New Results for Non-Preemptive Speed Scaling

Chien-Chung Huang
Sebastian Ott

MPI–I–2013–1–001 August 2013

Authors' Addresses

Chien-Chung Huang
Chalmers University
Göteborg, Sweden

Sebastian Ott
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Abstract

We consider the speed scaling problem introduced in the seminal paper of Yao et al. [25]. In this problem, a number of jobs, each with its own processing volume, release time, and deadline needs to be executed on a speed-scalable processor. The power consumption of this processor is $P(s) = s^\alpha$, where s is the processing speed, and $\alpha > 1$ is a constant. The total energy consumption is power integrated over time, and the goal is to process all jobs while minimizing the energy consumption.

The preemptive version of the problem, along with its many variants, has been extensively studied over the years. However, little is known about the non-preemptive version of the problem, except that it is strongly NP-hard and allows a constant factor approximation [6]. Up until now, the (general) complexity of this problem is unknown. In the present paper, we study an important special case of the problem, where the job intervals form a laminar family, and present a quasipolynomial-time approximation scheme for it, thereby showing that (at least) this special case is not APX-hard, unless NP $\subseteq$ DTIME($2^{poly(\log n)}$).

The second contribution of this work is a polynomial-time algorithm for the special case of equal-volume jobs, where previously only a 2^α approximation was known [8]. In addition, we show that two other special cases of this problem allow fully polynomial-time approximation schemes (FPTASs).

Contents

1 Introduction

Speed scaling is a widely applied technique for energy saving in modern microprocessors. Its general idea is to strategically adjust the processing speed, with the dual goal of finishing the tasks at hand in a timely manner while minimizing the energy consumption. The following theoretical model was introduced by Yao et al. in their seminal paper of 1995 [25]. We are given a set of jobs, each with its own *volume* v_j (number of CPU cycles needed for completion of this job), *release time* r_j (when the job becomes available), and *deadline* d_j (when the job needs to be finished), and a processor with power function $P(s) = s^\alpha$, where s is the processing speed, and $\alpha > 1$ is a constant (typically between two and three for modern microprocessors [15, 24]). The energy consumption is power integrated over time, and the goal is to process all given jobs in their allowed time intervals while minimizing the total energy consumption.

Most work in the literature focuses on the *preemptive* version, where the execution of a job may be interrupted and resumed at a later point of time. For this setting, Yao et al. [25] gave a polynomial-time exact algorithm to compute the optimal schedule. The *non-preemptive* version, where a job must be processed uninterruptedly until its completion, has so far received surprisingly little attention. From a theoretical point of view, the non-preemptive model is of interest, since it is a natural variation of Yao et al.'s original model. In practice, non-preemptive scheduling is often preferred or even unavoidable for the following reasons [20]:

- In many real-time applications, properties of device hardware or software make preemption impossible or prohibitively expensive.
- Non-preemptive algorithms cause lower overheads with respect to running time and energy costs.
- The overhead of preemptive algorithms is more difficult to characterize and predict.

- Non-preemptive algorithms are easier to implement.
- Non-preemptive scheduling guarantees exclusive access to shared resources.

So far, little is known about the complexity of the non-preemptive speed scaling problem [6, 8]. On the negative side, no lower bound is known, except that the problem is strongly NP-hard [6]. On the positive side, Antoniadis and Huang [6] showed that the problem has a constant factor approximation algorithm, although the obtained factor $2^{5\alpha-4}$ is very large.

1.1 Our Results and Techniques

In this paper, we work towards better understanding the complexity of the non-preemptive speed scaling problem, by considering several special cases and presenting (near)-optimal algorithms. In the following, we give a detailed overview on the different cases and our respective results.

Laminar Instances: An instance is said to be *laminar* if for any two different jobs j_1 and j_2, either $[r_{j_1}, d_{j_1}) \subseteq [r_{j_2}, d_{j_2})$, or $[r_{j_2}, d_{j_2}) \subseteq [r_{j_1}, d_{j_1})$, or $[r_{j_1}, d_{j_1}) \cap [r_{j_2}, d_{j_2}) = \emptyset$. The problem remains strongly NP-hard for this case, and the best known approximation ratio is $2^{4\alpha-3}$ [6]. We present the first $(1+\epsilon)$-approximation for this problem, with a quasipolynomial running time (i.e. a running time bounded by $2^{poly(\log n)}$ for any fixed $\epsilon > 0$); a so-called quasipolynomial-time approximation scheme (QPTAS). Our result implies that laminar instances are not APX-hard, unless NP $\subseteq$ DTIME($2^{poly(\log n)}$). We remark that laminar instances form a very important subclass of instances that not only arise commonly in practice (e.g. when jobs are created by recursive function calls [21]), but are also of great theoretical interest, as they highlight the difficulty of the non-preemptive speed scaling problem. Taking instances with an "opposite" structure, namely *agreeable instances* (here for any two jobs j_1 and j_2 with $r_{j_1} < r_{j_2}$, it holds that $d_{j_1} < d_{j_2}$), the problem becomes polynomial-time solvable [6]. On the other hand, further restricting the instances from laminar to *purely-laminar* (see next case) results in a problem that is only weakly NP-hard and admits an FPTAS.

Purely-Laminar Instances: An instance is said to be *purely-laminar* if for any two different jobs j_1 and j_2, either $[r_{j_1}, d_{j_1}) \subseteq [r_{j_2}, d_{j_2})$, or $[r_{j_2}, d_{j_2}) \subseteq [r_{j_1}, d_{j_1})$. We present a fully polynomial-time approximation

scheme (FPTAS) for this class of instances. This is the best possible result (unless P = NP), as the problem is still (weakly) NP-hard [6].

Equal-Volume Jobs: If all jobs have the same volume $v_1 = v_2 = \ldots = v_n = v$, we present a polynomial-time algorithm for computing an (exactly) optimal schedule. We thereby improve upon a recent result of Bampis et al. [8], who proposed a 2^α-approximation algorithm for this case, and answer their question for the complexity status of this problem.

Bounded Number of Time Windows: If the total number of different time windows is bounded by a constant, we present an FPTAS for the problem. This result is again optimal (unless P = NP), as the problem remains (weakly) NP-hard even if there are only two different time windows [6].

The basis for all our results is a discretization of the problem, in which we allow the processing of any job to start and end only at a carefully chosen set of *grid points* on the time axis. We then use various dynamic programs to optimize over a highly restricted set of schedules, exploiting the structural properties of specific optimal solutions. Technically, our QPTAS for laminar instances is of the most interest. It involves a lax representation of job sets in the bookkeeping, which is crucial to obtain a quasipolynomial running time. Roughly speaking, we "lose" a number of jobs during the recursion, but we ensure that these jobs can later be scheduled with only a small increment of energy cost.

1.2 Related Work

The study of dynamic speed scaling problems for reduced energy consumption was initiated by Yao, Demers, and Shenker in 1995. In their seminal paper [25], they present a polynomial-time algorithm for finding an optimal schedule when preemption of jobs is allowed. Furthermore, they also studied the online version of the problem (again with preemption of jobs allowed), where jobs become known only at their release times, and proposed two constant-competitive algorithms called *Average Rate* and *Optimal Available*. It was later shown by Bansal et al. [12], that Optimal Available is the better of the two, and achieves a competitive ratio of exactly α^α. In the same paper, they also provide a new online algorithm with a further improved competitive ratio of $2(\alpha/(\alpha-1))^\alpha e^\alpha$. The exponential dependence on α is unavoidable, as a lower bound on the competitive ratio of any online algorithm is $e^{\alpha-1}/\alpha$ [11].

Over the years, a rich spectrum of variations and generalizations of the original model have been investigated. Irani et al. [19], for instance, considered a setting where the processor additionally has a sleep state available. In their model, the power consumption is strictly positive even at zero speed, unless the processor is transitioned into the sleep state. To wake up a "sleeping" processor, a fixed amount of energy must be invested. Further work in that direction are [18] and a recent result of Albers and Antoniadis [1]. Another approach is to restrict the set of possible speeds that we may choose from, for example by allowing only a number of discrete speed levels [17, 22], or bounding the maximum possible speed [9, 16, 18]. The existence of a feasible schedule for all jobs is then no longer guaranteed, and maximizing the throughput naturally enters the objective function as an additional criterium. Variations with respect to the objective function have also been studied in the unbounded speed model, for instance by Albers and Fujiwara [3] and Bansal et al. [13], who try to minimize a combination of energy consumption and total flow time of the jobs. Finally, the problem has also been studied for arbitrary power functions [10], as well as for multiprocessor settings. In the latter, one has to distinguish whether migration of jobs between processors is allowed [2, 5, 14] or disallowed [4].

