

# Speed Scaling on Parallel Processors

Susanne Albers · Fabian Müller · Swen Schmelzer

Received: 29 February 2012 / Accepted: 30 July 2012 / Published online: 9 August 2012  
© Springer Science+Business Media, LLC 2012

**Abstract** In this paper we investigate dynamic speed scaling, a technique to reduce energy consumption in variable-speed microprocessors. While prior research has focused mostly on single processor environments, in this paper we investigate multiprocessor settings. We study the basic problem of scheduling a set of jobs, each specified by a release date, a deadline and a processing volume, on variable-speed processors so as to minimize the total energy consumption.

We first settle the problem complexity if unit size jobs have to be scheduled. More specifically, we devise a polynomial time algorithm for jobs with agreeable deadlines and prove NP-hardness results if jobs have **arbitrary deadlines**. For the latter setting we also develop a polynomial time algorithm achieving a constant factor approximation guarantee. Additionally, we study problem settings where jobs have arbitrary processing requirements and, again, develop constant factor approximation algorithms. We finally transform our offline algorithms into constant competitive on-line strategies.

---

A preliminary version of this paper has appeared in the *Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 289–298, 2007.

Most of this work was done while at the University of Freiburg and supported by the German Research Foundation, projects AL 464/4-1 and 4-2.

---

S. Albers (✉)

Department of Computer Science, Humboldt Universität zu Berlin, 10099 Berlin, Germany  
e-mail: [albers@informatik.hu-berlin.de](mailto:albers@informatik.hu-berlin.de)

F. Müller · S. Schmelzer

Department of Computer Science, University of Freiburg, Georges-Köhler-Allee 79, 79110 Freiburg, Germany

F. Müller

e-mail: [fmueLLer@informatik.uni-freiburg.de](mailto:fmueLLer@informatik.uni-freiburg.de)

S. Schmelzer

e-mail: [sschmelz@informatik.uni-freiburg.de](mailto:sschmelz@informatik.uni-freiburg.de)

**Keywords** Scheduling · Dynamic speed scaling · Energy efficiency · NP-hardness · Approximation algorithm · Online algorithm

## 1 Introduction

With increasing CPU clock speeds and higher levels of integration in processors, memories and controllers, power consumption has become a major concern in computer system design over the past years. **Power dissipation** is critical not only in battery operated mobile computing devices but also in desktop computers and servers. Electricity costs impose a substantial strain on the budget of data and computing centers, where CPUs account for 50–60% of the energy consumption [11]. In fact, Google engineers, maintaining thousands of servers, warned that if power consumption continues to grow, power costs can easily overtake hardware costs by a large margin [11]. In addition to costs, energy dissipation causes thermal problems. Most of the consumed energy is eventually converted into heat, resulting in wear and reduced reliability of hardware components.

On an algorithmic level there are two mechanisms to save energy. (1) *Speed scaling*: Microprocessors currently sold by chip makers such as AMD and Intel are able to operate at variable speed. The higher the speed, the higher the power consumption is. Speed scaling techniques dynamically adjust the speed of a processor executing a set of tasks. The goal is to construct energy-efficient schedules that preferably use low processing speeds and still guarantee a desired service. (2) *Sleep states*: When a system is idle, it can be transitioned into a low-power sleep state. Here one has to determine when to shut down the system, taking into account that a transition back to the active mode requires extra energy.

This paper focuses on dynamic speed scaling algorithms. Initiated by a seminal paper of Yao et al. [30] there has recently been considerable research interest in the design and analysis of speed scaling strategies, see e.g. [1–3, 5, 7, 14–16, 20, 23, 24, 27, 30] and references therein. While a considerable body of the literature considers single processor environments, in this paper we study multiprocessor settings. Multiprocessor speed scaling is a definite issue in desktop computers and servers that are typically equipped with dual or multiple processors. The topic is also interesting in laptops as computer manufacturers offer dual-core and quad-core notebooks. A general trend in hardware design is to develop architectures with multiple CPUs. In fact, Intel has done experiments with 80 cores on one chip [22].

We investigate a **classical** speed scaling problem in which jobs have deadlines. Consider  $n$  jobs, where job  $i$  is specified by a release date  $r(i)$ , a deadline  $d(i)$  and a processing volume  $p(i)$ . Job  $i$  can be feasibly scheduled in the time interval  $[r(i), d(i))$ . The processing volume  $p(i)$  represents the amount of work that must be finished to complete the job. The  $n$  jobs have to be assigned to  $m$  parallel processors, each of which can independently operate at variable speed. The power consumption function depending on speed  $s$  is defined as  $P(s) = s^\alpha$ , where  $\alpha > 1$  is a constant. If a processor runs at speed  $s$  for  $\delta$  time units, then a work of  $\delta s$  is finished and the consumed energy is  $\delta s^\alpha$ . At any time a processor can handle only one job and each job can be executed on one processor only. We allow job preemption, i.e. the processing

of a job may be interrupted and resumed later. However, we **disallow** job migration, i.e. over time a job may not be scheduled on several processors. This assumption is motivated by the fact that job migration is an **expensive** operation in many parallel processing environments. The goal is to find a feasible schedule that minimizes the total energy consumed on all the processors.

Both offline and online scenarios are of interest. In the offline setting all jobs and their characteristics are known in advance. In the online case, we learn about a job  $i$  at its release date  $r(i)$ . Following [29] we call an online algorithm  $c$ -competitive if, for any sequence of jobs, the incurred energy is upper bounded by  $c$  times the optimum energy for that sequence.

*Previous Work* The single processor variant of the speed scaling problem defined above was introduced by Yao et al. [30] and has been investigated the most among energy management problems, see [3, 4, 6, 8, 14, 15, 24, 30] for a selection of the relevant literature. Yao et al. [30] showed that on a single processor optimal schedules can be computed in polynomial time. They gave an efficient algorithm that repeatedly identifies time intervals of highest *density*. The density of an interval  $I$  is the total work to be completed in  $I$  divided by the length of  $I$ . The algorithm repeatedly schedules jobs in highest density intervals and takes care of reduced subproblems. Yao et al. [30] also proposed two online algorithms, called *Optimal Available* and *Average Rate*, and proved that *Average Rate* achieves a competitiveness of  $\alpha^\alpha 2^{\alpha-1}$ . A nearly matching lower bound on the performance of *Average Rate* was presented in [8]. Bansal et al. [3] analyzed *Optimal Available* and showed that its competitiveness is exactly  $\alpha^\alpha$ . Additionally, Bansal et al. developed a new algorithm with an improved competitive ratio of  $2(\alpha/(\alpha-1))^\alpha e^\alpha$ . Further improved performance guarantees can be achieved for small values of  $\alpha$  [6]. On the other hand, any randomized speed scaling algorithm has a competitive ratio of  $\Omega((4/3)^\alpha)$  [3]. For deterministic online algorithms the lower bound is at least  $e^{\alpha-1}/\alpha$  [6]. These lower bounds imply that the exponential dependence on  $\alpha$  is inherent in the algorithms' performance guarantees.

Much less is known for deadline-based scheduling in variable-speed multiprocessor environments. A simple reduction from 3-PARTITION implies that energy minimization is an NP-hard optimization problem if the jobs' processing requirements may take arbitrary values. This holds true even if all release dates and deadlines are identical, i.e.  $r(i) = r$  and  $d(i) = d$ , for some  $r$  and  $d$  and all  $i$ . Another simple observation is that for this case of identical release dates and deadlines a polynomial time approximation scheme can be derived using the PTAS for makespan minimization on parallel machines developed by Hochbaum and Shmoys [21]. A faster 1.13-approximation algorithm was given by Chen et al. [15]. They also showed that if job migration among processors is allowed, an optimal schedule can be computed in polynomial time.

Further related work studied scenarios where a variable-speed or a fixed-speed processor is equipped with an additional sleep state [9, 10, 17, 24]. Moreover, recent research has addressed the performance metrics of minimizing (a) the weighted sum of energy consumption and job flow times [2, 4, 5, 7, 20, 25–27] and (b) the makespan subject to an energy budget [13, 28].

*Our Contribution* This paper represents the first algorithmic study of multiprocessor speed scaling if jobs may have individual release dates and deadlines. Most of our paper concentrates on the offline scenario. In the first part of the paper we settle the complexity of the problem with unit size jobs. We may assume w.l.o.g. that  $p(i) = 1$ , for all  $i$ . We prove that if job deadlines are *agreeable*, an optimal multiprocessor schedule can be computed in polynomial time. Deadlines are agreeable if, for any two jobs  $i$  and  $i'$ , relation  $r(i) < r(i')$  implies  $d(i) \leq d(i')$ . In practice, instances with agreeable deadlines form a natural input class where, intuitively, jobs arriving at later times may be finished later. We then show that if the jobs' release dates and deadlines may take arbitrary values, the energy minimization problem is NP-hard, even on two processors. For a variable number of processors, energy minimization is strongly NP-hard. Furthermore, for arbitrary release dates and deadlines we develop a polynomial time algorithm that achieves a constant factor approximation guarantee of  $\alpha^\alpha 2^{4\alpha}$ .

In the second part of the paper we address multiprocessor speed scaling where the processing requirements  $p(i)$  may take arbitrary values. (Recall that the problem is NP-hard even for identical release dates and deadlines.) For agreeable deadlines we present constant factor approximation algorithms. If all jobs have a common release date or have a common deadline, the approximation factor is  $2(2 - 1/m)^\alpha$ . Otherwise the ratio is  $\alpha^\alpha 2^{4\alpha}$ . Finally, we show that our offline algorithms can be transformed into online strategies attaining constant competitive ratios.

