Speed Scaling of Tasks with Precedence Constraints

Kirk Pruhs[1,⋆], Rob van Stee[2], and Patchrawat "Patch" Uthaisombut[1]

[1] Computer Science Department, University of Pittsburgh,
Pittsburgh PA 15260 USA.
{kirk,utp}@cs.pitt.edu
[2] Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany.
vanstee@ira.uka.de.

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 $Pm \mid prec \mid C_{max}$. 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 $Qm \mid prec \mid C_{max}$ 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 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

⋆ Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, CCF-0448196, and CCF-0514058.

task finishes. We will denote this problem by $Sm \mid prec \mid C_{max}$. Without speed scaling, this problem is denoted by $Pm \mid prec \mid C_{max}$ 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) [16, 4]. 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 $O(a)$-energy $O(b)$-approximate if the makespan for S is at most bM and the energy used is at most aE 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, 20, 15] 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]. Research on algorithmic issues in power management is still at an early stage of development. Researchers are developing and analyzing algorithms to problems that appear particularly natural and/or that arise in some particular application. The eventual goal, after developing algorithms and analyses for many problems, is to develop a toolkit of widely applicable algorithmic methods for power management problems. While the algorithms and analyses that we present here are not extremely deep, we believe that our insights and techniques are quite natural, and have significant potential for future application in related problems.

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 $Sm \mid prec \mid C_{max}$ are relatively easy. If there is only one processor ($S1 \mid prec \mid C_{max}$), 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 ($Sm \mid\mid C_{max}$), 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.

We now turn to $Sm \mid prec \mid C_{max}$. 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. Or equivalently, if the cube-root rule holds (power equals speed cubed), the sum of cubes of the machines speeds should be constant over time. Schedules with this property are called *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 $Q \mid prec \mid C_{max}$. 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. We then show a trade-off between energy and makespan. That is, an $O(a)$-energy $O(b)$-approximate schedule for makespan can be converted into $O(b \cdot a^{1/\alpha})$-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 [2] and Bansal and Pruhs [3] 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. 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 weight (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 $Sm \mid prec \mid C_{\max}$.

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 also 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 weights 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 by Azar et al. [1]. Denote the optimal loads by $\textsc{opt}_1, \ldots, \textsc{opt}_m$. Similarly to (1), we have

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

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

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

or

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

Thus this gives us a PTAS for the problem $Sm \mid\mid C_{\max}$. For $\alpha > 2$, it even gives a better approximation for any fixed running time compared to the original PTAS.

4 Main results

4.1 One speed for all machines

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 $s^3(m^{1/3}+m)/s \le E = 2m$, or $s^2 \le 2m/(m^{1/3}+m)$. This clearly implies $s \le 2$, but then the makespan is at least $m^{1/3}/2$. Thus the approximation ratio is $\Omega(m^{1/3})$.

In contrast, we will use machines that have different speeds, but where the total power used by the machines is constant over time.

4.2 The power equality

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 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.
- 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).

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.

In this paper, a connected component of a directed graph G refers to a subgraph of G that corresponds to a connected component of the underlying undirected graph of G. Suppose C is a connected component of a power relation graph G. Define $H(C) = \{u \mid (v, u) \in C\}$ and $T(C) = \{v \mid (v, u) \in C\}$. 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. 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

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

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 all connected components of G satisfy the power equality.

Lemma 1. *If a schedule S is optimal, then each job is run at a constant speed.*

Proof (Proof sketch). Suppose S is an optimal schedule such that some job i does not run at a constant speed. By averaging the speeds in the interval that i runs, the execution time of i would not change, but the energy would be reduced, since the power is a convex function of the speed. A contradiction. □

Lemma 2. *If S is an optimal schedule for some energy level E, then the power relation graph G of S satisfies the power equality.*

Proof. The idea of the proof is to consider an arbitrary component C of the power relation graph G of an optimal schedule S. 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 relation.