In contrast to this diversity of results, the non-preemptive version of the speed scaling problem has been addressed very rarely in the literature. Only recently, in 2012, Antoniadis and Huang [6] proved that the problem is strongly NP-hard, and gave a $2^{5\alpha-4}$-approximation algorithm for the general case. They also considered so-called *laminar instances*, where the time windows of any two different jobs are either disjoint, or one is contained in the other. In this special case they could improve their approximation ratio to $2^{4\alpha-3}$. Finally, in a very recent paper, Bampis et al. [8] proposed a 2^{α}-approximation algorithm for the case that all jobs have the same volume. They also extend their studies to the multiprocessor setting and present a non-constant factor approximation algorithm for general instances.

1.3 Organization of the Paper

Our paper is organized as follows. In section 2 we give a formal definition of the problem and establish a couple of preliminaries. In section 3 we present a QPTAS for laminar instances, and in section 4 we present a polynomial-time algorithm for instances with equal-volume jobs. Our FPTASs for purely-laminar instances and instances with a bounded number of different time windows are deferred to the appendix.

2 Preliminaries and Notations

The input is given by a set $\mathcal{J}$ of n jobs, each having its own release time r_j, deadline d_j, and volume $v_j > 0$. The power function of the speed-scalable processor is $P(s) = s^\alpha$, with $\alpha > 1$, and the energy consumption is power integrated over time. A schedule is called feasible if every job is executed entirely within its time window $[r_j, d_j)$. Preemption is not allowed, meaning that once a job is started, it must be executed entirely until its completion. Our goal is to find a feasible schedule of minimum total energy consumption.

We use $E(S)$ to denote the total energy consumed by a given schedule S, and $E(S, j)$ to denote the energy used for the processing of job j in schedule S. Furthermore, we use OPT to denote the energy consumption of an optimal schedule. A crucial observation is that, due to the convexity of the power function $P(s) = s^\alpha$, it is never beneficial to vary the speed during the execution of a job. This follows from Jensen's Inequality in the continuous version. We can therefore assume that in an optimal schedule, every job is processed using a uniform speed.

In the following, we restate a proposition from [6], which allows us to speed up certain jobs without paying too much additional energy cost. A proof for this proposition appears in the appendix.

Proposition 1. *Let S and S' be two feasible schedules that process j using uniform speeds s and $s' > s$, respectively. Then $E(S', j) = (s'/s)^{\alpha-1} \cdot E(S, j)$.*

As mentioned earlier, all our results rely on a discretization of the time axis, using a carefully chosen set of *grid points*. Given such a set of grid points, we define *grid point schedules* as follows.

Definition 2 (Grid Point Schedule)**.** A schedule is called *grid point schedule* if the processing of every job starts and ends at a grid point.

We use two different sets of grid points, $\mathcal{P}_{\text{approx}}$ and $\mathcal{P}_{\text{exact}}$, each of which guarantees the existence of a "good" grid point schedule. The first set, $\mathcal{P}_{\text{approx}}$, is more universal as it can applied to any kind of instances, losing

only a small factor in comparison with OPT. On the contrary, the set $\mathcal{P}_{\text{exact}}$ is specialized for the case of equal-volume jobs, and on such instances guarantees the existence of a grid point schedule with energy consumption exactly OPT. We now give a detailed description of both sets. For convenience, let $\gamma := 1 + \lceil 1/\epsilon \rceil$, where $\epsilon > 0$ is the error parameter of our approximation schemes.

Definition 3 (Grid Point Set $\mathcal{P}_{\text{approx}}$)**.** Let us call a time point t an *event* if $t = r_j$ or $t = d_j$ for some job j, and let $t_1 < t_2 < \ldots < t_p$ be the set of ordered events. Furthermore, let us call the interval between two consecutive events t_i and t_{i+1} a *zone*. The set $\mathcal{P}_{\text{approx}}$ is obtained in the following way. First, create a grid point at every event. Secondly, for every zone (t_i, t_{i+1}), create $n^2\gamma - 1$ equally spaced grid points that partition the zone into $n^2\gamma$ many subintervals of equal length $L_i = \frac{t_{i+1}-t_i}{n^2\gamma}$. Now $\mathcal{P}_{\text{approx}}$ is simply the union of all created grid points.

Note that the total number of grid points in $\mathcal{P}_{\text{approx}}$ is at most $\mathcal{O}\big(n^3(1 + \frac{1}{\epsilon})\big)$, as there are $\mathcal{O}\big(n\big)$ zones, for each of which we create $\mathcal{O}\big(n^2\gamma\big)$ grid points.

Lemma 4. *There exists a grid point schedule $\mathcal{G}$ with respect to $\mathcal{P}_{\text{approx}}$, such that $E(\mathcal{G}) \leq (1+\epsilon)^{\alpha-1}\text{OPT}$.*

The proof of this lemma can be found in the appendix.

Definition 5 (Grid Point Set $\mathcal{P}_{\text{exact}}$)**.** For every pair of events $t_i \leq t_j$, and for every $k \in \{1, \ldots, n\}$, create $k - 1$ equally spaced grid points that partition the interval $[t_i, t_j]$ into k subintervals of equal length. Furthermore, create a grid point at every event. The union of all these grid points defines the set $\mathcal{P}_{\text{exact}}$.

Clearly, the total number of grid points in $\mathcal{P}_{\text{exact}}$ is $\mathcal{O}\big(n^4\big)$. The following lemma is proven in the appendix.

Lemma 6. *If all jobs have the same volume $v_1 = v_2 = \ldots = v_n = v$, there exists a grid point schedule $\mathcal{G}$ with respect to $\mathcal{P}_{\text{exact}}$, such that $E(\mathcal{G}) = \text{OPT}$.*

3 Laminar Instances

In this section, we assume that the given problem instance $\mathcal{I}$ is laminar, and present a QPTAS for this setting. Whenever we use grid points in this section, we refer to the set $\mathcal{P}_{\text{approx}}$. The main idea of our QPTAS is to stepwise compute schedules for subsets of jobs within specific time intervals via dynamic programming. The internal layout of the dynamic program (DP) is based on the tree-like structure of the time windows in laminar instances. Here we draw on some ideas of Muratore et al. [23] from a different scheduling problem. The essence of our technique is the definition of a binary tree T, whose vertices represent time intervals and subsets of jobs that must be processed within these intervals. The inclusion of intervals is reflected in the father-son-relation of the tree. Our DP stepwise constructs near-optimal schedules for entire subtrees of T in a bottom-up manner, that is starting at the leaves and moving towards the root of the tree. To compute these entries, we use the fact that a job from a father node can be scheduled anywhere inside the intervals of its children. We can therefore split up the jobs in the root of a subtree recursively among its children, and thus intuitively delegate the work to a deeper level of the tree. There are two main technical difficulties of this approach. One is that a job from a father node could also be scheduled "between" its children, starting in the interval of child one, stretching over its boundary, and entering the interval of child two. We overcome this issue by taking care of such jobs separately, and additionally listing the truncated child-intervals in the dynamic programming tableau. The second difficulty is the huge number of possible job sets that a child node could receive from its parent. Reducing this number was one of the most challenging tasks in the development of our QPTAS, and requires a controlled and purposeful "omitting" of small jobs during the recursion. We complement this approach with a rounding of job volumes and a condensed representation of job sets in the DP tableau. At any point of time, we ensure that "omitted" jobs only cause a small increment of energy cost when being added to the final schedule. We now start to elaborate the details, beginning with the rounding

of the job volumes.

Definition 7 (Rounded Instance)**.** The *rounded instance* $\mathcal{I}'$ is obtained by rounding down every job volume v_j to the next smaller number of the form $v_{\min}(1+\epsilon)^i$, where $i \in \mathbb{N}_{\geq 0}$ and $v_{\min}$ is the smallest volume of any job in the original instance. The numbers $v_{\min}(1+\epsilon)^i$ are called *size classes*, and a job belongs to *size class* $\mathcal{C}_i$ if its rounded volume is $v_{\min}(1+\epsilon)^i$.

Lemma 8. *Every feasible schedule S' for $\mathcal{I}'$ can be transformed into a feasible schedule S for $\mathcal{I}$ with $E(S) \leq (1+\epsilon)^\alpha E(S')$.*

Proof. The lemma easily follows by using the same execution intervals as S' and speeding up accordingly. As rounded and original volume of a job differ by at most a factor of $1+\epsilon$, we need to increase the speed at any time t by at most this factor. Therefore the energy consumption grows by at most a factor of $(1+\epsilon)^\alpha$. □

From now on, we restrict our attention to the rounded instance $\mathcal{I}'$. We proceed with a formal definition of the tree T.