All our algorithms are simple and fast, which is an important aspect in energy-efficient computing environments. In a first step the algorithms may divide jobs into classes and then assign them to processors using classical dispatching rules such as *Round Robin* or *List* scheduling. Once the assignment is done, each processor independently computes its own schedule. Hence, our algorithms can also be applied in fully distributed systems.

*Subsequent Work* After the conference publication of this paper, further results have been obtained. Greiner et al. [19] showed that any  $c$ -approximation algorithm for a single processor yields a randomized  $cB_\alpha$ -approximation algorithm for parallel processors, where  $B_\alpha$  is the  $\alpha$ -th Bell number, i.e. the number of partitions of a set of size  $\alpha$ . Similarly, any  $c$ -competitive online algorithm yields a randomized  $cB_\alpha$ -competitive algorithm for parallel processors [19]. These results give the first (randomized) performance guarantees for jobs with arbitrary release dates, deadlines and processing volumes. In the offline setting a deterministic  $B_\alpha$ -approximation can also be achieved [19]. A deterministic  $O(\log^\alpha P)$ -competitive online algorithm was presented by Bell and Wong [12]. Here  $P$  is the ratio of the largest to smallest processing volume. Finally we remark that our proposed approach of dividing jobs into classes and applying *Round Robin* was subsequently used in [12, 26, 27].

## 2 Unit Size Jobs with Agreeable Deadlines

In this section we consider problem instances consisting of unit size jobs with agreeable deadlines. We present a **polynomial time algorithm** that is essentially a *Round*

*Robin* strategy. More specifically, jobs are first sorted according to their release dates and are then assigned to processors using *Round Robin*. For each processor, given the assigned jobs, an optimal schedule is computed using the algorithm by Yao et al. [30].

### Algorithm RR

1. Number the jobs in order of non-decreasing release dates. Jobs having the same release date are numbered in order of non-decreasing deadlines. Ties may be broken arbitrarily.
2. Given the sorted list of jobs computed in Step 1, assign the jobs to processors using the *Round Robin* policy.
3. For each processor, given the jobs assigned to it, compute an optimal schedule.

**Theorem 1** *For a set of unit size jobs with agreeable deadlines, RR computes an optimal schedule.*

In order to prove the above theorem we need a technical lemma.

**Lemma 1** *Let  $c > 0$  and  $\alpha > 1$ . Then the function*

$$f(x) = \left(\frac{1}{x}\right)^{\alpha-1} + \left(\frac{1}{c-x}\right)^{\alpha-1}$$

*with  $x \in (0, c)$  takes its global minimum at  $x = c/2$ . The function is strictly decreasing for  $x < c/2$  and strictly increasing for  $x > c/2$ .*

*Proof* Computing the derivative of  $f$  we find

$$f'(x) = (1 - \alpha) \left( \left(\frac{1}{x}\right)^\alpha - \left(\frac{1}{c-x}\right)^\alpha \right)$$

and  $x = c/2$  is indeed a global minimum. □

*Proof of Theorem 1* In this proof, for simplicity, we number jobs from 0 to  $n - 1$  and processors from 0 to  $m - 1$ . Moreover, the jobs are numbered as described in Step 1 of RR. We will show that there exists an optimal schedule in which job  $i$  is assigned to processor  $i \bmod m$ , for  $0 \leq i \leq n - 1$ . This is exactly the job assignment computed by *Round Robin* in Step 2 of RR. The theorem then follows because the algorithm by Yao et al. [30] constructs optimal single processor schedules.

Consider an optimal schedule  $S_{OPT}$ . We may assume without loss of generality that in  $S_{OPT}$  each processor  $j$ ,  $0 \leq j \leq m - 1$ , executes the assigned jobs non-preemptively in order of increasing job number. While this is not the case and there exists a processor  $j$  that executes a portion of a job  $i'$  before a job  $i$ , with  $i < i'$ , is completely finished, we can modify the schedule as follows. At those times where jobs  $i$  and  $i'$  are processed, we first schedule job  $i$  completely and only then schedule  $i'$ . This modification maintains feasibility and does not increase the energy consumption of the schedule.

Furthermore, in  $S_{OPT}$  we number the processors in non-decreasing order of *start times*, where ties may be broken arbitrarily. The start time of a processor is the first time it executes any job. Let processor  $j$  be the  $(j + 1)$ -st element in this sorted order,  $0 \leq j \leq m - 1$ . In particular, processor 0 is one with the earliest start time and processor  $m - 1$  is one with the latest start time, among all processors. Let  $a(i)$  be the time when the processing of job  $i$  starts and let  $b(i)$  be the time when the processing ends,  $0 \leq i \leq n - 1$ . We show inductively, for  $i = 0, \dots, n - 1$ , that the following invariant holds.

- (I) There exists an optimal schedule in which job  $k$  is scheduled on processor  $k \bmod m$ , for  $0 \leq k \leq i$ . On each processor the jobs are executed non-preemptively in increasing order of job number. Furthermore  $a(i) \leq a(l)$ , for  $i < l \leq n - 1$ , and  $b(k) \leq b(k + 1)$ , for  $0 \leq k < i$ .

For  $i = n - 1$ , the invariant implies the desired statement.

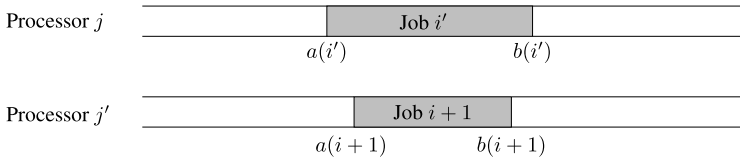
We first prove the induction basis of (I), for  $i = 0$ . If  $S_{OPT}$  executes job 0 on processor 0, then (I) holds. Suppose that in  $S_{OPT}$  the first job on processor 0 is job  $i_0$ , with  $i_0 \neq 0$ . Let  $j \neq 0$  be the processor processing job 0. Since each processor executes jobs in order of increasing job number, job 0 is the first one on processor  $j$ . It follows  $a(0) = a(i_0)$ : If  $a(0) > a(i_0)$ , then we would obtain a schedule with a strictly smaller energy consumption by starting job 0 already at time  $a(i_0)$ . The new schedule would be feasible as  $r(0) \leq r(i_0) \leq a(i_0)$ . Now, since  $a(0) = a(i_0)$ , we can simply swap processors 0 and  $j$ . Processors are still numbered in order of non-decreasing start times and we obtain  $a(0) \leq a(l)$ , for  $0 < l \leq n - 1$ . Hence the schedule satisfies invariant (I) for  $i = 0$ .

Suppose that invariant (I) holds for  $i$  and let  $S_{OPT}$  be the corresponding schedule. We prove that the invariant is also satisfied for  $i + 1$ . Let  $j = (i + 1) \bmod m$ . We first show that we may assume that in  $S_{OPT}$  processor  $j$  is one that first starts processing any job  $k$  with  $k \geq i + 1$ .

**Lemma 2** *There exists an optimal schedule that satisfies invariant (I) for value  $i$  and in which processor  $j = (i + 1) \bmod m$  is one that, among all processors, first starts processing any job  $l$  with  $l \geq i + 1$ . Processors are numbered in non-decreasing order of start times.*

*Proof* If in  $S_{OPT}$  processor  $j$  is one that first executes any job  $l \geq i + 1$ , then there is nothing to show. So suppose that this were not the case and that, instead, processor  $j'$  with  $j' \neq j$  is a processor satisfying this property. We distinguish two cases depending on whether or not  $i + 1 \leq m - 1$ .

If  $i + 1 \leq m - 1$ , then processor  $j = i + 1$  contains no jobs numbered smaller than  $i + 1$  because (I) holds for  $i$ . Furthermore, for  $0 \leq k < i + 1$ , job  $k$  is scheduled as first job on processor  $k$ . Since processors are numbered in non-decreasing order of start times, processor  $j'$  satisfies  $j' < j$  and, again, contains only job  $j'$  among jobs numbered smaller than  $i + 1$ . Now we swap the schedule on processor  $j$  with the schedule on  $j'$  after time  $b(j')$ . This is feasible because, by assumption, processor  $j'$  is one that first executes any job  $l \geq i + 1$ . After the swap processors are still numbered in order of non-decreasing start times because, since (I) holds for  $i$ ,  $a(i) \leq$



**Fig. 1** Jobs  $i'$  and  $i + 1$  if  $b(i') > b(i + 1)$

$a(l)$ , for any  $i < l$ , and hence the start time  $a(i)$  of processor  $j - 1 = i$  cannot be later than the new start time of processor  $j = i + 1$ . Moreover, since processor  $j'$  was one that first started executing any job  $l \geq i + 1$ , the new start time of processor  $j = i + 1$  cannot be later than the start time of any processor numbered at least  $i + 2$ .

If  $i + 1 > m - 1$ , then processors  $j$  and  $j'$  each have processed at least one job numbered smaller than  $i + 1$ . The last of these jobs executed on processor  $j$  is job  $i + 1 - m$  and, by the *Round-Robin* policy, the last of these jobs executed on processor  $j'$  is a job  $i'$  with  $i + 1 - m < i' < i + 1$ . Since invariant (I) holds for  $i$ , we have  $b(i + 1 - m) \leq b(i')$ . Thus we can swap the work assignment on processor  $j$  after  $b(i + 1 - m)$  with the work assignment on processor  $j'$  after  $b(i')$ . Again, this is feasible because processor  $j'$  is one that first executes any job  $l \geq i + 1$  and the original schedule on processor  $j$  after  $b(i + 1 - m)$  can safely be assigned to processor  $j'$ . Conversely, since  $b(i + 1 - m) \leq b(i')$ , the work on processor  $j'$  after  $b(i')$  can be moved to processor  $j$ .

