A Fully Polynomial-Time Approximation Scheme for Speed Scaling with a Sleep State

Antonios Antoniadis[1] · Chien-Chung Huang[2] · Sebastian Ott[3]

Abstract

We study classical deadline-based preemptive scheduling of jobs in a computing environment equipped with both dynamic speed scaling and sleep state capabilities: Each job is specified by a release time, a deadline and a processing volume, and has to be scheduled on a single, speed-scalable processor that is supplied with a sleep state. In the sleep state, the processor consumes no energy, but a constant wake-up cost is required to transition back to the active state. In contrast to speed scaling alone, the addition of a sleep state makes it sometimes beneficial to accelerate the processing of jobs in order to transition the processor to the sleep state for longer amounts of time and incur further energy savings. The goal is to output a feasible schedule that minimizes the energy consumption. Since the introduction of the problem by Irani et al. (ACM Trans Algorithms 3(4), 2007), its exact computational complexity has been repeatedly posed as an open question (see e.g. Albers and Antoniadis in ACM Trans Algorithms 10(2):9, 2014; Baptiste et al. in ACM Trans Algorithms 8(3):26, 2012; Irani and Pruhs in SIGACT News 36(2):63–76, 2005). The currently best known upper and lower bounds are a 4/3-approximation algorithm and NP-hardness due to Albers and Antoniadis (2014) and Kumar and Shannigrahi (CoRR, 2013. arXiv:1304.7373), respectively. We close the aforementioned gap between the upper and lower bound on the computational complexity of speed scaling with sleep state by presenting a fully polynomial-time approximation scheme for the problem. The scheme is based on a transformation to a non-preemptive variant of the problem, and a discretization that exploits a carefully defined lexicographical ordering among schedules.

A preliminary version of this paper appeared in SODA 2015 [3].

✉ Antonios Antoniadis
   aantonia@mpi-inf.mpg.de

Extended author information available on the last page of the article

1 Introduction

As energy efficiency in computing environments becomes more crucial, chip manufacturers are increasingly incorporating energy-saving functionalities to their processors. One of the most common functionalities is *dynamic speed scaling*, where the processor can adjust its speed dynamically. A higher speed yields a higher performance, but this performance comes at the cost of more energy consumption. On the other hand, a lower speed results in better energy-efficiency, but at the cost of performance degradation. In practice, it has been observed [7,11] that the power consumption of the processor is approximately proportional to its speed cubed. However, even when the processor is idling, it consumes a non-negligible amount of energy just for the sake of "being active" (for example because of leakage current). Due to this fact, additional energy can be saved by incorporating a *sleep state* to the processor. A processor in a sleep state consumes zero (or negligible) energy; however, there is an extra energy cost when it is transitioned back to the active state.

This article studies the offline problem of minimizing energy consumption of a processor which is equipped with both speed scaling and sleep state capabilities. This problem is called *speed scaling with sleep state*, first introduced by Irani et al. [18].

Let us state the problem more formally. The given processor has two states: the *active state*, during which it can execute jobs and consume some energy, and the *sleep state*, during which no jobs can be executed but also no energy is consumed. We assume that a *wake-up operation*, that is a transition from the sleep state to the active state, incurs a constant energy cost $C > 0$, whereas transitioning from the active state to the sleep state is free of charge. Further, as in [2,18], the power required by the processor in the active state is an arbitrary convex and non-decreasing function P of its speed s. In accordance to all previous work in the area, we make the necessary assumptions that the power function P is *fixed* and therefore not part of the input.[1] We also assume that $P(0) > 0$, since (i) as already mentioned, real-world processors are known to have leakage current and (ii) otherwise the sleep state would be redundant. Further motivation for considering arbitrary convex power functions for speed scaling can be found, for example, in [8].

The input is a set $\mathcal{J}$ of n jobs. Each job j is associated with a release time r_j, a deadline d_j and a processing volume v_j. One can think of the processing volume as the number of CPU cycles that are required in order to completely process the job, so that if job j is processed at a speed of s, then v_j/s time-units are required to complete the job. We call the interval $[r_j, d_j)$ the *allowed interval* of job j, and say that job j is active at time point t if and only if $t \in [r_j, d_j)$.[2] Furthermore, we may assume without loss of generality that $\min_{j \in \mathcal{J}} r_j = 0$, and that $v_{min} := \min_{j \in \mathcal{J}} v_j$ is normalized to 1 (if $v_{min} \neq 1$ is the case, we can scale the instance by dividing the r_j's, d_j's, and v_j's by v_{min}, and using the power function $P(s) \cdot v_{min}$ along with the original wake-up cost C). Further, let $d_{max} := \max_{j \in \mathcal{J}} d_j$ be the latest deadline of any job.

---

[1] It is assumed that we have access to an oracle which allows us to evaluate $P(s)$ for any value of s and perform simple operations involving values returned by P. Similarly the critical speed (defined later) is a known parameter of $P(s)$.

[2] Unless stated differently, throughout the text an interval will always have the form $[\cdot, \cdot)$.

A *schedule* is defined as a mapping of every time point t to the state of the processor, its speed, and the job being processed at t (or *null* if there is no job running at t). Note that the processing speed is zero whenever the processor sleeps, and that a job can only be processed when the speed is strictly positive. A schedule is called *feasible* when the whole processing volume of every job j is completely processed in j's allowed interval $[r_j, d_j)$. Preemption of jobs is allowed.

The energy consumption incurred by schedule $\mathcal{S}$ while the processor is in the active state, is its power integrated over time, i.e. $\int P(s(t))dt$, where $s(t)$ is the processing speed at time t, and the integral is taken over all time points in $[0, d_{max})$ during which the processor is active under $\mathcal{S}$. Assume that $\mathcal{S}$ performs k transitions from the sleep state to the active state. (We will assume throughout the paper that initially, prior to the first release time, as well as finally, after the last deadline, the processor is in the active state. However, our results can be easily adapted for the setting where the processor is initially and/or eventually in the sleep state). Then the total energy consumption of $\mathcal{S}$ is $E(\mathcal{S}) := \int P(s(t))dt + kC$, where again the integral is taken over all time points at which $\mathcal{S}$ keeps the processor in the active state. We are seeking a feasible schedule that minimizes the total energy consumption.

Observe that, by Jensen's inequality, and by the convexity of the power function, it is never beneficial to process a job with a varying speed. Irani et al. [18] observed the existence of a *critical speed* s_{crit}, which is the most efficient speed for processing jobs. This critical speed is the smallest speed that minimizes the function $P(s)/s$. Note that, by the convexity of $P(s)$, the only case where the critical speed s_{crit} is not well defined, is when $P(s)/s$ is always decreasing. However, this would render the setting unrealistic, and furthermore make the algorithmic problem trivial, since it would be optimal to process every job at an infinite speed. We may therefore assume that this case does not occur. Further, it can be shown (see [18]) that for any $s \geq s_{crit}$, the function $P(s)/s$ is non-decreasing.

1.1 Previous Work

The theoretical model for dynamic speed scaling was introduced in a seminal paper by Yao, Demers and Shenker [21]. They developed a polynomial time algorithm called *YDS*, that outputs a minimum-energy schedule for this setting. Irani, Shukla and Gupta [18] initiated the algorithmic study of speed scaling combined with a sleep state. Such a setting motivates the so-called *race to idle* technique: one saves overall energy by accelerating some jobs in order to transition the processor to the sleep state for longer periods of time (see [4,13,14,20] and references therein for more information regarding the race to idle technique). Irani et al. developed a 2-approximation algorithm for speed scaling with sleep state, but the computational complexity of the scheduling problem has remained open. The first step towards attacking this open problem was made by Baptiste [9], who gave a polynomial time algorithm for the special case where the processor must execute all jobs at a fixed speed, and all jobs are of unit size. Baptiste's algorithm is based on a clever dynamic programming formulation of the scheduling problem, and was later extended to (i) arbitrarily-sized jobs in [10], and (ii) a multiprocessor setting in [12].

More recently, Albers and Antoniadis [2] improved the upper bound on the approximation ratio of the general problem, by developing a 4/3-approximation algorithm. For the special case of agreeable deadlines and a power function of the form $P(s) = s^\alpha + \beta$ (with constant $\alpha > 1$ and $\beta > 0$), Bampis et al. [5] provided an exact polynomial time algorithm. With respect to the lower bound, [2] gave an NP-hardness reduction from the *partition* problem. The reduction uses a particular power function that is based on the partition instance, i.e., it is considered that the power function is part of the input. The reduction of [2] was later refined by Kumar and Shannigrahi [19], to show that the problem is NP-hard for any fixed, non-decreasing and strictly convex power function.

