Speed Scaling on Parallel Processors

Susanne Albers, Fabian Müller and Swen Schmelzer
Department of Computer Science, University of Freiburg
Georges-Köhler-Allee 79
79110 Freiburg, Germany
{salbers, fmueller, sschmelz}@informatik.uni-freiburg.de

ABSTRACT

In this paper we investigate algorithmic instruments leading to low power consumption in computing devices. While previous work on energy-efficient algorithms has mostly focused on single processor environments, in this paper we investigate multi-processor settings. We study the basic problem of scheduling a set of jobs, each specified by a release time, a deadline and a processing volume, on variable speed processors so as to minimize the total energy consumption.

We first settle the complexity of speed scaling with unit size jobs. More specifically, we devise a polynomial time algorithm for agreeable deadlines and prove NP-hardness results for arbitrary release dates and deadlines. For the latter setting we also develop a polynomial time algorithm achieving a constant factor approximation guarantee that is independent of the number of processors. Additionally, we study speed scaling of jobs with arbitrary processing requirements and, again, develop constant factor approximation algorithms. We finally transform our offline algorithms into constant competitive online strategies.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Scheduling*

General Terms

Algorithms, Theory

Keywords

multiprocessor scheduling, energy efficiency, approximation algorithms, online algorithms, NP-completeness

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 servers and, in particular, CPUs account for 50–60% of the energy consumption. In fact, Google engineers, maintaining thousands of servers, recently warned that if power consumption continues to grow, power costs can easily overtake hardware costs by a large margin [5]. In addition to costs, energy dissipation causes a thermal problem. Most of the consumed energy is 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 computing tasks. The goal is to construct energy-efficient schedules, using lower processing speeds, while still guaranteeing a determined service. (2) *Sleep states*: When a system is idle, it can be put into a low-power sleep state. One has to find out when to shut down a 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. [15] there has recently been considerable research interest in the design and analysis of speed scaling strategies, see e.g. [1, 2, 3, 5, 6, 7, 8, 11, 13, 15]. While most of the previous work considers single processor environments, in this paper we study multiprocessor settings. Multi-processor speed scaling is a definite issue in desktop computers and servers, equipped with dual or multiple processors. The topic is interesting in laptops as well, as computer manufacturers have just launched their first dual-processor notebooks. A general trend in hardware design is to develop architectures with multiple CPUs. AMD has introduced a "Quad-Core Design" and architectures with eight CPUs on one die are being developed. Furthermore, Intel has recently done experiments with 80 cores on one chip [10].

We will investigate the following basic speed scaling problem. We are given n jobs that have to be processed on m identical variable speed processors working in parallel. Each job i is specified by a release date $r(i)$, a deadline $d(i)$ and a processing volume $p(i)$. The processing volume represents the amount of work that must be finished to complete the job. Each of the m processors may independently operate at variable speed. The power consumption function depending

SPAA'07, June 9–11, 2007, San Diego, California, USA.

on speed s is given by $P(s) = s^\alpha$, where $\alpha > 1$ is a constant. If a processor runs at speed s for δ time units, then a work of δs is finished and the consumed energy is δs^α. At any time a processor can handle only one job, and each job can reside on only one processor. We allow preemption of jobs but disallow migration of jobs among processors. 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 [14] 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. [15] and has been investigated the most among the proposed energy management settings [3, 4, 6, 7, 11, 15]. Yao et al. 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 released and 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. [15] also proposed two online algorithms, called *Optimal Available* and *Average Rate*, and proved that *Average Rate* achieves a competitiveness of $\alpha^\alpha 2^{\alpha-1}$. Bansal et al. [3] analyzed *Optimal Available* and showed that its competitiveness is exactly α^α. Additionally, Bansal et al. developed a new algorithm with an improved competitive ratio of $2(\alpha/(\alpha-1))^\alpha e^\alpha$. They also gave a lower bound of $\Omega((4/3)^\alpha)$ on the competitive factor of any randomized speed scaling algorithm, demonstrating that the exponential dependence on α is inherent in the performance guarantees.