In either case, we obtain a feasible schedule satisfying the desired property that processor  $j$  is one that first executes any job  $l$  with  $l \geq i + 1$ . The start and finishing times of the jobs have not changed so that invariant (I), for value  $i$ , also holds for the modified schedule. □

Let  $S_{OPT}$  be an optimal schedule satisfying the properties of Lemma 2. We next show that job  $i + 1$  can be scheduled on processor  $j$ . Suppose that job  $i + 1$  is scheduled on processor  $j' \neq j$ . If  $i + 1 \leq m - 1$ , let  $i'$  denote the first job executed on processor  $j$ . If  $i + 1 > m - 1$ , let  $i'$  be the first job executed on processor  $j$  after job  $i + 1 - m$ . The latter one is the job with the highest number  $k < i + 1$  on processor  $j$ . Job  $i'$  has the smallest number among jobs  $l \geq i + 1$  on processor  $j$ . Moreover,  $a(i') \leq a(i + 1)$  because processor  $j$  is first starting the execution of any job  $l \geq i + 1$ . If  $b(i') \leq b(i + 1)$ , we swap jobs  $i + 1$  and  $i'$  in the current schedule. This is feasible because  $r(i + 1) \leq r(i') \leq a(i')$  and  $b(i + 1) \leq d(i + 1) \leq d(i')$ , i.e. job  $i + 1$  is started no earlier than its release date and job  $i'$  is finished by its deadline. As desired, we obtain a schedule in which job  $i + 1$  is scheduled on processor  $j$ .

We next consider the case that  $b(i') > b(i + 1)$ , cf. Figure 1. Recall that  $a(i') \leq a(i + 1)$ . We then change the schedule as follows. We swap the schedule of processor  $j$  starting at time  $a(i')$  with the schedule on processor  $j'$  starting at time  $a(i + 1)$ . The new starting time of job  $i'$  is  $a'(i') = a(i + 1)$  and that of job  $i + 1$  is  $a'(i + 1) = a(i')$ . All other starting and finishing times of jobs remain the same. The modified schedule is feasible because  $r(i + 1) \leq r(i') \leq a(i')$ . If, originally,  $a(i') = a(i + 1)$ , then the energy consumption of the schedule has not changed during the modification. Again we obtain an optimal schedule in which job  $i + 1$  is scheduled on processor  $j$ . We

finally show that if  $a(i') < a(i + 1)$ , then the modified schedule has a strictly smaller energy consumption than the original one, contradicting the fact that the original schedule was optimal. Hence this scenario cannot occur. During the schedule modification, only the energy of jobs  $i + 1$  and  $i'$  changes. The energy consumed by jobs  $i + 1$  and  $i'$  in the original schedule is

$$E = \left(\frac{1}{b(i + 1) - a(i + 1)}\right)^{\alpha-1} + \left(\frac{1}{b(i') - a(i')}\right)^{\alpha-1}$$

while in the modified schedule it is

$$E' = \left(\frac{1}{b(i + 1) - a(i')}\right)^{\alpha-1} + \left(\frac{1}{b(i') - a(i + 1)}\right)^{\alpha-1}.$$

Set  $c = b(i + 1) + b(i') - a(i + 1) - a(i')$  and  $x_0 = b(i + 1) - a(i + 1)$ . Moreover, set  $x_1 = \min\{b(i + 1) - a(i'), b(i') - a(i + 1)\}$ . Then  $E = (1/x_0)^{1-\alpha} + (1/(c - x_0))^{1-\alpha}$  and  $E' = (1/x_1)^{1-\alpha} + (1/(c - x_1))^{1-\alpha}$ . By assumption  $b(i') > b(i + 1)$ . If  $a(i') < a(i + 1)$ , we have  $x_0 < x_1$  because  $x_0 = b(i + 1) - a(i + 1) < \min\{b(i + 1) - a(i'), b(i') - a(i + 1)\} = x_1$ , see again Figure 1. Furthermore,  $x_1 \leq c/2$ . Lemma 1 implies  $E' < E$ .

As desired, we obtain an optimal schedule in which job  $i + 1$  is scheduled on processor  $j$ . Moreover,  $a(i + 1) \leq a(l)$ , for  $i + 1 < l \leq n - 1$ , because processor  $j$  is one that first starts processing any job  $l \geq i + 1$ .

It remains to prove  $b(i) \leq b(i + 1)$  which, together with the induction hypothesis, implies  $b(k) \leq b(k + 1)$ , for  $1 \leq k < i + 1$ . Job  $i$  is scheduled on processor  $(j - 1) \bmod m$ . By induction hypothesis,  $a(i) \leq a(i + 1)$ . Suppose  $b(i) > b(i + 1)$ . We modify the schedule as follows. We exchange the finishing times of job  $i$  and  $i + 1$ , i.e. the new finishing time of job  $i$  is  $b'(i) = b(i + 1)$  and that of job  $i + 1$  is  $b'(i + 1) = b(i)$ . Job  $i + 1$  is still completed before its deadline because  $b'(i + 1) = b(i) \leq d(i) \leq d(i + 1)$ . To obtain a feasible schedule we additionally swap the work assignment on processor  $(j - 1) \bmod m$  after time  $b(i)$  with the work assignment on processor  $j$  after time  $b(i + 1)$ . If  $a(i) = a(i + 1)$ , then the energy consumption has not changed during the schedule modification. We next show that if  $a(i) < a(i + 1)$ , then the modified schedule consumes strictly less energy than the original one, contradicting the fact that the original schedule is optimal. The energy consumption of jobs  $i$  and  $i + 1$  in the original schedule is  $E = (1/(b(i) - a(i)))^{1-\alpha} + (1/(b(i + 1) - a(i + 1)))^{1-\alpha}$  while that in the modified schedule is  $E' = 1/((b(i + 1) - a(i)))^{1-\alpha} + (1/(b(i) - a(i + 1)))^{1-\alpha}$ . Set  $c = b(i) + b(i + 1) - a(i) - a(i + 1)$ ,  $x_0 = b(i + 1) - a(i + 1)$  and  $x_1 = \min\{b(i + 1) - a(i), b(i) - a(i + 1)\}$ . Using Lemma 1 we obtain  $E' < E$ . We conclude  $b(i) \leq b(i + 1)$  and the proof is complete.  $\square$

### 3 Unit Size Jobs with Arbitrary Release Dates and Deadlines

We consider problem instances consisting of unit size jobs with arbitrary release dates and deadlines. We show that the problem of minimizing energy consumption on par-



allel processors is NP-hard. Moreover, we develop a polynomial time algorithm that achieves a constant factor approximation guarantee independent of  $m$ .

**Theorem 2** *Given a set of unit size jobs with arbitrary release dates and deadlines, the problem of minimizing the total energy on two processors is NP-hard.*

**Theorem 3** *Given a set of unit size jobs with arbitrary release dates and deadlines, the problem of minimizing the total energy on  $m$  processors is strongly NP-hard.*

Obviously, the decision variants of the speed scaling problems under consideration are in the class NP. The proofs of both Theorem 2 and Theorem 3 rely on a reduction from the problem MULTI-PARTITION, which contains PARTITION and 3-PARTITION as special cases.

**Definition 1** Given a finite set  $A \subset \mathbb{Z}^+$  and a number  $m \in \mathbb{N}$  there holds  $(A, m) \in \text{MULTI-PARTITION}$  if and only if there are subsets  $A_1, A_2, \dots, A_m$  such that  $\bigcup_{i=1}^m A_i = A$  and, for all  $i \neq j$ ,

$$A_i \cap A_j = \emptyset \quad \text{and} \quad \sum_{a \in A_i} a = \sum_{a \in A_j} a.$$

We describe the reduction from an instance of MULTI-PARTITION to an instance of our speed scaling problem. Let  $A = \{a_1, a_2, \dots, a_n\} \subset \mathbb{Z}^+$  and  $m \in \mathbb{N}$  be an instance of MULTI-PARTITION. We construct a set  $\mathcal{J}$  of unit size jobs as follows. For every  $a_i \in A$  we generate a job  $i$ . We assign  $i$  a release date  $r(i) = \sum_{j < i} a_j$  and a deadline  $d(i) = r(i) + a_i$ . Additionally we create  $m$  jobs  $n + 1, n + 2, \dots, n + m$ , where  $r(n + 1) = r(n + 2) = \dots = r(n + m) = 0$  and  $d(n + 1) = d(n + 2) = \dots = d(n + m) = 3d(n)$ .

For any job  $i$  with  $1 \leq i \leq n + m$ , let  $l(i) = d(i) - r(i)$  be the length of  $i$ . Furthermore, let  $B = (\sum_{a \in A} a)/m$ . We can show that  $(A, m) \in \text{MULTI-PARTITION}$  if and only if an optimal schedule for  $\mathcal{J}$  incurs an energy of

$$E_{OPT} = \sum_{i=1}^n \left( \frac{1}{l(i)} \right)^{\alpha-1} + m \left( \frac{1}{3d(n) - B} \right)^{\alpha-1}.$$

A detailed proof, which yields the two above theorems, is given in the [Appendix](#).