The online setting of the problem has also been studied. Irani et al. [18] gave a $(2^{2\alpha-2}\alpha^\alpha + 2^{\alpha-1+2})$-competitive online algorithm. Han et al. [15] improved upon this result by developing an $(\alpha^\alpha + 2)$-competitive algorithm for the problem. Both of the above results assume a power function of the form $P(s) = s^\alpha + \beta$, where $\alpha > 1$ and $\beta > 0$ are constants.

A more thorough discussion on the above scheduling problems can be found in the surveys [1,17].

1.2 Our Contribution

We study the offline setting of speed scaling with sleep state. Since the introduction of the problem by Irani et al. [18], its exact computational complexity has been repeatedly posed as an open question (see e.g. [2,10,17]). The currently best known upper and lower bounds are a 4/3-approximation algorithm and NP-hardness due to [2] and [2, 19], respectively. In this paper, we settle the open question by presenting a fully polynomial-time approximation scheme.

At the core of our approach is a transformation of the original preemptive problem into a non-preemptive scheduling problem of the same type. At first sight, this may seem counterintuitive, especially as Bampis et al. [6] showed that (for the problem of speed scaling alone), for the same instance, the ratio of an optimal preemptive solution against an optimal non-preemptive solution can be very high. However, this does not apply in our case, as we consider the non-preemptive problem on a modified instance, where each job is replaced by a polynomial number of *pieces*. Furthermore, in our analysis, we make use of a particular lexicographic ordering that does exploit the advantages of preemption.

In order to compute an optimal schedule for the modified instance via dynamic programming, we require a number of properties that pieces must satisfy in a valid schedule. The definition of these properties is based on a discretization of the time horizon by a polynomial number of time points. Roughly speaking, we focus on those schedules that start and end the processing of each piece at such time points, and satisfy a certain constraint on the processing order of the pieces. Proving that a near-optimal schedule in this class of schedules exists is the most subtle part of our approach.

On one hand, the processing order constraint can be exploited by the DP; on the other hand, such a constraint is difficult to establish in an optimal schedule with the introduced indivisible volumes (since pieces of different jobs might have different volumes and cannot easily be interchanged). To get around this, we first ensure the

right ordering in an optimal schedule for the preemptive setting, and then perform a series of transformations to a non-preemptive schedule with the above properties. Each of these transformations increases the energy consumption only by a small factor, and maintains the correct ordering among the pieces.

We remark that Baptiste [9] used a dynamic program of similar structure for the special case of unit-sized jobs and a fixed-speed processor equipped with a sleep state. His dynamic program is also based on a particular ordering of jobs, which, however, is not sufficient for our setting. Since we have pieces of different sizes, the swapping argument used in [9] fails.

In Sect. 2, we describe the YDS algorithm from [21] for the problem of speed scaling *without* a sleep state, and then show several properties that a schedule produced by YDS has for our problem of speed scaling with sleep state. We then, in Sect. 3, define a particular class of schedules that have a set of desirable properties, and show that there exists a schedule in this class, whose energy consumption is within a $(1 + \epsilon)$-factor from optimal. Finally, in Sect. 4, we develop an algorithm based on a dynamic program, that outputs, in polynomial time, a schedule of minimal energy consumption among all the schedules of the aforementioned class.

2 Preliminaries

We start by giving a short description of the YDS algorithm presented in [21]. For any interval I, let $B(I)$ be the set of jobs whose allowed intervals are contained in I. We define the *density* of I as

$$dens(I) = \frac{\sum_{j \in B(I)} v_j}{|I|}.$$

Note that the average speed that any feasible schedule uses during interval I is no less than $dens(I)$. YDS works in rounds. In the first round, the interval I_1 of maximal density is identified, and all jobs in $B(I_1)$ are scheduled during I_1 at a speed of $dens(I_1)$, according to the earliest deadline first policy. Then the jobs in $B(I_1)$ are removed from the instance and the time interval I_1 is "blacked out". In general, during round i, YDS identifies the interval I_i of maximal density and then processes all jobs in $B(I_i)$ at a uniform speed of $dens(I_i)$, removes the jobs in $B(I_i)$ from the instance, and "shrinks" the interval I_i to length zero. YDS terminates when all jobs are scheduled, and its running time is polynomial in the input size.

We remark that the speed used for the processing of jobs can never increase between two consecutive rounds, i.e., YDS schedules the jobs by order of non-increasing speeds. Furthermore, all the jobs scheduled in each round i have their allowed intervals within I_i.

Given any job instance $\mathcal{J}$, let FAST($\mathcal{J}$) be the subset of $\mathcal{J}$ that YDS processes at a speed greater than or equal to s_{crit}, and let SLOW($\mathcal{J}$) := $\mathcal{J} \setminus$ FAST($\mathcal{J}$). The following lemma is an extension of a fact proven by Irani et al. [18].

Lemma 1 *For any job instance $\mathcal{J}$, there exists an optimal schedule (w.r.t. speed scaling with sleep state) in which*

1. *Every job in FAST($\mathcal{J}$) is processed according to YDS.*
2. *Every job $k \in$ SLOW($\mathcal{J}$) is run at a uniform speed $s_k < s_{crit}$, and the processor never (actively) runs at a speed less than s_k during $[r_k, d_k)$.*

We call an optimal schedule with these properties a YDS-extension *for $\mathcal{J}$.*

Proof To break ties among different optimal schedules with equal energy consumption (which can occur when the power function is not strictly convex for any s), we introduce the pseudo cost function $\int s(t)^2 dt$ (this idea was first used in [18]). Consider a minimal pseudo cost schedule Y, so that Y satisfies property 1 of the lemma, and minimizes the energy consumption among all schedules satisfying this property. It was shown in [18] that Y is optimal for instance $\mathcal{J}$, and that under Y

> Every job $k \in$ SLOW($\mathcal{J}$) is run at a uniform speed s_k, and the processor never (actively) runs at a speed less than s_k during those portions of $[r_k, d_k)$ $\quad(*)$
> where no job from FAST($\mathcal{J}$) is processed.

It therefore remains to prove that the speeds s_k are no higher than s_{crit}. For the sake of contradiction, assume that there exists a job $j \in$ SLOW($\mathcal{J}$) which is processed at speed higher than s_{crit}. Let $\mathcal{I}$ be a maximal time interval, so that (i) $\mathcal{I}$ includes at least part of the execution of j, and (ii) at any time point $t \in \mathcal{I}$ the processor either runs strictly faster than s_{crit}, or executes a job from FAST($\mathcal{J}$). Then there must exist a job $k \in$ SLOW($\mathcal{J}$) (possibly $k = j$) which is executed to some extent during $\mathcal{I}$, and whose allowed interval is not contained in $\mathcal{I}$ (otherwise, when running YDS, the density of $\mathcal{I}$ after the jobs in FAST($\mathcal{J}$) have been scheduled is larger than s_{crit}, contradicting the fact that YDS processes all remaining jobs slower than s_{crit}). By the maximality of $\mathcal{I}$, there exists some interval $\mathcal{I}' \subseteq [r_k, d_k)$ right before $\mathcal{I}$ or right after $\mathcal{I}$, during which no job from FAST($\mathcal{J}$) is executed, and the processor either runs with speed at most s_{crit} or resides in the sleep state. The first case contradicts property $(*)$, as k is processed during $\mathcal{I}$ and thus at speed $s_k > s_{crit}$. In the second case, we can use a portion of $\mathcal{I}'$ to slightly slow down k to a new speed s', such that $s_{crit} < s' < s_k$. The resulting schedule Y' has energy consumption no higher than Y, as $P(s)/s$ is non-decreasing for $s \geq s_{crit}$. Furthermore, if $\mathcal{C}_p$ is the pseudo cost of Y, then Y' has pseudo cost $\mathcal{C}_p - v_k s_k + v_k s' < \mathcal{C}_p$. This contradicts our assumptions on Y. $\quad\square$

Clearly, in general a schedule produced by YDS only satisfies the first condition and is not a YDS-extension.

By the preceding lemma, we may use YDS to schedule the jobs in FAST($\mathcal{J}$), and need to find a good schedule only for the remaining jobs (which are exactly SLOW($\mathcal{J}$)). To this end, we transform the input instance $\mathcal{J}$ to an instance $\mathcal{J}'$, in which the jobs FAST($\mathcal{J}$) are replaced by dummy jobs. This introduction of dummy jobs bears resemblance to the approach of [2]. We then show in Lemma 2, that any schedule for $\mathcal{J}'$ with a certain property, can be transformed to a schedule for $\mathcal{J}$ without any degradation in the approximation factor.