Now we give the detail. Consider any connected component C of G. If C contains only one vertex, then it immediately follows that C satisfies the power equality because $T(C) = H(C) = \emptyset$. Therefore, suppose C contains two or more vertices. 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. For simplicity on the first reading, it is easy to think of ε as a small positive number. We create a new schedule S' from schedule S by increasing the execution time of every job in $T(C)$ by ε, and decreasing the execution time of every job in $H(C)$ by ε. 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.
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. 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. □

Let $p(k,t)$ be the power at which machine k runs at time t. By convention, if machine k is idle at time t, then $p(k,t) = 0$. Also, by convention if job i starts at time t_1 and completes at time t_2, we say that it runs in the open-close interval $(t_1, t_2]$. Therefore, $p(k,t)$ is well-defined at a time t when a job has just finished and another has just started; the value of $p(k,t)$ is equal to the power of the finishing job.

Lemma 3. *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. Consider any time t where $0 < t \leq C_{\max}^{S}$. Let t' be any time after t such that no jobs start or complete in the interval $(t, t']$. Note that this does not exclude the possibility that some jobs start or complete at time t. We will show that $\sum_{k=1}^{m} p(k,t) = \sum_{k=1}^{m} p(k,t')$. If this is the case, then the result follows.

On the one hand, if no jobs start or complete at time t, then the same set of jobs are running at time t and t'. From Lemma 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. Since the same set of jobs are running at time t and t', then $\sum_{k=1}^{m} p(k,t) = \sum_{k=1}^{m} p(k,t')$.

On the other hand, if some jobs start or complete at time t, then consider the power relation graph G of S. The jobs which start or complete at time t correspond to vertices in components occurring at time t. Note that if some component contains only one vertex, then S is not optimal, because the corresponding job could be run at a slower speed and start earlier (if it starts at time t) or finish later without violating any precedence constraints. This would reduce the total amount of energy used, which could be reinvested elsewhere to get a better schedule. Thus all components contain at least one edge. From Lemma 2, the sum of powers of jobs that complete at time t is equal to the sum of powers of jobs that start at time t. And since no jobs start or complete in the interval $(t, t']$, then again $\sum_{k=1}^{m} p(k,t) = \sum_{k=1}^{m} p(k,t')$. □

4.3 Algorithm

Lemma 3 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 $Sm \mid prec \mid C_{\max}$ to the problem $Q \mid prec \mid C_{\max}$. Then, we can use an approximation algorithm for the latter problem by Chekuri and Bender [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 $\textsc{opt}(I, E)$. Since energy is power times makespan, we have $\bar{p} = E/\textsc{opt}(I, E)$. However, an approximation algorithm does not know the value of $\textsc{opt}(I, E)$, 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-1} machines running at power $\bar{p}/i$ for $i = 1, 2, ..., \lfloor \log m \rfloor$. Denoting the total number of machines so far by m', there are an additional $m - m'$ machines running at power $\bar{p}/(1 + \lfloor \log m \rfloor)$. Thus there are m machines in the set $M(\bar{p})$, but the total power is at most $\bar{p}(1 + \log m)$. 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, it will always take the latter, since the makespan will be smaller.

Lemma 4. *We have*

$$\text{OPT}_{M(\bar{p})}(I) \leq \text{OPT}(I, E),$$

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

Proof. Consider the schedule of OPT with variable speed machines and energy bound E at some time t. Denote this OPT by OPT_1 and the OPT which uses the prescribed set of machines by OPT_2. Denote the power of machine i of OPT_1 at this time by p_i (note that this is a different notation from the one we use elsewhere in the paper) and sort the machines by decreasing p_i. Now we simply assign the job on machine 1 to the machine of power $\bar{p}$ of OPT_2, and for $i \geq 1$ we assign the jobs on machines $2^i, \ldots, 2^{i+1} - 1$ to the machines of power $\bar{p}/2^i$ of OPT_2.

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 OPT_1, this holds for all the machines. But since machine 2^i of OPT_2 has power exactly $\bar{p}/2^i$, this follows immediately.

It follows that OPT_2 allocates each individual job at least as much power as OPT_1 at time t. We can apply this transformation for any time t, where we only need to take into account that OPT_2 might finish some jobs earlier than OPT_1. So the schedule for OPT_2 might contain unnecessary gaps, but it is a valid schedule, which 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 by Chekuri and Bender [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 Figure 1.