We next develop a constant factor approximation algorithm. The algorithm, called *Classified Round Robin (CRR)*, first **divides** the given jobs into classes and then, when assigning jobs to processors, applies the *Round Robin* strategy independently to each class. Once the job assignment is done, for each processor, an optimal schedule is computed. Recall that each job has a processing requirement of  $p(i) = 1$ . Let  $\delta_i = 1/(d(i) - r(i))$  be the *density* of job  $i$ , which corresponds to the minimum average speed necessary to process the job in time if no other jobs were present. Let  $\mathcal{J}$  be the set of all jobs and  $\Delta = \max_{i \in \mathcal{J}} \delta_i$  be the maximum density of the jobs. We partition  $\mathcal{J}$  into classes  $C_k, k \geq 0$ , such that class  $C_0$  contains all jobs of density  $\Delta$  and  $C_k, k \geq 1$ , contains all jobs  $i$  with density  $\delta_i \in [\Delta 2^{-k}, \Delta 2^{-(k-1)})$ . Thus in each class job densities differ by a factor of at most 2.

**Algorithm CRR**

1. For each class  $C_k$ , first sort the jobs in non-decreasing order of release dates and then assign them to processors according to the *Round Robin* policy, ignoring job assignments done for other classes.
2. For each processor, given the jobs assigned to it, compute an optimal schedule.

We first present a lemma that relates energy consumptions in single-processor and  $m$ -processor schedules.

**Lemma 3** *For any set of jobs, the energy of an optimal schedule on  $m$  processors is at least  $1/m^{\alpha-1}$  times that of an optimal schedule on one processor.*

*Proof* For the given set of jobs, let  $S_{OPT}^m$  be an optimal schedule on  $m$  processors. We partition the time horizon of  $S_{OPT}^m$  into a set  $\mathcal{I}$  of intervals such that for any  $I \in \mathcal{I}$ , the speed does not change on any of the  $m$  processors throughout  $I$ . Let  $s_{I,j}$  be the speed of processor  $j$  in  $I$  and let  $s_I = \sum_{j=1}^m s_{I,j}$  be the total summed speed in that interval. The energy consumption of  $S_{OPT}^m$  is

$$E_{OPT} = \sum_{I \in \mathcal{I}} \sum_{j=1}^m |I| (s_{I,j})^\alpha \geq \sum_{I \in \mathcal{I}} |I| m (s_I/m)^\alpha = \frac{1}{m^{\alpha-1}} \sum_{I \in \mathcal{I}} |I| s_I^\alpha, \tag{1}$$

where the inequality follows from the convexity of the power consumption function.

Now consider the single processor schedule  $S^1$  in which the speed in interval  $I$  is set to  $s_I$ . Always processing the available job with the earliest deadline gives a feasible schedule: At any time the total amount of work that can be finished in  $S^1$  is exactly equal to that actually completed in  $S_{OPT}^m$ . In  $S^1$  we never run out of available jobs because there are jobs available in  $S_{OPT}^m$  at that time. Thus the work completed by  $S^1$  is exactly equal to that of  $S_{OPT}^m$ . Always sequencing available jobs according to the *Earliest Deadline* policy yields a feasible schedule. The energy consumption of  $S^1$  is  $\sum_{I \in \mathcal{I}} |I| s_I^\alpha \geq E_{OPT}^1$ , where  $E_{OPT}^1$  denotes the optimum energy of a single processor schedule. Combining the last inequality with (1) we obtain the lemma.  $\square$

In the following we analyze *CRR* for an arbitrary set  $\mathcal{J}$  of jobs. In a first step we transform  $\mathcal{J}$  into a set  $\mathcal{J}'$ . More specifically, for any job  $i \in \mathcal{J}$  belonging to class  $C_k$  we generate a unit size job  $i' \in \mathcal{J}'$  with release date  $r(i') = r(i)$  and deadline  $d(i') = r(i') + 2^k/\Delta$  in  $\mathcal{J}'$ . Hence the job’s density is  $\delta_{i'} = 1/(d(i') - r(i')) = \Delta/2^k$ , which is the smallest density in class  $C_k$ . We have  $d(i) \leq d(i')$  because  $d(i) = r(i) + 1/\delta_i \leq r(i') + 2^k/\Delta = d(i')$ . Thus  $\mathcal{J}'$  can be viewed as a relaxed problem instance in which jobs have later deadlines. Under the described transformation jobs do not change class and keep their original release date. Thus, *CRR* assigns jobs of  $\mathcal{J}'$  to exactly the same processors as jobs of  $\mathcal{J}$ . We will analyze *CRR* on the schedule for  $\mathcal{J}'$  and show that its energy consumption  $E'_{CRR}$  is bounded by  $\alpha^\alpha 2^{3\alpha}$  times the optimum energy  $E'_{OPT}$  for  $\mathcal{J}'$ . Obviously, the optimum energy  $E_{OPT}$  for the original set  $\mathcal{J}$  is at least  $E'_{OPT}$ . In a final step we will argue that the energy used by *CRR* on  $\mathcal{J}$  is at most  $2^\alpha$  times that spent by *CRR* on  $\mathcal{J}'$ . This establishes an approximation ratio of  $\alpha^\alpha 2^{4\alpha}$ .

We concentrate on job set  $\mathcal{J}'$ . The relevant scheduling horizon is  $[0, T)$ , where  $T = \max\{d(i') \mid i' \in \mathcal{J}'\}$ . For any  $t \in [0, T)$ , call a job  $i'$  *active at time  $t$*  if  $r(i') \leq t < d(i')$ . Let  $c_k(t)$  be the number of jobs of  $\mathcal{J}'$  that belong to  $C_k$  and are active at time  $t$ . The next lemma shows that *CRR* constructs balanced processor assignments with respect to each job class.

**Lemma 4** *For any time  $t$ , *CRR* assigns to each processor at most  $\lceil c_k(t)/m \rceil$  jobs  $i' \in \mathcal{J}'$  from  $C_k$  active at time  $t$ .*

*Proof* Fix a time  $t$  and a class  $C_k$ . All jobs of  $\mathcal{J}'$  belonging to  $C_k$  are active for exactly  $2^k/\Delta$  time units. When *CRR* has sorted the class  $C_k$  jobs in order of non-decreasing release dates, the jobs active at time  $t$  form a consecutive subsequence in the sorted job list. When the subsequence is assigned to processors using *Round Robin*, every  $m$ -th job is placed on a fixed processor. Thus each processor receives at most  $\lceil c_k(t)/m \rceil$  jobs. □

When analyzing *CRR* on  $\mathcal{J}'$ , rather than the optimal schedules constructed in Step 2 of the algorithm, we will consider schedules generated according to the *Average Rate (AVR)* algorithm by Yao et al. [30]. This algorithm sets processor speeds according to job densities. For any processor  $j$  and time  $t$ , where  $1 \leq j \leq m$  and  $t \in [0, T)$ , let  $c_{kj}(t)$  be the number of jobs from class  $C_k$  active at time  $t$  that have been assigned by *CRR* to processor  $j$ . Set the speed of processor  $j$  at time  $t$  to

$$s_j(t) = \sum_{k \geq 0} c_{kj}(t) \Delta / 2^k. \tag{2}$$

Sequencing available jobs on processor  $j$  according to the *Earliest Deadline* policy yields a feasible schedule. Let  $S'_{AVR,j}$  be the resulting schedule on processor  $j$  and  $E'_{AVR,j}$  be the energy consumption of  $S'_{AVR,j}$ . As *CRR* computes an optimal schedule for each processor, its total energy  $E'_{CRR}$  is bounded by

$$E'_{CRR} \leq \sum_{j=1}^m E'_{AVR,j}. \tag{3}$$

We next estimate the energy volumes  $E'_{AVR,j}$ ,  $1 \leq j \leq m$ . To this end we consider two energy bounds. Firstly, suppose that job  $i' \in \mathcal{J}'$  is processed at speed  $1/(d(i') - r(i'))$  throughout its active interval. The minimum energy necessary to complete the job is  $(d(i') - r(i'))^{1-\alpha}$  and hence the minimum energy necessary to complete all jobs  $i' \in \mathcal{J}'$  is at least

$$E'_{\min} = \sum_{i' \in \mathcal{J}'} (d(i') - r(i'))^{1-\alpha} = \sum_{k \geq 0} \sum_{i' \in C_k} (2^k/\Delta)^{1-\alpha}. \tag{4}$$

Secondly, we consider the single processor schedule  $S'_{AVR}$  constructed by *AVR* for  $\mathcal{J}'$ . More specifically, at time  $t$  the speed is set to

$$s(t) = \sum_{k \geq 0} c_k(t) \Delta / 2^k. \tag{5}$$

A result by Yao et al. [30] implies that the energy  $E'_{AVR}$  of  $S'_{AVR}$  is at most  $\alpha^\alpha 2^{\alpha-1}$  times the energy of an optimal single processor schedule. Using Lemma 3 we obtain that

$$E'_{AVR} \leq \alpha^\alpha 2^{\alpha-1} m^{\alpha-1} E'_{OPT}. \tag{6}$$

We will prove

$$\sum_{j=1}^m E'_{AVR,j} \leq 2^{2\alpha} (E'_{\min} + m^{1-\alpha} E'_{AVR}). \tag{7}$$

Combining inequalities (3), (6) and (7), and using the fact that  $E'_{\min} \leq E'_{OPT}$ , we obtain  $E'_{CRR} \leq \sum_{j=1}^m E'_{AVR,j} \leq \alpha^\alpha 2^{3\alpha} E'_{OPT}$ .

