Energy Efficient Frequency Scaling and Scheduling for Malleable Tasks

Peter Sanders and Jochen Speck

Department of Informatics
Karlsruhe Institute of Technology, Germany
{sanders,speck}@kit.edu

Abstract. We give an efficient algorithm for solving the following scheduling problem to optimality: Assign n jobs to m processors such that they all meet a common deadline T and energy consumption is minimized by appropriately controlling the clock frequencies of the processors. Jobs are malleable, i.e., their amount of parallelism can be flexibly adapted. In contrast to previous work on energy efficient scheduling we allow more realistic energy consumption functions including a minimum and maximum clock frequency and a linear term in energy consumption. We need certain assumptions on the speedup function of the jobs that we show to apply for a large class of practically occurring functions.

1 Introduction

In recent years the environmental impact of computing systems and their energy usage was increasingly recognized as an important issue. Additionally the energy costs of computers became a more important part of the total costs of computing systems. Hence much research effort was put into reducing the energy usage of computers. Frequency scaling and power gating (switch off entire processors if they are not needed) which were originally invented to increase battery life time of mobile devices became more and more common in PC and server systems. Theoretical research work in this area mostly considered NP-hard problems or jobs which are only sequential (see [7] and [3] for example). The well known YDS algorithm [8] which also computes an energy-optimal schedule uses a similar energy model as our work but also only considers one processor and serial jobs. This work is focused on the energy minimization problem for jobs with flexible parallelism called malleable jobs as defined in [4]. In this model we will be able to adapt previous work on makespan optimization [2], [1] and [6] to achieve polynomial time algorithms.

The main result of our work is that we can compute the optimal schedule and frequencies to minimize the energy usage for n jobs on m processors in time $\mathcal{O}(n\log(mn)\cdot\log m)$ or even in $\mathcal{O}\big((\frac{n}{m}+1)\log m\cdot\log(mn)\big)$ if we use all m processors for the computation. We need some assumptions about the speedup functions but show that these are fulfilled in many common cases. We also generalize the model to allow an additional linear term in energy consumption. Furthermore, we consider the problem with minimal and maximal clock frequencies, for which we give an ϵ-approximation. Note that this is an important step to bridge the gap between real world processors and the simple power laws usually used in theoretical results.

The rest of the paper is organized as follows: In Section 2 we describe the model. In Section 2.1 we show that the assumptions we made are commonly fulfilled. The problem for one job is solved in Section 3. In Section 4 we introduce noninteger processor numbers and the energy function for them. The algorithm is given in Section 5. The model is generalized in Section 6.

2 Model

We have a system consisting of m uniform (or physical identical) processors whose speed (processing frequency) can be adjusted independently. Since one usually has to increase the processor voltage as one increases the processing frequency and the voltage is proportional to the electrical current, it is plausible that the power consumption of a processor is proportional to the cubic frequency. For our model we set the power consumption of one processor proportional to f^α with $\alpha > 2$ and processing frequency f. Frequency changes of processors are immediate and with negligible overhead. If a processor is not used, the power consumption of that processor will be 0 during that time. In the initial machine model we have no maximum frequency and the power consumption of a processor approaches 0 if the frequency approaches 0. An enhanced model is introduced in Section 6. For the initial model we get

$$E = p \cdot f^\alpha \cdot T$$

for the energy consumption E of a job which runs on p processors with frequency f for a time T.

The n jobs we want to schedule on this machine are ready at time 0 and have a common deadline T. Job $j \in \{1, \dots, n\}$ requires w_j clock cycles on a single processor. The jobs are malleable which means that the number of processors working on them can be changed over time. There is no penalty for changing the number of processors. However, a function $s_j(p)$ gives the speedup achieved by using p processors for job j, i.e., the amount of work done for a job j during a time interval of length t on p processors running at frequency f is $f \cdot s_j(p) \cdot t$ (we will drop the j of s_j when there is only one job or the job is clear).

For the rest of the article we assume that two restrictions are met for the speedup functions s. In Section 2.1 we argue that these conditions are met by a large class of relevant speedup functions.
The **first restriction** is that all speedup functions are concave at least between 0 and the processor number $\hat{p}$ where the maximal speedup is reached. Of course the speedup for 0 processors is 0 and the speedup for 1 processor is 1.

The **second restriction** is that the function h defined as $h(0) = 0$ and for other p as

$$h : p \mapsto \sqrt[\alpha-1]{\frac{s^\alpha(p)}{p}}$$

is monotonically increasing for all $p < \bar{p}$ for a $\bar{p} \in \mathbb{N}$ and monotonically decreasing for all $p > \bar{p}$. Additionally h should be concave on $(0, \bar{p}]$. We set $\bar{p}$ to be the smallest value for which h reaches its maximum then h is strictly increasing on $(0, \bar{p}]$ because of the concavity on $(0, \bar{p}]$. Both functions (h, s) are in general only known for integer values of p. We can see directly $\bar{p} \leq \hat{p}$.

As this work heavily depends on convexity or concavity properties of functions we restate the following definition known from calculus:

Definition 1. *A function f is said to be concave (convex) on an interval $[a, b]$ when for all $x, y \in (a, b)$ with $x < y$ the following holds:*
For each $r \in (0, 1)$ with $z = ry + (1-r)x$ we have $f(z) \geq (1-r)f(x) + rf(y)$ $(f(z) \leq (1-r)f(x) + rf(y))$.

