Theory Comput Syst (2008) 43: 67–80
DOI 10.1007/s00224-007-9070-1

Speed Scaling of Tasks with Precedence Constraints

Kirk Pruhs · Rob van Stee · Patchrawat Uthaisombut

Published online: 16 October 2007
© Springer Science+Business Media, LLC 2007

Abstract We consider the problem of speed scaling to conserve energy in a multiprocessor setting where there are precedence constraints between tasks, and where the performance measure is the makespan. That is, we consider an energy bounded version of the classic problem $Pm|prec|C_{\max}$. We extend the standard 3-field notation and denote this problem as $Sm|prec, energy|C_{\max}$. We show that, without loss of generality, one need only consider constant power schedules. We then show how to reduce this problem to the problem $Qm|prec|C_{\max}$ to obtain a poly-log(m)-approximation algorithm.

1 Introduction

1.1 Motivation

Power is now widely recognized as a first-class design constraint for modern computing devices. This is particularly critical for mobile devices, such as laptops, that

A preliminary version of this paper appears in Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA 2005).

Research of K. Pruhs supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, CCF-0448196, CCF-0514058, and IIS-0534531.
Research of R. van Stee supported by Alexander von Humboldt-Stiftung.

K. Pruhs · P. Uthaisombut
Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA

K. Pruhs
e-mail: kirk@cs.pitt.edu

P. Uthaisombut
e-mail: utp@cs.pitt.edu

R. van Stee (✉)
Fakultät für Informatik, Universität Karlsruhe, 76128 Karlsruhe, Germany
e-mail: vanstee@ira.uka.de

rely on batteries for energy. While the power consumption of devices has been growing exponentially, battery capacities have been growing at a (modest) linear rate. One common technique for managing power is speed/voltage/power scaling. For example, current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. The motivation for speed scaling as an energy saving technique is that, as the speed to power function $P(s)$ in all devices is strictly convex, less aggregate energy is used if a task is run at a slower speed. The application of speed scaling requires a policy/algorithm to determine the speed of the processor at each point in time. The processor speed should be adjusted so that the energy/power used is in some sense justifiable by the improvement in performance attained by running at this speed.

In this paper, we consider the problem of speed scaling to conserve energy in a multiprocessor setting where there are precedence constraints between tasks, and where the performance measure is the makespan, the time when the last task finishes. We will denote this problem by $Sm|prec, energy|C_{\max}$. Without speed scaling, this problem is denoted by $Pm|prec|C_{\max}$ in the standard three field scheduling notation [9]. Here m is the number of processors. This is a classic scheduling problem considered by Graham in his seminal paper [8] where he showed that list scheduling produces a $(2-\frac{1}{m})$-approximate solution. In our speed scaling version, we make a standard assumption that there is a continuous function $P(s)$, such that if a processor is run at speed s, then its power, the amount of energy consumed per unit time, is $P(s)=s^{\alpha}$, for some $\alpha>1$. For example, the well known cube-root rule for CMOS-based devices states that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s)=s^3$ (the power is proportional to the speed cubed) [4, 16]. Our second objective is to minimize the total energy consumed. Energy is power integrated over time. Thus we consider a bicriteria problem, in that we want to optimize both makespan and total energy consumption. Bicriteria problems can be formalized in multiple ways depending on how one values one objective in relationship to the other. We say that a schedule S is a b-energy c-approximate if the makespan for S is at most cM and the energy used is at most bE where M is the makespan of an optimal schedule which uses E units of energy. The most obvious approach is to bound one of the objective functions and optimize the other. In our setting, where the energy of the battery may reasonably be assumed to be fixed and known, it seems perhaps most natural to bound the energy used, and to optimize makespan.

Power management for tasks with precedence constraints has received some attention in computer systems literature, see for example [10, 14, 15, 20] and the references therein. These papers describe experimental results for various heuristics.

In the last few years, interest in power management has seeped over from the computer systems communities to the algorithmic community. For a survey of recent literature in the algorithmic community related to power management, see [11].

1.2 Summary of Our Results

For simplicity, we state our results when we have a single objective of minimizing makespan, subject to a fixed energy constraint, although our results are a bit more general.

We begin by noting that several special cases of $Sm|prec, energy|C_{\max}$ are relatively easy. If there is only one processor ($S1|prec, energy|C_{\max}$), then it is clear from the convexity of $P(s)$ that the optimal speed scaling policy is to run the processor at a constant speed; if there were times where the speeds were different, then by averaging the speeds one would not disturb the makespan, but the energy would be reduced. If there are no precedence constraints ($Sm|energy|C_{\max}$), then the problem reduces to finding a partition of the jobs that minimizes the ℓ_α norm of the load. A PTAS for this problem is known [1]. One can also get an $O(1)$-approximate constant-speed schedule using Graham's list scheduling algorithm. So for these problems, speed scaling doesn't buy you more than an $O(1)$ factor in terms of energy savings. Note that the $O(1)$ notation mentioned above means that the multiplicative factor is a constant that is independent of the input parameters even when they are taken into consideration.