In order to prove (7), fix a processor  $j$  and a time  $t$ . Let  $K_1$  be the set of job class indices  $k$  such that exactly one job  $i' \in \mathcal{J}'$  from  $C_k$  active at time  $t$  is assigned by  $CRR$  to processor  $j$ . Set  $k_1 = \min\{k \mid k \in K_1\}$ . Similarly, let  $K_2$  be the set of job class indices  $k$  such that at least two jobs  $i' \in \mathcal{J}'$  from  $C_k$  active at time  $t$  are assigned by  $CRR$  to processor  $j$ . Using (2) and Lemma 4 we obtain

$$\begin{aligned} s_j(t) &= \sum_{k \in K_1} \Delta/2^k + \sum_{k \in K_2} c_{kj}(t) \Delta/2^k \leq \Delta/2^{k_1-1} + \sum_{k \in K_2} \lceil c_k(t)/m \rceil \Delta/2^k \\ &\leq \Delta/2^{k_1-1} + \sum_{k \in K_2} (2c_k(t)/m) \Delta/2^k. \end{aligned}$$

Using (5) we find

$$s_j(t) \leq 4 \max \left\{ \Delta/2^{k_1}, \frac{1}{m} s(t) \right\}. \tag{8}$$

Note that  $\Delta/2^{k_1}$  is the minimum average speed necessary to complete the job  $i' \in \mathcal{J}'$  from class  $C_{k_1}$  active at time  $t$  that was assigned by  $CRR$  to processor  $j$ . Let  $\mathcal{J}'_j$  be the set of jobs assigned by  $CRR$  to processor  $j$ . We integrate  $s_j(t)^\alpha$ , first over all  $t$  where the first term of the maximum of (8) is dominating, and then over all  $t$  where the second term of the maximum of (8) is dominating. Integration of the first term gives an upper bound on the energy consumption that is at most  $4^\alpha$  times the minimum energy necessary to complete jobs assigned to processor  $j$ , which is  $4^\alpha \sum_{k \geq 0} \sum_{i' \in C_k \cap \mathcal{J}'_j} (2^k/\Delta)^{1-\alpha}$ . Integration of the second term gives an upper bound of  $4^\alpha \frac{1}{m^\alpha} E'_{AVR}$ . Hence

$$E'_{AVR,j} \leq 4^\alpha \left( \sum_{k \geq 0} \sum_{i' \in C_k \cap \mathcal{J}'_j} (2^k/\Delta)^{1-\alpha} + \frac{1}{m^\alpha} E'_{AVR} \right).$$

Summing over all  $j$  and applying (4) we obtain (7). As argued above we may conclude  $E'_{CRR} \leq \alpha^\alpha 2^{3\alpha} E'_{OPT}$ . We finally observe that a job  $i \in \mathcal{J}$  has a density that is at most twice as high as that of the corresponding job  $i' \in \mathcal{J}'$ . Hence a doubling of the speeds in the schedules  $S_{AVR,j}$  yields a feasible schedule for  $\mathcal{J}$ . This establishes the following theorem.

**Theorem 4** For unit size jobs, algorithm CRR achieves an approximation ratio of  $\alpha^\alpha 2^{4\alpha}$ .

#### 4 Jobs with Arbitrary Processing Requirements

In this section we study the scenario that the jobs' processing requirements  $p(i)$  may take arbitrary values. We first assume that all jobs are released at time 0 and have individual deadlines. We present a polynomial time algorithm that achieves an approximation factor of  $2(2 - \frac{1}{m})^\alpha$ . The strategy can also be used to handle jobs with individual release dates and a common deadline. We then consider the setting with agreeable deadlines.

Suppose that we are given jobs with  $r(i) = 0$ , for all  $i$ . The deadlines  $d(i)$  may take arbitrary values. Our strategy combines *Earliest Deadline* and *List scheduling* to assign jobs to processors. At any time, let the *load* of a processor be the sum of the  $p(i)$ 's currently assigned to it.

##### Algorithm EDL

1. Number the jobs in order of non-decreasing deadlines, i.e.  $d(1) \leq \dots \leq d(n)$ .
2. Consider the jobs one by one in the order computed in Step 1. Assign each job to the processor that currently has the smallest load.
3. For each processor, given the jobs assigned to it, compute an optimal schedule using the optimal offline algorithm for a single processor.

In the following we evaluate *EDL* and first give an outline of the analysis. For any processor  $j$ , we define a speed function and prove that using this speed function all jobs assigned by *EDL* to processor  $j$  can be completed by their deadline. As *EDL* computes an optimal schedule for the jobs on processor  $j$ , its energy on processor  $j$  cannot be larger than the energy  $E_j$  used by our speed function. In a second step we show that  $\sum_{j=1}^m E_j$  is upper bounded by  $2(2 - \frac{1}{m})^\alpha$  times the total energy incurred by an optimal solution.

We assume that every processor in *EDL*'s schedule processes at least one job since otherwise every processor processes at most one job and the global schedule is optimal. For any job  $i$ , let  $L(i) = \sum_{i'=1}^i p(i')$  be the total processing volume up to job  $i$ . Fix a processor  $j$  and let  $S_j$  be the set of jobs scheduled by *EDL* on processor  $j$ . In order to define the speed function, we have to consider load densities over the entire time horizon. The load density of an interval is the total work to be completed during that interval divided by the length of the interval. We identify an integer sequence  $\lambda_1^j < \lambda_2^j < \dots < \lambda_{i_j}^j$  such that the highest density occurs in interval  $[0, d(\lambda_1^j))$  among all  $[0, d(i))$  with  $i \in S_j$ , the second highest density occurs in interval  $[d(\lambda_1^j), d(\lambda_2^j))$  among all  $[d(\lambda_1^j), d(i))$  with  $i \in S_j$  and so on. Formally, let  $\lambda_0^j = 0$ ,  $d(0) = 0$  and  $L(0) = 0$ . Suppose that  $\lambda_0^j < \dots < \lambda_i^j$  have been defined and that  $\lambda_i^j$  is not equal to the highest job number in  $S_j$ . Then  $\lambda_{i+1}^j$  identifies a highest density interval after

$d(\lambda_l^j)$  assuming that a load of exactly  $L(\lambda_l^j)$  is completed by  $d(\lambda_l^j)$ , i.e.

$$\lambda_{l+1}^j = \operatorname{argmax}_{k \in S_j, k > \lambda_l^j} \frac{L(k) - L(\lambda_l^j)}{d(k) - d(\lambda_l^j)}.$$

Assuming that a load of exactly  $L(\lambda_{l-1}^j)$  is finished by time  $d(\lambda_{l-1}^j)$ , a load of  $L(\lambda_l^j) - L(\lambda_{l-1}^j)$  has to be processed on the  $m$  processors between time  $d(\lambda_{l-1}^j)$  and  $d(\lambda_l^j)$  and, for  $l = 1, \dots, l_j$ , we define

$$s_l^j = \frac{1}{m} \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)} \tag{9}$$

as the minimum average speed to accomplish this. We observe that

$$s_1^j \geq s_2^j \geq \dots \geq s_{l_j}^j, \tag{10}$$

for, if there were an index  $l$  with  $s_l^j < s_{l+1}^j$ , then  $\frac{L(\lambda_{l+1}^j) - L(\lambda_l^j)}{d(\lambda_{l+1}^j) - d(\lambda_l^j)} > \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)}$  and hence

$$\frac{L(\lambda_{l+1}^j) - L(\lambda_{l-1}^j)}{d(\lambda_{l+1}^j) - d(\lambda_{l-1}^j)} = \frac{L(\lambda_{l+1}^j) - L(\lambda_l^j) + L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_{l+1}^j) - d(\lambda_l^j) + d(\lambda_l^j) - d(\lambda_{l-1}^j)} > \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)},$$

contradicting the choice of  $\lambda_l^j$ . We are now ready to specify the speed function.

**Speed function for processor  $j$**

1. Initial setting: For any  $l = 1, \dots, l_j$ , set the speed in interval  $[d(\lambda_{l-1}^j), d(\lambda_l^j))$  to  $(2 - \frac{1}{m})s_l^j$ .
2. Adjustment: For any  $i \in S_j$  with  $p(i) > L(i)/m$  consider the time interval  $[0, d(i))$ . For any interval  $I \subseteq [0, d(i))$  in which the speed is strictly lower than  $(2 - \frac{1}{m})p(i)/d(i)$  raise the speed to that value.

**Lemma 5** *Using the above speed function, all jobs in  $S_j$  are completed by their deadline.*

*Proof* On processor  $j$  we schedule the jobs in  $S_j$  in increasing order of job number. Thus the jobs are scheduled in non-decreasing order of deadlines. We first consider any job  $i \in S_j$  with  $p(i) \leq L(i)/m$  and then any  $i \in S_j$  with  $p(i) > L(i)/m$ . In both cases we will prove that the job is finished by its deadline.