Irani et al. [11] studied an extended scenario where, in addition to speed scaling, a scheduler may take advantage of a sleep state to save energy. They presented an offline algorithm that achieves a 3-approximation and developed a strategy that transforms an online algorithm for the setting without sleep state into an algorithm for the setting with sleep state. Energy minimization with several sleep states was addressed in [2]. Recently, Baptiste [4] considered the problem of minimizing the number of idle periods when a set of unit size jobs must be scheduled on a processor with one sleep state. He showed that the offline variant is polynomially solvable.

Much less is known for multi-processor speed scaling. 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 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 [9]. A faster 1.13-approximation algorithm was given by Chen et al. [7]. They also showed that if job migration among processors is allowed, an optimal schedule can be computed in polynomial time. Pruhs et al. [12] consider speed scaling on parallel processors if there are precedence constraints among jobs and the goal is to minimize the makespan.

Our contribution: We present the first algorithmic study of multi-processor speed scaling where 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. In practice, instances with agreeable deadlines form a natural input class where, intuitively, jobs arriving at later times may be finished later. Formally, deadlines are agreeable if, for any two jobs i and i', relation $r(i) < r(i')$ implies $d(i) \leq d(i')$. 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 multi-processor 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 of our algorithms are simple and fast, which is an important aspect in energy-efficient computing environments. In a first step the algorithms assign jobs to processors, using classical dispatching rules such as *Round Robin* or *List* scheduling. Once the assignment is done, each processor independently computes its own service schedule. Hence, our algorithms can also be applied in fully distributed systems as well.

2. UNIT SIZE JOBS WITH AGREEABLE DEADLINES

The polynomial time algorithm we develop is essentially a *Round Robin* strategy. We first sort the jobs according to their release dates and then assign them to processors using Round Robin. For each processor, given the assigned jobs, an optimal schedule is computed using the algorithm by Yao et al. [15].

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 service schedule.

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

In the remainder of this section we prove Theorem 1. We first state a simple fact regarding the execution order of jobs on processors.

FACT 1. *Given a feasible schedule for jobs with agreeable deadlines, on any processor the assigned jobs can be reordered such that they are executed in order of increasing job index. This reordering does not cause a higher energy consumption.*

In this proof, we index the processors from 0 to $m-1$. We show that there exists an optimal schedule in which the i-th job of the sorted list computed in step 1 of *RR* is assigned to processor $i \bmod m$. This is exactly the job assignment computed by Round Robin. The theorem then follows since the algorithm by Yao et al. [15] constructs optimal single processor schedules. deadline.

Consider an optimal schedule and let S_{OPT} be the corresponding optimal schedule where jobs are executed in order of increasing job index on any processor as described in Fact 1. Let $a(i)$ be the time when processing of job i starts and let $b(i)$ be the time when processing ends in S_{OPT}. We show inductively for increasing values of i that the following invariant holds.

(I) There exists an optimal schedule in which any job k with $0 \leq k \leq i$ is scheduled on processor $k \bmod m$. Furthermore $b(k-1) \leq b(k)$, for $1 \leq k \leq i$.

For the inductive basis we show that there exists an optimal schedule in which job 0 is scheduled on processor 0. By Fact 1 we know that job 0 must be the first job executed on some processor j. If $j = 0$ there is nothing so show. Otherwise, we swap the complete work assignment of processors 0 and j and we are done again. Thus the invariant holds for $i = 0$.

Suppose the invariant holds for i and let S_{OPT} be the corresponding schedule. We assume that on each processor jobs are executed in order of increasing job index; this property follows from (I) for jobs indexed at most i and from Fact 1 for jobs indexed larger than i. We prove that (I) holds for $i+1$ as well. We first show that there exists an optimal schedule in which job $i+1$ is executed on processor $(i+1) \bmod m$. Assume that $i+1$ is scheduled on processor $j \neq (i+1) \bmod m$. By (I) and Fact 1 we have that the first job executed on processor $(i+1) \bmod m$ with job index greater than i has to be some job $i+k$ for some $k \geq 2$. Note that if no such job $i+k$ exists, job $i+1$ is can obviously be scheduled on processor $(i+1) \bmod m$ without increasing energy consumption since by invariant (I) on all other processors the finishing time of the last job with index at most i must be greater or equal to that on processor $(i+1) \bmod m$. We now distinguish three cases.