We now turn to $Sm|prec, energy|C_{\max}$. We start by showing that there are instances where every schedule, in which all machines have the same fixed speed, has a makespan that is a factor of $\omega(1)$ more than the optimal makespan. The intuition is that if there are several jobs, on different processors, that are waiting for a particular job j, then j should be run with higher speed than if it were the case that no jobs were waiting on j. In contrast, we show that what should remain constant is the aggregate powers of the processors. That is, we show that in any locally optimal schedule, the sum of the powers at which the machines run is constant over time. If the cube-root rule holds (power equals speed cubed), this means the sum of cubes of the machines speeds should be constant over time. We call schedules with this property *constant power schedules*. We then show how to reduce our energy minimization problem to the problem of scheduling on machines of different speeds (without energy considerations). In the three field scheduling notation, this problem is denoted by $Q|prec|C_{\max}$. Using the $O(\log m)$-approximate algorithms from [5, 7], we can then obtain a $O(\log^2 m)$-energy $O(\log m)$-approximate algorithm for makespan for our problem. We then show a trade-off between energy and makespan for our problem. That is, an $O(b)$-energy $O(c)$-approximate schedule for makespan can be converted into $O(c \cdot b^{1/\alpha})$-approximate schedule. Thus we can then get an $O(\log^{1+2/\alpha} m)$-approximate algorithm for makespan.

We believe that the most interesting insight from these investigations is the observation that one can restrict one's attention to constant power schedules. This fact will also hold for several related problems.

1.3 Related Results

We will be brief here, and refer the reader to the recent survey [11] for more details. Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shankar [18]. They considered the problem of minimizing energy usage when each task has to be finished on one machine by a predetermined deadline. Most of the results in the literature to date focus on deadline feasibility as the measure for the quality of the schedule. Yao, Demers, and Shankar [18] give an optimal offline greedy algorithm. The running time of this algorithm can be improved if the jobs form a tree structure [13]. Bansal, Kimbrel, and Pruhs [3] and Bansal and Pruhs [2] extend the results in [18] on online algorithms and introduce the problem of speed

scaling to manage temperature. For jobs with a fixed priority, Yun and Kim [19] show that it is NP-hard to compute a minimum energy schedule. In this model, priorities of jobs are given as part of the input, and an available job with the highest priority should be run at any time. They also give an FPTAS for the problem. Kwon and Kim [12] give a polynomial-time algorithm for the case of a processor with discrete speeds. Chen, Kuo and Lu [6] give a PTAS for some special cases of this problem. Pruhs, Uthaisombut, and Woeginger [17] give some results on the flow time objective function.

2 Formal Problem Description

The setting for our problems consists of m variable-speed machines. If a machine is run at speed s, its power is $P(s) = s^\alpha$, $\alpha > 1$. The energy used by each machine is power integrated over time.

An instance consists of n jobs and an energy bound E. All jobs arrive at time 0. Each job i has an associated work (or size) w_i. If this job is run consistently at speed s, it finishes in w_i/s units of time. There are precedence constraints among the jobs. If $i \prec j$, then job j cannot start before job i completes.

Each job must be run non-preemptively on some machine. The machines can change speed continuously over time. Although it is easy to see by the convexity of $P(s)$ that it is best to run each job at a constant speed.

A *schedule* specifies, for each time and each machine, which job to run and at what speed. A schedule is *feasible at energy level* E if it completes all jobs and the total amount of energy used is at most E. Suppose S is a schedule for an input instance I. We define a number of concepts which depend on S. The completion time of job i is denoted C_i^S. The makespan of S, denoted $C_{\max}^S$, is the maximum completion time of any job. A schedule is *optimal for energy level* E if it has the smallest makespan among all feasible schedules at energy level E. The goal of the problem is to find an optimal schedule for energy level E. We denote the problem as $Sm|prec, energy|C_{\max}$.

We use s_i^S to denote the speed of job i. The execution time of i is denoted by x_i^S. Note that $x_i^S = w_i/s_i^S$. The power of job i is denoted by p_i^S. Note that $p_i^S = (s_i^S)^\alpha$. We use E_i^S to denote the energy used by job i. Note that $E_i^S = p_i^S x_i^S$. The total energy used in schedule S is denoted E^S. Note that $E^S = \sum_{i=1}^n E_i^S$. We drop the superscript S if the schedule is clear from the context.

3 No Precedence Constraints

As a warm-up, we consider the scheduling of tasks without precedence constraints. In this case we know that each machine will run at a fixed speed, since otherwise the energy use could be decreased without affecting the makespan by averaging the speed.