Fix any  $i \in S_j$  with  $p(i) \leq L(i)/m$ . We will show that after the initial speed setting in Step 1 of the speed function definition, the job is finished by  $d(i)$ . As the speed can only increase in the adjustment Step 2, the lemma then holds for this job  $i$ . Let  $k$

be the largest integer such that  $\lambda_k^j \leq i$ . By time  $d(\lambda_k^j)$  a total load of

$$\begin{aligned} & \sum_{l=1}^k \left(2 - \frac{1}{m}\right) s_l^j (d(\lambda_l^j) - d(\lambda_{l-1}^j)) \\ &= \left(2 - \frac{1}{m}\right) \sum_{l=1}^k \frac{1}{m} \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)} (d(\lambda_l^j) - d(\lambda_{l-1}^j)) \\ &= \left(2 - \frac{1}{m}\right) L(\lambda_k^j) / m \end{aligned} \tag{11}$$

can be completed on processor  $j$ . If  $i > \lambda_k^j$ , then between time  $d(\lambda_k^j)$  and  $d(i)$  a load of

$$\begin{aligned} \left(2 - \frac{1}{m}\right) s_{k+1}^j (d(i) - d(\lambda_k^j)) &= \left(2 - \frac{1}{m}\right) \frac{1}{m} \frac{L(\lambda_{k+1}^j) - L(\lambda_k^j)}{d(\lambda_{k+1}^j) - d(\lambda_k^j)} (d(i) - d(\lambda_k^j)) \\ &\geq \left(2 - \frac{1}{m}\right) \frac{1}{m} \frac{L(i) - L(\lambda_k^j)}{d(i) - d(\lambda_k^j)} (d(i) - d(\lambda_k^j)) \\ &= \left(2 - \frac{1}{m}\right) (L(i) - L(\lambda_k^j)) / m \end{aligned} \tag{12}$$

can be completed. The inequality follows from the definition of  $\lambda_{k+1}^j$ . Combining (11) and (12) we find that a total load of at least  $(2 - \frac{1}{m})L(i)/m$  can be finished on processor  $j$  by time  $d(i)$ . It remains to argue that the total processing requirement of jobs scheduled on processor  $j$  before job  $i$  and including  $p(i)$  is at most  $(2 - \frac{1}{m})L(i)/m$ . To this end consider the event when *EDL* assigns job  $i$  to processor  $j$ . As the job is placed on the least loaded processor, just after the assignment processor  $j$  has a load of at most  $\frac{1}{m} \sum_{i' < i} p(i') + p(i) \leq (2 - \frac{1}{m})L(i)/m$ , and we are done because jobs assigned to processor  $j$  at a later stage are scheduled after job  $i$ .

Next we examine a job  $i$  with  $p(i) > L(i)/m$ . After the speed adjustment in Step 2 of the speed function definition, processor  $j$  runs at a speed of at least  $(2 - \frac{1}{m})p(i)/d(i)$  throughout  $[0, d(i))$ . Thus a total work of at least  $(2 - \frac{1}{m})p(i)$  can be finished by  $d(i)$ . Again, when *EDL* assigns job  $i$  to processor  $j$ , the total load on the processor is upper bounded by  $\frac{1}{m} \sum_{i' < i} p(i') + p(i) \leq (2 - \frac{1}{m})p(i)$  and this is indeed the total work of jobs scheduled on processor  $j$  up to (and including) job  $i$ . □

We compare the energy incurred using our speed function to the energy of an optimal solution. Let

$$E_j^1 = \sum_{l=1}^{l_j} (s_l^j)^\alpha (d(\lambda_l^j) - d(\lambda_{l-1}^j)).$$

This expression represents the energy consumed by processor  $j$  if a speed of  $s_l^j$  is used in the interval  $[d(\lambda_{l-1}^j), d(\lambda_l^j))$ , for  $l = 1, \dots, l_j$ . Hence, compared to the speed function defined in the initial setting, the speeds are reduced by a factor of  $2 - \frac{1}{m}$ .

**Lemma 6** *An optimal solution uses a total energy on the  $m$  processors of at least  $mE_j^1$ , for any  $1 \leq j \leq m$ .*

*Proof* Given an optimal schedule, let  $s_{l,\text{opt}}$  be the average speed of the  $m$  processors during the time interval  $[d(\lambda_{l-1}^j), d(\lambda_l^j))$ , for  $l = 1, \dots, l_j$ . By the convexity of the power function, the total energy used by the optimal solution is

$$E_{\text{OPT}} \geq m \sum_{l=1}^{l_j} (s_{l,\text{opt}})^\alpha (d(\lambda_l^j) - d(\lambda_{l-1}^j)).$$

The speeds  $s_{l,\text{opt}}$  must satisfy the constraint that at time  $d(\lambda_k^j)$  a load of at least  $L(\lambda_k^j)$  is completed, for  $k = 1, \dots, l_j$ . In the following let  $\delta_l^j = d(\lambda_l^j) - d(\lambda_{l-1}^j)$ . We next show that the speeds  $s_l^j$ , with  $1 \leq l \leq l_j$ , defined in (9) minimize the function  $f(x_1, \dots, x_{l_j}) = m \sum_{l=1}^{l_j} x_l^\alpha \delta_l^j$  subject to the constraint

$$m \sum_{l=1}^k x_l \delta_l^j \geq L(\lambda_k^j), \tag{13}$$

for  $k = 1, \dots, l_j$ . This implies, as desired,  $E_{\text{OPT}} \geq mE_j^1$ . Suppose  $(y_1, \dots, y_{l_j})$  with  $(y_1, \dots, y_{l_j}) \neq (s_1^j, \dots, s_{l_j}^j)$  is an optimal solution. Note that

$$m \sum_{l=1}^k y_l \delta_l^j = L(\lambda_k^j), \tag{14}$$

for  $k = 1, \dots, l_j$ . Thus there must exist a  $k$  with  $y_k > s_k^j$ : If  $y_l \leq s_l^j$  held for  $l = 1, \dots, l_j$ , then there would be a  $k'$  with  $y_{k'} < s_{k'}^j$  and hence  $m \sum_{l=1}^{k'} y_l \delta_l^j < L(\lambda_{k'}^j)$ , resulting in a violation of constraint (13) for  $k = k'$ . Let  $k_1$  be the smallest index such that  $y_{k_1} > s_{k_1}^j$ . We have  $y_l = s_l^j$ , for  $l = 1, \dots, k_1 - 1$  since otherwise, using the same argument as before, constraint (13) would be violated for  $k = k_1 - 1$ . Let  $k_2$  with  $k_2 > k_1$  be the smallest index such that  $y_{k_2} > s_{k_2}^j$ . Such an index exists because otherwise invariant (10) implies  $y_l > s_l^j$ , for  $l = k_1, \dots, l_j$ , and we find  $m \sum_{l=1}^{l_j} y_l \delta_l^j > L(\lambda_{l_j}^j)$ . In this case we could reduce  $y_{l_j}$ , achieving a smaller objective function value  $f$  and hence a contradiction to the optimality of the  $y_l, 1 \leq l \leq l_j$ .

We now decrease  $y_{k_1}$  by  $\epsilon$  and increase  $y_{k_2}$  by  $\epsilon \delta_{k_1} / \delta_{k_2}$ , where  $0 < \epsilon \leq \min\{y_{k_1} - s_{k_1}^j, (y_{k_1} - y_{k_2}) / (1 + \delta_{k_1} / \delta_{k_2})\}$ . We argue that constraints (13) are still satisfied. There is nothing to show for  $k = 1, \dots, k_1 - 1$ . Also for  $k = k_2, \dots, l_j$  there is nothing to show because the work reduction in interval  $[d(\lambda_{k_1-1}^j), d(\lambda_{k_1}^j))$  is  $\epsilon \delta_{k_1}$  while the work



increase in interval  $[d(\lambda_{k_2-1}^j), d(\lambda_{k_2}^j))$  is  $\delta_{k_2} \epsilon \delta_{k_1} / \delta_{k_2} = \epsilon \delta_{k_1}$ , yielding a net change of 0. By the choice of  $\epsilon$  we have  $y_{k_1} - \epsilon \geq s_{k_1}^j$ , and  $y_l \geq y_{k_1} > s_{k_1}^j$  as well as (10) imply  $y_l > s_l^j$  for  $l = k_1 + 1, \dots, k_2 - 1$ . Using the fact that equations (14) hold we obtain that constraints (13) are satisfied for  $l = k_1, \dots, k_2 - 1$ .

We finally show that the modification of  $y_{k_1}$  and  $y_{k_2}$  leads to a strict reduction in the value of  $f$ . The reduction is given by  $g(\epsilon) = \delta_{k_1} (y_{k_1}^\alpha - (y_{k_1} - \epsilon)^\alpha) - \delta_{k_2} ((y_{k_2} + \epsilon \delta_{k_1} / \delta_{k_2})^\alpha - y_{k_2}^\alpha)$ . This function is strictly positive for the considered range of  $\epsilon$  because  $g(0) = 0$  and  $g(\epsilon)$  is increasing since the first derivative  $g'(\epsilon) = \alpha \delta_{k_1} ((y_{k_1} - \epsilon)^{\alpha-1} - (y_{k_2} + \epsilon \delta_{k_1} / \delta_{k_2})^{\alpha-1})$  is positive for  $\epsilon < (y_{k_1} - y_{k_2}) / (1 + \delta_{k_1} / \delta_{k_2})$ . We conclude that  $(y_1, \dots, y_l)$  is not optimal.  $\square$

We now combine the speed functions for all processors  $j$ .

**Lemma 7** *An optimal solution uses a total energy of at least  $\sum_{j=1}^m E_j^1$ .*

*Proof* Using Lemma 6 and summing over all  $j$ , we find  $mE_{\text{OPT}} \geq m \sum_{j=1}^m E_j^1$ , where  $E_{\text{OPT}}$  is the energy of an optimal solution. Dividing by  $m$  we obtain the desired statement.  $\square$

For any  $j$ , let  $S'_j$  be the set of jobs  $i$  with  $i \in S_j$  and  $p(i) > L(i)/m$ . Define  $E_j^2 = \sum_{i \in S'_j} (p(i)/d(i))^\alpha d(i)$ .