Definition 9 (Tree T)**.** For every interval $[t_i, t_{i+1})$ between two consecutive events t_i and t_{i+1}, we introduce a vertex v. Additionally, we introduce a vertex for every time window $[r_j, d_j)$, $j \in \mathcal{J}$ that is not represented by a vertex yet. If several jobs share the same allowed interval, we add only one single vertex for this interval. The interval corresponding to a vertex v is denoted by I_v. We also associate a (possibly empty) set of jobs J_v with each vertex v, namely the set of jobs j whose allowed interval $[r_j, d_j)$ is equal to I_v. Finally, we specify a distinguished root node r as follows. If there exists a vertex v with $I_v = [r^*, d^*)$, where r^* is the earliest release time and d^* the latest deadline of any job in $\mathcal{J}$, we set $r := v$. Otherwise, we introduce a new vertex r with $I_r := [r^*, d^*)$ and $J_r := \emptyset$. The edges of the tree are defined in the following way. A node u is the son of a node v if and only if $I_u \subset I_v$ and there is no other node w with $I_u \subset I_w \subset I_v$. As a last step, we convert T into a binary tree by repeating the following procedure as long as there exists a vertex v with more than two children. Let v_1 and v_2 be two "neighboring" sons of v, such that $I_{v_1} \cup I_{v_2}$ forms a contiguous interval. Now create a new vertex u with $I_u := I_{v_1} \cup I_{v_2}$ and $J_u := \emptyset$, and make u a new child of v, and the new parent of v_1 and v_2. This procedure eventually results in a binary tree T with $\mathcal{O}(n)$ vertices.

The main idea of our dynamic program is to stepwise compute schedules for subtrees of T, i.e. for the jobs associated with the vertices in the subtree (including its root), in a bottom-up manner. In order to allow a recursive

computation of these schedules, we do the following. When we consider a particular subtree T' with root r', we not only schedule the jobs in T', but also a given set of "inherited" jobs from the ancestors of r'. This enables us to recursively hand down jobs to children and deeper levels of the tree. However, we cannot enumerate all possible sets of heritable jobs, as this would burst the limits of our DP tableau. Instead, we use a lax representation of those sets via so-called *job vectors*, focussing only on a logarithmic number of size classes and ignoring jobs that are too small to be covered by any of these. To this end, let δ be the smallest integer such that $n/\epsilon \leq (1+\epsilon)^\delta$, and note that δ is $\mathcal{O}(\log n)$ for any fixed $\epsilon > 0$.

Definition 10 (Job Vector)**.** A *job vector* $\overrightarrow{\lambda}$ is a vector of $\delta + 1$ integers $\lambda_0, \ldots, \lambda_\delta$. The first component λ_0 specifies a size class, and we require $\lambda_0 \geq \delta - 1$. The remaining δ components specify a number of jobs between 0 and n for each of the size classes $\mathcal{C}_{\lambda_0}$, $\mathcal{C}_{\lambda_0-1}$, $\ldots$, $\mathcal{C}_{\lambda_0-\delta+1}$ in this order. We refer to the set of jobs described by a job vector $\overrightarrow{\lambda}$ as $J(\overrightarrow{\lambda})$.

Definition 11 (Heritable Job Vector)**.** A job vector $\overrightarrow{\lambda}$ is *heritable* to a vertex v of T if:

1. $J(\overrightarrow{\lambda})$ describes a subset of $\bigcup_{u \text{ ancestor of } v} J_u$, and
2. $\lambda_1 > 0$ or $\lambda_0 = \delta - 1$.

The conditions on a heritable job vector ensure that for a fixed vertex v, λ_0 can take only $\mathcal{O}(n)$ different values, as it must specify a size class that really occurs in the rounded instance, or be equal to $\delta - 1$. Therefore, in total, we can have at most $\mathcal{O}(n^{\delta+1})$ different job vectors that are heritable to a fixed vertex of the tree. In order to control the error caused by the laxity of our job set representation, we introduce the concept of δ-omitted schedules.

Definition 12 (δ-omitted Schedule)**.** Let J be a given set of jobs. A *δ-omitted schedule* for J is a feasible schedule for a subset $R \subseteq J$, s.t. for every job $j \in J \setminus R$, there exists a job $\text{big}(j) \in R$ with volume at least $v_j(1+\epsilon)^\delta$ that is scheduled entirely inside the allowed interval of j. The jobs in $J \setminus R$ are called *omitted jobs*, the ones in R *non-omitted jobs*.

Lemma 13. *Every δ-omitted schedule S' for a set of jobs J can be transformed into a feasible schedule S for all jobs in J, such that $E(S) \leq (1+\epsilon)^\alpha E(S')$.*

The proof of Lemma 13 can be found in the appendix. Essentially, this lemma ensures that representing the δ largest size classes of an inherited job

set is sufficient if we allow a small increment of energy cost. The smaller jobs can then be added safely to the final schedule in the end. We now turn to the central definition of the dynamic program.

Definition 14. All schedules in this definition are with respect to the rounded instance $\mathcal{I}'$. For any vertex v in the tree T, any job vector $\overrightarrow{\lambda}$ that is heritable to v, and any pair of grid points $g_1 \leq g_2$ with $[g_1, g_2) \subseteq I_v$, let $G(v, \overrightarrow{\lambda}, g_1, g_2)$ denote a minimum cost grid point schedule for the jobs in the subtree of v (including v itself) plus the jobs $J(\overrightarrow{\lambda})$ (these are allowed to be scheduled anywhere inside $[g_1, g_2)$) that uses only the interval $[g_1, g_2)$. Furthermore, let $S(v, \overrightarrow{\lambda}, g_1, g_2)$ be a δ-omitted schedule for the same set of jobs in the same interval $[g_1, g_2)$, satisfying $E\big(S(v, \overrightarrow{\lambda}, g_1, g_2)\big) \leq E\big(G(v, \overrightarrow{\lambda}, g_1, g_2)\big)$.

Dynamic Program. Our dynamic program computes the schedules $S(v, \overrightarrow{\lambda}, g_1, g_2)$. For ease of exposition, we focus only on computing the energy consumption values $E(v, \overrightarrow{\lambda}, g_1, g_2) := E\big(S(v, \overrightarrow{\lambda}, g_1, g_2)\big)$, and omit the straightforward bookkeeping of the corresponding schedules. The base cases are the leaves of T. For a particular leaf node ℓ, we set

$$E(\ell, \overrightarrow{\lambda}, g_1, g_2) := \begin{cases} 0 & \text{if } J_\ell \cup J(\overrightarrow{\lambda}) = \emptyset \\ \frac{V^\alpha}{(g_2 - g_1)^{\alpha-1}} & \text{otherwise,} \end{cases}$$

where V is the total volume of all jobs in $J_\ell \cup J(\overrightarrow{\lambda})$. This corresponds to executing $J_\ell \cup J(\overrightarrow{\lambda})$ at uniform speed using the whole interval $[g_1, g_2)$. The resulting schedule is feasible, as no release times or deadlines occur in the interior of I_ℓ. Furthermore, it is also optimal by the convexity of the power function. Thus $E(\ell, \overrightarrow{\lambda}, g_1, g_2) \leq E\big(G(\ell, \overrightarrow{\lambda}, g_1, g_2)\big)$.

When all leaves have been handled, we move on to the next level, i.e. the parents of the leaves. For this and also the following levels up to the root r, we compute the values $E(v, \overrightarrow{\lambda}, g_1, g_2)$ recursively, using the procedure Compute in Figure 3.1. An intuitive description of the procedure is given below.

Our first step is to iterate through all possible options for a potential "crossing" job j, whose execution interval $[\tilde{g_1}, \tilde{g_2})$ stretches from child v_1 into the interval of child v_2. For every possible choice, we combine the optimal energy cost E for this job (obtained by using a uniform execution speed) with the best possible way to split up the remaining jobs between the truncated intervals of v_1 and v_2. Here we consider only the δ largest size classes of the remaining jobs $\tilde{J}$, and omit the smaller jobs. This omitting happens during the construction of a vector representation for $\tilde{J}$ using the procedure Vector. Finally, we also try the option that no "crossing" job exists and all

COMPUTE $(v, \overrightarrow{\lambda}, g_1, g_2)$:

Let v_1 and v_2 be the children of v, such that I_{v_1} is the earlier of the intervals I_{v_1}, I_{v_2}. Furthermore, let g be the grid point at which I_{v_1} ends and I_{v_2} starts.

Initialize MIN $:= \infty$.

For all gridpoints $\tilde{g}_1, \tilde{g}_2$, s.t. $g_1 \leq \tilde{g}_1 < g < \tilde{g}_2 \leq g_2$, and all jobs $j \in J_v \cup J(\overrightarrow{\lambda})$, **do:**

$E := \frac{{v_j}^\alpha}{(\tilde{g}_2 - \tilde{g}_1)^{\alpha-1}}$.