We may assume that there are at least as many jobs as there are machines. (If $m > n$, we simply ignore the last $m - n$ machines.) We then know that each machine will finish at the same time, since otherwise some energy from a machine

which finishes early could be transferred to machines which finish late, decreasing the makespan. Furthermore there will be no gaps in the schedule.

For any schedule, denote the makespan by M, and denote the load on machine j, which is the sum of the work of the jobs on machine j, by L_j. Since each machine runs at a fixed speed, in this section we denote by s_j the speed of *machine* j, by p_j its power, and by E_j its energy used. By our observations so far we have $s_j = L_j/M$.

The energy used by machine j is

$$E_j = p_j M = s_j^\alpha M = \frac{L_j^\alpha}{M^{\alpha-1}}.$$

We can sum this over all the machines and rewrite it as

$$M^{\alpha-1} = \frac{1}{E}\sum_j L_j^\alpha. \tag{1}$$

It turns out that minimizing the makespan is equivalent to minimizing the ℓ_α norm of the loads. For this we can use the PTAS for identical machines given in [1]. Denote the optimal loads by $\text{OPT}_1, \ldots, \text{OPT}_m$. Similarly to (1), we have

$$\text{OPT}^{\alpha-1} = \frac{1}{E}\sum_j \text{OPT}_j^\alpha, \tag{2}$$

where OPT is the optimal makespan. For any $\varepsilon > 0$, we can find loads $L_1, \ldots, L_m$ in polynomial time such that $\sum_j L_j^\alpha \le (1+\varepsilon)\sum_j \text{OPT}_j^\alpha$. For the corresponding makespan M it now follows from (1) and (2) that

$$M^{\alpha-1} = \frac{1}{E}\sum_j L_j^\alpha \le (1+\varepsilon)\cdot\frac{1}{E}\sum_j \text{OPT}_j^\alpha = (1+\varepsilon)\text{OPT}^{\alpha-1}$$

or

$$M \le (1+\varepsilon)^{1/(\alpha-1)}\text{OPT}.$$

Thus this gives us a PTAS for the problem $Sm|energy|C_{\max}$.

4 Main Results

4.1 One Speed for All Machines

In the remainder of the paper, we only consider the case with precedence constraints. Suppose all machines run at a fixed speed s. We show that under this constraint, it is not possible to get a good approximation of the optimal makespan. For simplicity, we only consider the special case $\alpha = 3$.

Consider the following input: one job of size $m^{1/3}$ and m jobs of size 1, which can only start after the first job has finished. Suppose the total energy available is $E = 2m$. It is possible to run the large job at a speed of $s_1 = m^{1/3}$ and all others

at a speed of 1. The makespan of this schedule is 2, and the total amount of energy required is $s_1^3 + m = 2m$.

Now consider an approximation algorithm with a fixed speed s. The total time for which this speed is required is the total size of all the jobs divided by s. Thus s must satisfy $s^3(m^{1/3} + m)/s \leq E = 2m$, or $s^2 \leq 2m/(m^{1/3} + m)$. This clearly implies $s \leq 2$, but then the makespan is at least $m^{1/3}/2$. Thus the approximation ratio is $\Omega(m^{1/3}) = \omega(1)$.

4.2 The Power Equality

To discuss the relationship among the powers of jobs in an optimal schedule, we need the following definitions. Given a schedule S of an input instance I, we define the *schedule-based constraint* $\prec_S$ among jobs in I as follows. For any jobs i and j, $i \prec_S j$ if and only if $i \prec j$ in I, or i runs before j on the same machine in S. Suppose S is a schedule where each job is run at a constant speed. The *power relation graph* of a schedule S of an instance I is a vertex-weighted directed graph $G = (V, E)$ created as follows:

- For each job i, create vertices u_i and v_i, each with weight p_i where p_i is the power at which job i is run. Vertex u_i corresponds to the *start* of job i. Vertex v_i corresponds to the *completion* of job i.
- In S, if $i \prec_S j$ and job j starts as soon as job i finishes (maybe on different machines), then create a directed edge (v_i, u_j).
- Two dummy vertices v_0 and u_{n+1} are added. In S, if job i starts at time 0, then create a directed edge (v_0, u_i). In S, if job i completes at time $C_{\max}^S$, then create a directed edge (v_i, u_{n+1}). Let $p_0 = \sum_{i:(v_0,u_i)\in E} p_i$, and let the weight of v_0 be p_0. Let $p_{n+1} = \sum_{i:(v_i,u_{n+1})\in E} p_i$, and let the weight of u_{n+1} be p_{n+1}.

Basically, the power relation graph G tells us which pairs of jobs on the same machine run back to back, and which pairs of jobs with precedence constraint $\prec$ between them run back to back. For an example, see Fig. 1.

