

# The Bell is Ringing in Speed-Scaled Multiprocessor Scheduling

Gero Greiner and Tim Nonner\* and Alexander Souza  
Albert Ludwigs University of Freiburg, Germany

January 6, 2010

## Abstract

This paper investigates the problem of scheduling jobs on multiple speed-scaled processors without migration, i.e., we have constant  $\alpha > 1$  such that running a processor at speed  $s$  results in energy consumption  $s^\alpha$  per time unit. We consider the general case where each job has a monotonously increasing cost function that penalizes delay. This includes the so far considered cases of deadlines and flow time. For any type of delay cost functions, we obtain the following results: Any  $\beta$ -approximation algorithm for a single processor yields a randomized  $\beta B_\alpha$ -approximation algorithm for multiple processors without migration, where  $B_\alpha$  is the  $\alpha$ th Bell number, that is, the number of partitions of a set of size  $\alpha$ . Analogously, we show that any  $\beta$ -competitive online algorithm for a single processor yields a  $\beta B_\alpha$ -competitive online algorithm for multiple processors without migration. Finally, we show that any  $\beta$ -approximation algorithm for multiple processors with migration yields a deterministic  $\beta B_\alpha$ -approximation algorithm for multiple processors without migration. These facts improve several approximation ratios and lead to new results. For instance, we obtain the first constant factor online and offline approximation algorithm for multiple processors without migration for arbitrary release times, deadlines, and job sizes.

---

\*Corresponding author: nonner@informatik.uni-freiburg.de, supported by DFG research program No 1103 *Embedded Microsystems*

# 1 Introduction

Saving energy is a major concern in today's information technology, which more and more consists of mobile battery-powered devices. One way to deal with this problem is to embed speed-scaled processors that adjust their speed dynamically according to the current need. This is of great advantage, since for most CMOS based systems it has been observed that the energy consumption grows proportional to the cube of the processor speed  $s$  [10]. We consider the more general case that there is a constant  $\alpha > 1$  such that running a processor at speed  $s$  results in energy consumption  $s^\alpha$  (typically,  $\alpha = 2$  or  $\alpha = 3$  [10]). More specifically, this paper investigates energy minimization for a set  $J := \{1, \dots, n\}$  of jobs, whereas each job  $j \in J$  has a *release time*  $r_j \in \mathbb{R}^+$  and a *size*  $p_j \in \mathbb{R}^+$ . A job volume of  $\delta s$  is processed by a processor running at speed  $s$  for  $\delta$  time units, resulting in energy consumption  $\delta s^\alpha$ . Hence, the slower a processor runs, the fewer energy is consumed. Preemption is allowed, i.e., a job may be interrupted at any time, and then resumed at the point of preemption with no penalty. Let  $s : \mathbb{R}^+ \mapsto \mathbb{R}^+$  be the function that describes the speed of the processor over time with respect to some schedule  $\sigma$ . We conclude that the total *energy/power consumption/cost* of  $\sigma$  is then

$$\text{energy}(\sigma) := \int_0^\infty s(t)^\alpha dt.$$

To penalize delay, each job  $j$  has a monotonously increasing *delay cost function*  $h_j : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ . Specifically, if  $j$  is completed at time  $c_j > r_j$  in schedule  $\sigma$ , then its delay is  $c_j - r_j$ , and hence, the *delay cost* of  $\sigma$  is

$$\text{delay}(\sigma) := \sum_{j=1}^n h_j(c_j - r_j).$$

The objective is then to minimize the combined *cost* of  $\sigma$ , that is,

$$\text{cost}(\sigma) := \text{energy}(\sigma) + \text{delay}(\sigma).$$

We denote this problem with SS. The cost structure described above includes the two so far considered cases of deadlines and flow time. More specifically, we can model deadlines by setting the delay cost to infinity after some time, and we can model flow time with linear delay cost functions, that is,  $h_j(x) = x$ , for each job  $j$ . Speed-scaled scheduling was originally introduced by Yao, Demers, and Shenker [23] with deadlines. On the other hand, flow time was introduced to speed-scaled scheduling by Albers and Fujiwara [2]. For a survey on algorithmic problems in power management, we refer to [17].

