)^{\alpha-1}$.

Our formulation is based on the configuration linear program which was proposed in [9]. In [15], an additional constraint was used for the single-processor non-preemptive problem. This constraint implies that the life interval of a job cannot be included to the execution interval of another job. We explicitly incorporate this constraint in the definition of the set of configurations for each job. More specifically, for a job $j \in \mathcal{J}$, we define a configuration $c$ to be a set

of consecutive slots in $[r_j, d_j]$ such that there is not another job $j'$ whose life interval $[r_{j'}, d_{j'}]$ is included in $c$. Let $\mathcal{C}_j$ be the set of all possible configurations for the job $j$. We introduce a binary variable $x_{j,c}$ which is equal to one if the job $j$ is executed according the configuration $c \in \mathcal{C}_j$. Let $|c|$ be the length (in time) of the configuration $c$. Note that the number of configurations is polynomial as they only contain consecutive slots and the number of slots is also polynomial.

For notational convenience, we write $t \in c$ if the slot $t \in \mathcal{T}$ is part of the configuration $c \in \mathcal{C}_j$ of job $j \in \mathcal{J}$. By the convexity of the power function, each job in an optimal schedule runs in a constant speed (see for example [22]). Hence, the quantity $\frac{w_j^{\alpha}}{|c|^{\alpha-1}}$ corresponds to the energy consumed by $j$ if it is executed according to $c$, as the constant speed that will be used for $j$ is equal to $\frac{w_j}{|c|}$. Consider the following integer linear program.

$$\min \sum_{j \in \mathcal{J}} \sum_{c \in \mathcal{C}_j} x_{j,c} \frac{w_j^{\alpha}}{|c|^{\alpha-1}}$$

$$\sum_{c \in \mathcal{C}_j} x_{j,c} \geq 1 \ \ \forall j \in \mathcal{J}$$

$$\sum_{j \in \mathcal{J}} \sum_{c \in \mathcal{C}_j : t \in c} x_{j,c} \leq 1 \ \ \forall t \in \mathcal{T}$$

$$x_{j,c} \in \{0,1\} \ \ \forall j \in \mathcal{J}, c \in \mathcal{C}_j$$

The first constraint ensures that each job is executed according to a configuration. The second constraint implies that at each slot at most one configuration and hence at most one job can be executed.

We consider the randomized rounding procedure proposed in [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem, adapted to the single processor environment. More specifically, for each job $j \in \mathcal{J}$ we choose at random with probability $x_{j,c}$ a configuration $c \in \mathcal{C}_j$. By doing this, more than one jobs may be assigned in a slot $t \in \mathcal{T}$ which has as a result to get a non-feasible schedule. In order to deal with this infeasibility, for each slot $t \in \mathcal{T}$ we perform an appropriate speed-up that leads to a feasible preemptive schedule. The above procedure is described formally in Algorithm 1.

---
**Algorithm 1**
---
1: Solve the configuration LP relaxation.
2: For each job $j \in \mathcal{J}$, choose a configuration at random with probability $x_{j,c}$.
3: Let $w_j(t)$ be the amount of work executed for job $j$ during the slot $t \in \mathcal{T}$ according to its chosen configuration.
4: Set the processor's speed during $t$ as if $\sum_{j \in \mathcal{J}} w_j(t)$ units of work are executed with constant speed during the entire $t$, i.e., $\sum_{j \in \mathcal{J}} w_j(t)/|t|$, where $|t|$ is the length of $t$.
5: **return** the obtained schedule $\mathcal{S}_{pr}$.
---

The analysis of the above procedure in [9] is done independently for each slot, while the speed-up performed leads to a loss of a factor of $\tilde{B}_\alpha$ to the approximation ratio. We can use exactly the same analysis and get the same approximation guarantee for our problem, that is

$$E(\mathcal{S}_{pr}) \leq \tilde{B}_\alpha \cdot LP^* \tag{1}$$

where $LP^*$ is the objective value of an optimal solution of the configuration LP relaxation.

In what follows, given the feasible preemptive schedule $\mathcal{S}_{pr}$ obtained by Algorithm 1, we will create a feasible non-preemptive schedule $\mathcal{S}_{npr}$ of energy consumption at most $E(\mathcal{S}_{pr})$. In fact, we first create a restricted agreeable instance $\mathcal{I}'$ of our initial instance $\mathcal{I}$ based on $\mathcal{S}_{pr}$. Then, we will apply the algorithm proposed by Yao et al. [22] that finds the optimal preemptive schedule on a single processor, which turns to be a non-preemptive schedule since the instance is agreeable. A formal description of our algorithm follows.