In this paper, we define a connected component of a directed graph G to be a subgraph of G that corresponds to a connected component of the underlying *undirected* graph of G. Note that an isolated vertex will form a connected component by itself. Suppose C is a connected component of a power relation graph G. Define $H(C) = \{u|(v, u) \in C\}$ and $T(C) = \{v|(v, u) \in C\}$. Note that $H(C)$ and $T(C)$ is the set of vertices at the heads and tails, respectively, of directed edges in C. If C contains only one vertex, then $H(C) = T(C) = \emptyset$. The completion of jobs in $T(C)$ and the start of jobs in $H(C)$ all occur at the same time. This holds simply because for each edge (v_i, u_j) in C, the completion time of job i is the starting time of job j by definition of an edge. Travelling through all the edges of a component shows that all completions and starts occur at a common time. If time t is when this occurs, we say that C *occurs at time* t. We say that a connected component C satisfies the *power equality* if

$$\sum_{i:u_i\in H(C)} p_i = \sum_{i:v_i\in T(C)} p_i.$$

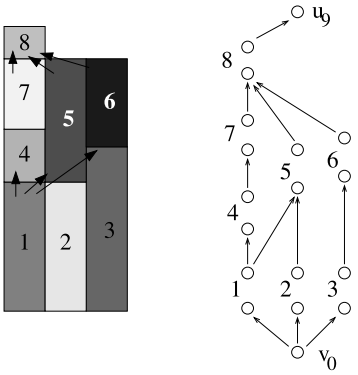

Fig. 1 An example of a schedule and the corresponding power relation graph. In the schedule on the *left*, the *arrows* denote precedence constraints between jobs. Note that the precedence constraint between jobs 1 and 6 is not represented in the power relation graph. However, in the power relation graph there is an edge between jobs 2 and 5 since they run back to back on the same machine. In this example, the graph has six connected components

Note that p_i is the power at which job i is run, and is also the weight of vertices u_i and v_i. We say that a power relation graph G satisfies the *power equality* if each connected component of G has at least one edge, and each connected component of G satisfies the power equality. We now need to establish some properties of optimal schedules. The following observation is an obvious consequence of the convexity of the speed to power function.

Observation 1 *If S is an optimal schedule for some energy level E, then each job is run at a constant speed. This also implies that each job is run at a constant power.*

Lemma 2 *If S is an optimal schedule for some energy level E, then in the power relation graph G of S, each component contains at least one edge.*

Proof Let C be any connected component of the power relation graph G of an optimal schedule S. Assume to reach a contradiction that C contains no edges, that is, C contains only one vertex x. Let t be the time in S corresponding to the occurrence of C. Vertex x either corresponds to the start of some job i ($x = u_i$), or the completion of some job i ($x = v_i$).

If $x = u_i$ for some job i, then x corresponds to the start of some job i. Since there is no edge incident to u_i, then no jobs complete at time t. Thus, the machine that job i runs on is idle right before time t. We can modify the schedule S by starting job i earlier and running job i at a slower speed without violating the precedence constraints. Slowing down the job reduces the energy used. The energy saved could be reinvested elsewhere to get a better makespan. This contradicts the fact that S is optimal.

If $x = v_i$ for some job i, then x corresponds to the completion of job i. Since there is no edge incident from v_i, then no jobs start at time t. Thus, the machine that job i runs on is idle right after time t. We can modify the schedule S by running job i at a slower speed so that it completes later without violating the precedence constraints. Also this does not increase the makespan because job i could not be the last job to finish from the construction of G. Slowing down the job reduces the energy used. The energy saved could be reinvested elsewhere to get a better makespan. This contradicts the fact that S is optimal. □

Lemma 3 *If S is an optimal schedule for some energy level E, then the power relation graph G of S satisfies the power equality.*

Proof Let G be the power relation graph of an optimal schedule S. From Lemma 2, every component of G contains at least one edge. Thus, it only remains to show that each component of G satisfies the power equality. The idea of the proof is to consider an arbitrary component C of G. Then create a new schedule S' from S by slightly stretching and compressing jobs in C. Since S is optimal, S' cannot use a smaller amount of energy. By creating an equality to represent this relationship and solving it, we have that C must satisfy the power equality.

Now we give the details. C contains at least two vertices by Lemma 2. Let $\varepsilon \neq 0$ be a small number such that $x_i + \varepsilon > 0$ for any job i in $T(C)$, and $x_i - \varepsilon > 0$ for any job i in $H(C)$. Note that we allow ε to be either positive or negative.

We create a new schedule S' by modifying schedule S in the following manner. Increase the execution time of every job in $T(C)$ by ε, and decrease the execution time of every job in $H(C)$ by ε. All other jobs are unchanged. Note the following:

(1) The execution time of job i in $T(C)$ in S' is positive because $x_i + \varepsilon > 0$.
(2) The execution time of job i in $H(C)$ in S' is positive because $x_i - \varepsilon > 0$.
(3) For $|\varepsilon|$ small enough, S' has the same power relation graph as S.