Case 1: We first assume $a(i+k) \geq a(i+1)$ (see Figure 1a): In this case suppose first that $i+1 \geq m$. We observe that by (I) job $i+1-m$ is the job executed on processor $(i+1) \bmod m$ immediately before $i+k$. Let l denote the job with highest job index less than or equal to i executed on processor j (where $i+1$ is scheduled in S_{OPT}). Then by (I) we have $a(i+1) \geq b(l) \geq b(i+1-m)$. Therefore, we can just swap the complete work assignments of processors $(i+1) \bmod m$ and j after time $a(i+1)$. If $i+1 < m$, the same argument works because job $i+k$ has no predecessor on processor $(i+1) \bmod m$. In either case the schedule remains feasible and the energy consumption does not increase.

Case 2: In this case we assume $a(i+k) < a(i+1)$ and $b(i+k) \leq b(i+1)$ (see Figure 1b). Our goal is to swap jobs $i+1$ and $i+k$. To this end we exchange start and finishing times of jobs $i+1$ and $i+k$ as follows. Let $a'(i+k) := a(i+1)$, $b'(i+k) := b(i+1)$ and $a'(i+1) := a(i+k)$, $b'(i+1) := b(i+k)$. We can now execute job $i+1$ on processor $(i+1) \bmod m$ (where $i+k$ was scheduled earlier) and job $i+k$ on processor j. The new schedule is feasible since $r(i+1) \leq r(i+k)$ and $d(i+1) \leq d(i+k)$ by the agreeable deadline property. The energy consumption did not change because the total energy consumed by $i+1$ and $i+k$ remains unchanged for they have unit size.

Case 3: For the last case we assume $a(i+k) < a(i+1)$ but $b(i+k) > b(i+1)$ (see Figure 1c). We can now exchange the start times of jobs $i+1$ and $i+k$ by setting $a'(i+1) := a(i+k)$ and $a'(i+k) := a(i+1)$. Since start times are exchanged, we can now swap the complete work assignment on processor $(i+1) \bmod m$ after (and including) job $i+k$ with the work assignment on processor j after (and including) job $i+1$. The schedule is feasible since (by agreeable deadlines) $r(i+1) \leq r(i+k)$. The power consumption of jobs $i+1$ and $i+k$ in the original schedule is $(b(i+1)-a(i+1))^{1-\alpha} + (b(i+k)-a(i+k))^{1-\alpha}$ while we have $(b(i+1)-a(i+k))^{1-\alpha} + (b(i+k)-a(i+1))^{1-\alpha}$ in the modified schedule. By the convexity of the power function the latter expression is smaller because $a(i+k) < a(i+1)$.

Note that the invariant (I) was not violated by any of the above manipulations since we never modified the location or finishing time of any job with index smaller than $i+1$. Moreover in the resulting schedule, as desired, job $i+1$ is scheduled on processor $(i+1) \bmod m$. It remains to show that $b(i+1) \geq b(i)$.

Assume on the contrary that $b(i+1) < b(i)$ in S_{OPT}. Now determine job k with minimum value $b(k)$ such that $b(i+1) < b(k)$. The index k satisfies $1 \leq k \leq i$. Let j be the processor handling job k. We argue that $a(k) \leq a(i+1)$. This obviously holds if job k is the first job processed on processor j because $r(k) \leq r(i+1)$ and job k can be started as early as job $i+1$. If job k has a predecessor on processor j, then by the inductive hypothesis, this is job $k-m$ while the predecessor of job $i+1$ on processor $(i+1) \bmod m$ is indexed $i+1-m$. Again the inductive hypothesis implies $b(k-m) \leq b(i+1-m)$ and job k can be started as early as $a(i+1)$. We now exchange the finishing times of jobs $i+1$ and k, i.e. $b'(i+1) := b(k)$ and $b'(k) := b(i+1)$, and simultaneously swap the complete work assignments of processors $i+1$ and j after time $b(i+1)$ and $b(k)$, respectively (this is depicted as first step in Figure 2). The resulting schedule is feasible because $d(k) \leq d(i+1)$. The power consumption of jobs k and $i+1$ in the original schedule is $(b(k)-a(k))^{1-\alpha} + (b(i+1)-a(i+1))^{1-\alpha}$ while that in the modified schedule is $(b(i+1)-a(k))^{1-\alpha} + (b(k)-a(i+1))^{1-\alpha}$. By the convexity of the power function the latter expression is smaller because $a(k) \leq a(i+1)$ and $b(k) > b(i+1)$. Note, that the invariant (I) is not violated by this manipulation since we chose the smallest job k such that $b(k) > b(i+1)$ for exchanging finishing times. So the ordering of the finishing times of any two jobs excluding job $i+1$ is not affected.