$\tilde{J} := \big(J_v \cup J(\overrightarrow{\lambda})\big) \setminus \{j\}$.

$\overrightarrow{\gamma} :=$ VECTOR $(\tilde{J})$.

MIN $:= \min\big\{$MIN,

$\min\{E + E(v_1, \overrightarrow{\gamma_1}, g_1, \tilde{g}_1) + E(v_2, \overrightarrow{\gamma_2}, \tilde{g}_2, g_2) \ : \ J(\overrightarrow{\gamma_1}) \cup J(\overrightarrow{\gamma_2}) = J(\overrightarrow{\gamma})\}\big\}$.

$\tilde{J} := J_v \cup J(\overrightarrow{\lambda})$.

$\overrightarrow{\gamma} :=$ VECTOR $(\tilde{J})$.

$a_1 := \min\{g_1, g\}$; $a_2 := \min\{g_2, g\}$; $b_1 := \max\{g_1, g\}$; $b_2 := \max\{g_2, g\}$.

$E(v, \overrightarrow{\lambda}, g_1, g_2) := \min\big\{$MIN,

$\min\{E(v_1, \overrightarrow{\gamma_1}, a_1, a_2) + E(v_2, \overrightarrow{\gamma_2}, b_1, b_2) : J(\overrightarrow{\gamma_1}) \cup J(\overrightarrow{\gamma_2}) = J(\overrightarrow{\gamma})\}\big\}$.

VECTOR $(\tilde{J})$:

Let $\mathcal{C}_\ell$ be the largest size class of any job in $\tilde{J}$.

$i := \max\{\ell, \delta - 1\}$.

For $k := i - \delta + 1, \ldots, i$ **do:** $x_k := |\{p \in \tilde{J} : p$ belongs to size class $\mathcal{C}_k\}|$.

Return $(i, x_i, x_{i-1}, \ldots, x_{i-\delta+1})$.

Figure 3.1: Procedure for computing the remaining entries of the DP.

jobs are split up between v_1 and v_2. In this case we need to take special care of the subproblem boundaries, as $g_1 > g$ or $g_2 < g$ are also valid arguments for COMPUTE.

Lemma 15. *The schedules $S(v, \overrightarrow{\lambda}, g_1, g_2)$ constructed by the above dynamic program are δ-omitted schedules for the jobs in the subtree of v plus the jobs*

$J(\overrightarrow{\lambda})$. *Furthermore, they satisfy* $E\big(S(v, \overrightarrow{\lambda}, g_1, g_2)\big) \leq E\big(G(v, \overrightarrow{\lambda}, g_1, g_2)\big)$.

A proof for this lemma is given in the appendix. We can now combine Lemmas 4, 8, 13, and 15 to obtain the following theorem.

Theorem 16. *The non-preemptive speed scaling problem admits a QPTAS if the instance is laminar.*

Proof. Let r^* be the earliest release time, and d^* be the latest deadline of any job in $\mathcal{J}$. Furthermore, let r be the root of the tree T, and let $\overrightarrow{0}$ denote the (heritable) job vector representing the empty set, i.e. $\overrightarrow{0} := (\delta - 1, 0, \ldots, 0)$. We consider the schedule $S(r, \overrightarrow{0}, r^*, d^*)$, which is a δ-omitted schedule for the rounded instance by Lemma 15, and turn it into a feasible schedule S_r for the whole set of (rounded) jobs, using Lemma 13. Finally, we apply Lemma 8 to turn S_r into a feasible schedule S for the original instance $\mathcal{I}$, and obtain

$$\begin{aligned} E(S) &\leq (1+\epsilon)^{\alpha} E(S_r) \leq (1+\epsilon)^{2\alpha} E\big(S(r, \overrightarrow{0}, r^*, d^*)\big) \leq (1+\epsilon)^{2\alpha} E\big(G(r, \overrightarrow{0}, r^*, d^*)\big) \\ &\leq (1+\epsilon)^{3\alpha-1}\mathrm{OPT} = \big(1 + \mathcal{O}(\epsilon)\big)\mathrm{OPT}. \end{aligned}$$

Here the third inequality holds by Lemma 15, and the fourth inequality follows from Lemma 4 and the fact that $G(r, \overrightarrow{0}, r^*, d^*)$ is an optimal grid point schedule for the rounded instance (with smaller job volumes). The quasipolynomial running time of the algorithm is easily verified, as we have only a polynomial number of grid points, and at most a quasipolynomial number of job vectors that are heritable to any vertex of the tree. □

4 Equal-Volume Jobs

In this section, we consider the case that all jobs have the same volume $v_1 = v_2 = \ldots = v_n = v$. We present a dynamic program that computes an (exactly) optimal schedule for this setting in polynomial time. All grid points used for this purpose relate to the set $\mathcal{P}_{\text{exact}}$.

As a first step, let us order the jobs such that $r_1 \leq r_2 \leq \ldots \leq r_n$. Furthermore, let us define an ordering on schedules as follows.

Definition 17 (Completion Time Vector)**.** Let $C_1, \ldots, C_n$ be the completion times of the jobs $j_1, \ldots, j_n$ in a given schedule S. The vector $\overrightarrow{S} := (C_1, \ldots, C_n)$ is called the *completion time vector* of S.

Definition 18 (Lexicographic Ordering)**.** A schedule S is said to be *lexicographically smaller* than a schedule S' if the first component in which their completion time vectors differ is smaller in $\overrightarrow{S}$ than in $\overrightarrow{S'}$.

We now elaborate the details of the DP, focusing on energy consumption values only.

Definition 19. Let $i \in \{1, \ldots, n\}$ be a job index, and let g_1, g_2, and g_3 be grid points satisfying $g_1 \leq g_2 \leq g_3$. We define $E(i, g_1, g_2, g_3)$ to be the minimum energy consumption of a grid point schedule for the jobs $\{j_k \in \mathcal{J} : k \geq i \ \wedge \ g_1 < d_k \leq g_3\}$ that uses only the interval $[g_1, g_2)$.

Dynamic Program. Our goal is to compute the values $E(i, g_1, g_2, g_3)$. To this end, we let

$$E(i, g_1, g_2, g_3) := \begin{cases} 0 & \text{if } \{j_k \in \mathcal{J} : k \geq i \ \wedge \ g_1 < d_k \leq g_3\} = \emptyset \\ \infty & \text{if } \exists k \geq i : g_1 < d_k \leq g_3 \ \wedge \ [r_k, d_k) \cap [g_1, g_2) = \emptyset. \end{cases}$$

Note that if $g_1 = g_2$, one of the above cases must apply. We now recursively compute the remaining values, starting with the case that g_1 and g_2 are consecutive grid points, and stepwise moving towards cases with more and

more grid points in between g_1 and g_2. The recursion works as follows. Let $E(i, g_1, g_2, g_3)$ be the value we want to compute, and let j_q be the smallest index job in $\{j_k \in \mathcal{J} : k \geq i \ \wedge \ g_1 < d_k \leq g_3\}$. Furthermore, let $\mathcal{G}$ denote a lexicographically smallest optimal grid point schedule for the jobs $\{j_k \in \mathcal{J} : k \geq i \ \wedge \ g_1 < d_k \leq g_3\}$, using only the interval $[g_1, g_2)$. Our first step is to "guess" the grid points b_q and e_q that mark the beginning and end of j_q's execution interval in $\mathcal{G}$, by minimizing over all possible options. We then use the crucial observation that in $\mathcal{G}$, all jobs $J^- := \{j_k \in \mathcal{J} : k \geq q+1 \ \wedge \ g_1 < d_k \leq e_q\}$ are processed completely before j_q, and all jobs $J^+ := \{j_k \in \mathcal{J} : k \geq q+1 \ \wedge \ e_q < d_k \leq g_3\}$ are processed completely after j_q. For J^- this is obviously the case because of the deadline constraint. For J^+ this holds as all these jobs have release time at least r_q by the ordering of the jobs, and deadline greater than e_q by definition of J^+. Therefore any job in J^+ that is processed before j_q could be swapped with j_q, resulting in a lexicographic smaller schedule; a contradiction. Hence, we can use the following recursion to compute $E(i, g_1, g_2, g_3)$.

$$E(i, g_1, g_2, g_3) := \min \Big\{ \frac{{v_q}^\alpha}{(e_q - b_q)^{\alpha-1}} + E(q+1, g_1, b_q, e_q) + E(q+1, e_q, g_2, g_3) : \\ (g_1 \leq b_q < e_q \leq g_2) \wedge (b_q \geq r_q) \wedge (e_q \leq d_q) \Big\}.$$

Once we have computed all values, we output the schedule S corresponding to $E(1, r^*, d^*, d^*)$, where r^* is the earliest release time and d^* the latest deadline of any job in $\mathcal{J}$. Lemma 6 implies that $E(S) = \mathrm{OPT}$, and the running time of the algorithm is clearly polynomial as there are at most $\mathcal{O}(n^4)$ grid points in $\mathcal{P}_{\text{exact}}$. Hence, we obtain the following theorem.