In particular, we choose ε such that $|\varepsilon|$ is less than the smallest difference between two successive times at which connected components occur (i.e., at which the set of jobs being executed changes). Therefore, S' is a feasible schedule having the same power relation graph as S. Observe that the makespan of S' remains the same as that of S. All that has changed is the timing of some inner changeover point.

As an example, in Fig. 1 we might take the connected component consisting of v_1, v_2, u_4, u_5. Changing the execution times in this component as described above means that the horizontal line between jobs 1 and 2 and jobs 4 and 5 gets moved slightly up or down, without affecting the rest of the schedule. Our restriction (3) on ε means that this line is not for instance moved above the starting point of job 6, which would violate a precedence constraint and give an infeasible schedule.

The change in the energy used, $\Delta E(\varepsilon)$, is

$$\begin{aligned}\Delta E(\varepsilon) &= E^{S'} - E^{S} \\ &= \sum_{i:v_i \in T(C)} \left(E_i^{S'} - E_i^{S}\right) + \sum_{i:u_i \in H(C)} \left(E_i^{S'} - E_i^{S}\right) \\ &= \sum_{i:v_i \in T(C)} \left(\frac{w_i^{\alpha}}{(x_i+\varepsilon)^{\alpha-1}} - \frac{w_i^{\alpha}}{x_i^{\alpha-1}}\right) + \sum_{i:u_i \in H(C)} \left(\frac{w_i^{\alpha}}{(x_i-\varepsilon)^{\alpha-1}} - \frac{w_i^{\alpha}}{x_i^{\alpha-1}}\right).\end{aligned}$$

Since S is optimal, $\Delta E(\varepsilon)$ must be non-negative. Otherwise, we could reinvest the energy saved by this change to obtain a schedule with a better makespan. Since the derivative $\Delta E'(\varepsilon)$ is continuous for $|\varepsilon|$ small enough, we must have $\Delta E'(0) = 0$. We have

$$\Delta E'(\varepsilon) = \sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^{\alpha}}{(x_i+\varepsilon)^{\alpha}} + \sum_{i:u_i \in H(C)} \frac{(\alpha-1)w_i^{\alpha}}{(x_i-\varepsilon)^{\alpha}}.$$

Substitute $\varepsilon = 0$ and solve for $\Delta E'(0) = 0$.

$$\begin{aligned}
\Delta E'(0) &= 0,\\
\sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} - \sum_{i:u_i \in H(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} &= 0,\\
\sum_{i:v_i \in T(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha} &= \sum_{i:u_i \in H(C)} \frac{(1-\alpha)w_i^\alpha}{x_i^\alpha},\\
\sum_{i:v_i \in T(C)} s_i^\alpha &= \sum_{i:u_i \in H(C)} s_i^\alpha,\\
\sum_{i:v_i \in T(C)} p_i &= \sum_{i:u_i \in H(C)} p_i.
\end{aligned}$$

Thus, this connected component C satisfies the power equality. Since C is an arbitrarily chosen connected component in G, then G satisfies the power equality, and the result follows. □

Note that the above proof also establishes that the power equality must also hold for any schedule that locally optimal schedule with respect to the change considered in the proof.

Let $p_i(t)$ be the power at which job j runs at time t. Let $p(k,t)$ be the power at which machine k runs at time t. By convention if job i starts at time t_1 and completes at time t_2, we say that it runs in the close-open interval $[t_1, t_2)$. If a job has just finished at time t and another has just start at time t on machine k, then $p(k,t)$ is equal to the power of the *starting* job. We will use $p(k,t^-)$ to denote the power of the *completing* job. Also by convention, if no job is running at time t on machine k, then $p(k,t) = 0$.

Lemma 4 *If S is an optimal schedule for some energy level E, there exists a constant p such that at any time t, $\sum_{k=1}^m p(k,t) = p$, i.e. the sum of the powers of all machines at time t is p.*

Proof Suppose S is an optimal schedule. Let $t_0 = 0$. For $i \geq 1$, let t_i be the earliest time, if it exists, strictly after t_{i-1} at which some job completes or starts. Suppose t_l is the completion time of the last job. For $i = 0, \ldots, l-1$ and for any time t' such that $t_i < t' < t_{i+1}$, we will show that

$$\sum_{k=1}^m p(k,t_i) = \sum_{k=1}^m p(k,t') \quad \text{and} \tag{3}$$

$$\sum_{k=1}^m p(k,t_i) = \sum_{k=1}^m p(k,t_{i+1}). \tag{4}$$

If this is the case, then the result follows.