---

**Algorithm 2**

---

1: Run Algorithm 1 in the initial instance $\mathcal{I}$ and get the schedule $\mathcal{S}_{pr}$.
2: **for** each job $j \in \mathcal{J}$ **do**
3:     Let $b_j$ be the time at which the first piece of $j$ begins in $\mathcal{S}_{pr}$.
4:     Let $e_j$ be the time at which the last piece of $j$ ends in $\mathcal{S}_{pr}$.
5:     Select $r'_j$ and $d'_j$ such that
        – $r_j \leq r'_j \leq b_j$ and $r'_j$ is minimum;
        – $e_j \leq d'_j \leq d_j$ and $d'_j$ is maximum;
        – for any other job $i \in \mathcal{J} \setminus \{j\}$, it **cannot** hold that $r'_j < r_i < d_i < d'_j$.
6: Create the instance $\mathcal{I}'$ in which each job $j \in \mathcal{J}$ has:
    – release date $r'_j$,
    – deadline $d'_j$,
    – work $w_j$.
7: Run the algorithm proposed in [22] in the transformed instance $\mathcal{I}'$ and get the schedule $\mathcal{S}_{npr}$.
8: **return** $\mathcal{S}_{npr}$.

---

An example of the above transformation is given in Fig. 1. In this picture, the life intervals of jobs $J_1$ and $J_4$ are shortened. For example, in the preemptive schedule $\mathcal{S}_{pr}$ the job $J_4$ is executed on the right of the job $J_5$. Hence, in the restricted instance we cut down the part of the life interval of $J_4$ which is on the left of the release date of $J_5$. Intuitively, we decide if $J_4$ should be executed on the left or on the right of $J_5$ with respect to $\mathcal{S}_{pr}$ and we transform the initial instance appropriately.

**Lemma 1.** *The restricted instance $\mathcal{I}'$ is agreeable.*

*Proof.* Assume for contradiction that there are two jobs $i, j \in \mathcal{J}$ in $\mathcal{I}'$ such that $r'_j < r'_i < d'_i < d'_j$. The algorithm did not select a smaller $r'_i$ because either
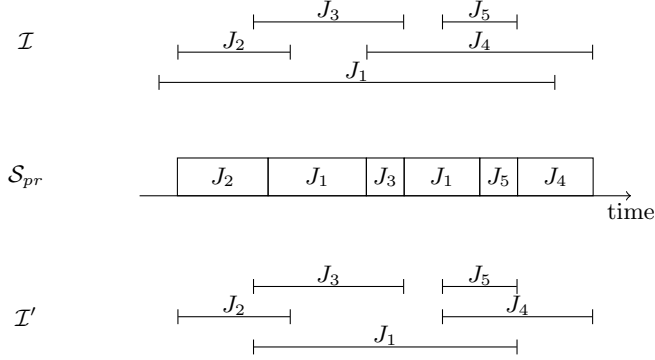
**Fig. 1.** The transformation of an instance $\mathcal{I}$ into a restricted (agreeable) instance $\mathcal{I}'$ based on the feasible preemptive schedule $\mathcal{S}_{pr}$.

there is a job $k \in \mathcal{J}$ such that $r'_i = r_k < d_k < d'_i$ or $r'_i = r_i$. In the first case, we have that $r'_j < r_k < d_k < d'_j$, which is a contradiction to the definition of configurations and the selection of $r'_j$. In the second case, the algorithm did not select a bigger $d'_i$ because either there is a job $\ell \in \mathcal{J}$ such that $r'_i < r_\ell < d_\ell = d'_i$ or $d'_i = d_i$. In both last subcases we have again a contradiction, as either $r'_j < r_\ell < d_\ell < d'_j$ or $r'_j < r_i < d_i < d'_j$. $\qquad\square$

**Theorem 1.** *Algorithm 2 achieves an approximation ratio of $(1 + \varepsilon)^{\alpha-1}\tilde{B}_\alpha$ for the single-processor non-preemptive speed-scaling problem.*

*Proof.* By construction, the life interval of each job $j \in \mathcal{J}$ in the restricted instance $\mathcal{I}'$ is a superset of its execution interval in $\mathcal{S}_{pr}$, i.e., $[b_j, e_j] \subseteq [r'_j, d'_j]$. Hence, the schedule $\mathcal{S}_{npr}$ is a feasible preemptive schedule for $\mathcal{I}'$.