It is still possible that $b(i+1)$ is smaller than $b(i)$, but the value of $b(i+1)$ has increased and is equal to a former finishing time. Hence iterating the above exchanges finitely often (in fact, at most m times), we obtain a schedule satisfying $b(i+1) \geq b(i)$. Two steps of this procedure are shown

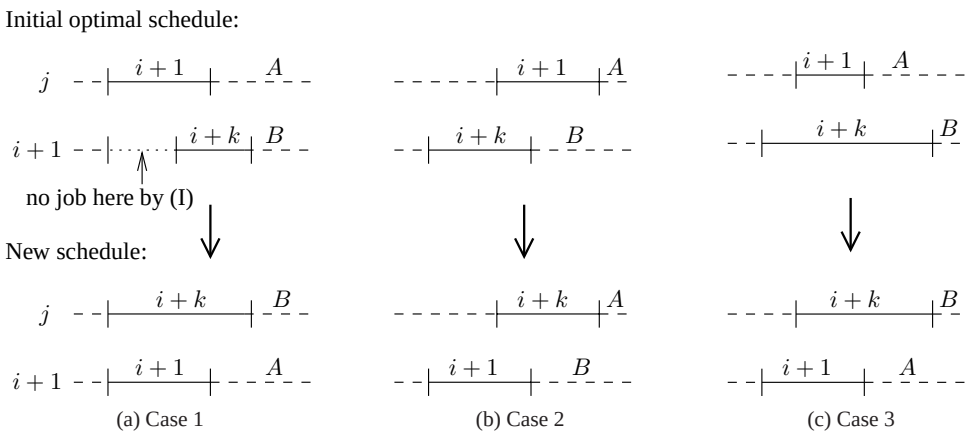

Figure 1: Distinction of cases. Processor numbers are modulo m.

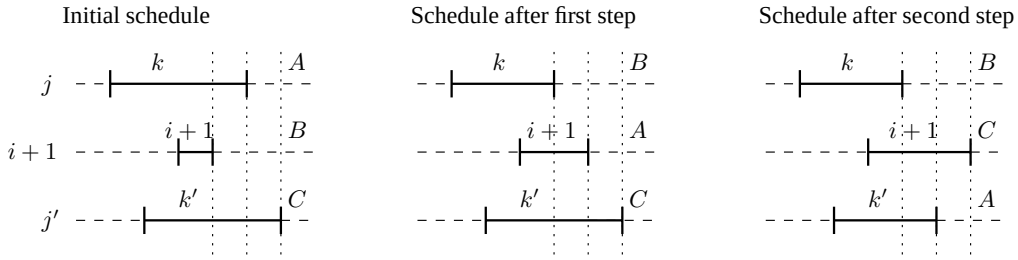

Figure 2: We adjust the finishing time of job $i+1$ step by step.

in Figure 2, where some job $k' \leq i$ on some processor j' has strictly greater finishing time than job $i+1$.

3. UNIT SIZE JOBS WITH ARBITRARY RELEASE DATES AND DEADLINES

We consider unit size jobs with arbitrary release dates and deadlines. We first show that the problem of minimizing the total consumed energy on parallel processors is NP-hard. We then develop a polynomial time algorithm that achieves a constant factor approximation guarantee independent of m. The proofs of the following NP-hardness results are given in the appendix.

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.*

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 Δ 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 service schedule.

We start with a lemma that relates energy consumptions in single-processor and m-processor schedules.