Let i be an index such that $0 \leq i \leq l-1$. Let t' be any time such that $t_i < t' < t_{i+1}$. We now prove (3). Since no jobs start or complete in the interval $(t_i, t']$, then the same

set of jobs are running at time t and t'. By Observation 1, each job runs at a constant speed at all time. This also means that each job runs at a constant power at all time. Thus, (3) follows.

We now prove (4). Let A be the set of jobs that are running (or have just started) at time t_i. Let B be the set of jobs that are running (or have just started) at time t_{i+1}. Since no jobs start or finish during (t_i, t_{i+1}), then $A - B$ is the set of jobs that completes at time t_{i+1}, $B - A$ is the set of jobs that starts at time t_{i+1}, and $A \cap B$ is the set of jobs that has been running since time t_i (or earlier) and until after t_{i+1}. If X is a set of jobs, then let $M(X)$ be the set of machines on which jobs in X run.

$$\begin{aligned}
\sum_{k=1}^{m} p(k, t_i) &= \sum_{j \in A} p_j(t_i) \\
&= \sum_{j \in A-B} p_j(t_i) + \sum_{j \in A \cap B} p_j(t_i) \\
&= \sum_{j \in A-B} p_j(t_{i+1}^-) + \sum_{j \in A \cap B} p_j(t_{i+1}) \quad \text{by the same argument as (3)} \\
&= \sum_{k \in M(A-B)} p(k, t_{i+1}^-) + \sum_{j \in A \cap B} p_j(t_{i+1}) \\
&= \sum_{k \in M(B-A)} p(k, t_{i+1}) + \sum_{j \in A \cap B} p_j(t_{i+1}) \quad \text{from Lemma 3} \\
&= \sum_{j \in B-A} p_j(t_{i+1}) + \sum_{j \in A \cap B} p_j(t_{i+1}) \\
&= \sum_{j \in B} p_j(t_{i+1}) \\
&= \sum_{k=1}^{m} p(k, t_{i+1}).
\end{aligned}$$

□

4.3 Algorithm

Lemma 4 implies that the total power at which all the machines run is constant over time (only the distribution of the power over the machines may vary). We will describe a scheme to use this lemma to relate $Sm|prec, energy|C_{\max}$ to the problem $Q|prec|C_{\max}$. Then, we can use an approximation algorithm for the latter problem given in [5] to obtain an approximate schedule. The schedule is then scaled so that the total amount of energy used is within the energy bound E.

Let $\bar{p}$ be the sum of powers at which the machines run in the optimal schedule $\text{OPT}(I, E)$. Since energy is power times makespan, we have $\bar{p} = E/\text{OPT}(I, E)$. However, an approximation algorithm does not know the value of $\text{OPT}(I, E)$, so it cannot immediately compute $\bar{p}$. Nevertheless, we will assume that we know the value of $\bar{p}$. The value of $\bar{p}$ can be approximated using binary search, and this will be discussed

later. Given $\bar{p}$, define the set $M(\bar{p})$ to consist of the following *fixed speed* machines: 1 machine running at power $\bar{p}$, 2 machines running at power $\bar{p}/2$, and in general 2^i machines running at power $\bar{p}/2^i$ for $i = 0, 1, \ldots, \lfloor \log(m+1) \rfloor - 1$. Denoting the total number of machines so far by m', there are an additional $m - m'$ machines running at power $\bar{p}/2^{\lfloor \log(m+1) \rfloor}$. Thus there are m machines in the set $M(\bar{p})$, but the total power is at most $(\log m + 1)\bar{p}$. We show in the following lemma that if the optimal algorithm is given the choice between m variable speed machines with total energy E and the set $M(\bar{p})$ of machines just described, where it is allowed to use preemptions, it will always take the latter, since the makespan will be smaller.

Lemma 5 *We have*

$$\textsc{PreemptiveOpt}_{M(\bar{p})}(I) \leq \textsc{opt}(I, E),$$

where $\textsc{PreemptiveOpt}_{M(\bar{p})}(I)$ *is the makespan of the optimal preemptive schedule using fixed speed machines in the set* $M(\bar{p})$, *and* $\textsc{opt}(I, E)$ *is the makespan of the optimal schedule using* m *variable-speed machines with energy bound* E.

Proof In an abuse of notation, we let $\textsc{PreemptiveOpt}_{M(\bar{p})}(I)$ and $\textsc{opt}(I, E)$ refer both to the makespans of the two optimal schedules and those respective schedules themselves. We will create a preemptive schedule S using fixed speed machines in the set $M(\bar{p})$. We will consider each time t and assign jobs in $\textsc{opt}(I, E)$ to machines in S. We will show that the assignment can be feasibly done. We abuse the notation by using S to refer to the makespan of schedule S. Thus, $\textsc{PreemptiveOpt}_{M(\bar{p})}(I) \leq S \leq \textsc{opt}(I, E)$.