**Lemma 8** *An optimal solution uses a total energy of at least  $\sum_{j=1}^m E_j^2$ .*

*Proof* Consider an optimal solution and suppose that it processes job  $i \in S'_j$ , with  $1 \leq j \leq m$ , at speed  $s$ . Then the energy used to complete the job is  $s^\alpha p(i)/s = s^{\alpha-1} p(i)$  and this expression is increasing in  $s$ . The minimum speed necessary to finish the job in time is  $p(i)/d(i)$  and hence the energy used for job  $i$  is at least  $(p(i)/d(i))^{\alpha-1} p(i) = (p(i)/d(i))^\alpha d(i)$  and the lemma follows by summing the latter expression for all  $i \in S_j$  and all processors  $j$ .  $\square$

**Theorem 5** *For arbitrary size jobs released at time 0, EDL achieves an approximation ratio of at most  $2(2 - \frac{1}{m})^\alpha$ .*

*Proof* We first evaluate the total energy  $E_{SF}$  used by the speed functions on all  $m$  processors. For any processor  $j$ , the initial Step 1 of the speed function definition requires a speed of  $(2 - \frac{1}{m})^\alpha E_1^j$ . The adjustment Step 2 requires a total energy of at most  $(2 - \frac{1}{m})^\alpha E_2^j$  in the intervals modified. Thus  $E_{SF} \leq (2 - \frac{1}{m})^\alpha \sum_{j=1}^m (E_1^j + E_2^j)$  and, using Lemmas 7 and 8, we find  $E_{SF} \leq (2 - \frac{1}{m})^\alpha 2E_{\text{OPT}}$ , where  $E_{\text{OPT}}$  is the total energy of an optimal solution. In Lemma 5 we showed that the speed functions give feasible schedules for any  $S_j$ . Algorithm *EDL* computes a feasible schedule with minimum energy for any  $S_j$ . We conclude that the total energy of *EDL* is bounded by  $E_{SF}$ .  $\square$

Obviously, by interchanging release dates and deadlines, *EDL* can also handle the case of jobs with individual release dates but a common deadline.

**Corollary 1** *For arbitrary size jobs with individual release dates that have to be finished by a common deadline, EDL achieves an approximation ratio of at most  $2(2 - \frac{1}{m})^\alpha$ .*

We next consider the scenario where jobs have (general) agreeable deadlines. Again the jobs’ processing requirements may take arbitrary values. It turns out that we can apply the algorithm *CRR* presented in Section 3. We only have to generalize the definition of job densities. Here, for any job  $i$ , the density is  $\delta_i = p(i)/(d(i) - r(i))$ , which represents again the minimum speed to finish the job in time. Let  $\mathcal{J}$  be the set of all jobs and  $\Delta = \max_{i \in \mathcal{J}} \delta_i$  be the maximum density. We partition  $\mathcal{J}$  into job classes as before. We can then apply *CRR* to a given job instance, where within each class  $C_k$  jobs having the same release date are sorted in order of non-decreasing deadlines.

**Theorem 6** *For arbitrary size jobs with agreeable deadlines, algorithm CRR achieves an approximation ratio of  $\alpha^\alpha 2^{4\alpha}$ .*

*Proof* The proof is similar to that of Theorem 4 and we just sketch the difference. Again we introduce a job set  $\mathcal{J}'$ . Here we just scale the job densities without changing release dates or deadlines. More specifically, for any  $\mathcal{J}$  of class  $C_k$  we introduce a job  $i'$  with  $r(i') = r(i)$ ,  $d(i') = d(i)$  and density  $\delta_{i'} = \Delta/2^k$ , i.e.  $p(i') = (d(i') - r(i'))\Delta/2^k$ . Lemma 4 carries over. The proof of Lemma 4 for unit size jobs made use of the fact that, for  $\mathcal{J}$ , job deadlines are agreeable within each class  $C_k$ . In our considered scenario with arbitrary size jobs the deadlines are agreeable anyway. In the further analysis we also consider schedules constructed by *Average Rate (AVR)*. Let  $S'_{AVR,j}$  be the schedule generated by *AVR* on the job set  $\mathcal{J}'_j$  that is assigned to processor  $j$  by *CRR*. Let  $S'_{AVR}$  be the single processor schedule of *AVR* on the entire set  $\mathcal{J}'$ . The corresponding energy volumes of *AVR*’s schedule are denoted by  $E'_{AVR,j}$  and  $E'_{AVR}$ , respectively. We can prove  $\sum_{j=1}^m E'_{AVR,j} \leq 2^{2\alpha} (E'_{\min} + m^{1-\alpha} E'_{AVR})$ , where

$$E'_{\min} = \sum_{i' \in \mathcal{J}'} (p(i')/(d(i') - r(i')))^{\alpha-1} p(i') = \sum_{k \geq 0} \sum_{i' \in C_k} (2^k/\Delta)^{1-\alpha} p(i')$$

is the minimum energy to complete all jobs in  $\mathcal{J}'$ . We then derive  $E'_{CRR} \leq \sum_{j=1}^m E'_{AVR,j} \leq \alpha^\alpha 2^{3\alpha} E'_{OPT}$ . Since  $\delta_i$  and  $\delta_{i'}$  differ by a factor of at most 2, a doubling of the speeds in  $S'_{AVR,j}$  yields optimal schedules for  $\mathcal{J}$ . The theorem follows.  $\square$

### 5 Online Algorithms

The algorithms we have presented in the previous sections can be modified so that they work in an online scenario where jobs arrive over time. More specifically, a

job  $i$  together with its characteristics  $d(i)$  and  $p(i)$  becomes available at its release date  $r(i)$ . The job must be assigned to a processor without knowledge of future jobs arriving at times  $t > r(i)$ .

All our offline algorithms first assign jobs to processors and then, on each processor, construct an optimal schedule for the job set assigned to it. In the online setting, we keep the assignment of jobs to processors but, instead of constructing optimal schedules, apply a single processor **online algorithm**. For unit size jobs with agreeable deadlines, we again assign the incoming jobs to processors using *Round Robin*. On each processor we apply an online algorithm by Bansal et al. [3] that achieves a competitive ratio of  $2(\alpha/(\alpha - 1))^\alpha e^\alpha$ . Let *RR-ON* be the resulting algorithm.

**Theorem 7** *For unit size jobs with agreeable deadlines, CRR-ON achieves a competitive ratio of  $2(\alpha/(\alpha - 1))^\alpha e^\alpha$ .*

As for the algorithm *CRR*, we define jobs classes that are centered around  $\delta_1$ , the density of the first incoming job. As usual, job densities within each class differ by a factor less than 2. More precisely, for densities of at least  $\delta_1$ , there are job classes  $[2^k \delta_1, 2^{k+1} \delta_1)$ , for  $k \geq 0$ . For smaller densities there are classes  $[2^{-k} \delta_1, 2^{-(k-1)} \delta_1)$ , for  $k \geq 1$ . While jobs are assigned to processors, instead of computing optimal schedules, which would be impossible, we apply the *Average Rate* algorithm [30]. Let *CRR-ON* denote the resulting strategy. Our analyses of *CRR* in Sections 3 and 4 in fact assumed that *Average Rate* is executed on each processor. Thus the proven approximation ratios do not increase.

**Theorem 8** *For unit size jobs with arbitrary release dates and deadlines and for arbitrary size jobs with agreeable deadlines, algorithm CRR-ON achieves a competitive ratio of  $\alpha^\alpha 2^{4\alpha}$ .*

## 6 Conclusions and Open Problems

This paper contains the first algorithmic study of classical deadline-based scheduling in variable-speed multiprocessor environments assuming that jobs may have individual release dates and deadlines. We proposed simple algorithms based on job classification and natural dispatching rules that can also be used in distributed environments. The major open problem is to devise a deterministic constant-competitive online algorithm for the general setting where jobs have arbitrary release dates, deadlines and processing volumes, or to show that such a constant factor performance guarantee is impossible. Another interesting working direction is to consider speed-bounded processors as well as extended architectures where processors are equipped with an additional sleep state into which they can be transitioned when idle.

## Appendix

In this appendix we establish the NP-hardness results stated in Theorems 2 and 3 based on the reduction described in Section 3. Given a schedule  $S$ , let  $l_S(i)$  be the

total length of intervals in  $S$  where job  $i$  is executed. The energy consumption of job  $i$  is at least  $(1/l_S(i))^{\alpha-1}$ . Furthermore, for a set  $\mathcal{S}$  of jobs (or intervals in  $S$ ) let  $l(\mathcal{S})$  ( $l_S(\mathcal{S})$ , respectively) denote the sum of the lengths of all jobs (intervals) in  $\mathcal{S}$ .

We will determine the minimum energy necessary to schedule  $\mathcal{J}$  on  $m$  processors. To this end we need two lemmas on the structure of an optimal schedule for  $\mathcal{J}$ .

**Lemma 9** *In any optimal schedule  $S$  for  $\mathcal{J}$  no two jobs numbered larger than  $n$  are executed on the same processor.*

*Proof* Consider an optimal schedule  $S$  and assume for the sake of contradiction there were jobs  $i > n$  and  $i' > n$  running on the same processor  $j$ . Then one of the two jobs, say job  $i$ , is executed in intervals of total length at most  $3d(n)/2$  and incurs an energy consumption of at least  $(2/(3d(n)))^{\alpha-1}$ . Furthermore, there must exist one processor on which no job is executed in the interval  $[d(n), 3d(n))$  because we have only  $m$  jobs with a deadline greater than  $d(n)$  and two of these are executed on processor  $j$ . We can now schedule job  $i$  in this idle period, generating an energy consumption of at most  $(1/(2d(n)))^{\alpha-1}$  for job  $i$ . This is less than the initial consumption, contradicting the optimality of the considered schedule.  $\square$

**Lemma 10** *In any optimal schedule  $S$  for  $\mathcal{J}$  each job  $i$ , with  $1 \leq i \leq n$ , is processed continuously throughout its execution interval  $[r(i), d(i))$ .*

*Proof* Consider any optimal schedule  $S$ . A first observation is that  $S$  does not contain processor idle times, i.e. each processor executes jobs throughout  $[0, 3d(n))$ : Suppose there were some processor  $j$  and an interval  $I \subseteq [0, 3d(n))$  during which the processor would not execute any job. By Lemma 9 some job  $n+k$ , where  $1 \leq k \leq m$ , is executed in processor  $j$ . We now modify  $S$  by executing job  $n+k$  additionally during  $I$ . Thereby we reduce the speed needed to process job  $n+k$  and hence the total energy of the schedule.