Consider the schedule S_{YDS} that algorithm YDS produces on $\mathcal{J}$. Let $I_i = [y_i, z_i)$, $i = 1, \ldots, \ell$ be the ith maximal interval in which S_{YDS} continuously runs at a speed

greater than or equal to s_{crit}, and let $T_1, \ldots, T_m$ be the remaining maximal intervals in $[0, d_{max})$ not covered by intervals $I_1, I_2, \ldots, I_\ell$. Furthermore, let $\mathcal{T} := \cup_{1 \le k \le m} T_k$. Note that each maximal interval I_i or T_i may contain subintervals identified in different rounds of YDS, and thus more than one speed levels may be employed in each such maximal interval. Also note that the intervals I_p and T_q ($1 \le p \le \ell$ and $1 \le q \le m$) partition the time horizon $[0, d_{max})$, and furthermore, by the way YDS is defined, every job $j \in \text{FAST}(\mathcal{J})$ is active in exactly one interval I_i, and is not active in any interval T_i. On the other hand, a job $j \in \text{SLOW}(\mathcal{J})$ may be active in several (consecutive) intervals I_i and $T_{i'}$. We transform $\mathcal{J}$ to a job instance $\mathcal{J}'$ as follows:

- For every job $j \in \text{SLOW}(\mathcal{J})$, if there exists an i such that $r_j \in I_i$ (resp. $d_j \in I_i$), then we set $r_j := z_i$ (resp. $d_j := y_i$), else we keep the job as it is.
- For each I_i, we replace all jobs $j \in \text{FAST}(\mathcal{J})$ that are active in I_i by a single job j_i^d (d stands for "dummy") with release time at y_i, deadline at z_i, and processing volume v_i^d equal to the total volume that S_{YDS} schedules in I_i, i.e. $v_i^d = \sum_{j \in B(I_i)} v_j$.

Clearly, the above transformation can be done in polynomial time. Note that after the transformation, there is no release time or deadline in the interior of any interval I_i. Furthermore, we have the following proposition:

Proposition 1 $FAST(\mathcal{J}') = \{j_i^d : 1 \le i \le \ell\}$ and $SLOW(\mathcal{J}') = SLOW(\mathcal{J})$.

Proof Since $\mathcal{J}' = \{j_i^d : 1 \le i \le \ell\} \cup \text{SLOW}(\mathcal{J})$, and furthermore $\text{SLOW}(\mathcal{J}')$ and $\text{FAST}(\mathcal{J}')$ are disjoint sets, it suffices to show that (i) $\text{FAST}(\mathcal{J}') \supseteq \{j_i^d : 1 \le i \le \ell\}$ and that (ii) $\text{SLOW}(\mathcal{J}') \supseteq \text{SLOW}(\mathcal{J})$.

For (i), we observe that no job j_i^d can be feasibly scheduled at a uniform speed less than s_{crit}. As YDS uses a uniform speed for each job, these jobs must belong to $\text{FAST}(\mathcal{J}')$.

For (ii), consider the execution of YDS on $\mathcal{J}'$. More specifically, consider the first round when a job from $\text{SLOW}(\mathcal{J})$ is scheduled. Let $\mathcal{I}$ be the maximal density interval of this round, and let $\mathcal{J}_S$ and $\mathcal{J}_d$ be the sets of jobs from $\text{SLOW}(\mathcal{J})$ and $\{j_i^d : 1 \le i \le \ell\}$, respectively, that are scheduled in this round (note that $\mathcal{I}$ contains the allowed intervals of these jobs). As the speed used by YDS is non-increasing from round to round, it suffices to show that $dens(\mathcal{I}) < s_{crit}$.

Consider a partition of $\mathcal{I}$ into maximal intervals $\Lambda_1, \ldots, \Lambda_a$, s.t. each Λ_k is contained in some interval I_i or T_i. Then

$$
\begin{aligned}
dens(\mathcal{I}) &= \frac{\sum_{j \in \mathcal{J}_d} v_j}{|\mathcal{I}|} + \frac{\sum_{j \in \mathcal{J}_S} v_j}{|\mathcal{I}|} \\
&= \sum_{\Lambda_k \not\subseteq \mathcal{T}} \left(\frac{|\Lambda_k|}{|\mathcal{I}|} dens(\Lambda_k) \right) + \frac{\sum_{\Lambda_k \subseteq \mathcal{T}} |\Lambda_k|}{|\mathcal{I}|} \cdot \frac{\sum_{j \in \mathcal{J}_S} v_j}{\sum_{\Lambda_k \subseteq \mathcal{T}} |\Lambda_k|} \\
&\le \left(\sum_{\Lambda_k \not\subseteq \mathcal{T}} \frac{|\Lambda_k|}{|\mathcal{I}|} \right) dens(\mathcal{I}) + \left(1 - \sum_{\Lambda_k \not\subseteq \mathcal{T}} \frac{|\Lambda_k|}{|\mathcal{I}|} \right) \cdot \frac{\sum_{j \in \mathcal{J}_S} v_j}{\sum_{\Lambda_k \subseteq \mathcal{T}} |\Lambda_k|},
\end{aligned}
$$

since no Λ_k can have a density larger than $dens(\mathcal{I})$ (because $\mathcal{I}$ is the interval of maximal density). It follows that

$$dens(\mathcal{I}) \leq \frac{\sum_{j \in \mathcal{J}_S} v_j}{\sum_{\Lambda_k \subseteq \mathcal{T}} |\Lambda_k|} .$$

Furthermore, by the definition of SLOW($\mathcal{J}$), it is possible to schedule all jobs in $\mathcal{J}_S$ during $\mathcal{I} \cap \mathcal{T}$, at a speed slower than s_{crit} (since none of the steps in the transformation from $\mathcal{J}$ to $\mathcal{J}'$ reduce the time any job is active during $\mathcal{T}$). Together with the previous inequality, this implies $dens(\mathcal{I}) < s_{crit}$. □

The following lemma suggests that for obtaining an FPTAS for instance $\mathcal{J}$, it suffices to give an FPTAS for instance $\mathcal{J}'$, as long as we schedule the jobs j_i^d exactly in their allowed intervals I_i.

Lemma 2 *Let S' be a schedule for input instance $\mathcal{J}'$, that (i) processes each job j_i^d exactly in its allowed interval I_i (i.e. from y_i to z_i), and (ii) is a c-approximation for $\mathcal{J}'$. Then S' can be transformed in polynomial time into a schedule S that is a c-approximation for input instance $\mathcal{J}$.*

Proof Given such a schedule S', we leave the processing in the intervals $T_1, \dots, T_m$ unchanged, and replace for each interval I_i the processing of job j_i^d by the original YDS-schedule S_{YDS} during I_i. It is easy to see that the resulting schedule S is a feasible schedule for $\mathcal{J}$. We now argue about the approximation factor.

Let OPT be a YDS-extension for $\mathcal{J}$, and let OPT$'$ be a YDS-extension for $\mathcal{J}'$. Recall that $E(\cdot)$ denotes the energy consumption of a schedule (including wake-up costs). Additionally, let $E^I(S)$ denote the total energy consumption of S in all intervals $I_1, \dots, I_\ell$ without wake-up costs (i.e. the energy consumption for processing or being active but idle during those intervals), and define similarly $E^I(S')$, $E^I(OPT)$, and $E^I(OPT')$ for the schedules S', OPT, and OPT$'$, respectively. Since S' is a c-approximation for $\mathcal{J}'$, we have

$$E(S') \leq cE(OPT').$$

Note that OPT$'$ schedules exactly the job j_i^d in each I_i (using the entire interval for it) by Proposition 1, and thus each of the schedules S, S', OPT, and OPT$'$ keeps the processor active during every entire interval I_i. Therefore

$$E(S) - E(S') = E^I(S) - E^I(S'),$$

since S and S' have the same wake-up costs and do not differ in the intervals T_1, $\dots, T_m$. Moreover,

$$E(OPT) - E(OPT') = E^I(OPT) - E^I(OPT'),$$

as $E(OPT) - E^I(OPT)$ and $E(OPT') - E^I(OPT')$ are both equal to the optimal energy consumption during $\mathcal{T}$ of any schedule that processes the jobs SLOW($\mathcal{J}$)

in $\mathcal{T}$ and resides in the active state during each interval I_i (including all wake-up costs of the schedule). Clearly, $E^I(S) = E^I(OPT)$, and since both S' and OPT$'$ schedule exactly the job j_i^d in each I_i (using the entire interval for it), we have that $E^I(S') \geq E^I(OPT')$. Therefore

$$E(S) - E(S') \leq E(OPT) - E(OPT').$$

We next show that $0 \leq E^I(OPT) - E^I(OPT') = E(OPT) - E(OPT')$, which implies

$$\begin{aligned}
E(S) &\leq E(OPT) - E(OPT') + E(S') \\
&\leq c(E(OPT) - E(OPT')) + E(S') \\
&\leq c(E(OPT) - E(OPT')) + cE(OPT') \\
&\leq cE(OPT).
\end{aligned}$$