4.4 Analysis

Lemma 5. *Suppose* $p = E/OPT(I, E)$*. Subroutine* $FindSchedule(I, p)$ *creates a schedule which has makespan* $O(\log^{1+2/\alpha} m)\text{OPT}(I, E)$ *and uses energy* $O(1)E$*.*

FindSchedule(I, p)

1. Find a schedule for instance I and machines in the set $M(p)$ using Chekuri and Bender's algorithm [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 $p^* = \left(\frac{E}{mW}\right)^{\frac{\alpha}{\alpha-1}}$ where W is the total weight of all jobs divided by m.
2. Using binary search on $[0, p^*]$ with p as the search variable, find the largest value for p such that these 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. 1. Our speed scaling algorithm. The input consists a set of jobs I and an energy bound E.

Proof. Let S_1 and S_2 denote the schedules obtained in steps 1 and 2 of the subroutine *FindSchedule*(I, p), respectively. In an abuse of notation, we will also use S_1 and S_2 to refer to the makespans of these schedules. Schedule S_2 is the one returned by *FindSchedule*. First we analyze the makespan. From Chekuri and Bender [5], $S_1 = O(\log m)\text{OPT}_{M(p)}(I)$. 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 4, $\text{OPT}_{M(p)}(I) \leq \text{OPT}(I, e)$. Therefore, taken together, we have

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

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

$$p(1 + \log m)\, O(\log m) \text{OPT}_{M(p)}(I) \;\leq\; O(\log^2 m)\, p\, \text{OPT}(I, A) \;=\; O(\log^2 m)\, A \quad (3)$$

where the inequality follows from Lemma 4. 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 (3), this is $O(1)A$. □

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 weight of all the jobs divided by m. Completing W units of work at a speed of s requires $s^{\alpha-1}W$ units of energy. If each of the m machines processes W units of work, then it takes a total $mWs^{\alpha-1}$ units of energy. This must be less than E. For the speed we find $s^{\alpha-1} \leq E/mW$ and thus $p^{\frac{\alpha-1}{\alpha}} \leq E/mW$. This gives us an initial upper bound for p for the binary search:

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

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

From Lemma 5 and our analysis above, the following theorem holds.

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

5 Conclusions

Speed scaling to manage power is an important area of application that is worthy of further academic investigation. For a survey, including proposed avenues for further investigations, we recommend the survey paper [11].

References

1. Noga Alon, Yossi Azar, Gerhard Woeginger, and Tal Yadid. Approximation schemes for scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.
2. Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, pages 520 – 529, 2004.
3. Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *Symposium on Theoretical Aspects of Computer Science*, pages 460–471, 2005.

4. David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
5. Chandra Chekuri and Michael A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41:212–224, 2001.
6. Jian-Jia Chen, Tei-Wei Kuo, and Hsueh-I Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures*, 2005. To appear.
7. Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 581–590, 1997.
8. Ronald L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Techical Journal*, 45:1563–1581, 1966.
9. Ronald L. Graham, Eugene Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
10. Flavius Gruian and Krzysztof Kuchcinski. Lenes: Task-scheduling for low-energy systems using variable voltage processors. In *Asia South Pacific - Design Automation Conference*, pages 449–455, 2001.
11. Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 2005.
12. Woo-Cheol Kwon and Taewhan Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):211–230, 2005.
13. Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. In *11th International Computing and Combinatorics Conference (COCOON 2005)*, 2005. To appear.
14. Jiong Luo and Niraj K. Jha. 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*, pages 357–364, 2000.
15. Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Moss, and Rami G. Melhem. Energy aware scheduling for distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, page 21, 2003.
16. Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.
17. Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, pages 14–25, 2004.
18. F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science (FOCS 1995)*, pages 374–382, 1995.
19. Han-Saem Yun and Jihong Kim. On energy-optimal voltage scheduling for fixed priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3):393–430, 2003.
20. Yumin Zhang, Xiaobo Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference*, pages 183–188, 2002.