Consider any time t in $\textsc{opt}(I, E)$. Denote the power of machine k of $\textsc{opt}(I, E)$ at this time by P_k. Suppose the machines are labeled so that $P_1 \geq P_2 \geq \cdots \geq P_m$. Now we simply assign the job on machine 1 to the machine of power $\bar{p}$ in S. And for $i \geq 1$ we assign the jobs on machines $2^i, \ldots, 2^{i+1} - 1$ to the machines of power $\bar{p}/2^i$ in S.

Clearly, $P_1 \leq \bar{p}$, since no machine can use more than $\bar{p}$ power at any time. In general, we have that $P_j \leq \bar{p}/j$ for $j = 1, \ldots, m$. If we can show that the first machine in any power group has at least as much power as the corresponding machine of $\textsc{opt}(I, E)$, this holds for all the machines. But since machine 2^i in S has power exactly $\bar{p}/2^i$, this follows immediately.

It follows that S allocates each individual job at least as much power as $\textsc{opt}(I, E)$ at time t. We can apply this transformation for any time t, where we only need to take into account that S might finish some jobs earlier than $\textsc{opt}(I, E)$. So the schedule for S might contain unnecessary gaps, but it is a valid schedule, at least when we allow preemptions. This proves the lemma. □

To construct an approximate schedule, we assume the value of $\bar{p}$ is known, and the set of fixed speed machines in $M(\bar{p})$ will be used. The schedule is created using the algorithm given in [5]. The schedule created may use too much energy. To fix this, the speeds of all jobs are decreased so that the total energy used is within E at the expense of having a longer makespan. The steps are given in subroutine *FindSchedule* in Fig. 2.

FindSchedule(I, p)

1. Find a schedule for instance I and machines in the set $M(p)$ using the algorithm in [5].
2. Reduce the speed of all machines by a factor of $\log^{2/\alpha} m$.
3. Return the resulting schedule.

ALG(I, E)

1. Set $p^* = (\frac{E}{W})^{\frac{\alpha}{\alpha-1}}$ where W is the total work of all jobs.
2. Using binary search on $[0, p^*]$ with p as the search variable, find the largest value for p such that this 2-step process returns true. Binary search terminates when the binary search interval is shorter than 1.
   (a) Call *FindSchedule*(I, p).
   (b) If for the schedule obtained we have $\sum_{i=1}^{n} s_i^{\alpha-1} w_i \leq E$, return true.

Fig. 2 Our speed scaling algorithm. The input consists a set of jobs I and an energy bound E

4.4 Analysis

Lemma 6 *Suppose* $p = E/\text{OPT}(I, E)$. *Subroutine FindSchedule*(I, p) *creates a schedule which has makespan* $O(\log^{1+2/\alpha} m)\text{OPT}(I, E)$ *and uses energy* $O(E)$.

Proof Let S_1 and S_2 denote the schedules obtained in steps 1 and 2 of the subroutine *FindSchedule*(I, p), respectively. Schedule S_2 is the one returned by *FindSchedule*. First we analyze the makespan.

From the results in [5], $C_{\max}^{S_1} = O(\log m)\text{PREEMPTIVEOPT}_{M(p)}(I)$. This holds because although their algorithm does not use preemptions, it has this approximation ratio even when compared against an optimal preemptive algorithm. In step 2, the speed of every job decreases by a factor of $\log^{2/\alpha} m$. Thus, the makespan increases by a factor of $\log^{2/\alpha} m$. From Lemma 5, $\text{PREEMPTIVEOPT}_{M(p)}(I) \leq \text{OPT}(I, E)$. Therefore, taken together, we have

$$\begin{aligned} C_{\max}^{S_2} &= (\log^{2/\alpha} m) C_{\max}^{S_1} = (\log^{2/\alpha} m) O(\log m)\text{PREEMPTIVEOPT}_{M(p)}(I) \\ &= O(\log^{1+2/\alpha} m)\text{OPT}(I, E). \end{aligned}$$

Next we analyze the energy. The machines in the schedule $\text{OPT}(I, E)$ run for $\text{OPT}(I, E)$ time units at the total power of $p = E/\text{OPT}(I, E)$ consuming a total energy of E. Recall that if all machines in $M(p)$ are busy, the total power is at most $p(1 + \log m)$.