Since YDS (when applied to $\mathcal{J}$) processes a volume of exactly v_i^d in each interval I_i, the average speed of OPT in I_i is $v_i^d/|I_i|$. On the other hand, OPT$'$ runs with a speed of exactly $v_i^d/|I_i|$ during I_i, and therefore $E^I(OPT) \geq E^I(OPT')$.   □

3 Discretizing the Problem

After the transformation in the previous section, we have an instance $\mathcal{J}'$. In this section, we show that there exists a "discretized" schedule for $\mathcal{J}'$, whose energy consumption is at most $1 + \epsilon$ times that of an optimal schedule for $\mathcal{J}'$. In the next section, we will show how such a discretized schedule can be found by dynamic programming.

Before presenting formal definitions and technical details, we here first sketch the ideas behind our approach.

A major challenge of the original problem is that we need to deal with an infinite number of possible schedules. We overcome this intractability by "discretizing" the problem as follows: (1) we break each job in SLOW($\mathcal{J}'$) into smaller pieces, and (2) we create a set of time points and introduce the additional constraint that each piece of a job has to start and end at these time points. The number of the introduced time points and job pieces are both polynomial in the input size and $1/\epsilon$, which substantially reduces the amount of guesswork we have to do in the dynamic program. The challenge is how to find such a discretization and argue that it does not increase the optimal energy consumption by too much.

3.1 Further Definitions and Notation

We first define the set W of time points. Given an error parameter $\epsilon > 0$, let $\delta := \min\{\frac{1}{4}, \frac{\epsilon}{4}\frac{P(s_{crit})}{P(2s_{crit})-P(s_{crit})}\}$. Intuitively, δ is defined in such a way that speeding up the processor by a factor $(1 + \delta)^3$ does not increase the power consumption by more than a factor $1 + \epsilon$ (see Lemma 5).
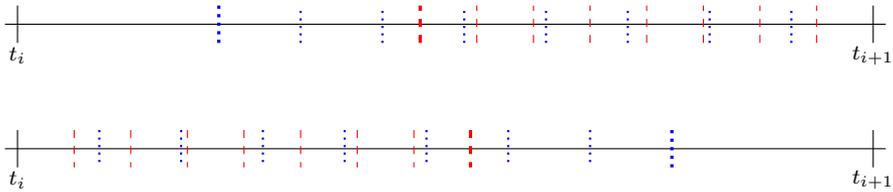
Fig. 1 We assume that $r = 1 \ldots 8$ and that $x(i) = 2$. The red dashed points correspond to $j = 1$ and the blue dotted points to $j = 2$. For clarity, we drew the points defined from t_i and from t_{i+1} in two separate pictures. Note that for each j the number of points is the same and the points of the same color are at equal distance from each other (Color figure online)

Let $W' := \bigcup_{j \in \mathcal{J}'} \{r_j, d_j\}$, and consider the elements of W' in sorted order. Let $t_i, 1 \leq i \leq |W'|$ be the ith element of W' in this order. We call an interval $[t_i, t_{i+1})$ for $1 \leq i \leq |W'| - 1$ a *zone*, and observe that every zone is either equal to some interval I_i or contained in some interval T_i.

For each i in $1, \ldots, |W'| - 1$, let $x(i)$ be the largest integer j so that

$$(1 + \delta)^j \frac{1}{4n^2 s_{crit}(1 + \delta)\lceil 1/\delta \rceil} \leq t_{i+1} - t_i.$$

We are now ready to define the set W of time points as follows:

$$W := W' \bigcup_{\substack{i \text{ s.t. } [t_i, t_{i+1}) \subseteq \mathcal{T} \\ 0 \leq j \leq x(i) \\ 1 \leq r \leq 16n^6 \lceil 1/\delta \rceil^2 (1 + \lceil 1/\delta \rceil)}} \left\{ t_i + r \cdot \frac{(1 + \delta)^j \frac{1}{4n^2 s_{crit}(1+\delta)\lceil 1/\delta \rceil}}{16n^6 \lceil 1/\delta \rceil^2 (1 + \lceil 1/\delta \rceil)}, \right.$$
$$\left. t_{i+1} - r \cdot \frac{(1 + \delta)^j \frac{1}{4n^2 s_{crit}(1+\delta)\lceil 1/\delta \rceil}}{16n^6 \lceil 1/\delta \rceil^2 (1 + \lceil 1/\delta \rceil)} \right\}.$$

Let us explain how these time points in W come about. As we will show later (Lemma 3(2)), there exists a certain optimal schedule for $\mathcal{J}'$ in which each zone $[t_i, t_{i+1}) \subseteq \mathcal{T}$ contains at most one contiguous maximal processing interval, and this interval "touches" either t_i or t_{i+1} (or both). The geometric series

$$(1 + \delta)^j \frac{1}{4n^2 s_{crit}(1 + \delta)\lceil 1/\delta \rceil}$$

of time points are used to approximate the ending/starting time of this maximal processing interval. For each guess of the ending/starting time, we split the guessed interval, during which the job pieces (to be defined formally immediately) are to be processed, into $16n^6 \lceil 1/\delta \rceil^2 (1 + \lceil 1/\delta \rceil)$ many intervals of equal length. An example of the set W for a given zone can be seen in Fig. 1.

Note that $|W|$ is polynomial in the input size and $1/\epsilon$.

Definition 1 We split each job $j \in \text{SLOW}(\mathcal{J}')$ into $4n^2 \lceil 1/\delta \rceil$ equal sized *pieces*, and also consider each job $j_i^d \in \text{FAST}(\mathcal{J}')$ as a single piece on its own. For every piece u of some job j, let $job(u) := j$, $r_u := r_j$, $d_u := d_j$, and $v_u := v_j/(4n^2 \lceil 1/\delta \rceil)$ if $j \in \text{SLOW}(\mathcal{J}')$, and $v_u := v_j$ otherwise. Furthermore, let D denote the set of all pieces derived from all jobs in $\mathcal{J}'$.

Note that $|D| = \ell + |\text{SLOW}(\mathcal{J}')| \cdot 4n^2 \lceil 1/\delta \rceil$ is polynomial in the input size and $1/\epsilon$. We now define an ordering of the pieces in D.

Definition 2 Fix an arbitrary ordering of the jobs in $\mathcal{J}'$, s.t. for any two different jobs j and j', $j \prec j'$ implies $r_j \leq r_{j'}$. Now extend this ordering to the set of pieces, s.t. for any two pieces u and u', there holds

$$u \prec u' \Rightarrow job(u) \preceq job(u').$$

We point out that any schedule for $\mathcal{J}'$ can also be seen as a schedule for D, by implicitly assuming that the pieces of any fixed job are processed in the above order.

We are now ready to define the class of discretized schedules.

Definition 3 A *discretized schedule* is a schedule for $\mathcal{J}'$ that satisfies the following two properties:

(i) Every piece is completely processed in a single zone, and without preemption.
(ii) The execution of every piece starts and ends at a time point from the set W.

A discretized schedule S is called *well-ordered* if and only if

(iii) For any time point t, such that in S a piece u ends at t, S schedules all pieces $u' \succ u$ with $d_{u'} \geq t$ after t.

Finally, we define a particular ordering over possible schedules, which will be useful in our analysis.

Definition 4 Consider a given schedule. For every job $j \in \mathcal{J}'$, and every $x \leq v_j$, let $c_j(x)$ denote the earliest time point at which volume x of job j has been finished under this schedule. Furthermore, for any $j \in \mathcal{J}'$, we define

$$q_j := \int_0^{v_j} c_j(x)dx.$$

Let $j_1 \prec j_2 \prec \cdots \prec j_{|\mathcal{J}'|}$ be the jobs in $\mathcal{J}'$. A schedule S is *lexicographically smaller* than a schedule S' if and only if it is lexicographically smaller with respect to the vector $(q_{j_1}, q_{j_2}, \ldots, q_{j_{|\mathcal{J}'|}})$.