By Lemma 1, $\mathcal{I}'$ is an agreeable instance. Thus, by applying the algorithm proposed by Yao et al. [22], the schedule $\mathcal{S}_{npr}$ is a non-preemptive schedule for $\mathcal{I}'$. Moreover, the life interval of each job $j \in \mathcal{J}$ in $\mathcal{I}'$ is a subset of its life interval in the initial instance $\mathcal{I}$, i.e., $[r'_j, d'_j] \subseteq [r_j, d_j]$. Hence, the schedule $\mathcal{S}_{npr}$ is a feasible non-preemptive schedule for $\mathcal{I}$.

Concerning the energy consumption, it holds that $E(\mathcal{S}_{npr}) \leq E(\mathcal{S}_{pr})$ since $\mathcal{S}_{npr}$ is an optimal schedule for $\mathcal{I}'$ for both preemptive and non-preemptive versions. Hence, by using Equation (1) we have that $E(\mathcal{S}_{npr}) \leq \tilde{B}_\alpha \cdot LP^*$. Finally, taking into account the factor we loose by the discretization of the time proposed in [18], the theorem follows. $\qquad\square$

## 3 Parallel Processors

In this section we consider the fully heterogeneous multiprocessor case and we propose an approximation algorithm of ratio $\tilde{B}_\alpha((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^\alpha$, generalizing the recent result by Cohen-Addad et al. [15] from the homogeneous with different

8

$w_{i,j}$'s to the fully heterogeneous environment and improving their ratio by a factor of $(\frac{w_{\max}}{w_{\min}})^{\alpha}(\frac{5}{2})^{\alpha-1}$. Our algorithm uses the following result proposed in [8].

**Theorem 2.** [8] *There is an approximation algorithm for the single-processor non-preemptive speed-scaling problem that returns a schedule $\mathcal{S}$ with energy consumption $E(\mathcal{S}) \leq (1 + \frac{w_{\max}}{w_{\min}})^{\alpha} E(\mathcal{S}_{pr}^*) \leq (1 + \frac{w_{\max}}{w_{\min}})^{\alpha} E(\mathcal{S}_{npr}^*)$, where $\mathcal{S}_{pr}^*$ and $\mathcal{S}_{npr}^*$ are the optimal schedules for the preemptive and the non-preemptive case, respectively.*

The key observation in the above theorem concerns the first inequality of Theorem 2 that the energy consumption of the non-preemptive schedule $\mathcal{S}$ created by the algorithm in [8] is bounded within a factor of $(1 + \frac{w_{\max}}{w_{\min}})^{\alpha}$ by the energy consumption of the optimal preemptive schedule $\mathcal{S}_{pr}^*$. Based on this, we propose Algorithm 3 which uses the $(1+\epsilon)^{\alpha}\tilde{B}_{\alpha}$-approximation algorithm proposed in [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem to find a good assignment of the jobs to the processors and then applies Theorem 2 to create a non-preemptive schedule independently for each processor.

---

**Algorithm 3**

---

1: Find a preemptive non-migratory schedule $\mathcal{S}$ using the algorithm proposed in [9] for the fully heterogeneous environment.
2: **for** each processor $i \in \mathcal{P}$ **do**
3:     Let $\mathcal{J}_i$ be the set of jobs assigned to processor $i$ according to $\mathcal{S}$.
4:     Find a single-processor non-preemptive schedule $\mathcal{S}_{i,npr}$ using the algorithm proposed in [8] (Theorem 2) with input $\mathcal{J}_i$.
5: **return** the non-preemptive schedule $\mathcal{S}_{npr}$ which consists of the non-preemptive schedules $\mathcal{S}_{i,npr}$, $1 \leq i \leq m$.

---

**Theorem 3.** *Algorithm 3 achieves an approximation ratio of $\tilde{B}_{\alpha}((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^{\alpha}$ for the fully heterogeneous non-preemptive speed-scaling problem.*

*Proof.* Consider first the schedule $\mathcal{S}$ obtained in Line 1 of the algorithm, and let $\mathcal{S}_{i,pr}$ be the (sub)schedule of $\mathcal{S}$ that corresponds to the processor $i \in \mathcal{P}$. In other words, each $\mathcal{S}_{i,pr}$ is a feasible preemptive schedule of the subset of jobs $\mathcal{J}_i$. As $\mathcal{S}$ is a non-migratory schedule the subsets of jobs $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_m$ are pairwise disjoint. Hence, we have that

$$\sum_{i \in \mathcal{P}} E(\mathcal{S}_{i,pr}) = E(\mathcal{S}) \leq (1 + \epsilon)^{\alpha} \tilde{B}_{\alpha} E(\mathcal{S}^*) \tag{2}$$

where $\mathcal{S}^*$ is the optimal non-preemptive schedule for our problem and the inequality holds by the result in [9] and the fact that the energy consumption in an optimal preemptive-non-migratory schedule is a lower bound to the energy consumption of $\mathcal{S}^*$.