For functions which are two times continuously differentiable it is known that concavity (convexity) is equivalent to $f''(z) \leq 0 \;\; (f''(z) \geq 0) \; \forall z \in (a, b)$.

2.1 Common Speedup Functions

All speedup functions considered in this section are given as closed formulas and are two times continuously differentiable except for a finite number of points. We want to show in this section that the quite technical restrictions from the model hold for many typical speedup functions and hence do not restrict the use of the results of this paper.

Linear Speedup Functions One of the easiest types of parallel jobs are those with linear speedup between 1 and p^* processors and $s(p) = s(p^*)$ for all $p > p^*$ (even for embarrassingly parallel jobs the maximal speedup is reached for $p = m$). If we set $\bar{p} = p^*$ then h is monotonically decreasing for all $p > \bar{p}$ and $h(p) = p$ for all $p \in (0, \bar{p})$ which is monotonically increasing and concave. The speedup function is also concave between 0 and p^*. Hence these kind of jobs fulfills the restrictions.

Jobs with Amdahl's Law Another kind of jobs one often sees are these which fulfill Amdahl's Law. If the amount of sequential work is $\frac{w}{k}$ and the amount of parallelizable work is $(1 - \frac{1}{k})w$, we get

$$s(p) = \frac{w}{\frac{w}{k} + (1 - \frac{1}{k})\frac{w}{p}} = \frac{k \cdot p}{p + k - 1}$$

as speedup function. Doing the math we get $h'(p) > 0 \Leftrightarrow p < (\alpha - 1)(k - 1)$ and thus $\bar{p} = (\alpha - 1)(k - 1)$. For $h''(p)$ we get $h''(p) < 0 \Leftrightarrow p < 2 \cdot (\alpha - 1)(k - 1)$. Hence h is concave for all $p \in (0, \bar{p})$ and monotonically increasing in $(0, \bar{p})$ and monotone decreasing in $(\bar{p}, \infty)$. The speedup function is concave for all p. Thus these kind of jobs fulfills the restrictions.

Jobs with Parallelization Overhead A situation also very common in parallel computing is that one can parallelize all the work but one has to pay an additional overhead g depending on the number of processors with $g(p) \geq 0$ and

$g'(p) \geq 0$ for all $p \geq 1$. Then we get this speedup function:

$$s(p) = \frac{w + g(1)}{\frac{w}{p} + g(p)} = \frac{p \cdot w + p \cdot g(1)}{w + p \cdot g(p)}$$

With some calculus we get:

$$\begin{aligned} h'(p) > 0 &\Leftrightarrow (\alpha - 1)w > p \cdot (g(p) + \alpha p g'(p)) \\ h''(p) < 0 &\Leftrightarrow \\ p \cdot (g(p) + pg'(p))^2 \frac{2\alpha - 1}{\alpha - 1} &< (w + pg(p))(2g(p) + 4pg'(p) + p^2 g''(p)) \end{aligned}$$

All p which fulfill $h'(p) > 0$ fulfill also $h''(p) < 0$ if $g'(p) + pg''(p) \geq 0$. To have only one local maximum of h the term $p \cdot (g(p) + \alpha p g'(p))$ has to be strictly increasing but this is the case if $2g'(p) + pg''(p) > 0$. Hence if $2g'(p) + pg''(p) > 0$ holds for an overhead g then the second restriction is fulfilled.

If $w = p^2 g'(p)$ then $s'(p) = 0$ and if the function $p \mapsto p^2 g'(p)$ is strictly increasing this only holds for only one $p = p^*$. If $2g'(p) + pg''(p) > 0$ then $p \mapsto p^2 g'(p)$ is strictly increasing and $s''(p) < 0$ for all $p \in (0, p^*]$. For $\hat{p} = p^*$ the first restriction is fulfilled.

Hence if $2g'(p) + pg''(p) > 0$ and $g(p) \geq 0$ and $g'(p) \geq 0$ hold for all $p \geq 1$ both restrictions are fulfilled. We can check this condition for some overheads:

$g(p)$	$g'(p)$	$g''(p)$	$2g'(p) + pg''(p)$
p	1	0	2
$\log p$	p^{-1}	$-p^{-2}$	p^{-1}
$p \log p$	$\log p + 1$	p^{-1}	$3 + 2\log p$
$\sqrt{p}$	$(2\sqrt{p})^{-1}$	$-(4p\sqrt{p})^{-1}$	$3 \cdot (4\sqrt{p})^{-1}$
p^2	$2p$	2	$6p$
$\log^2 p$	$2p^{-1} \log p$	$2p^{-2} \cdot (1 - \log p)$	$2p^{-1} \cdot (1 + \log p)$

So the condition is fulfilled for all of these overheads and all possible sums of these overheads.

3 The Optimal Solution for the Single-Job Case

In this section we want to compute the optimal number of processors in terms of energy usage for a job which fulfills the restriction from Section 2. Now we will give some lemmata which also will be useful for the multi-job case:

Lemma 1. *In an optimal solution for any number of processors p used during the computation of the job the processing frequency is always the same if you use p processors.*

The detailed proof will be given in the full paper.