Observe that shifting the processing interval of any fraction of some job j to an earlier time point (without affecting the other processing times of j) decreases the value of q_j.

3.2 Existence of a Near-Optimal Discretized Schedule

In this section, we first show that there exists a YDS-extension for $\mathcal{J}'$ with certain nice properties (recall that a YDS-extension is an optimal schedule satisfying the properties of Lemma 1). We then explain how such a YDS-extension can be transformed into a well-ordered discretized schedule, and prove that the speed of the latter, at all times, is at most $(1 + \delta)^3$ times that of the former. This fact essentially guarantees the existence of a well-ordered discretized schedule with energy consumption at most $1 + \epsilon$ that of an optimal schedule for $\mathcal{J}'$. The transformation is depicted in Fig. 2.

Lemma 3 *Let OPT be a lexicographically minimal YDS-extension for $\mathcal{J}'$. Then the following hold:*

1. *Every job j_i^d is scheduled exactly in its allowed interval I_i.*
2. *Every zone $[t_i, t_{i+1}) \subseteq \mathcal{T}$ has the following two properties:*

   (a) *There is at most one contiguous maximal processing interval within $[t_i, t_{i+1})$, and this interval either starts at t_i and/or ends at t_{i+1}. We call this interval the block of zone $[t_i, t_{i+1})$.*

   (b) *OPT uses a uniform speed of at most s_{crit} during this block.*

3. *There exist no two jobs $j' > j$, such that a portion of j is processed after some portion of j', and before $d_{j'}$.*

Proof 1. Since $\text{FAST}(\mathcal{J}') = \{j_i^d : 1 \le i \le \ell\}$ (by Proposition 1), and OPT is a YDS-extension, it follows that each j_i^d is processed exactly in its allowed interval I_i.

2. (a) Assume for the sake of contradiction that $[t_i, t_{i+1}) \subseteq \mathcal{T}$ contains a number of maximal intervals $N_1, N_2, \ldots, N_\psi$ (ordered from left to right[3]) during which jobs are being processed, with $\psi \ge 2$. Let $M_1, M_2, \ldots, M_{\psi'}$ (again ordered from left to right) be the remaining maximal intervals in $[t_i, t_{i+1})$, so that $N_1, \ldots, N_\psi$ and $M_1, \ldots, M_{\psi'}$ partition the zone $[t_i, t_{i+1})$. Furthermore, note that for each $i = 1, \ldots, \psi'$, the processor is either active but idle or asleep during the whole interval M_i, since otherwise setting the processor asleep during the whole interval M_i would incur a strictly smaller energy consumption. We modify the schedule by shifting the intervals $N_i, i = 2, \ldots, \psi$ to the left, so that $N_1, N_2, \ldots, N_\psi$ now form a single contiguous processing interval. The intervals M_k lying to the right of N_1 are moved further right and merge into a single (longer) interval M' during which no jobs are being processed. If the processor was active during each of these intervals M_k, then we keep the processor active during the new interval M', else we transition it to the sleep state. We observe that the resulting schedule is still a YDS-extension (note that its energy consumption is at most that of the initial schedule), but is lexicographically smaller.

For the second part of the statement, assume that there exists exactly one contiguous maximal processing interval N_1 within $[t_i, t_{i+1})$, and that there exist

---

[3] For any two time points $t_1 < t_2$, we say that t_1 is to the *left* of t_2, and t_2 is to the *right* of t_1.

two M-intervals, M_1 and M_2 before and after N_1, respectively.
We consider two cases:

- The processor is active just before t_i, or the processor is asleep both just before t_i and just after t_{i+1}: In this case we can shift N_1 left by $|M_1|$ time units, so that it starts at t_i. Again, we keep the processor active during $[t_i + |N_1|, t_{i+1})$ only if it was active during both M_1 and M_2. As before, the resulting schedule remains a YDS-extension, and is lexicographically smaller.

- The processor is in the sleep state just before t_i but active just after t_{i+1}: In this case we shift N_1 by $|M_2|$ time units to the right, so that its right endpoint becomes t_{i+1}. During the new idle interval $[t_i, t_i + |M_1| + |M_2|)$ we set the processor asleep. Note that in this case the processor was asleep during M_1. The schedule remains a YDS-extension, but its energy consumption becomes strictly smaller: (i) either the processor was asleep during M_2, in which case the resulting schedule uses the same energy while the processor is active but has one wake-up operation less, or (ii) the processor was active and idle during M_2, in which case the resulting schedule saves the idle energy that was spent during M_2.

(b) The statement follows directly from the second property of Lemma 1 and the fact that all jobs processed during $[t_i, t_{i+1})$ belong to SLOW($\mathcal{J}'$) and are active in the entire zone.

3. Assume for the sake of contradiction that there exist two jobs $j' \succ j$, such that a portion of j is processed during an interval $Z = [\zeta_1, \zeta_2)$, $\zeta_2 \le d_{j'}$, and some portion of j' is processed during an interval $Z' = [\zeta_1', \zeta_2')$, with $\zeta_2' \le \zeta_1$. We first observe that both jobs belong to SLOW($\mathcal{J}'$). This follows from the fact that both jobs are active during the whole interval $[\zeta_1', \zeta_2)$, and processed during parts of this interval, whereas any job j_i^d (which are the only jobs in FAST($\mathcal{J}'$)) is processed exactly in its entire interval $[y_i, z_i)$ (by statement 1 of the lemma).

By the second property of Lemma 1, both j and j' are processed at the same speed. We can now apply a swap argument. Let $L := \min\{|Z|, |Z'|\}$. Note that OPT schedules only j' during $[\zeta_2' - L, \zeta_2')$ and only j during $[\zeta_2 - L, \zeta_2)$. Swap the part of the schedule OPT in $[\zeta_2' - L, \zeta_2')$ with the schedule in the interval $[\zeta_2 - L, \zeta_2)$. Given the above observations, it can be easily verified that the resulting schedule (i) is feasible and remains a YDS-extension, and (ii) is lexicographically smaller than OPT. $\qquad\square$

The next lemma shows how to transform the lexicographically minimal YDS-extension for $\mathcal{J}'$ of the previous lemma into a well-ordered discretized schedule. This is the most crucial part of our approach. Roughly speaking, the transformation needs to guarantee that (1) in each zone, the volume of a job $j \in$ SLOW($\mathcal{J}'$) processed is an integer multiple of $v_j / (4n^2 \lceil 1/\delta \rceil)$ (this is tantamount to making sure that each zone has integral job pieces to deal with), (2) the job pieces start and end at the time points in W, and (3) all the job pieces are processed in the "right order". As we will show, the new schedule may run at a higher speed than the given lexicographically minimal YDS-extension, but not by too much.

Lemma 4 *Let OPT be a lexicographically minimal YDS-extension for $\mathcal{J}'$, and let $s_{\mathcal{S}}(t)$ denote the speed of schedule $\mathcal{S}$ at time t, for any $\mathcal{S}$ and t. Then there exists a well-ordered discretized schedule F, such that at any time point $t \in \mathcal{T}$, there holds*

$$s_F(t) \leq (1 + \delta)^3 s_{OPT}(t),$$

and for every $t \notin \mathcal{T}$, there holds

$$s_F(t) = s_{OPT}(t).$$

Proof Through a series of three transformations, we will transform OPT to a well-ordered discretized schedule F, while upper bounding the increase in speed caused by each of these transformations. More specifically, we will transform OPT to a schedule F_1 satisfying (i) and (iii) of Definition 3, then F_1 to F_2 where we slightly adapt the block lengths, and finally F_2 to F which satisfies all three properties of Definition 3. Each of these transformations can increase the speed by at most a factor $(1 + \delta)$ for any $t \in \mathcal{T}$ and does not affect the speed in any interval I_i.

Transformation 1 (OPT $\rightarrow F_1$): We will transform the schedule so that

(i) For each job $j \in \text{SLOW}(\mathcal{J}')$, an integer multiple of $v_j/(4n^2\lceil 1/\delta \rceil)$ volume of job j is processed in each zone, and the processing order of jobs within each zone is determined by $\prec$. Together with property 1 of Lemma 3, this implies that F_1 (considered as a schedule for pieces) satisfies Definition 3(i).

(ii) The well-ordered property of Definition 3 is satisfied.