9

Consider now, for each processor $i \in \mathcal{P}$, the schedule $\mathcal{S}_{i,npr}$ created in Line 4 of the algorithm. By Theorem 2 we have that

$$E(\mathcal{S}_{i,npr}) \leq \left(1 + \frac{w_{\max}}{w_{\min}}\right)^{\alpha} E(\mathcal{S}_{i,pr}^*)$$

where $\mathcal{S}_{i,pr}^*$ is an optimal preemptive schedule for the subset of jobs $\mathcal{J}_i$. As $\mathcal{S}_{i,pr}^*$ and $\mathcal{S}_{i,pr}$ are schedules concerning the same set of jobs and $\mathcal{S}_{i,pr}^*$ is the optimal preemptive schedule, we have that

$$E(\mathcal{S}_{i,npr}) \leq \left(1 + \frac{w_{\max}}{w_{\min}}\right)^{\alpha} E(\mathcal{S}_{i,pr}) \tag{3}$$

Since $\mathcal{S}_{npr}$ is the concatenation of $\mathcal{S}_{i,npr}$ for all $i \in \mathcal{P}$, and by using Equations (2) and (3), the theorem follows. $\qquad\square$

Algorithm 3 can be also used for the case where all jobs have equal work on each processor, i.e., each job $j \in \mathcal{J}$ has to execute an amount of work $w_{i,j} = w_i$ if it is assigned on processor $i \in \mathcal{P}$. In this case we get the following result.

**Corollary 1.** *Algorithm 3 achieves a constant-approximation ratio of $\tilde{B}_{\alpha}(2(1+\epsilon))^{\alpha}$ for the fully heterogeneous non-preemptive speed-scaling problem when all jobs have equal work on each processor.*

## 4    Conclusions

In this paper, we have presented algorithms with improved approximation ratios for both the single-processor and the multiprocessor environments. A challenging question left open in this work is the existence of a constant approximation ratio algorithm for the multiprocessor case. Also, there is a need for non-approximability results in the same vein as the one presented in [15].

## References

1. S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.
2. S. Albers. Algorithms for dynamic speed scaling. In *STACS*, volume 9 of *LIPIcs*, pages 1–11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
3. S. Albers, A. Antoniadis, and G. Greiner. On multi-processor speed scaling with migration: extended abstract. In *SPAA*, pages 279–288. ACM, 2011.
4. S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298. ACM, 2007.
5. E. Angel, E. Bampis, and V. Chau. Throughput maximization in the speed-scaling setting. *CoRR*, abs/1309.1732, 2013.
6. E. Angel, E. Bampis, F. Kacem, and D. Letsios. Speed scaling on parallel processors with migration. In *Euro-Par*, volume 7484 of *LNCS*, pages 128–140, 2012.
7. A. Antoniadis and C.-C. Huang. Non-preemptive speed scaling. In *SWAT*, volume 7357 of *LNCS*, pages 249–260. Springer, 2012.

8. E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, and I. Nemparis. From preemptive to non-preemptive speed-scaling scheduling. In *COCOON*, volume 7936 of *LNCS*, pages 134–146. Springer, 2013.

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

10. E. Bampis, D. Letsios, and G. Lucarelli. Green scheduling, flows and matchings. In *ISAAC*, volume 7676 of *LNCS*, pages 106–115. Springer, 2012.

11. E. Bampis, D. Letsios, I. Milis, and G. Zois. Speed scaling for maximum lateness. In *COCOON*, volume 7434 of *LNCS*, pages 25–36. Springer, 2012.

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

13. D. P. Bunde. Power-aware scheduling for makespan and flow. In *SPAA*, pages 190–196. ACM, 2006.

14. H.-L. Chan, W.-T. Chan, T. W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *SODA*, pages 795–804, 2007.

15. V. Cohen-Addad, Z. Li, C. Mathieu, and I. Milis. Energy-efficient algorithms for non-preemptive speed-scaling. In *WAOA*, LNCS. Springer, 2014.

16. G. Greiner, T. Nonner, and A. Souza. The bell is ringing in speed-scaled multiprocessor scheduling. In *SPAA*, pages 11–18. ACM, 2009.

17. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

18. C.-C. Huang and S. Ott. New results for non-preemptive speed scaling. In *MFCS*, volume 8635 of *LNCS*, pages 360–371. Springer, 2014.

19. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal on Computing*, 35:658–671, 2006.

20. K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43:67–80, 2008.

21. A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM*, pages 2007–2015. IEEE, 2009.

22. F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.