Schedule S_1 runs the machines for $O(\log m)\text{PREEMPTIVEOPT}_{M(p)}(I)$ time units at the total power at most $p(1 + \log m)$. Thus, it uses energy at most

$$\begin{aligned} &p(1 + \log m) O(\log m)\text{PREEMPTIVEOPT}_{M(p)}(I) \\ &\leq O(\log^2 m) p\text{OPT}(I, E) = O(\log^2 m) E, \end{aligned} \tag{5}$$

where the inequality follows from Lemma 5. The speeds at which the machines in S_2 run are $\log^{2/\alpha} m$ slower than those in $M(p)$, which S_1 uses. Thus, the total power at which the machines in S_2 run is $\log^2 m$ times smaller than that of S_1. By (5), this is $O(E)$. □

Note that when we decrease the speed in S_2 by some constant factor, the makespan increases by that factor and the energy decreases by a larger constant factor. To find the value of $\bar{p}$, we use binary search in the interval $[0, p^*]$ where p^* is an initial upper bound to be computed shortly. We continue until the length of the interval is at most 1. We then use the left endpoint of this interval as our power. Now we compute the initial upper bound p^*. For a given schedule, the total energy used is

$$\sum_{i=1}^{n} p_i x_i = \sum_{i=1}^{n} s_i^{\alpha} w_i / s_i = \sum_{i=1}^{n} s_i^{\alpha-1} w_i.$$

The best scenario that could happen for the optimal algorithm is when the work is evenly distributed on all the machines and all the machines run at the same speed at all time. Let W be the total work of all the jobs. Completing x units of work at a speed of s requires $s^{\alpha-1}x$ units of energy. If each of the m machines processes W/m units of work, then it takes a total $Ws^{\alpha-1}$ units of energy. This must be less than E. For the speed we find $s^{\alpha-1} \le E/W$ and thus $p^{\frac{\alpha-1}{\alpha}} \le E/W$. This gives us an initial upper bound for p for the binary search:

$$p \le p^* = \left(\frac{E}{W}\right)^{\frac{\alpha}{\alpha-1}}.$$

OPT does not use a higher power than this, because then it would run out of energy before all jobs complete.

From Lemma 6 and our analysis above, the following theorem holds.

Theorem 7 ALG *is an* $O(\log^{1+2/\alpha} m)$*-approximation algorithm for the problem* $Sm|prec, energy|C_{\max}$ *where the power is equal to the speed raised to the power of* α *and* $\alpha > 1$.

References

1. Alon, N., Azar, Y., Woeginger, G., Yadid, T.: Approximation schemes for scheduling on parallel machines. J. Sched. **1**(1), 55–66 (1998)
2. Bansal, N., Pruhs, K.: Speed scaling to manage temperature. In: Symposium on Theoretical Aspects of Computer Science, pp. 460–471 (2005)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. In: IEEE Symposium on Foundations of Computer Science, pp. 520–529 (2004)
4. 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)
5. Chekuri, C., Bender, M.A.: An efficient approximation algorithm for minimizing makespan on uniformly related machines. J. Algorithms **41**, 212–224 (2001)

6. Chen, J.-J., Kuo, T.-W., Lu, H.-I.: Power-saving scheduling for weakly dynamic voltage scaling devices. In: Workshop on Algorithms and Data Structures, pp. 338–349 (2005)
7. Chudak, F.A., Shmoys, D.B.: Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. J. Algorithms **30**(2), 323–343 (1999)
8. Graham, R.L.: Bounds for certain multiprocessor anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
9. Graham, R.L., Lawler, E., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979)
10. Gruian, F., Kuchcinski, K.: Lenes: task-scheduling for low-energy systems using variable voltage processors. In: Asia South Pacific—Design Automation Conference, pp. 449–455 (2001)
11. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News **32**(2), 63–76 (2005)
12. Kwon, W.-C., Kim, T.: Optimal voltage allocation techniques for dynamically variable voltage processors. ACM Trans. Embed. Comput. Syst. (TECS) **4**(1), 211–230 (2005)
13. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. In: 11th International Computing and Combinatorics Conference (COCOON 2005), pp. 283–296 (2005)
14. Luo, J., Jha, N.K.: Power-conscious joint scheduling of periodic task graphs and aperiodic task graphs in distributed real-time embedded systems. In: International Conference on Computer Aided Design, pp. 357–364 (2000)
15. Mishra, R., Rastogi, N., Zhu, D., Mossé, D., Melhem, R.G.: Energy aware scheduling for distributed real-time systems. In: International Parallel and Distributed Processing Symposium, p. 21 (2003)
16. Mudge, T.: Power: a first-class architectural design constraint. Computer **34**(4), 52–58 (2001)
17. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. In: Scandinavian Workshop on Algorithms and Theory, pp. 14–25 (2004)
18. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced cpu energy. In: IEEE Symposium on Foundations of Computer Science (FOCS 1995), pp. 374–382 (1995)
19. Yun, H.-S., Kim, J.: On energy-optimal voltage scheduling for fixed priority hard real-time systems. ACM Trans. Embed. Comput. Syst. **2**(3), 393–430 (2003)
20. Zhang, Y., Hu, X., Chen, D.Z.: Task scheduling and voltage selection for energy minimization. In: Design Automation Conference, pp. 183–188 (2002)