Theorem 20. *The non-preemptive speed scaling problem admits a polynomial time algorithm if all jobs have the same volume.*

5 Conclusion

In this paper, we made a first step to narrow down the complexity of the non-preemptive speed scaling problem. For most of the studied cases our results are optimal, unless P = NP. The only exception are laminar instances, where our QPTAS strongly indicates that in fact a polynomial-time approximation scheme should be possible. This is an obvious direction of future research and should be the next step in order to settle the precise approximability of the non-preemptive speed scaling problem. We hope that our paper will initiate an enhanced interest in this problem, which is very fundamental for a deeper understanding of speed scaling in general.

Bibliography

[1] Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, pages 1266–1285. SIAM, 2012.

[2] Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration: extended abstract. In *SPAA*, pages 279–288. ACM, 2011.

[3] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 621–633. Springer, 2006.

[4] Susanne Albers, Fabian Müller, and Swen Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298. ACM, 2007.

[5] Eric Angel, Evripidis Bampis, Fadi Kacem, and Dimitrios Letsios. Speed scaling on parallel processors with migration. In *Euro-Par*, volume 7484 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2012.

[6] Antonios Antoniadis and Chien-Chung Huang. Non-preemptive speed scaling. In *SWAT*, volume 7357 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2012.

[7] Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. All-norm approximation algorithms. *J. Algorithms*, 52(2):120–133, 2004.

[8] Evripidis Bampis, Alexander Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Ioannis Nemparis. From preemptive to non-preemptive speed-scaling scheduling. In *COCOON*, volume 7936 of *Lecture Notes in Computer Science*, pages 134–146. Springer, 2013.

[9] Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 2008.

[10] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701. SIAM, 2009.

[11] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2009.

[12] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS*, pages 520–529. IEEE Computer Society, 2004.

[13] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. In *SODA*, pages 805–813. SIAM, 2007.

[14] Brad D. Bingham and Mark R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *ISPA*, pages 153–161. IEEE, 2008.

[15] David Brooks, Pradip Bose, Stanley Schuster, Hans M. Jacobson, Prabhakar Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor V. Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[16] Ho-Leung Chan, Joseph Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Optimizing throughput and energy in online deadline scheduling. *ACM Transactions on Algorithms*, 6(1), 2009.

[17] Jian-Jia Chen, Tei-Wei Kuo, and Hsueh-I Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *WADS*, volume 3608 of *Lecture Notes in Computer Science*, pages 338–349. Springer, 2005.

[18] Xin Han, Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Deadline scheduling and power management for speed bounded processors. *Theor. Comput. Sci.*, 411(40-42):3587–3600, 2010.

[19] Sandy Irani, Sandeep K. Shukla, and Rajesh K. Gupta. Algorithms for power savings. In *SODA*, pages 37–46. ACM/SIAM, 2003.

[20] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the Twelfth IEEE Real-Time Systems Symposium*, pages 129–139, 1991.

[21] Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. In *COCOON*, volume 3595 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2005.

[22] Minming Li and Frances F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. In *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

[23] Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Oper. Res. Lett.*, 38(1):47–50, 2010.

[24] Adam Wierman, Lachlan L. H. Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Perform. Eval.*, 69(12):601–622, 2012.

[25] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *FOCS*, pages 374–382. IEEE Computer Society, 1995.

Appendix A Proof of Proposition 1

Proposition 1. *Let S and S' be two feasible schedules that process j using uniform speeds s and $s' > s$, respectively. Then $E(S', j) = (s'/s)^{\alpha-1} \cdot E(S, j)$.*

Proof.

$$E(S', j) = P(s')\frac{v_j}{s'} = (s')^{\alpha-1}v_j = (\frac{s'}{s})^{\alpha-1}s^{\alpha-1}v_j$$
$$= (\frac{s'}{s})^{\alpha-1}P(s)\frac{v_j}{s} = (\frac{s'}{s})^{\alpha-1}E(S, j).$$

□

Appendix B Proof of Lemma 4

Lemma 4. *There exists a grid point schedule $\mathcal{G}$ with respect to $\mathcal{P}_{\text{approx}}$, such that $E(\mathcal{G}) \leq (1+\epsilon)^{\alpha-1}\text{OPT}$.*

Proof. Let $\mathcal{S}^*$ be an optimal schedule, that is $E(\mathcal{S}^*) = \text{OPT}$. We show how to modify $\mathcal{S}^*$ by shifting and compressing certain jobs, s.t. every execution interval starts and ends at a grid point. For the proof we focus on one particular zone (t_i, t_{i+1}), and the lemma follows by applying the transformation to each other zone individually.

Let us consider the jobs that $\mathcal{S}^*$ processes within the zone (t_i, t_{i+1}). If a job's execution interval overlaps partially with this zone, we consider only its fraction inside (t_i, t_{i+1}) and treat this fraction as if it were a job by itself. We denote the set of (complete and partial) jobs in zone (t_i, t_{i+1}) by J. If $J = \emptyset$, nothing needs to be done. Otherwise, we can assume that $\mathcal{S}^*$ uses the entire zone (t_i, t_{i+1}) without any idle periods to process the jobs in J. If this were not the case, we could slow down the processing of any job in J without violating a release time or deadline constraint, and thus obtain a feasible schedule with lower energy cost than $\mathcal{S}^*$, a contradiction. Consequently, the total time for processing J in $\mathcal{S}^*$ is $\gamma n^2 L_i$ (recall that $L_i = \frac{t_{i+1}-t_i}{n^2\gamma}$), and as $|J| \leq n$, there must exist a job $j \in J$ with execution time $T_j \geq \gamma n L_i$.

We now partition the jobs in $J \setminus j$ into J^+, the jobs processed after j, and J^-, the jobs processed before j. First, we restrict our attention to J^+. Let $q_1, \ldots, q_{|J^+|}$ denote the jobs in J^+ in the order they are processed by $\mathcal{S}^*$. Starting with the last job $q_{|J^+|}$, and going down to q_1, we modify the schedule as follows. We keep the end of $q_{|J^+|}$'s execution interval fixed, and shift its start to the next earlier grid point, reducing its uniform execution speed accordingly. At the same time, to not produce any overlappings, we shift the execution intervals of all $q_k,\ k < |J^+|$ by the same amount, in the direction of earlier times (leaving their lengths unchanged). Eventually, we

also move the execution end point of j by the same amount towards earlier times (leaving its start point fixed). This shortens the execution interval of j and "absorbs" the shifting of the jobs in J^+. The shortening of j's execution interval is compensated by an appropriate increase of speed. We then proceed with $q_{|J^+|-1}$, keeping its end (which now already resides at a grid point) fixed, and moving its start to the next earlier grid point. Again, the shift propagates to earlier jobs in J^+, which are moved by the same amount, and shortens j's execution interval once more. When all jobs in J^+ have been modified in this way, we turn to J^- and apply the same procedure there. This time, we keep the start times fixed and instead shift the right end points of the execution intervals towards later times. As before, j "absorbs" the propagated shifts, as we increase its start time accordingly. After this modification, the execution intervals of all jobs in J start and end at grid points only.

To complete the proof, we need to analyze the changes made in terms of energy consumption. Let $\mathcal{G}$ denote the schedule obtained by the above modification of $\mathcal{S}^*$. Obviously, for all $j' \in J \setminus j$, we have that $E(\mathcal{G}, j') \leq E(\mathcal{S}^*, j')$, as the execution intervals of those jobs are only prolonged during the transformation process, resulting in a less or equal execution speed. The only job whose processing time is possibly shortened, is j. Since $|J| \leq n$, it can be shortened at most n times, each time by a length of at most L_i. Remember that the execution time of j in $\mathcal{S}^*$ was $T_j \geq \gamma n L_i$. Therefore, in $\mathcal{G}$, its execution time is at least $T_j - nL_i \geq T_j - T_j/\gamma$. Thus the speedup factor of j in $\mathcal{G}$ compared to $\mathcal{S}^*$ is at most

$$\frac{T_j}{T_j - \frac{T_j}{\gamma}} = \frac{1}{1 - \frac{1}{\gamma}} \leq 1 + \epsilon,$$

where the last inequality follows from the definition of γ. Hence, Proposition 1 implies that $E(\mathcal{G}, j) \leq (1+\epsilon)^{\alpha-1} E(\mathcal{S}^*, j)$, and the lemma follows by summing up the energy consumptions of the individual jobs. □