Current trends lead towards multiprocessor architectures on a single chip, giving rise to the question of balancing work among processors. While the single processor case has received a considerable amount of attention [23, 2, 7, 9, 5, 12, 8, 20, 22, 13, 6, 18], much less is known about the multiprocessor case. Therefore, this paper investigates the problem of scheduling jobs on  $m$  identical speed-scaled processors without migration (SSM), i.e., each job needs to be assigned to a single processor. This contrasts to the case with migration which allows that a job is processed by different processors, whereas a job must never be processed by two processors in parallel. As mentioned in [19], finding algorithms with constant approximation guarantees is easy for a constant number of processors, since we can then simply schedule all jobs on a single processor with known single processor algorithms. Hence, the main difficulty stems from the fact that the number of processors  $m$  is unbounded.

## 1.1 Previous work for a single processor (SS)

In this subsection, we describe the best known algorithms for SS in detail, since we use them as a building block to construct multiprocessor algorithms.

In the deadline case, each job  $j$  additionally has a deadline  $d_j > r_j$  until when it needs to be completed. Let then  $J[t, t'] := \{j \mid t \leq r_j < d_j \leq t'\}$  be the jobs which need to be processed in some time interval  $[t, t']$ . Yao, Demers, and Shenker [23] presented the following surprisingly simple polynomial time algorithm for SS with deadlines:

---

### YDS( $J$ )

---

1. While  $J \neq \emptyset$ :

(a) Let  $[t, t']$  be the interval with maximum density, i.e., that maximizes

$$\frac{\sum_{j \in J[t, t']} p_j}{t' - t}.$$

(b) Process the jobs  $J[t, t']$  with speed equal to this maximum density.

(c) Remove the jobs  $J[t, t']$  from  $J$ , and then adjust the remaining jobs  $J$  as if the time interval  $[t, t']$  does not exist, i.e., for each job  $j \in J$ , if  $r_j \geq t$ , then set  $r_j \leftarrow \max\{t, r_j - (t' - t)\}$ , and if  $d_j \geq t$ , then set  $d_j \leftarrow \max\{t, d_j - (t' - t)\}$ .

2. Return the resulting schedule.

---

The optimality of **YDS** relies on the convexity of the objective function. On the other hand, every online algorithm must satisfy the following scheme: At each time  $t$ , select two parameters with respect to the currently known jobs, (1) the current speed of the processor and (2) the currently processed job. Yao et al. [23] showed that the strategy of scheduling each uncompleted job  $j$  with speed equal to its *density*  $p_j/(d_j - r_j)$  is  $2^{\alpha-1}\alpha^\alpha$ -competitive. This algorithm is denoted **AVR**, short for average rate. Since we can round-robin process all uncompleted jobs, **AVR** satisfies the scheme for online algorithms given above. Moreover, they proposed to schedule all uncompleted jobs optimally at each time in order to select the speed and currently processed job according to this schedule. This algorithm is denoted **OA**, short for optimal available. Bansal, Kimbrel, and Pruhs [7] showed that **OA** is  $\alpha^\alpha$ -competitive. They also presented the following algorithm with improved competitive ratio  $2(\frac{\alpha}{\alpha-1})e^\alpha$ , which is asymptotically optimal for large  $\alpha$  [7]:

---

**BKP**( $J$ )

---

At each time  $t$ , schedule the uncompleted job  $j$  with earliest deadline  $d_j$  at speed

$$\max_{t' > t} \frac{\sum_{j \in J[et - (e-1)t', t']; r_j \leq t} p_j}{t' - t}.$$


---

On the other hand, it is often more reasonable to penalize delay instead of having fixed deadlines, which can be done by adding the flow time  $\sum_j (c_j - r_j)$  to the objective function, where  $c_j$  is the completion time of job  $j$  in some schedule  $\sigma$ . Note that this model can be modified to fractional flow time [6], or extended to weighted flow time [9], where we have a weight  $w_j$  for each job  $j$  and add  $\sum_j w_j (c_j - r_j)$  to the objective function. However, we only review results for the classical flow time case in what follows.