LEMMA 1. *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^m_{OPT} be an optimal schedule on m processors. We partition the time horizon of S^m_{OPT} 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^m_{OPT} is

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

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^m_{OPT}. In S^1 we never run out of available jobs because there are jobs available in S^m_{OPT} at that time. Thus the work completed by S^1 is exactly equal to that of S^m_{OPT}. Always sequencing available jobs according to the *Earliest Deadline* policy yields a feasible schedule. The energy consumption of S^1 is $\sum_{I\in\mathcal{I}} |I|s_I^\alpha \geq E^1_{OPT}$, where E^1_{OPT} denotes the optimum energy of a single processor schedule. Combining the last inequality with (1) we obtain the lemma. □

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 introduce a unit size job $i' \in \mathcal{J}'$ with release date $r(i') = r(i)$ and deadline $d(i') = r(i') + 2^k/\Delta$. 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 larger 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^α 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 $r(i') \leq t \leq d(i')$. 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 2. *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. [15]. 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

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

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 $E'_{AVR,j}$ the energy consumption of $S'_{AVR,j}$. As *CRR* computes an optimal schedule for each processor, its total energy E'_{CRR} is bounded by

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

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

$$E'_{\min} = \sum_{i'\in\mathcal{J}'} (d(i') - r(i'))^{1-\alpha} = \sum_{k\geq 0}\sum_{i'\in C_k} (2^k/\Delta)^{1-\alpha}. \quad (3)$$

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

$$s(t) = \sum_{k\geq 0} c_k(t)\Delta/2^k. \qquad (4)$$

A result by Yao et al. [15] 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 1 we obtain that

$$E'_{AVR} \leq \alpha^\alpha 2^{\alpha-1} m^{\alpha-1} E'_{OPT}. \qquad (5)$$

We will prove

$$\sum_{j=1}^{m} E'_{AVR,j} \leq 2^{2\alpha}(E'_{\min} + m^{1-\alpha}E'_{AVR}). \qquad (6)$$

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 2 we obtain

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

Using (4) we find

$$s_j(t) \leq 4 \max\{\Delta/2^{k_1}, \frac{1}{m} s(t)\}. \quad (7)$$

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 (7) is dominating, and then over all t where the second term of (7) is dominating. Integration of the first term gives an upper bound on the energy consumption that is at most 4^α 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

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

Summing over all j and applying (3) we obtain (6). Combining (5) and (6) and using the fact that $E'_{\min} \leq E'_{OPT}$, we conclude $E'_{CRR} \leq \sum_{j=1}^m E'_{AVR,j} \leq \alpha^\alpha 2^{3\alpha} E'_{OPT}$. 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 $\mathcal{J}$.

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 problem that the jobs' processing requirements $p(i)$ may take arbitrary values. We first assume that all jobs are released at time 0 but 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 study the scenario of arbitrary 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. $d(1) \leq \ldots \leq d(n)$.
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 speed sequence 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 $\lambda_1^j < \lambda_2^j < \ldots < \lambda_{l_j}^j$ 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 $\lambda_0^j < \ldots < \lambda_l^j$ have been defined and that λ_l^j is not equal to the highest job number in S_j. Then λ_{l+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.

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

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 $l = 1, \ldots, l_j$, we define

$$s_l^j = \frac{1}{m} \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)} \quad (8)$$

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

$$s_1^j \geq s_2^j \geq \ldots \geq s_{l_j}^j, \quad (9)$$

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

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

contradicting the choice of λ_l^j. We are now ready to specify the speed function.

Speed function for processor j:

1. Initial setting: For any $l = 1, \ldots, l_j$, 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 3. *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

$$\begin{aligned}
&\sum_{l=1}^{k} \left(2 - \frac{1}{m}\right) s_l^j (d(\lambda_l^j) - d(\lambda_{l-1}^j)) \\
&\quad = \left(2 - \frac{1}{m}\right) \sum_{l=1}^{k} \frac{1}{m} \frac{L(\lambda_l^j) - L(\lambda_{l-1}^j)}{d(\lambda_l^j) - d(\lambda_{l-1}^j)} (d(\lambda_l^j) - d(\lambda_{l-1}^j)) \\
&\quad = \left(2 - \frac{1}{m}\right) L(\lambda_k^j)/m \qquad (10)
\end{aligned}$$