(iii) For all $t \in \mathcal{T}$ it holds that $s_{F_1}(t) \leq (1+\delta)s_{OPT}(t)$, and for every $t \notin \mathcal{T}$ it holds that $s_{F_1}(t) = s_{OPT}(t)$.

Note that by Lemma 3, every zone is either empty, filled exactly by a job j_i^d, or contains a single block. For any job $j \in \text{SLOW}(\mathcal{J}')$, and every zone $[t_i, t_{i+1})$, let V_j^i be the processing volume of job j that OPT schedules in zone $[t_i, t_{i+1})$. Since there can be at most $2n$ different zones, for every job j there exists some index $h(j)$, such that $V_j^{h(j)} \geq v_j/(2n)$.

For every job $j \in \text{SLOW}(\mathcal{J}')$, and every $i \neq h(j)$, we reduce the load of job j processed in $[t_i, t_{i+1})$, by setting it to

$$\bar{V}_j^i = \left\lfloor V_j^i / \frac{v_j}{4n^2\lceil 1/\delta \rceil)} \right\rfloor \cdot \frac{v_j}{4n^2\lceil 1/\delta \rceil}.$$

Finally, we set the volume of j processed in $[t_{h(j)}, t_{h(j)+1})$ to $\bar{V}_j^{h(j)} = v_j - \sum_{i \neq h(j)} \bar{V}_j^i$. To keep the schedule feasible, we process the new volume of each non-empty zone $[t_i, t_{i+1}) \subseteq \mathcal{T}$ in the zone's original block B_i, at a uniform speed of $\sum_{j \in \text{SLOW}(\mathcal{J}')}(\bar{V}_j^i)/|B_i|$. Here, the processing order of the jobs within the block is determined by $\prec$.

Note that in the resulting schedule F_1, a job may be processed at different speeds in different zones, but each zone uses only one constant speed level.
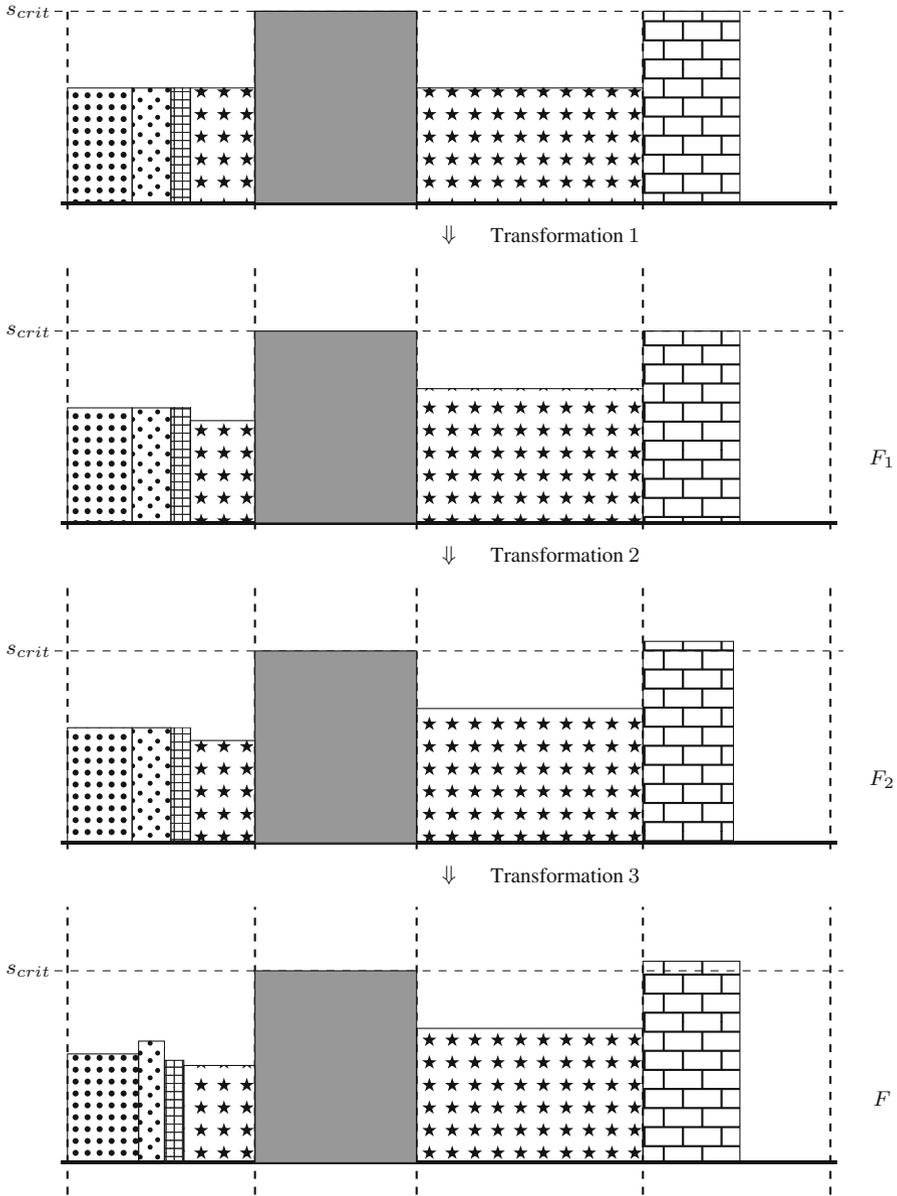
Fig. 2 An illustration of the three transformations. In Transformation 1, volume is shifted between the blocks so that each block contains an integer number of pieces for each job. Note how volume was shifted between the two star-patterned blocks. In Transformation 2, the block lengths are adapted, so that both block-endpoints align with W, see for instance brick-patterned block. Finally, in Transformation 3, some pieces within each block are acellerated while others decelerated in order to make sure that the execution of any piece starts and ends at a timepoint in W. See for example the leftmost pieces. We note that each of the three transformations is designed in a way so that no speed increases by more than an $(1 + \delta)$-factor

It is easy to see that F_1 is a feasible schedule in which for each job $j \in \text{SLOW}(\mathcal{J}')$, an integer multiple of $v_j/(4n^2\lceil 1/\delta \rceil)$ volume of j is processed in each zone, and that $\bar{\mathcal{V}}_j^i \leq V_j^i$ for all $i \neq h(j)$. Furthermore, if $i = h(j)$, we have that $\bar{\mathcal{V}}_j^i - V_j^i \leq v_j/(2n\lceil 1/\delta \rceil)$, and $V_j^i \geq v_j/(2n)$. It follows that $\bar{\mathcal{V}}_j^i \leq V_j^i + V_j^i/\lceil 1/\delta \rceil \leq (1+\delta)V_j^i$ in this case, and therefore $s_{F_1}(t) \leq (1+\delta)s_{OPT}(t)$ for all $t \in \mathcal{T}$. We note here, that for every job j_i^d, and the corresponding interval I_i, nothing changes during the transformation.

We finally show that F_1 satisfies the well-ordered property of Definition 3. Assume for the sake of contradiction that there exists a piece u ending at some t, and there exists a piece $u' \succ u$ with $d_{u'} \geq t$ that is scheduled before t. Recall that we can implicitly assume that the pieces of any fixed job are processed in the corresponding order $\prec$. Therefore $job(u') \succ job(u)$, by definition of the ordering $\prec$ among pieces. Furthermore, if $[t_k, t_{k+1})$ and $[t_{k'}, t_{k'+1})$ are the zones in which u and u', respectively, are scheduled, then $k' < k$, as $k' = k$ would contradict F_1's processing order of jobs inside a zone. Also note that $d_{u'} \geq t_{k+1}$, since $t \in (t_k, t_{k+1}]$, and (t_k, t_{k+1}) does not contain any deadline. This contradicts property 3 of Lemma 3, as the original schedule OPT must have processed some volume of $job(u')$ in $[t_{k'}, t_{k'+1})$, and some volume of $job(u)$ in $[t_k, t_{k+1})$.

Transformation 2 ($F_1 \rightarrow F_2$): In this transformation, we slightly modify the block lengths, as a preparation for Transformation 3. For every non-empty zone $[t_i, t_{i+1}) \subseteq \mathcal{T}$, we increase the uniform speed of its block until it has a length of $(1+\delta)^j \frac{1}{4n^2 s_{crit}(1+\delta)\lceil 1/\delta \rceil}$ for some integer $j \geq 0$, keeping one of its endpoints fixed at t_i or t_{i+1}. Note that in F_1, the block had length at least $\frac{1}{4n^2 s_{crit}(1+\delta)\lceil 1/\delta \rceil}$, since it contained a volume of at least $1/(4n^2\lceil 1/\delta \rceil)$, and the speed in this zone was at most $(1+\delta)s_{crit}$. The speedup needed for this modification is clearly at most $(1+\delta)$.