Assume that in  $S$  some job  $i$  is not executed over its full possible interval  $I = [r(i), d(i))$ , i.e.  $l_S(i) = l(i) - \epsilon$  for some  $0 < \epsilon < l(i)$ . Since the execution intervals of all jobs  $i'$  with  $1 \leq i' \leq n$  are pairwise disjoint, no such job  $i' \neq i$  can be scheduled in  $I$ . Hence, by the above observation, some job  $n+k$ , with  $1 \leq k \leq m$ , is partially executed in  $I$  on the processor where  $i$  is scheduled. As shown in Lemma 9 only this one job is processed in  $I$ . Furthermore,  $l_S(n+k) > 2d(n)$ . Now we can construct a better schedule by executing job  $i$  over its full possible length  $l(i)$  and reducing the execution interval of job  $n+k$  by  $\epsilon$ . This modified schedule has a strictly smaller energy consumption because the energy consumed by jobs  $i$  and  $n+k$  satisfies

$$\underbrace{\left(\frac{1}{l_S(i)}\right)^{\alpha-1} + \left(\frac{1}{l_S(n+k)}\right)^{\alpha-1}}_{\text{energy consumption in initial schedule}} > \underbrace{\left(\frac{1}{l_S(i) + \epsilon}\right)^{\alpha-1} + \left(\frac{1}{l_S(n+k) - \epsilon}\right)^{\alpha-1}}_{\text{energy consumption in new schedule}}.$$

This follows from Lemma 1, by setting  $c = l_S(n+k) + l_S(i)$  and  $x = l_S(i)$ . We note that  $l_S(i) < l_S(i) + \epsilon = l(i) < c/2$  as  $l(i) \leq d(n)$  and  $l_S(n+k) > 2d(n)$ . This contradicts the assumption that schedule  $S$  was optimal.  $\square$

**Theorem 9** Let  $A \subset \mathbb{Z}^+$ ,  $m \in \mathbb{N}$  and  $B = (\sum_{a \in A} a)/m$ .  $(A, m) \in \text{MULTI-PARTITION}$  if and only if an optimal schedule for  $\mathcal{J}$  incurs an energy of

$$E_{OPT} = \sum_{i=1}^n \left( \frac{1}{l(i)} \right)^{\alpha-1} + m \left( \frac{1}{3d(n) - B} \right)^{\alpha-1}.$$

*Proof* By Lemma 10, in any optimal schedule each job  $i$ , with  $1 \leq i \leq n$ , is processed continuously throughout its execution interval  $[r(i), d(i))$ . Hence it requires an energy of exactly  $(1/l(i))^{\alpha-1}$ , and the total energy spent for jobs  $1, \dots, n$  is  $\sum_{i=1}^n (1/l(i))^{\alpha-1}$ . Thus the energy consumption of an optimal solution depends only on the energy consumed by jobs  $n+1, n+2, \dots, n+m$ . Let  $k \in \{1, 2, \dots, m\}$ . By Lemma 9, all these jobs are executed on separate machines. We assume w.l.o.g. that job  $n+k$  is executed on processor  $k$ . Let  $\mathcal{J}_k \subseteq \{1, 2, \dots, n\}$  be the set of jobs scheduled on processor  $k$ . We can now easily compute the energy used by  $n+k$  as

$$\left( \frac{1}{l_S(n+k)} \right)^{\alpha-1} = \left( \frac{1}{3d(n) - \sum_{i \in \mathcal{J}_k} l(i)} \right)^{\alpha-1}.$$

By Lemma 1 we find that the sum of the energy consumptions of jobs  $n+1, n+2, \dots, n+m$  is minimal if and only if  $l_S(n+1) = l_S(n+2) = \dots = l_S(n+m)$ . This is the case if and only if  $\sum_{i \in \mathcal{J}_1} l(i) = \sum_{i \in \mathcal{J}_2} l(i) = \dots = \sum_{i \in \mathcal{J}_m} l(i) = d(n)/m = B$ . By our construction this is possible if and only if there exist sets  $A_1, A_2, \dots, A_m \subseteq A$  with  $\bigcup_{i=1}^m A_i = A$  and, for all  $i \neq j$ , there holds  $A_i \cap A_j = \emptyset$  as well as  $\sum_{a \in A_i} a = \sum_{a \in A} a$ . Finally, this is the case if and only if  $(A, m) \in \text{MULTI-PARTITION}$ .  $\square$

We are now ready to derive the desired NP-hardness results. Setting  $m = 2$ , the NP-hard PARTITION problem [18] is a special case of our MULTI-PARTITION problem. Theorem 9 yields Theorem 2. Another special case of MULTI-PARTITION is 3-PARTITION. In this case  $A$  satisfies the property that there exists a  $B \in \mathbb{Z}^+$  such that, for all  $a \in A$ , we have  $B/4 < a < B/2$  and  $\sum_{a \in A} a = mB$ . Since 3-PARTITION is strongly NP-hard [18], we obtain Theorem 3 as a consequence of Theorem 9.

## References

1. Albers, S.: Energy-efficient algorithms. *Commun. ACM* **53**(5), 86–96 (2010)
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms* **3**(4), 49 (2007)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. *J. ACM* **54**(1), 3 (2007)
4. Bansal, N., Chan, H.-L., Lam, T.-W., Lee, K.-L.: Scheduling for speed bounded processors. In: *Proc. 35th International Colloquium on Automata, Languages and Programming*. LNCS, vol. 5125, pp. 409–420. Springer, Berlin (2008)
5. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 693–701 (2009)
6. Bansal, N., Chan, H.-L., Pruhs, K., Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: *Proc. 36th International Colloquium on Automata, Languages and Programming*. LNCS, vol. 5555, pp. 144–155. Springer, Berlin (2009)

7. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. *SIAM J. Comput.* **39**(4), 1294–1308 (2009)
8. Bansal, N., Bunde, D.P., Chan, H.-L., Pruhs, K.: Average rate speed scaling. *Algorithmica* **60**(4), 877–889 (2011)
9. Baptiste, P.: Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In: Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 364–367 (2006)
10. Baptiste, P., Chrobak, M., Dürr, C.: Polynomial time algorithms for minimum energy scheduling. In: Proc. 15th Annual European Symposium on Algorithms. LNCS, vol. 4698, pp. 136–150. Springer, Berlin (2007)
11. Barroso, L.A.: The price of performance. *ACM Queue* **3**(7), 48–53 (2005)
12. Bell, P.C., Wong, P.W.H.: Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. In: Proc. 8th Annual Conference on Theory and Applications of Models of Computation (TAMC). LNCS, vol. 6648, pp. 27–36. Springer, Berlin (2011)
13. Bunde, D.P.: Power-aware scheduling for makespan and flow. *J. Sched.* **12**, 489–500 (2009)
14. Chan, H.-L., Chan, J.W.-T., Lam, T.W., Lee, L.-K., Mak, K.-S., Wong, P.W.H.: Optimizing throughput and energy in online deadline scheduling. *ACM Trans. Algorithms* **6**(1), 10 (2009)
15. Chen, J.-J., Hsu, H.-R., Chuang, K.-H., Yang, C.-L., Pang, A.-C., Kuo, T.-W.: Multiprocessor energy-efficient scheduling with task migration considerations. In: Proc. 16th Euromicro Conference of Real-Time Systems, pp. 101–108 (2004)
16. Chen, J.-J., Kuo, T.-W., Lu, H.-I.: Power-saving scheduling for weakly dynamic voltage scaling devices. In: Proc. 9th International Workshop on Algorithms and Data Structures. LNCS, vol. 3608, pp. 338–349. Springer, Berlin (2005)
17. Demaine, E.D., Ghodsi, M., Hajiaghayi, M.T., Sayedi-Roshkhar, A.S., Zadimoghaddam, M.: Scheduling to minimize gaps and power consumption. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 46–54 (2007)
18. Garay, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
19. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: Proc. 21st Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 11–18 (2009)
20. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., Pruhs, K.: Scheduling heterogeneous processors isn't as easy as you think. In: Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1242–1253 (2012)
21. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM* **34**, 144–162 (1987)
22. Intel pressroom. <http://www.intel.com/pressroom/kits/teraflops/> or [http://download.intel.com/pressroom/kits/Teraflops/Teraflops\\_Research\\_Chip\\_Overview.pdf](http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf)
23. Irani, S., Pruhs, K.: Algorithmic problems in power management. *SIGACT News* **36**(2), 63–76 (2005)
24. Irani, S., Shukla, S., Gupta, R.: Algorithms for power savings. *ACM Trans. Algorithms* **3**(4), 41 (2007)
25. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Energy efficient deadline scheduling in two processor systems. In: Proc. 18th International Symposium on Algorithms and Computation. LNCS, vol. 4835, pp. 476–487. Springer, Berlin (2007)
26. Lam, T.W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Nonmigratory multiprocessor scheduling for response time and energy. *IEEE Trans. Parallel Distrib. Syst.* **19**(11), 1527–1539 (2008)
27. Lam, T.W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Improved multi-processor scheduling for flow time and energy. *J. Sched.* **15**(1), 105–116 (2012)
28. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. *Theory Comput. Syst.* **43**(1), 67–80 (2008)
29. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. *Commun. ACM* **28**, 202–208 (1985)
30. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. 36th Annual Symposium on Foundations of Computer Science, pp. 374–382 (1995)