Appendix C Proof of Lemma 6

Lemma 6. *If all jobs have the same volume $v_1 = v_2 = \ldots = v_n = v$, there exists a grid point schedule $\mathcal{G}$ with respect to $\mathcal{P}_{\text{exact}}$, such that $E(\mathcal{G}) = \text{OPT}$.*

Proof. Let $\mathcal{S}^*$ be an optimal schedule. W.l.o.g., we can assume that $\mathcal{S}^*$ changes the processing speed only at events (recall that an event is either a release time or a deadline of some job), as a constant average speed between any two consecutive events minimizes the energy consumption (this follows from Jensen's Inequality) without violating release time or deadline constraints. Given this property, we will show that $\mathcal{S}^*$ is in fact a grid point schedule with respect to $\mathcal{P}_{\text{exact}}$. To this end, we partition the time horizon of $\mathcal{S}^*$ into *phases* of constant speed, that is time intervals of maximal length during which the processing speed is unchanged. As every job itself is processed using a uniform speed, no job is processed only partially within a phase. Each phase is therefore characterized by a pair of events $t_i \leq t_j$ indicating its beginning and end, and a number x of jobs that are processed completely between t_i and t_j at constant speed. It is clear that the grid points created for the pair (t_i, t_j) and $k := x$ in the definition of $\mathcal{P}_{\text{exact}}$ correspond exactly to the start and end times of the jobs in this phase. Since this is true for every phase, $\mathcal{S}^*$ is indeed a grid point schedule. □

Appendix D Proof of Lemma 13

Lemma 13. *Every δ-omitted schedule S' for a set of jobs J can be transformed into a feasible schedule S for all jobs in J, such that $E(S) \leq (1+\epsilon)^\alpha E(S')$.*

Proof. Let R be the set of non-omitted jobs in S'. W.l.o.g., we can assume that S' executes each job in R at a uniform speed, as this minimizes the energy consumption. For every $j \in R$, define $\text{SMALL}(j) := \{x \in J \setminus R : \text{big}(x) = j\}$. Note that every omitted job occurs in exactly one of the sets $\text{SMALL}(j),\ j \in R$. The schedule S is constructed as follows. For all $j \in R$, we process the jobs $\{j\} \cup \text{SMALL}(j)$ using the execution interval of j in S' and a uniform speed. The processing order may be chosen arbitrarily. Clearly, the resulting schedule is feasible by the definition of $\text{big}(x)$. In order to finish the total volume V_j of the jobs $\{j\} \cup \text{SMALL}(j)$ within the interval of j in S', we need to raise the speed in this interval by the factor V_j/v_j. As $|\text{SMALL}(j)| \leq n$, and $v_x \leq v_j(1+\epsilon)^{-\delta}$ for all $x \in \text{SMALL}(j)$, we have that

$$V_j \leq v_j + nv_j(1+\epsilon)^{-\delta} \leq v_j + nv_j\frac{\epsilon}{n} \leq (1+\epsilon)v_j,$$

where the second inequality follows from the definition of δ. For the speedup factor, we therefore obtain $V_j/v_j \leq 1+\epsilon$. Hence, the energy consumption grows by at most the factor $(1+\epsilon)^\alpha$. □

Appendix E Proof of Lemma 15

Lemma 15. *The schedules $S(v, \overrightarrow{\lambda}, g_1, g_2)$ constructed by the above dynamic program are δ-omitted schedules for the jobs in the subtree of v plus the jobs $J(\overrightarrow{\lambda})$. Furthermore, they satisfy $E\big(S(v, \overrightarrow{\lambda}, g_1, g_2)\big) \leq E\big(G(v, \overrightarrow{\lambda}, g_1, g_2)\big)$.*

Proof. We prove the lemma by induction. In the base cases, i.e. at the leaves of T, we already argued that the schedules are feasible and optimal. Since no jobs are omitted at all, the lemma is obviously true at this level. We now perform the induction step. To this end, let us consider a fixed schedule $S(v, \overrightarrow{\lambda}, g_1, g_2)$, and let us assume the lemma is true for the children v_1 and v_2 of v. We first show that $S(v, \overrightarrow{\lambda}, g_1, g_2)$ is indeed a δ-omitted schedule. The only point where jobs are omitted in the recursive procedure is the call of VECTOR $(\tilde{J})$, where a vector-representation $\overrightarrow{\gamma}$ of $\tilde{J}$ is constructed. This vector $\overrightarrow{\gamma}$ only represents a subset of the jobs $\tilde{J}$, namely the jobs in the δ largest size classes of $\tilde{J}$. Let $j_{\max}$ denote a job in $\tilde{J}$ with maximum volume, i.e. a job belonging to the largest size class. Then every omitted job j_{om} has volume at most $v_{j_{\max}}(1+\epsilon)^{-\delta}$, and we can choose $\text{big}(j_{om}) := j_{\max}$ to satisfy the requirements of Definition 12, as long as $j_{\max}$ is indeed contained in one of the subschedules that we combine in the recursion step. If, however, $j_{\max}$ is omitted in the corresponding subschedule, then there exists a job $\text{big}(j_{\max})$ as required in Definition 12, by induction hypothesis. In this case we can choose $\text{big}(j_{om}) := \text{big}(j_{\max})$. This proves that $S(v, \overrightarrow{\lambda}, g_1, g_2)$ is indeed a δ-omitted schedule.

We now argue about the energy consumption. Let J_1 and J_2 denote the subsets of jobs that $G(v, \overrightarrow{\lambda}, g_1, g_2)$ processes entirely within I_{v_1} and I_{v_2}, respectively. If there exists a "crossing" job spanning from I_{v_1} into I_{v_2}, we denote this job by j_c. Now we look at the iteration that handles exactly this situation, i.e. when $j = j_c$ and $\tilde{g}_1, \tilde{g}_2$ mark the beginning and end of j_c's execution interval in $G(v, \overrightarrow{\lambda}, g_1, g_2)$, or the passage after the for-loop for

the case without "crossing" job. As mentioned earlier, the procedure possibly omits certain jobs and only splits up a subset of $J_v \cup J(\overrightarrow{\lambda})$ between the children v_1 and v_2. Here, all possible splits are tried. One option for the min-operation is therefore to combine the subschedules that process the non-omitted subset of J_1 within I_{v_1}, and the non-omitted subset of J_2 within I_{v_2}. By induction hypothesis, and since we only schedule subsets of J_1 and J_2, the energy consumption of these subschedules is at most the energy spent by $G(v, \overrightarrow{\lambda}, g_1, g_2)$ for executing J_1 and J_2, respectively. Furthermore, if there exists a "crossing" job j_c, then executing this job from $\tilde{g_1}$ to $\tilde{g_2}$ at uniform speed costs at most the energy that $G(v, \overrightarrow{\lambda}, g_1, g_2)$ pays for this job. Summing up the different parts, we get that the considered option has an energy consumption of at most $E\big(G(v, \overrightarrow{\lambda}, g_1, g_2)\big)$. The lemma follows as we choose the minimum over all possible options. □

Appendix F Purely-Laminar Instances

In this section, we present an FPTAS for purely-laminar instances $\mathcal{I}$. W.l.o.g., we assume that the jobs are ordered by inclusion of their time windows, that is $[r_1, d_1) \subseteq [r_2, d_2) \subseteq \cdots \subseteq [r_n, d_n)$. Furthermore, whenever we refer to grid points in this section, we refer to the set $\mathcal{P}_{\text{approx}}$. Our FPTAS uses dynamic programming to construct an optimal grid point schedule for $\mathcal{I}$, satisfying the following structural property:

Property 21. *For any $k > 1$, jobs $j_1, \ldots, j_{k-1}$ are either all processed before j_k, or all processed after j_k.*

This structure can easily be established in any schedule for $\mathcal{I}$ by performing a sequence of energy-preserving swaps. According to this, the following lemma is a straightforward extension of Lemma 4 to the purely-laminar case.