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

$$\begin{aligned}
&\left(2 - \frac{1}{m}\right) s_{k+1}^j (d(\lambda_{k+1}^j) - d(\lambda_k^j)) \\
&\quad = \left(2 - \frac{1}{m}\right) \frac{1}{m} \frac{L(\lambda_{k+1}^j) - L(\lambda_k^j)}{d(\lambda_{k+1}^j) - d(\lambda_k^j)} (d(i) - d(\lambda_k^j)) \\
&\quad \geq \left(2 - \frac{1}{m}\right) \frac{1}{m} \frac{L(i) - L(\lambda_k^j)}{d(i) - d(\lambda_k^j)} (d(i) - d(\lambda_k^j)) \\
&\quad = \left(2 - \frac{1}{m}\right) (L(i) - L(\lambda_k^j))/m \qquad (11)
\end{aligned}$$

is completed. The inequality follows from the definition of λ_{k+1}^j. Combining (10) and (11) we find that a total load of at least $(2 - \frac{1}{m})L(i)/m$ is 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 $(2 - \frac{1}{m})p(i)$ gets 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 by the 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 used by our speed function on processor j after the initial setting when speeds are reduced by a factor of $2 - \frac{1}{m}$.

LEMMA 4. *An optimal solution uses a total energy on the m processors of at least mE_j^1.*

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 $l = 1, \ldots, l_j$. 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 $k = 1, \ldots, l_j$. 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 (8) minimize the function $f(x_1, \ldots, x_{l_j}) = m\sum_{l=1}^{l_j} x_l^\alpha \delta_l^j$ subject to the constraint

$$m \sum_{l=1}^{k} x_l \delta_l^j \geq L(\lambda_k^j), \qquad (12)$$

for $k = 1, \ldots, l_j$. Suppose $(y_1, \ldots, y_{l_j})$ with $(y_1, \ldots, y_{l_j}) \neq (s_1^j, \ldots, s_{l_j}^j)$ is an optimal solution. Note that

$$m \sum_{l=1}^{k} s_l^j \delta_l^j = L(\lambda_k^j), \qquad (13)$$

for $k = 1, \ldots, l_j$. Thus there must exist a k with $y_k > s_k^j$: If $y_l \leq s_l^j$ held for $l = 1, \ldots, l_j$, then there would be a k' with $y_{k'} < s_{k'}^j$ and hence $m\sum_{l=1}^{k'} y_l^\alpha \delta_l^j < L(\lambda_{k'}^j)$, resulting in a violation of constraint (12) 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 $l = 1, \ldots, k_1 - 1$ since otherwise, using the same argument as before, constraint (12) would be violated for $k = k_1 - 1$. Let k_2 with $k_2 > k_1$ be the smallest index such that $y_{k_1} > y_{k_2}$. Such an index exists because otherwise invariant (9) implies $y_l > s_l^j$, for $l = k_1, \ldots, l_j$, 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 ϵ and increase y_{k_2} by $\epsilon\delta_{k_1}/\delta_{k_2}$, where $\epsilon \leq \min\{y_{k_1} - s_{k_1}^j, (y_{k_1} - y_{k_2})/(1 + \delta_{k_1}/\delta_{k_2})\}$. We argue that constraints (12) are still satisfied. There is nothing to show for $k = 1, \ldots, k_1 - 1$. Also for $k = k_2, \ldots, l_j$ 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 ϵ 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 (9) imply $y_l > s_l^j$ for $l = k_1 + 1, \ldots, k_2 - 1$. Using the fact that equations (13) hold we obtain that constraints (12) are satisfied for $l = k_1, \ldots, k_2 - 1$.

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 ϵ 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 $(y_1, \ldots, y_{l_l})$ is not optimal. □

We now combine the speed functions for all processors j.

LEMMA 5. *An optimal solution uses a total energy of at least* $\sum_{j=1}^m E_j^1$.