As this transformation does not change the processing order of any pieces nor the zone in which any piece is scheduled, it preserves the well-ordered property of Definition 3.

Transformation 3 ($F_2 \rightarrow F$): In this final transformation, we want to establish Definition 3(ii). To this end, we shift and compress certain pieces in F_2, such that every execution interval starts and ends at a time point from W (this is already true for pieces corresponding to jobs j_i^d). The procedure resembles a transformation done in [16]. For any zone $[t_i, t_{i+1}) \subseteq \mathcal{T}$, we do the following: Consider the pieces that F_2 processes within the zone $[t_i, t_{i+1})$, and denote this set of pieces by D_i. If $D_i = \emptyset$, nothing needs to be done. Otherwise, let γ be the integer such that $(1+\delta)^\gamma \frac{1}{4n^2 s_{crit}(1+\delta)\lceil 1/\delta \rceil}$ is the length of the block in this zone, and let

$$\Delta := \frac{1}{64n^8\lceil 1/\delta \rceil^3 s_{crit}(1+\delta)(1+\lceil 1/\delta \rceil)}.$$

Note that in the definition of W, we introduced $16n^6\lceil 1/\delta \rceil^2(1+\lceil 1/\delta \rceil)$ many time points (for $j = \gamma$ and $r = 1, \ldots, 16n^6\lceil 1/\delta \rceil^2(1+\lceil 1/\delta \rceil)$) that subdivide this block into $16n^6\lceil 1/\delta \rceil^2(1+\lceil 1/\delta \rceil)$ intervals of length Δ. Furthermore, since $|D_i| \leq 4n^3\lceil 1/\delta \rceil$,

there must exist a piece $u \in D_i$ with execution time $\Gamma_u \geq 4n^3\lceil 1/\delta\rceil(1+\lceil 1/\delta\rceil)\Delta$. We now partition the pieces in $D_i \setminus \{u\}$ into D^+, the pieces processed after u, and D^-, the pieces processed before u. First, we restrict our attention to D^+. Let $q_1, \ldots, q_{|D^+|}$ denote the pieces in D^+ in the order they are processed by F_2. Starting with the last piece $q_{|D^+|}$, and going down to q_1, we modify the schedule as follows. We keep the end of $q_{|D^+|}$'s execution interval fixed, and shift its start to the next earlier time point in W, reducing its uniform execution speed accordingly. At the same time, in order to avoid overlaps, we shift the execution intervals of all q_k, $k < |D^+|$ by the same amount to the left (leaving their lengths unchanged). Eventually, we also move the execution end point of u by the same amount to the left (leaving its start point fixed). This shortens the execution interval of u and "absorbs" the shifting of the pieces in D^+ (note that the processing speed of u increases as its interval gets shorter). We then proceed with $q_{|D^+|-1}$, keeping its end (which now already resides at a time point in W) fixed, and moving its start to the next earlier time point in W. Again, the shift propagates to earlier pieces in D^+, which are moved by the same amount, and shortens u's execution interval once more. When all pieces in D^+ have been modified in this way, we turn to D^- 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 further to the right. As before, u "absorbs" the propagated shifts, as we increase its start time accordingly. After this modification, the execution intervals of all pieces in D_i start and end at time points in W.

To complete the proof, we need to argue that the speedup of piece u is bounded by a factor $(1 + \delta)$. Since $|D_i| \leq 4n^3\lceil 1/\delta\rceil$, u's execution interval can be shortened at most $4n^3\lceil 1/\delta\rceil$ times, each time by a length of at most Δ. Furthermore, recall that the execution time of u was $\Gamma_u \geq 4n^3\lceil 1/\delta\rceil(1 + \lceil 1/\delta\rceil)\Delta$. Therefore, its new execution time is at least $\Gamma_u - 4n^3\lceil 1/\delta\rceil\Delta \geq \Gamma_u - \frac{\Gamma_u}{1+\lceil 1/\delta\rceil}$, and the speedup factor thus at most

$$\frac{\Gamma_u}{\Gamma_u - \frac{\Gamma_u}{1+\lceil 1/\delta\rceil}} = \frac{1}{1 - \frac{1}{1+\lceil 1/\delta\rceil}} \leq 1 + \delta.$$

Again, the transformation does not change the processing order of any pieces nor the zone in which any piece is scheduled, and thus preserves the well-ordered property of Definition 3. □

We now show that the speedup used in our transformation does not increase the energy consumption by more than a factor of $1 + \epsilon$. To this end, observe that for any $t \in \mathcal{T}$, the speed of the schedule OPT in Lemma 4 is at most s_{crit}, by Lemma 3(2). Furthermore, note that the final schedule F has speed zero whenever OPT has speed zero. This allows F to use exactly the same sleep phases as OPT (resulting in the same wake-up costs). It therefore suffices to prove the following lemma, in order to bound the increase in energy consumption.

Lemma 5 *For any $s \in [0, s_{crit}]$, there holds*

$$\frac{P\big((1 + \delta)^3 s\big)}{P(s)} \leq 1 + \epsilon.$$

Proof

$$\frac{P\big((1+\delta)^3 s\big)}{P(s)} \overset{(1)}{\leq} \frac{P\big((1+4\delta)s\big)}{P(s)}$$

$$= \frac{P(s) + 4\delta s \frac{P(s+4\delta s) - P(s)}{4\delta s}}{P(s)}$$

$$\overset{(2)}{\leq} \frac{P(s) + 4\delta s \frac{P(s+s_{crit}) - P(s)}{s_{crit}}}{P(s)}$$

$$\overset{(3)}{\leq} \frac{P(s) + 4\delta s \frac{P(2s_{crit}) - P(s_{crit})}{s_{crit}}}{P(s)}$$

$$\overset{(4)}{\leq} 1 + 4\delta \frac{s_{crit}}{P(s_{crit})} \cdot \frac{P(2s_{crit}) - P(s_{crit})}{s_{crit}}$$

$$\overset{(5)}{\leq} 1 + \epsilon.$$

In the above chain of inequalities, (1) holds since $\delta \leq \frac{1}{4}$ and $P(s)$ is non-decreasing. (2) and (3) follow from the convexity of $P(s)$, and the fact that $4\delta s \leq s_{crit}$. Inequality (4) holds since s_{crit} minimizes $P(s)/s$ (and thus maximizes $s/P(s)$), and (5) follows from the definition of δ. □

We summarize the major result of this section in the following lemma.

Lemma 6 *There exists a well-ordered discretized schedule with an energy consumption no more than $(1 + \epsilon)$ times the optimal energy consumption for $\mathcal{J}'$.*

4 The Dynamic Program

In this section, we show how to use dynamic programming to find a well-ordered discretized schedule with minimum energy consumption. In the following, we discuss only how to find the minimum energy consumption of this target schedule, as the actual schedule can be easily retrieved by proper bookkeeping in the dynamic programming process.

Recall that D is the set of all pieces and W the set of time points. Let $u_1, u_2, \ldots, u_{|D|}$ be the pieces in D, and w.l.o.g. assume that $u_1 \prec u_2 \prec \ldots \prec u_{|D|}$.

Definition 5 For any $k \in \{1, \ldots, |D|\}$, and $\tau_1 \leq \tau_2$, $\tau_1, \tau_2 \in W$, we define $E_k(\tau_1, \tau_2)$ as the minimum energy consumption during the interval $[\tau_1, \tau_2]$, of a well-ordered discretized schedule so that

1. all pieces $\{u \succeq u_k : \tau_1 < d_u \leq \tau_2\}$ are processed in the interval $[\tau_1, \tau_2)$, and
2. the machine is active right before τ_1 and right after τ_2.

In case that there is no such feasible schedule, let $E_k(\tau_1, \tau_2) = \infty$.

The DP proceeds by filling the entries $E_k(\tau_1, \tau_2)$ by decreasing index of k. The base cases are

$$E_{|D|+1}(\tau_1, \tau_2) := \min\{P(0)(\tau_2 - \tau_1), C\},$$

for all $\tau_1, \tau_2 \in W, \tau_1 \leq \tau_2$. For the recursion step, suppose that we are about to fill in $E_k(\tau_1, \tau_2)$. There are two possibilities.