Lemma 2. *For only one job running on the machine it is optimal to use $\bar{p}$ processors for the whole time $[0, T]$.*

Proof. If we want a job to do an amount of work w during a time interval I of length t with a constant number of processors, the operating frequency depends only on the speedup reached during I: $f = \frac{w}{s(p)\cdot t}$. Thus the energy becomes now a function only dependent on p:

$$E = \frac{w^\alpha}{t^{\alpha-1}} \cdot \frac{p}{s^\alpha(p)} = \frac{w^\alpha}{t^{\alpha-1}} \cdot \frac{1}{h^{\alpha-1}(p)}$$

Thus the energy consumption is minimized if h is maximized and it is optimal to use $\bar{p}$ processors during I (this is also true for $w = 0$).

Hence this is true for all subintervals of $[0, T]$ and thus it is optimal to use $\bar{p}$ processors all the time. □

Because of Lemma 1 the work done in an interval I for a job which runs on $\bar{p}$ processors for the whole time $[0, T]$ is proportional to the length of I. Because of Lemma 2 the algorithm to find the optimal number of processors for a job consists only of the search for the integer value $\bar{p}$ that maximizes h. It is also obvious that E is strictly decreasing on $[1, \bar{p}]$. It is possible that $\bar{p} > m$ but as h is strictly increasing in $(0, \bar{p})$, the minimal E is reached for $p = m$ in this case.

4 Energy Function for Non-Integer Processor Numbers

Until this section all processor numbers were integer. In order to solve the energy-optimal scheduling problem for multiple jobs we need to handle noninteger processor numbers. For a noninteger processor number we introduce the notation $p + \tau$ for the rest of the paper where p is the integer part and $\tau \in [0, 1)$. The main goal of this Section is to compute the minimal energy consumption (and according optimal schedule) of a job with work w to be done in time T when the job can use an average number of $p+\tau$ processors. A job runs on an average processor number $p + \tau = T^{-1} \sum_{p=0}^{m} t_p p$ during time $T = \sum_{p=0}^{m} t_p$ if it runs on p processors for time t_p. We prove that for any average processor number $p + \tau$ Consider a job that runs on an average processor number $p + \tau$. We prove that it is optimal to only use p and $p + 1$ processors for the job. Then we define the the energy function E of the job to be the mapping from the average processor number $p + \tau$ to the minimal possible energy consumption with this processor number. In the end we will show some useful properties of E. All processor numbers in this section are not bigger than $\bar{p}$ otherwise you could shrink all larger processor numbers to $\bar{p}$ without using more energy (because of Lemma 2).

Lemma 3. *If the average number of processors is τ then it is optimal to run on 1 processor for time τT and not to run for time $(1-\tau)T$.*

Proof. We consider the case of a job which runs on p processors for a time interval I_1 of length t_1 and runs on 0 processors (or does not run) for a time interval I_2 of length t_2. Let w be the total work done in I_1 (and I_2) and $E_{old} = \frac{w^\alpha}{t_1^{\alpha-1}} \cdot \frac{p}{s^\alpha(p)}$ be the energy used. Set $\tau := \frac{t_1 \cdot p}{t_1+t_2}$. We will now use 1 processor for time $\tau(t_1 + t_2)$ and 0 processors for time $(1-\tau)(t_1 + t_2)$ and show that this does the same work with no more energy. The energy used to do work w with 0 and 1 processor is $E_{new} = \frac{w^\alpha}{(\tau\cdot(t_1+t_2))^{\alpha-1}} \cdot \frac{1}{s^\alpha(1)} = \frac{s^\alpha(p)}{p^\alpha} \cdot E_{old} \leq E_{old}$.

The repeated use of this argument for all intervals where the job runs on more than one processor shows that if the average number of processors used during time T is $\tau \leq 1$ then it is optimal to use one processor during time $\tau \cdot T$ and 0 processors during time $(1-\tau)\cdot T$. $\square$

Lemma 4. *If the average number of processors is* $p+\tau \geq 1$ *then it is optimal to run on at least* 1 *processor throughout* $[0,T]$.

The detailed proof will be given in the full paper.

Lemma 5. *If the average number of processors is* $p+\tau \geq 1$ *then it is optimal to run on* $p+1$ *processors for time* τT *and to run on* p *processors for time* $(1-\tau)T$.

Proof. We consider the case of a job which runs on p_1 processors for a time interval I_1 of length t_1 and runs on p_2 processors for a time interval I_2 of length t_2 (because of Lemma 4 $p_1, p_2 > 0$) w.l.o.g. $0 < p_1 < p_2 \leq \bar{p}$. Let w_i be the work done in I_i and $E_i = \frac{w_i^\alpha}{t_i^{\alpha-1}} \cdot \frac{p_i}{s^\alpha(p_i)}$ be the energy used in I_i for $i \in \{1,2\}$. Let $w = w_1 + w_2$ be the total work done. We now compute how the work w is optimally distributed between I_1 and I_2. If we do βw work during I_1 and $(1-\beta)w$ work during I_2 we get the following energy as a function of $\beta \in (0,1)$:

$$E(\beta) = \frac{\beta^\alpha w^\alpha}{t_1^{\alpha-1}} \cdot \frac{p_1}{s^\alpha(p_1)} + \frac{(1-\beta)^\alpha w^\alpha}{t_2^{\alpha-1}} \cdot \frac{p_2}{s^\alpha(p_2)}$$

We now have to find the minimum of $E(\beta)$. We do this by computing the β with $E'(\beta) = 0$. This β is a minimum because $E''(\beta) > 0$ for all $\beta \in (0,1)$. For $A = \frac{p_1}{s^\alpha(p_1)t_1^{\alpha-1}}$ and $B = \frac{p_2}{s^\alpha(p_2)t_2^{\alpha-1}}$ the minimizing β is

$$\beta = \frac{\sqrt[\alpha-1]{B}}{\sqrt[\alpha-1]{A} + \sqrt[\alpha-1]{B}}$$

Thus the value with the optimal β for E is $E = w^\alpha \cdot (t_1 h(p_1) + t_2 h(p_2))^{-\alpha+1}$.