PROOF. Using Lemma 4 and summing over all j, we find $mE_{\text{OPT}} \geq m\sum_{j=1}^m E_j^1$, where E_{OPT} is the energy of an optimal solution. Dividing by m we obtain the desired statement. □

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 6. *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. □

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 $E_{SF} \leq (2-\frac{1}{m})^\alpha \sum_{j=1}^m (E_1^1 + E_2^j)$ and, using Lemmas 4 and 5, we find $E_{SF} \leq \left(2-\frac{1}{m}\right)^\alpha 2E_{\text{OPT}}$, where E_{OPT} is the total energy of an optimal solution. In Lemma 3 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}. □

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 $\delta_{i'} = \Delta/2^k$. Lemma 2 carries over. The proof of Lemma 2 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

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

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 δ_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. □

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 of 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 generate jobs classes dynamically as jobs arrive. The classes are centered around δ_1, the density of the first incoming job. As usual, job densities within each class differ by a factor less than 2. More precisely, we open a new job class of smaller densities when,

for the first time, a job of density at most $2^{-k}3\delta_1/2$ arrives, where $k \geq 1$. We open a job class of higher densities when, for the first time, a job of density greater than $2^k 3\delta_1/2$ arrives, where $k \geq 0$. After jobs have been assigned to processors, instead of computing optimal schedules, we apply the *Average Rate* algorithm [15]. 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. REFERENCES

[1] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *Proc. 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Springer LNCS 3884, 621–633, 2006.

[2] J. Augustine, S. Irani and C. Swamy. Optimal power-down strategies. *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, 530-539, 2004.

[3] N. Bansal, T. Kimbrel and K. Pruhs. Dynamic speed scaling to manage energy and temperature. *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, 520–529, 2004.

[4] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 364–367, 2006.

[5] L.A. Barroso. The price of performance. *ACM Queue*, 3(7), September 2005.

[6] N. Bansal and K. Pruhs. Speed scaling to manage temperature. *Proc. 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, Springer LNCS 3404, 460–471, 2005.

[7] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. *Proc. 16th Euromicro Conference of Real-Time Systems*, 101–108, 2004.

[8] J.-J. Chen, T.-W. Kuo, H.-I. Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. *Proc. 9th International Workshop on Algorithms and Data Structures*, Springer LNCS 3608, 338–349, 2005.

[9] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[10] Intel pressroom. `http://www.intel.com/pressroom/kits/teraflops/` or `http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf`

[11] S. Irani, S. Shukla and R. Gupta. Algorithms for power savings. *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 37–46, 2003.

[12] K. Pruhs, R. van Stee, P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Proc. 3rd International Workshop on Approximation and Online Algorithms (WAOA)*, Springer LNCS 3879, 307–319, 2005.

[13] K. Pruhs, P. Uthaisombut and G. Woeginger. Getting the best response for your erg. *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, Springer LNCS 3111, 15–25, 2004.

[14] D.D. Sleator und R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202-208, 1985.

[15] F. Yao, A. Demers and S. Shenker. A scheduling model for reduced CPU energy. *Proc. 36th Annual Symposium on Foundations of Computer Science*, 374–382, 1995.

APPENDIX

In this appendix we establish the NP-hardness results stated in Theorems 2 and 3. We will present 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* $m \in \mathbb{N}$ *we say* $(A, m) \in$ MULTI-PARTITION *if and only if there are subsets* $A_1, A_2, \ldots, A_m$ *such that* $\bigcup_{i=1}^m A_i = A$ *and, for all* $i \neq j$,

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

We describe the reduction from an instance of the MULTI-PARTITION problem to an instance of the scheduling problem under consideration. To this end we consider an instance $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{Z}^+$ and (A, m) 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 $n+1, n+2, \ldots, n+m$, where $r(n+1) = r(n+2) = \ldots = r(n+m) = 0$ and $d(n+1) = d(n+2) = \ldots = d(n+m) = 3d(n)$.

In the following, for any job i with $1 \leq i \leq n+m$, let $l(i) = d(i) - r(i)$ be the length of i. 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 7. *In an optimal schedule* S *for* $\mathcal{J}$ *no two jobs with index 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 at least $(2/3d(n))^{\alpha-1}$. Furthermore, there must exist one processor where no job is executed in the interval $[d(n), 3d(n)]$ as we have only m jobs with deadline greater than $d(n)$ and two jobs 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. □

In order to establish the second lemma we need a technical property.