Lemma 22. *If the problem instance is purely-laminar, there exists a grid point schedule $\mathcal{G}$ with respect to $\mathcal{P}_{\text{approx}}$ that satisfies Property 21 and has energy cost $E(\mathcal{G}) \leq (1+\epsilon)^{\alpha-1}\text{OPT}$.*

Proof. Consider an optimal schedule $\mathcal{S}^*$ for $\mathcal{I}$, and let J^- and J^+ be the jobs executed before and after j_1, respectively. Now rearrange the execution intervals (without changing their lengths) of the jobs in J^+ into *smallest index first* order (SIF), by repeatedly swapping two consecutively processed jobs j_a preceding j_b, with $a > b$. For the swap, we let the execution interval of j_b now start at j_a's original starting time, and directly append j_a's execution interval once j_b is finished. Note that each such swap maintains feasibility, as no release times occurs during the execution of the jobs in J^+, and $a > b$ implies $d_a \geq d_b$. Similarly, we rearrange the execution intervals of the jobs in J^- into *largest index first* order (LIF), and denote the resulting schedule by S'. Clearly, $E(S') = \text{OPT}$, as the rearrangements preserve the energy cost of every individual job. Furthermore, S' satisfies Property 21. To see

this, let us fix $k > 1$ and distinguish whether j_k is in J^- or in J^+. In the first case, when $j_k \in J^-$, all $j \in J^+$ are scheduled after j_k by definition of J^-/J^+, and all $j_i \in J^-, i < k$ are scheduled after j_k by the LIF-order. In the second case, when $j_k \in J^+$, all $j \in J^-$ are scheduled before j_k by definition of J^-/J^+, and all $j_i \in J^+, i < k$ are scheduled before j_k by the SIF-order. As a final step, we now apply the transformation from the proof of Lemma 4 to S'. Since this transformation does not change the order of any jobs, the resulting grid point schedule $\mathcal{G}$ still satisfies Property 21, and has energy cost $E(\mathcal{G}) \leq (1+\epsilon)^{\alpha-1}\mathrm{OPT}$. □

Dynamic Program. For any $k \leq n$ and grid points $g_1 \leq g_2$, let $S(k, g_1, g_2)$ denote a minimum cost grid point schedule for $j_1, \ldots, j_k$ that satisfies Property 21 and uses only the time interval between g_1 and g_2. The corresponding energy cost of $S(k, g_1, g_2)$ is denoted by $E(k, g_1, g_2)$, where $E(k, g_1, g_2) := \infty$ if no such schedule exists. For ease of exposition, we only show how to compute the energy consumption values $E(k, g_1, g_2)$, and omit the straightforward bookkeeping of the corresponding schedules. The base cases are given by $E(0, g_1, g_2) = 0$, for all $g_1 \leq g_2$. All remaining entries can be computed with the following recursion.

$$E(k{+}1, g_1, g_2) = \begin{cases} \infty & \text{if } \ (g_1 = g_2) \vee (g_1 \geq d_{k+1}) \vee (g_2 \leq r_{k+1}). \\ \min \Big\{ \frac{{v_{k+1}}^\alpha}{(g_2' - g_1')^{\alpha-1}} + \min\{E(k, g_1, g_1'), E(k, g_2', g_2)\} : & \\ \qquad (g_1 \leq g_1' < g_2' \leq g_2) \wedge (g_1' \geq r_{k+1}) \wedge (g_2' \leq d_{k+1}) \Big\} & \text{otherwise.} \end{cases}$$

Intuitively, we minimize over all possible combinations of grid points g_1' and g_2' that could mark the beginning and end of j_{k+1}'s execution. For fixed g_1' and g_2', it is best to process j_{k+1} at uniform speed, resulting in the energy cost ${v_{k+1}}^\alpha/(g_2' - g_1')^{\alpha-1}$ for this job. The remaining jobs $j_1, \ldots, j_k$ must then be scheduled either before or after j_{k+1}, to satisfy Property 21. This fact is captured in the second min-operation of the formula. The constraints on g_1' and g_2' ensure that j_{k+1} can be feasibly scheduled in the chosen interval.

Once we have computed all values $E(k, g_1, g_2)$ (and their corresponding schedules), we output the schedule $\tilde{S} := S(n, r^*, d^*)$, where r^* is the earliest release time and d^* the latest deadline of any job in $\mathcal{I}$. Note that $\tilde{S}$ is an optimal grid point schedule with Property 21 for $\mathcal{I}$. Hence, Lemma 22 implies that $E(\tilde{S}) \leq (1+\epsilon)^{\alpha-1}\mathrm{OPT} = \big(1 + \mathcal{O}(\epsilon)\big)\mathrm{OPT}$. Finally, it is easy to verify that the running time of the algorithm is polynomial in n and $1/\epsilon$, since the total number of grid points in $\mathcal{P}_{\text{approx}}$ is $\mathcal{O}\big(n^3(1 + \frac{1}{\epsilon})\big)$. We therefore obtain the following theorem.

Theorem 23. *The non-preemptive speed scaling problem admits an FPTAS if the instance is purely-laminar.*

Appendix G Bounded Number of Time Windows

Let us consider a problem instance $\mathcal{I}$, and group together jobs that share the same time window. We refer to the group of jobs with time window $[r, d)$ as the *type* T_{rd}.

Theorem 24. *The non-preemptive speed scaling problem admits an FPTAS if the total number of types is at most a constant c.*

Our FPTAS draws on ideas of Antoniadis and Huang [6], as we transform the problem into an instance $\mathcal{I}'$ of *unrelated machine scheduling* with ℓ_α-norm objective. In this problem, one is given a set of machines $\mathcal{M}$, a set of jobs $\mathcal{J}$, and numbers p_{ij} that specify the processing time of job j on machine i. The goal is to find an assignment A of the jobs to the machines that minimizes $\text{Cost}(A) = (\sum_{i \in \mathcal{M}} (\sum_{j:A(j)=i} p_{ij})^\alpha)^{1/\alpha}$. In general, this problem is APX-hard [7]. Our instance, however, will have only a constant number of machines, and for this special case an FPTAS exists [7].

The transformation works as follows. Let $\mathcal{G}$ be an optimal grid point schedule with respect to $\mathcal{P}_{\text{approx}}$, and for each type T_{rd} let $b(T_{rd})$ and $e(T_{rd})$ denote the grid points at which $\mathcal{G}$ starts to process the first job of T_{rd} and finishes the last job of T_{rd}, respectively. Our first step is to "guess" the entire set of grid points $b(\cdot)$ and $e(\cdot)$, by minimizing over all possible options with $r \leq b(T_{rd}) < e(T_{rd}) \leq d$ for every type T_{rd}. Note that the total number of choices that we have to make is at most $\mathcal{O}\big(n^{6c}(1+\frac{1}{\epsilon})^{2c}\big)$, and thus polynomial in both n and $1/\epsilon$. For one particular guess, let $g_1 < g_2 < \ldots < g_k$ be the ordered set of distinct grid points $b(\cdot)$ and $e(\cdot)$. The instance $\mathcal{I}'$ has a machine i for every interval $[g_i, g_{i+1})$, and a job j for every job of the original instance. The processing times p_{ij} are given as $p_{ij} := \frac{v_j}{(g_{i+1}-g_i)^{1-1/\alpha}}$ if $[g_i, g_{i+1}) \subseteq [r_j, d_j)$, and $p_{ij} := \infty$ otherwise.

Note that the total number of machines in $\mathcal{I}'$ is $k - 1 < 2c$. Hence, the FPTAS of [7] can be applied to obtain an assignment A with $\text{Cost}(A) \leq$

$(1+\epsilon)\mathrm{OPT}'$, where OPT' denotes the cost of an optimal assignment for $\mathcal{I}'$. The following two lemmas imply Theorem 24.

Lemma 25. *Every finite-cost assignment A for $\mathcal{I}'$ can be transformed into a schedule S for $\mathcal{I}$, such that $E(S) = \big(\mathit{Cost}(A)\big)^{\alpha}$.*

Proof. For any $i \in \mathcal{M}$, let A_i denote the set of jobs that A assigns to machine i. In order to create the schedule S, we iterate through all $i \in \mathcal{M}$ and process the jobs in A_i within the interval $[g_i, g_{i+1})$, using the uniform speed $(\sum_{j\in A_i} v_j)/(g_{i+1}-g_i)$. The resulting schedule is clearly feasible, as A has finite cost and every $j \in A_i$ thus satisfies $[g_i, g_{i+1}) \subseteq [r_j, d_j)$. For the energy consumption of S we get

$$E(S) = \sum_{i\in\mathcal{M}} \left(\frac{\sum_{j\in A_i} v_j}{g_{i+1}-g_i}\right)^{\alpha} (g_{i+1}-g_i) = \sum_{i\in\mathcal{M}} \left(\frac{\sum_{j\in A_i} v_j}{(g_{i+1}-g_i)^{1-1/\alpha}}\right)^{\alpha} = \sum_{i\in\mathcal{M}} \left(\sum_{j\in A_i} p_{ij}\right)^{\alpha}$$

$$= \Big(\mathrm{Cost}(A)\Big)^{\alpha}.$$

□

Lemma 26. *If the grid points $b(\cdot)$ and $e(\cdot)$ are guessed correctly, there exists an assignment A for $\mathcal{I}'$ with $\mathit{Cost}(A) \le \big((1+\epsilon)^{\alpha-1}\mathrm{OPT}\big)^{1/\alpha}$.*