We set $p := \lfloor \frac{t_1 p_1 + t_2 p_2}{t_1+t_2} \rfloor$ and $\tau := \frac{t_1 p_1 + t_2 p_2}{t_1+t_2} - p$ then $p+\tau$ is the average number of processors used during I_1 and I_2 and $p_1 \leq p < p+1 \leq p_2$. We now want to show that using p processors during time $(1-\tau)(t_1+t_2)$ and $p+1$ processors during time $\tau\cdot(t_1+t_2)$ is an optimal solution to do the work w during $I_1 \cup I_2$. In order to do this it is sufficient to show that $t_1 h(p_1) + t_2 h(p_2) \leq \tau\cdot(t_1+t_2)h(p+1) + (1-\tau)(t_1+t_2)h(p)$.

If we set $r := \tau\frac{t_1+t_2}{t_1} \cdot \frac{p_2-(p+1)}{p_2-p_1}$ then $1-r = (1-\tau)\frac{t_1+t_2}{t_1} \cdot \frac{p_2-p}{p_2-p_1}$ and with $s := \tau\frac{t_1+t_2}{t_2} \cdot \frac{p+1-p_1}{p_2-p_1}$ we get $1-s = (1-\tau)\frac{t_1+t_2}{t_2} \cdot \frac{p-p_1}{p_2-p_1}$. With this we have:

$$\begin{aligned} t_1 h(p_1) \ + \ t_2 h(p_2) &= r t_1 h(p_1) + s t_2 h(p_2) \ + \ (1-r)t_1 h(p_1) + (1-s)t_2 h(p_2) \\ &= \tau\cdot(t_1+t_2)\left(\frac{p_2-(p+1)}{p_2-p_1}h(p_1) + \frac{p+1-p_1}{p_2-p_1}h(p_2)\right) \\ &\quad +(1-\tau)(t_1+t_2)\left(\frac{p_2-p}{p_2-p_1}h(p_1) + \frac{p-p_1}{p_2-p_1}h(p_2)\right) \\ &\leq \tau\cdot(t_1+t_2)h(p+1) \ + \ (1-\tau)(t_1+t_2)h(p) \end{aligned}$$

because h is concave for $p_1, p, (p+1), p_2 \in (0, \bar{p}]$.

The repeated use of this argument for all intervals with different numbers of processors shows that if the average number of processors used during time T is $p + \tau$ with $p \geq 1$ then it is optimal to use $p+1$ processors during time $\tau \cdot T$ and p processors during time $(1-\tau) \cdot T$. $\square$

With Lemma 3 and Lemma 5 we can define the energy usage for an average number of processors $p + \tau$ as the optimal energy usage of this case:

Definition 2. *A job which does work w during time T on an average number of processors $p + \tau$ with $p \in \mathbb{N}_0$ uses energy*

$$E(p+\tau) := E(p,\tau) := \frac{w^\alpha}{T^{\alpha-1}} \cdot (\tau \cdot h(p+1) + (1-\tau)h(p))^{-\alpha+1}$$

It is immediately clear that E is a continuous function on $(0, \infty)$ and has the same values as E from Section 3 on integer p

Lemma 6. *The function $E(p+\tau)$ as defined in Definition 2 is strictly convex on $(0, \bar{p}]$ and has its minimum at $\bar{p}$.*

Proof. The thing left to show is that $E(p+\tau)$ is strictly convex on $(0, \bar{p})$ and the minimum at $\bar{p}$. We will first show that $\frac{\partial^2 E(p,\tau)}{\partial \tau^2} = E_{\tau\tau}(p,\tau) > 0$ for all $p+1 \leq \bar{p}$.

$$E_\tau(p,\tau) = -\frac{w^\alpha}{T^{\alpha-1}} \cdot (\alpha-1)(\tau \cdot h(p+1) + (1-\tau)h(p))^{-\alpha}(h(p+1) - h(p))$$

$$E_{\tau\tau}(p,\tau) = \frac{w^\alpha}{T^{\alpha-1}} \cdot \alpha(\alpha-1)(\tau \cdot h(p+1) + (1-\tau)h(p))^{-\alpha-1}(h(p+1) - h(p))^2$$

Thus $E_{\tau\tau}(p,\tau) > 0 \Leftrightarrow h(p+1) - h(p) > 0 \Leftrightarrow p+1 \leq \bar{p}$. It remains to check that

$$\lim_{\tau \to 1} E_\tau(p,\tau) \leq \lim_{\tau \to 0} E_\tau(p+1,\tau)$$
$$\Leftrightarrow \; h(p+1) - h(p) \geq h(p+2) - h(p+1)$$

The last inequality is true for $p+2 \leq \bar{p}$ because h is concave and true for $p+1 = \bar{p}$ because $h(p+1) \geq h(p), h(p+2)$. Thus we have shown that E is strictly convex for $p+1 \leq \bar{p}$ and E_τ is strictly increasing for $p+1 \leq \bar{p}$.