LEMMA 8. *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 derivate 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. □

LEMMA 9. *In an optimal schedule S for $\mathcal{J}$ the energy consumption of any jobs i with $1 \le i \le n$ is $(1/l(i))^{\alpha-1}$.*

PROOF. Assume that in S some job i is not executed over its full possible interval $I = [r(i), d(i)]$, i.e. $l(i) = l_S(i) + \varepsilon$ for some $0 < \varepsilon < l(i)$. Since the execution intervals of all jobs i' with $1 \le i' \le n$ are pairwise disjoint, no such job i' can be scheduled in I. We now investigate two cases.

Case 1: No job besides i is executed in interval I on the processor where i is processed. Then the schedule is not optimal since it would obviously be cheaper to execute i over its full interval I.

Case 2: Job $n+k$, with $1 \le k \le m$, is partially executed in I on the processor where i is processed. As shown in Lemma 7 only this one job can overlap I. Furthermore, $l_S(n+k) > 2d(n)$. Now we can construct a better schedule by executing job i over its full possible length $l(i)$, reducing the execution interval of job $n+k$ by ϵ since

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

This follows from Lemma 8 setting $c = l_S(n+k) + l_S(j)$ and $x = l_S(i)$. We note that $l_S(i) < l_S(i) + \varepsilon = l(i) < c/2$ as $l_S(i) < d(n)$ and $l_S(n+k) > 2d(n)$. This contradicts the assumption that schedule S was optimal and we conclude that $(1/l(i))^{\alpha-1}$ is the energy consumption of job i in an optimal schedule. □

Let E_{OPT} denote the energy of an optimal schedule for $\mathcal{J}$.

THEOREM 9. *Let $A \subset \mathbb{Z}^+$, $m \in \mathbb{N}$ and $B = \left(\sum_{a\in A} a\right)/m$. $(A, m) \in$ MULTI-PARTITION if and only if*

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

PROOF. By Lemma 9 in an optimal schedule the execution of all jobs $i \in \{1, 2, \ldots, n\}$ requires an energy volume of exactly $\sum_{i=1}^{n} (1/l(i))^{\alpha-1}$. Thus the energy consumption of an optimal solution depends only on the execution energy consumed by jobs $n+1, n+2, \ldots, n+m$. Let $k \in \{1, 2, \ldots, m\}$. By Lemma 7, all these jobs are executed on separate machines. We assume w.l.o.g. that job $n+k$ is executed on processor k. Let $\mathcal{J}_k \subseteq \{1, 2, \ldots, n\}$ be the set of jobs scheduled on processor k. We can now easily compute the energy used by $n+k$ as

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

By Lemma 8 we find that the sum of the energy consumptions of $n+1, n+2, \ldots, n+m$ is minimal if and only if $l_S(n+1) = l_S(n+2) = \ldots = l_S(n+m)$. This is the case if and only if $\sum_{i\in\mathcal{J}_1} l(i) = \sum_{i\in\mathcal{J}_2} l(i) = \ldots = \sum_{i\in\mathcal{J}_m} l(i) = d(n)/m$. Note that the value asserted in the theorem is taken at this point. By our construction this is possible if and only if there exist sets $A_1, A_2, \ldots, A_m \subseteq A$ with $\bigcup_{i=1}^{m} A_i = A$ and for all $i \neq j$ it holds $A_i \cap A_j = \emptyset$ and $\sum_{a\in A_i} a = \sum_{a\in A_j} a$. Finally, this is the case if and only if $(A, m) \in$ MULTI-PARTITION. □

We are now ready to derive the first desired NP-hardness result. Setting $m = 2$, the NP-hard PARTITION problem is a special case of our MULTI-PARTITION problem. Theorem 9 gives the following result.

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.*

Another special case of MULTI-PARTITION is 3-PARTITION. In this case A underlies the restriction that there exists a $B \in \mathbb{Z}^+$ such that for all $a \in A$ we have $B/4 < a < B/2$ and $\sum_{a\in A} a = 3B$. If any of these conditions is not satisfied, the input can be rejected in polynomial time. Otherwise, we can use MULTI-PARTITION to solve the problem on input (A, B). Since 3-PARTITION is strongly NP-hard, we obtain the second desired NP-hardness result as a consequence of Theorem 9.

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.*