- Suppose that $d_{u_k} \notin (\tau_1, \tau_2]$. Then clearly $E_k(\tau_1, \tau_2) = E_{k+1}(\tau_1, \tau_2)$.
- Suppose that $d_{u_k} \in (\tau_1, \tau_2]$. By definition, piece u_k needs to be processed in the interval $[\tau_1, \tau_2]$. We need to guess its actual execution period $[b, e) \subseteq [\tau_1, \tau_2]$, and process the remaining pieces $\{u \succeq u_{k+1} : \tau_1 < d_u \leq \tau_2\}$ in the two intervals $[\tau_1, b)$ and $[e, \tau_2)$. We first rule out some guesses of $[b, e)$ that are bound to be wrong.

  - By Definition 3(i), in a discretized schedule, a piece has to be processed completely inside a zone $[t_i, t_{i+1})$ (recall that $t_i \in W'$ are the release times and deadlines of the jobs). Therefore, in the right guess, the interior of $[b, e)$ does not contain any release times or deadlines; more precisely, there is no time point $t_i \in W'$ so that $b < t_i < e$.
  - By Definition 3(iii), in a well-ordered discretized schedule, if piece u_k ends at time point e, then all pieces $u' \succ u_k$ with deadline $d_{u'} \geq e$ are processed *after* u_k. However, consider the guess $[b, e)$, where $e = d_{u'}$ for some $u' \succ u_k$ (notice that the previous case does not rule out this possibility). Then u' cannot be processed anywhere in a well-ordered schedule. Thus, such a guess $[b, e)$ cannot be right.

By the preceding discussion, if the guess (b, e) is right, the two sets of pieces $\{u \succeq u_{k+1} : \tau_1 < d_u \leq b\}$ and $\{u \succeq u_{k+1} : e < d_u \leq \tau_2\}$, along with piece u_k, comprise all pieces to be processed that are required by the definition of $E_k(\tau_1, \tau_2)$. Clearly, the former set of pieces $\{u \succeq u_{k+1} : \tau_1 < d_u \leq b\}$ has to be processed in the interval $[\tau_1, b)$; the latter set of pieces, in a well-ordered schedule, must be processed in the interval $[e, \tau_2)$ if $[b, e)$ is the correct guess for the execution of the piece u_k.

We therefore have that

$$
\begin{aligned}
&E_k(\tau_1, \tau_2) \\
&= \min_{\substack{b,e \in W, \ [b,e) \subseteq [\tau_1, \tau_2), \\ [b,e) \subseteq [r_{u_k}, d_{u_k}), \\ \nexists t_i \in W', \ \text{s.t.} \ b < t_i < e, \\ \nexists u' \succ u_k, \ \text{s.t.} \ d_{u'} = e.}} \left\{ E_{k+1}(\tau_1, b) + P\left(\frac{v_{u_k}}{e-b}\right)(e-b) + E_{k+1}(e, \tau_2) \right\}
\end{aligned}
$$

if there exist $b, e \in W$ with the properties stated under the min-operator, and $E_k(\tau_1, \tau_2) = \infty$ otherwise.

Theorem 1 *There exists a fully polynomial-time approximation scheme (FPTAS) for speed scaling with sleep state. Its time complexity is $O(\frac{n^{32}}{\epsilon^{17}}(\log \frac{n^2 d_{\max}}{\epsilon})^4)$.*

Proof Given an arbitrary instance $\mathcal{J}$ for speed scaling with sleep state, we can transform it in polynomial time to an instance $\mathcal{J}'$, as seen in Sect. 2. We then apply the dynamic programming algorithm that was described in this section to obtain a well-ordered discretized schedule $\mathcal{S}'$ of minimal energy consumption for instance $\mathcal{J}'$. By

Lemma 6, we have that $\mathcal{S}'$ is a $(1+\epsilon)$-approximation for instance $\mathcal{J}'$. Furthermore, note that every discretized schedule (and therefore also $\mathcal{S}'$) executes each job j_i^d exactly in its allowed interval $I_i = [y_i, z_i)$. This holds because there are no time points from the interior of I_i included in W, and any discretized schedule must therefore choose to run j_i^d precisely from $y_i \in W$ to $z_i \in W$. Therefore, by Lemma 2, we can transform $\mathcal{S}'$ to a schedule $\mathcal{S}$ in polynomial time and obtain a $(1 + \epsilon)$-approximation for $\mathcal{J}$.

We next analyze the running time. The pre-processing of the instance and the YDS algorithm are easily dominated by the dynamic program. By construction in Sect. 3.1, $|W| = O(\frac{n^7}{\epsilon^4} \log \frac{n^2 d_{\max}}{\epsilon})$, $|W'| = O(n)$ and $|D| = O(\frac{n^3}{\epsilon})$. The total number of entries in the dynamic program is $O(|D||W|^2)$. For each single entry of $E_k(\tau_1, \tau_2)$, we need to check $O(\binom{|W|}{2})$ possibilities. For each possibility, we need $O(n)$ time. In sum, this gives the running time of $O(n|D||W|^4) = O(\frac{n^{32}}{\epsilon^{17}} (\log \frac{n^2 d_{\max}}{\epsilon})^4)$.   □

References

1. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
2. Albers, S., Antoniadis, A.: Race to idle: new algorithms for speed scaling with a sleep state. ACM Trans. Algorithms **10**(2), 9 (2014)
3. Antoniadis, A., Huang, C.-C., Ott, S.: A fully polynomial-time approximation scheme for speed scaling with sleep state. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015 (2015)
4. Bailis, P., Reddi, V.J., Gandhi, S., Brooks, D., Seltzer, M.I.: Dimetrodon: processor-level preventive thermal management via idle cycle injection. In: DAC, pp. 89–94. ACM (2011)
5. Bampis, E., Dürr, C., Kacem, F., Milis, I.: Speed scaling with power down scheduling for agreeable deadlines. Sustain. Comput.: Inf. Syst. **2**(4), 184–189 (2012)
6. Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., Nemparis, I.: From preemptive to non-preemptive speed-scaling scheduling. In: COCOON, pp. 134–146. Springer, New York (2013)
7. Bansal, N., Chan, H.-L., Katz, D., Pruhs, K.: Improved bounds for speed scaling in devices obeying the cube-root rule. Theory Comput. **8**(1), 209–229 (2012)
8. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. ACM Trans. Algorithms **9**(2), 18 (2013)
9. Baptiste, P.: Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In: SODA, pp. 364–367. ACM Press, New York (2006)
10. Baptiste, P., Chrobak, M., Dürr, C.: Polynomial-time algorithms for minimum energy scheduling. ACM Trans. Algorithms **8**(3), 26 (2012)
11. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors. IEEE Micro **20**(6), 26–44 (2000)
12. Demaine, E.D., Ghodsi, M., Hajiaghayi, M., Sayedi-Roshkhar, A.S., Zadimoghaddam, M.: Scheduling to minimize gaps and power consumption. J. Sched. **16**(2), 151–160 (2013)
13. Gandhi, A., Harchol-Balter, M., Das, R., Lefurgy, C.: Optimal power allocation in server farms. In: SIGMETRICS/Performance, pp. 157–168. ACM, New York (2009)
14. Garrett, M.: Powering down. ACM Queue **5**(7), 16–21 (2007)

15. Han, X., Lam, T.W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Deadline scheduling and power management for speed bounded processors. Theor. Comput. Sci. **411**(40–42), 3587–3600 (2010)
16. Huang, C., Ott, S.: New results for non-preemptive speed scaling. In: MFCS, pp. 360–371. Springer, New York(2014)
17. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News **36**(2), 63–76 (2005)
18. Irani, S., Shukla, S.K., Gupta, R.: Algorithms for power savings. ACM Trans. Algorithms **3**(4) (2007)
19. Kumar, G., Shannigrahi, S.: NP-hardness of speed scaling with a sleep state. CoRR (2013). arXiv:1304.7373
20. Raghavan, A., Emurian, L., Shao, L., Papaefthymiou, M.C., Pipe, K.P., Wenisch, T.F., Martin, M.M.K.: Utilizing dark silicon to save energy with computational sprinting. IEEE Micro **33**(5), 20–28 (2013)
21. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: FOCS, pp. 374–382. IEEE Computer Society (1995)

Affiliations

Antonios Antoniadis[1] · **Chien-Chung Huang[2]** · **Sebastian Ott[3]**

Chien-Chung Huang
cchuang@di.ens.fr

Sebastian Ott
se-ott@web.de

1   Saarland University and Max-Planck-Institut für Informatik, Saarbrücken, Germany

2   École Normale Supérieure, Paris, France

3   Max-Planck-Institut für Informatik, Saarbrücken, Germany