The fact that E has its minimum at $\bar{p}$ directly comes from the fact that h has its maximum at $\bar{p}$. $\square$

Definition 3. *We define the left derivative of E for integer processor numbers as $\overrightarrow{E}(p) := \lim_{\tau \to 1} E_\tau(p-1,\tau)$ and the right derivative as $\overleftarrow{E}(p) := \lim_{\tau \to 0} E_\tau(p,\tau)$. For noninteger $p+\tau$ the left and right derivative are the same and we define $\overrightarrow{E}(p+\tau) = \overleftarrow{E}(p+\tau) = E_\tau(p,\tau) =: E'(p+\tau)$.*

Lemma 7. *We have $\overleftarrow{E}(0) = -\infty$ and $\overleftarrow{E}(\bar{p}) \geq 0$ and $\overrightarrow{E}(p+\tau) \leq \overleftarrow{E}(p+\tau)$ $\forall p+\tau \in (0, \bar{p}]$ and $\overrightarrow{E}(p+\tau), \overleftarrow{E}(p+\tau)$ are strictly increasing on $(0, \bar{p}]$.*

This lemma is obvious and needs no proof. With Lemma 7 we can define the inversion of the derivative of the energy function:

Definition 4. *We have E as in Definition 2 and the left and right derivatives as in Definition 3. Then for any $c \in (-\infty, 0]$ we define $(E')^{-1}(c) := p^* + \tau^*$ for $p^* + \tau^* \in (0, \bar{p}]$ with $\overrightarrow{E}(p^* + \tau^*) \le c \le \overleftarrow{E}(p^* + \tau^*)$.*

Lemma 8. *$(E')^{-1}$ as defined in Definition 4 is a continuous and monotonously increasing function on $(-\infty, 0]$.*

The detailed proof will be given in the full paper.

5 The Optimal Solution for the Multi-Job Case

After the technical section we are now ready to prove the main theorem:

Theorem 1. *We have n jobs and for each job j an energy function E_j as in Definition 2 and the left and right derivatives as in Definition 3 and the inverse of the derivative of the energy function $(E'_j)^{-1}$ as in Definition 4. If we want to minimize $\sum_j E_j(p_j + \tau_j)$ under the restriction $\sum_j (p_j + \tau_j) \le m$ then there exists a $c \in \mathbb{R}$ such that $\overrightarrow{E_j}(p^*_j + \tau^*_j) \le c \le \overleftarrow{E_j}(p^*_j + \tau^*_j)$ for all j holds for an optimal solution $(p^*_1 + \tau^*_1, \ldots, p^*_n + \tau^*_n)$.*

If we have found a c such that $\sum_j (E'_j)^{-1}(c) = m$ or $\sum_j (E'_j)^{-1}(c) < m$ and $(E'_j)^{-1}(c) = \bar{p}_j$ for all j then we have found an optimal solution.

Proof. We do the first part of the proof by contradiction. Suppose there are $i, j \in \{1, \ldots, n\}$ with $\overrightarrow{E_j}(p^*_j + \tau^*_j) > \overleftarrow{E_i}(p^*_i + \tau^*_i)$ in an optimal solution. Because $\overrightarrow{E_j}$ is continuous from the left and $\overleftarrow{E_i}$ is continuous from the right there exists a ϵ such that for all $y \in [p^*_j + \tau^*_j - \epsilon, p^*_j + \tau^*_j]$ and all $z \in [p^*_i + \tau^*_i, p^*_i + \tau^*_i + \epsilon]$ we have $\overrightarrow{E_j}(y) \ge \overleftarrow{E_i}(z) + \epsilon$. Thus we have

$$E_j(p^*_j + \tau^*_j) - E_j(p^*_j + \tau^*_j - \epsilon) \ge \inf_{y \in [p^*_j + \tau^*_j - \epsilon, p^*_j + \tau^*_j]} \overrightarrow{E_j}(y) \cdot \epsilon$$

$$\ge \sup_{z \in [p^*_i + \tau^*_i, p^*_i + \tau^*_i + \epsilon]} (\overleftarrow{E_i}(z) + \epsilon) \cdot \epsilon \ge E_i(p^*_i + \tau^*_i + \epsilon) - E_i(p^*_i + \tau^*_i) + \epsilon^2$$

$$\Leftrightarrow E_j(p^*_j + \tau^*_j) + E_i(p^*_i + \tau^*_i) \ge E_j(p^*_j + \tau^*_j - \epsilon) + E_i(p^*_i + \tau^*_i + \epsilon) + \epsilon^2$$

Hence we have shown that there exists a better solution than the optimal solution which leads to the contradiction.

In case of $\sum_j (E'_j)^{-1}(c) < m$ and $(E'_j)^{-1}(c) = \bar{p}_j$ for all j every job runs with its optimal number of processors. Thus no job can save energy trough running on a different number of processors. $(E'_j)^{-1}(c) = \bar{p}_j$ is always the case for $c = 0$.