Proof. Remember that $\mathcal{G}$ is an optimal grid point schedule for $\mathcal{I}$, and that the grid points $b(T_{rd})$ and $e(T_{rd})$ mark the time points at which $\mathcal{G}$ starts to process the first job of type T_{rd} and finishes the last job of T_{rd}, respectively. Now observe that in $\mathcal{G}$, every job j is processed entirely within some interval $[g_i, g_{i+1})$, satisfying $[g_i, g_{i+1}) \subseteq [r_j, d_j)$. This is true because $r_j \le b(T_{r_j d_j}) < e(T_{r_j d_j}) \le d_j$, and no job can stretch from an interval $[g_{x-1}, g_x)$ into $[g_x, g_{x+1})$ since g_x indeed marks the beginning or end of some job. Let A_i denote the set of jobs which are entirely processed within $[g_i, g_{i+1})$, and let A be the assignment that maps all jobs from A_i to machine i. The cost of A is given as

$$\mathrm{Cost}(A) = \left(\sum_{i\in\mathcal{M}} \Big(\sum_{j\in A_i} p_{ij}\Big)^{\alpha}\right)^{1/\alpha} = \left(\sum_{i\in\mathcal{M}} \Big(\frac{\sum_{j\in A_i} v_j}{(g_{i+1}-g_i)^{1-1/\alpha}}\Big)^{\alpha}\right)^{1/\alpha}$$

$$= \left(\sum_{i\in\mathcal{M}} \Big(\frac{\sum_{j\in A_i} v_j}{g_{i+1}-g_i}\Big)^{\alpha} (g_{i+1}-g_i)\right)^{1/\alpha}$$

$$\le \Big(E(\mathcal{G})\Big)^{1/\alpha} \le \Big((1+\epsilon)^{\alpha-1}\mathrm{OPT}\Big)^{1/\alpha}.$$

Here the last two inequalities follow from the convexity of the power function and Lemma 4, respectively. □

Below you find a list of the most recent research reports of the Max-Planck-Institut für Informatik. Most of them are accessible via WWW using the URL `http://www.mpi-inf.mpg.de/reports`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
– Library and Publications –
Campus E 1 4

D-66123 Saarbrücken

E-mail: `library@mpi-inf.mpg.de`

MPI-I-2013-RG1-002	P. Baumgartner, U. Waldmann	Hierarchic superposition with weak abstraction
MPI-I-2013-5-002	F. Makari, B. Awerbuch, R. Gemulla, R. Khandekar, J. Mestre, M. Sozio	A distributed algorithm for large-scale generalized matching
MPI-I-2012-RG1-002	A. Fietzke, E. Kruglov, C. Weidenbach	Automatic generation of inductive invariants by SUP(LA)
MPI-I-2012-RG1-001	M. Suda, C. Weidenbach	Labelled superposition for PLTL
MPI-I-2012-5-004	F. Alvanaki, S. Michel, A. Stupar	Building and maintaining halls of fame over a database
MPI-I-2012-5-003	K. Berberich, S. Bedathur	Computing n-gram statistics in MapReduce
MPI-I-2012-5-002	M. Dylla, I. Miliaraki, M. Theobald	Top-k query processing in probabilistic databases with non-materialized views
MPI-I-2012-5-001	P. Miettinen, J. Vreeken	MDL4BMF: Minimum Description Length for Boolean Matrix Factorization
MPI-I-2012-4-001	J. Kerber, M. Bokeloh, M. Wand, H. Seidel	Symmetry detection in large scale city scans
MPI-I-2011-RG1-002	T. Lu, S. Merz, C. Weidenbach	Towards verification of the pastry protocol using TLA+
MPI-I-2011-5-002	B. Taneva, M. Kacimi, G. Weikum	Finding images of rare and ambiguous entities
MPI-I-2011-5-001	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Temporal index sharding for space-time efficiency in archive search
MPI-I-2011-4-005	A. Berner, O. Burghard, M. Wand, N.J. Mitra, R. Klein, H. Seidel	A morphable part model for shape manipulation
MPI-I-2011-4-002	K.I. Kim, Y. Kwon, J.H. Kim, C. Theobalt	Efficient learning-based image enhancement : application to compression artifact removal and super-resolution
MPI-I-2011-4-001	M. Granados, J. Tompkin, K. In Kim, O. Grau, J. Kautz, C. Theobalt	How not to be seen inpainting dynamic objects in crowded scenes
MPI-I-2010-RG1-001	M. Suda, C. Weidenbach, P. Wischnewski	On the saturation of YAGO
MPI-I-2010-5-008	S. Elbassuoni, M. Ramanath, G. Weikum	Query relaxation for entity-relationship search
MPI-I-2010-5-007	J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum	YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia
MPI-I-2010-5-006	A. Broschart, R. Schenkel	Real-time text queries with tunable term pair indexes
MPI-I-2010-5-005	S. Seufert, S. Bedathur, J. Mestre, G. Weikum	Bonsai: Growing Interesting Small Trees
MPI-I-2010-5-004	N. Preda, F. Suchanek, W. Yuan, G. Weikum	Query evaluation with asymmetric web services
MPI-I-2010-5-003	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Efficient temporal keyword queries over versioned text
MPI-I-2010-5-002	M. Theobald, M. Sozio, F. Suchanek, N. Nakashole	URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules
MPI-I-2010-5-001	K. Berberich, S. Bedathur, O. Alonso, G. Weikum	A language modeling approach for temporal information needs
MPI-I-2010-1-001	C. Huang, T. Kavitha	Maximum cardinality popular matchings in strict two-sided preference lists

MPI-I-2009-RG1-005	M. Horbach, C. Weidenbach	Superposition for fixed domains
MPI-I-2009-RG1-004	M. Horbach, C. Weidenbach	Decidability results for saturation-based model building
MPI-I-2009-RG1-002	P. Wischnewski, C. Weidenbach	Contextual rewriting
MPI-I-2009-RG1-001	M. Horbach, C. Weidenbach	Deciding the inductive validity of $\forall\exists^*$ queries
MPI-I-2009-5-007	G. Kasneci, G. Weikum, S. Elbassuoni	MING: Mining Informative Entity-Relationship Subgraphs
MPI-I-2009-5-006	S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, G. Weikum	Scalable phrase mining for ad-hoc text analytics
MPI-I-2009-5-005	G. de Melo, G. Weikum	Towards a Universal Wordnet by learning from combined evidenc
MPI-I-2009-5-004	N. Preda, F.M. Suchanek, G. Kasneci, T. Neumann, G. Weikum	Coupling knowledge bases and web services for active knowledge
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-002	M. Ramanath, K.S. Kumar, G. Ifrim	Generating concise and readable summaries of XML documents
MPI-I-2009-4-006	C. Stoll	Optical reconstruction of detailed animatable human body models
MPI-I-2009-4-005	A. Berner, M. Bokeloh, M. Wand, A. Schilling, H. Seidel	Generalized intrinsic symmetry detection
MPI-I-2009-4-004	V. Havran, J. Zajac, J. Drahokoupil, H. Seidel	MPI Informatics building model as data for your research
MPI-I-2009-4-003	M. Fuchs, T. Chen, O. Wang, R. Raskar, H.P.A. Lensch, H. Seidel	A shaped temporal filter camera
MPI-I-2009-4-002	A. Tevs, M. Wand, I. Ihrke, H. Seidel	A Bayesian approach to manifold topology reconstruction
MPI-I-2009-4-001	M.B. Hullin, B. Ajdin, J. Hanika, H. Seidel, J. Kautz, H.P.A. Lensch	Acquisition and analysis of bispectral bidirectional reflectance distribution functions
MPI-I-2008-RG1-001	A. Fietzke, C. Weidenbach	Labelled splitting
MPI-I-2008-5-004	F. Suchanek, M. Sozio, G. Weikum	SOFI: a self-organizing framework for information extraction
MPI-I-2008-5-003	G. de Melo, F.M. Suchanek, A. Pease	Integrating Yago into the suggested upper merged ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single phase construction of optimal DAG-structured QEPs
MPI-I-2008-5-001	G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, G. Weikum	STAR: Steiner tree approximation in relationship-graphs
MPI-I-2008-4-003	T. Schultz, H. Theisel, H. Seidel	Crease surfaces: from theory to extraction and application to diffusion tensor MRI
MPI-I-2008-4-002	D. Wang, A. Belyaev, W. Saleem, H. Seidel	Estimating complexity of 3D shapes using view similarity
MPI-I-2008-1-001	D. Ajwani, I. Malinger, U. Meyer, S. Toledo	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for finite domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : a large ontology from Wikipedia and WordNet
MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A time machine for text search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: searching and ranking knowledge
MPI-I-2007-4-008	J. Gall, T. Brox, B. Rosenhahn, H. Seidel	Global stochastic optimization for robust and accurate human motion capture