In case of $\sum_j (E'_j)^{-1}(c) = m$ we have used all processors available. As we have $c \le 0$ and $(E'_j)^{-1}(c) \le \bar{p}_j$ for all j in this case we would increase energy usage if we used less processors. As $\overrightarrow{E_j}(p^*_j + \tau^*_j + \delta) > \overleftarrow{E_i}(p^*_i + \tau^*_i - \delta)$ for all $\delta > 0$ and all i, j it is not possible to improve the solution by transferring an amount δ of processors from job i to job j. Hence the solution can not be improved and thus is optimal. $\square$

With Theorem 1 we can now give the algorithm. Let c^* be the c of an optimal solution as in Theorem 1 and $p_j^* + \tau_j^*$ be the amount of processors of job j in the optimal solution. If we have a c such that $\sum_j (E_j')^{-1}(c) < m$ and $(E_j')^{-1}(c) < \bar{p}_j$ for at least one j then we know $c < c^*$ because all $(E_j')^{-1}$ are monotonously increasing. If we have a c such that $\sum_i (E_i')^{-1}(c) > m$ then we know $c > c^*$ also because all $(E_i')^{-1}$ are monotonously increasing. Hence we can use an interval halving technique to find the optimal $c = c^*$.

We also know in which interval to search. The maximal possible c is 0. If we use the same amount of $p_m + \tau_m = \min\{\frac{m}{n}, 1\}$ processors for each job then we know that c^* can not be smaller than $c_m = \min_i \overleftarrow{E}_i(p_m + \tau_m)$ because $p_m + \tau_m \leq \bar{p}_i$ for all i and $\sum_i (E_i')^{-1}(c_m) \leq m$.

For each job i we have a set of $2\bar{p}_i \leq 2m$ bend points. These are the points $\overleftarrow{E}_i(p)$ and $\overrightarrow{E}_i(p)$ for all $p \in \{1, \ldots, \bar{p}\}$. If we know c^* lies in an interval (c_ℓ, c_u) which contains no bend points, we can solve the problem directly. For i with $\overleftarrow{E}_i(p) \leq c_\ell < c_u \leq \overrightarrow{E}_i(p+1)$ for a certain p we have $p_i^* + t_i^* \in (p, p+1)$ (**case 1**). For i with $\overrightarrow{E}_i(p) \leq c_\ell < c_u \leq \overleftarrow{E}_i(p)$ for a certain p we have $p_i^* + t_i^* = p$ (**case 2**) thus $t_i^* = 0$. Other cases can not exist.

The computation of $(E_i')^{-1}(c)$ is done in two steps. First we search for the two adjacent bend points of c. If we are in case 2 we are done. In case 1 we know p in $E_i'(p+\tau) = c$. With some algebra we get $(\tau \cdot (h_i(p+1) - h_i(p)) + h_i(p))^{-\alpha} \cdot D_i = c$ for a positive D_i which does not depend on τ and thus we can compute τ with $\tau = \frac{\sqrt[\alpha]{c^{-1} \cdot D_i} - h_i(p)}{h_i(p+1) - h_i(p)}$ and then $(E_i')^{-1}(c) = p + \tau$.

We will now compute the exact solution if all bend points are eliminated. W.l.o.g. let the n jobs be ordered such that the jobs $1, \ldots, n_1$ are from case 1 and the jobs $n_1 + 1, \ldots, n$ are from case 2. We set $m_r = m - \sum_{i=1}^{n} p_i^*$ thus $\sum_{i=1}^{n_1} \tau_i^* = m_r$. For all jobs from case 1 the derivative of the energy function $E_i'(p_i^*, \tau_i^*)$ has to be the same ($= c^*$). With some algebra we get:

$$\begin{aligned}
&D_1 \cdot (\tau_1^* h_1(p_1 + 1) + (1 - \tau_1^*) h_1(p_1))^{-\alpha} = \ldots = \\
&D_{n_1} \cdot (\tau_{n_1}^* h_{n_1}(p_{n_1} + 1) + (1 - \tau_{n_1}^*) h_{n_1}(p_{n_1}))^{-\alpha} = c^* \\
\Leftrightarrow\ &D_1^{\frac{-1}{\alpha}} \cdot (\tau_1^* h_1(p_1 + 1) + (1 - \tau_1^*) h_1(p_1)) = \ldots = \\
&D_{n_1}^{\frac{-1}{\alpha}} \cdot (\tau_{n_1}^* h_{n_1}(p_{n_1} + 1) + (1 - \tau_{n_1}^*) h_{n_1}(p_{n_1})) = (c^*)^{\frac{-1}{\alpha}}
\end{aligned}$$

With $A_i = D_i^{\frac{-1}{\alpha}} \cdot (h_i(p_i + 1) - h_i(p_i))$ and $B_i = D_i^{\frac{-1}{\alpha}} h_i(p_i)$ and $G = (c^*)^{\frac{-1}{\alpha}}$ we get $A_i \tau_i^* + B_i = G$ and thus $\tau_i^* = \frac{G}{A_i} - \frac{B_i}{A_i}$ for all i of case 1. We can compute G through

$$m_r = \sum_{i=1}^{n_1} \tau_i^* = G \sum_{i=1}^{n_1} \frac{1}{A_i} - \sum_{i=1}^{n_1} \frac{B_i}{A_i}$$

and with G we can compute the τ_i^* and thus the final solution.

With these prerequisites we can now describe the algorithm:

1. Set $c_u = 0$ and $c_\ell = c_m$ and the range of bend points $[1, \bar{p}_i]$ for each job i.
2. Pick a randomly chosen bend point in (c_ℓ, c_u) and set c accordingly.

3. Check if $\sum_i (E_i')^{-1}(c) > m$ then $c_u = c$ else $c_\ell = c$ and update the range of bend points for each job.
4. If there are bend points left in (c_ℓ, c_u) goto 2.
5. Do the exact calculation as above.

The expected number of times the loop 2-4 is executed is in $\mathcal{O}(\log(mn))$ because there are $\Theta(nm)$ bend points. Each time we have to choose one random bend point this is possible in time $\mathcal{O}(n)$. In 3 we have to invert n functions. Each can be done in time $\mathcal{O}(\log m)$ if all values of $h_i(p)$ are given. Additionally we have to update n ranges of breakpoints in 3. Each update can be done in $\mathcal{O}(1)$ with the p computed during function inversion done for the same job. The check in 4 can be done in $\mathcal{O}(n)$. The exact calculation also takes time in $\mathcal{O}(n)$.

Altogether the algorithm takes time in $\mathcal{O}(n \log(mn) \cdot \log m)$. It remains to compute the frequencies and to place the jobs onto the $m \times T$ processor $\times$ time rectangle.

We now know p_i^* and t_i^* for each job i. We reserve p_i^* processors for each job i for the whole interval $[0, T]$. For the τ_i^* and the remaining processors we can use McNaughton's wrap-around rule [5]. This can be done in time $\mathcal{O}(n)$ and each job i runs on p_i^* processors for time $(1 - \tau_i^*)T$ and on $p_i + 1$ processors for time $\tau_i^* T$ additionally any job changes its number of processors at most two times. A more detailed solution for a similar problem is given in [1]. With τ_i^* and p_i^* we can compute the work distribution between the phase with p_i^* processors and the phase with $p_i^* + 1$ processors in a similar way as we computed β in the proof of Lemma 5. Then the time, work and number of processors of both phases are known and the frequency can be computed with $w = s(p) \cdot f \cdot t$.

The parallelization is done in a similar way as we did it in [6] for another problem. If each processor does the computation of $(E_i')^{-1}(c)$ for $\frac{n}{m}$ jobs and takes care of their list of bend points we only have to use collective operations for sum, broadcast of c and a distributed random pick. Such operations cost time $\Theta(\log m)$. In the loop 2-4 each processor computes $(E_i')^{-1}(c)$ for $\frac{n}{m}$ jobs in $\mathcal{O}\big(\frac{n}{m} \log m\big)$ and takes part in the collective sum in time $\mathcal{O}\big(\frac{n}{m} + \log m\big)$. Thus the main part of the algorithm runs in $\mathcal{O}\big((\frac{n}{m} + 1) \log m \cdot \log(mn)\big)$. Frequency computation can be done in time $\mathcal{O}\big(\frac{n}{m} + 1\big)$. The placement can be done with prefix sum in time $\mathcal{O}\big(\frac{n}{m} + \log m\big)$.

6 The Enhanced Model

If we have an additional linear term in the energy usage (maybe from memory or other parts which can not change their frequency) like in $E_{new} = p \cdot f^\alpha \cdot T + p \cdot \delta \cdot T$ with δ being independent of the job then $\overrightarrow{E}_{new}(p, t) = \overrightarrow{E}(p, t) + \delta T$ and $\overleftarrow{E}_{new}(p, t) = \overleftarrow{E}(p, t) + \delta T$. The optimal solution for the single job case is $\bar{p}_{new} = (E_{new}')^{-1}(0) \le \bar{p}$ which can be noninteger. As E_{new}' is just E' with an additional constant we can use the same techniques as in the original model to get an optimal solution for the multi-job case.

Many processors have minimal and maximal operating frequencies (because of memory requirements or the length of signal paths). For this case we have to do a bit more theory first. We will restrict the presentation here to case of a maximal frequency f_G because the case of a minimal frequency is analogous.

There are two amounts of resources which are interesting. One is the absolute minimal amount of resources for our job $p_1 + \tau_1$ and the other is the minimal amount of resources $p_2 + \tau_2$ above we can use the standard techniques already introduced. Let $f_q(p + \tau)$ be the frequency for the part of the job which runs on q processors. f_q is defined on $(q - 1, q + 1)$. When we use β from the proof of Lemma 5 we can compute $f_p(p + \tau)$ and $f_{p+1}(p + \tau)$ for the optimal solution through $\beta w = s(p + 1)f_{p+1}(p + \tau)T \cdot \tau$ and $(1 - \beta)w = s(p)f_p(p + \tau)T \cdot (1 - \tau)$. Doing some algebra we get:

$$\frac{f_p(p + \tau)}{f_{p+1}(p + \tau)} = \sqrt[\alpha-1]{\frac{s(p)(p + 1)}{s(p + 1)p}} \geq 1$$

Thus $f_p(p + \tau) \geq f_{p+1}(p + \tau)$. Because the energy usage E is a continuous decreasing function of $p + \tau$ we know that f_{p+1} and f_p are continuous decreasing functions. Hence we can use the techniques for the case with unrestricted frequencies as long as $f_p(p + \tau) \leq f_G$ this gives us $p_2 + \tau_2$. The absolute minimum of resources needed for our job can be computed trough $w = s(p_1)f_GT(1-\tau_1)+s(p_1+1)f_GT\tau_1$. Let p be such that $f_{p+1}(p+1) \leq f_G \leq f_p(p)$. As $f_p(p) \geq f_p(p_2+\tau_2) \geq f_{p+1}(p_2+\tau_2) \geq f_{p+1}(p+1)$ and $s(p)f_GT \leq w \leq s(p+1)f_GT$ we know that $p \leq p_1 + \tau_1 \leq p_2 + \tau_2 \leq p + 1$ and thus $p = p_1 = p_2$.

Now we can define the energy function $\tilde{E}$ on $(p+\tau_1, p+\tau_2)$. The frequencies on $(p+\tau_1, p+\tau_2)$ will be $\tilde{f}_p$ and $\tilde{f}_{p+1}$. We know from the proof of Lemma 5 that $E(\beta)$ is convex hence for an optimal energy usage $\tilde{f}_p(p + \tau) = f_G$ for all $\tau \in [\tau_1, \tau_2]$. Thus we can compute $\tilde{f}_{p+1}(p+\tau)$ trough $w = s(p)f_GT \cdot (1-\tau) + s(p+1)\tilde{f}_{p+1}(p+\tau)T\tau$. With $\tilde{f}_{p+1}(p + \tau)$ we can compute $\tilde{E}(p + \tau)$. Doing some analysis we see that $\tilde{E}$ is convex on $(p+\tau_1, p+\tau_2)$. Obviously $\tilde{E}$ is continuous, $\tilde{E}(p+\tau) \geq E(p+\tau)$ and $\tilde{E}(p+\tau_2) = E(p+\tau_2)$. Hence we also have $\overrightarrow{\tilde{E}}(p+\tau_2) \leq \overrightarrow{E}(p+\tau_2)$. We also can compute $M := \overleftarrow{\tilde{E}}(p+\tau_1) = Tf_G^{\alpha}(s(p+1) - \alpha \cdot (p+1)(s(p+1) - s(p)))s^{-1}(p+1)$ which is a function of the input values.

Let c be as in Section 5 then we get: If $c \leq \overleftarrow{\tilde{E}}(p+\tau_1)$ our job just gets $p+\tau_1$ processors because it is the minimal number so we can put this amount away and continue with the other jobs. If $c \geq \overrightarrow{\tilde{E}}(p+\tau_2)$ we can use the technique from Section 5. If we know that the optimal solution lies between the bend points for $p + \tau_1$ and $p + \tau_2$ (these are new bend points for the enhanced model) then we do not know how to compute an exact solution. But then we know the minimal derivative M (the same is true for a minimal frequency).

If we want to compute a solution which only uses an amount of ϵ more energy than the optimal solution we can do it in the following way: For each job for which the amount of processors is already known we sum up these amounts. Let the difference between these amounts and m be $\tilde{m}$. We now do interval halving on $(M, \overrightarrow{\tilde{E}}(p + \tau_2))$. For every c we invert the E' and $\tilde{E}'$ (depending on which one applies) with an maximal additive error of $\frac{\epsilon}{8n}$. Then we compare

$\sum(E')^{-1}(c) + \sum(\tilde{E}')^{-1}(c)$ with $\tilde{m}$. If the remaining interval is smaller than $\epsilon \cdot (M \cdot 2m)^{-1}$ we know that the lower end of the interval stands for a feasible solution with a maximal additive error of ϵ.

The algorithm needs time in $\mathcal{O}\big(n \log(8n\epsilon^{-1}) \cdot \log(2mM^2\epsilon^{-1})\big)$ and can be parallelized in a similar way as above.

The placement is done as in Section 5. The frequency calculations for jobs with energy function $\tilde{E}$ is clear for the others we can do it in the same way as in Section 5.

7 Conclusion

We have shown that with two restrictions it is possible to solve our energy efficient scheduling problem optimally in near linear time. The major step to solve the problem was to build continuous convex energy functions and to use some calculus on them. This is somehow surprising because many related problems are known to be NP-hard. The two restrictions do not limit the applicability too much because many classes of parallel jobs fit into these restrictions.

Acknowledgment This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Invasive Computing" (SFB/TR 89). We also like to thank the unknown reviewer who gave a very helpful two page review.

References

1. Jacek Blazewicz, Mikhail Y. Kovalyov, Maciej Machowiak, Denis Trystram, and Jan Weglarz. Preemptable malleable task scheduling problem. *IEEE Transactions on Computers*, 55, 2006.
2. Jacek Blazewicz, Maciej Machowiak, Jan Weglarz, Mikhail Y. Kovalyov, and Denis Trystram. Scheduling malleable tasks on parallel processors to minimize the makespan: Models and algorithms for planning and scheduling problems. *Annals of Operations Research*, 129, 2004.
3. Jian-Jia Chen and Tei-Wei Kuo. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *Parallel Processing, 2005. ICPP 2005. International Conference on.*
4. Joseph Y-T. Leung, editor. *Handbook of Scheduling.* CRC, 2004.
5. Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1), 1959.
6. Peter Sanders and Jochen Speck. Efficient parallel scheduling of malleable tasks. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International.*
7. Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '05.
8. Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE*, pages 374–382. IEEE Computer Society, 1995.