For the special case of unit size jobs ( $p_j = 1$ , for each job  $j$ ), Albers and Fujiwara [2] presented a polynomial time algorithm for SS with flow time. They also proposed the natural algorithm to set the speed such that the power consumption is proportional to the number of uncompleted jobs. This matches the observation of Pruhs, Uthaisombut, and Woeginger [22] that in any locally-optimal schedule for flow time, each job  $j$  is run at a power proportional to the number of jobs that would be delayed if  $j$  would be delayed. However, Albers and Fujiwara only showed that a batched variant of this natural algorithm has competitive ratio  $8.3e^{(\frac{3+\sqrt{5}}{2})^\alpha}$ . On the other hand, Bansal, Pruhs, and Stein [9] showed that this

algorithm is indeed 4-competitive. All these results are for unit size jobs. The first constant competitive algorithm for arbitrary size jobs was also presented in [9]. The competitive ratio of this algorithm is  $O(\frac{\alpha^2}{\log^2 \alpha})$ . This was improved to  $O(\frac{\alpha}{\log \alpha})$  by Lam et al.[18], and recently to  $O(1)$  by Bansal, Chan, and Pruhs [6] with the following algorithm, whereas they considered a more general model with arbitrary power functions:

---

**BCP( $J$ )**

---

At each time  $t$ , schedule the uncompleted job with minimum remaining processing volume at speed  $s$  such that the power consumption  $s^\alpha$  equals the number of currently uncompleted jobs plus one.

---

Bansal, Chan, and Pruhs [6] showed that this algorithm is  $(3 + \epsilon)$ -competitive. However, very recently, Andrew, Wierman, and Tang [4] improved the competitive analysis of this algorithm by showing that it is even  $(2 + \epsilon)$ -competitive.

Moreover, Chan et al. [13] gave a  $O(\alpha^3)$ -competitive algorithm for SS with flow time and nonclairvoyant jobs, that is, the sizes of the jobs are not known when they arrive. Finally, it is worth mentioning that there are also many interesting results for SS with bounded speed processors [12, 5, 19, 20], that is, we have some constant  $T$  that bounds the speed of each processor.

## 1.2 Previous work for multiple processors (SSM)

In contrast to SS, the problem SSM with deadlines is clearly strongly NP-hard, since it contains 3-Partition as a special case. Specifically, given a 3-Partition instance with weights  $w_1, w_2, \dots, w_n$ , we simply represent each weight by a job  $j$  with  $p_j = w_j$ ,  $r_j = 0$ , and  $d_j = 1$ . Consequently, if the weights  $w_1, w_2, \dots, w_n$  can be partitioned into three sets of equal weight, then an optimal multiprocessor schedule for  $m = 3$  processors will assign the jobs to the processors according to these sets. Albers, Müller, and Schmelzer [3] showed that SSM with deadlines is strongly NP-hard, even for unit size jobs. A common scheme to design multiprocessor algorithms is to first define some dispatching strategy that assigns jobs to processors, and then schedule the assigned jobs separately on each processor with some single processor algorithm. Albers et al. [3] presented an  $\alpha^\alpha 2^{4\alpha}$ -approximation algorithm using this scheme. Specifically, this algorithm clusters the jobs according to their densities, and then dispatches the jobs round-robin to the processors independently for each cluster. The optimal algorithm **YDS** is

then applied separately to each processor.

The problem SSM was first discussed by Bunde [11] for unit size jobs. However, Lam et al. [20] presented the first constant competitive online algorithm for arbitrary job sizes. Specifically, the competitive ratio is  $2\eta_\epsilon$  times the competitive ratio of the single processor algorithm in [9], where  $\eta_\epsilon := (1 + \epsilon)^\alpha((1 + \epsilon)^{\alpha-1} + (1 - 1/\alpha)(2 + \epsilon)/\epsilon^2)$  for an arbitrary  $\epsilon > 0$ . As in [3], the jobs are first clustered, and then round-robin dispatched to the processors independently for each cluster in order to apply the single processor algorithm in [9]. However, the jobs are clustered according to their sizes instead of their densities.

### 1.3 The Bell and Stirling numbers

Let  $B_\alpha$  denote the  $\alpha$ th Bell number [16], that is, the number of partitions of a set of size  $\alpha$ . The Bell numbers have the well-known asymptotic behavior

$$B_\alpha \sim \frac{1}{\sqrt{\alpha}} [\lambda(\alpha)]^{\alpha+1/2} e^{\lambda(\alpha)-\alpha-1},$$

where  $\lambda(\alpha) = \alpha/W(\alpha)$  and  $W$  is the Lambert  $W$  function [21]. Hence,  $B_\alpha < \alpha^\alpha$ . For instance, the first seven Bell numbers starting with  $B_1$  are 1,2,5,15,52,203,877. Moreover, Dobinski's formula [15] gives the surprising closed form

$$B_\alpha = \frac{1}{e} \sum_{i=0}^{\infty} \frac{i^\alpha}{i!}. \quad (1)$$

Analogously, the  $j$ th Stirling number for  $\alpha$  elements,  $S(\alpha, j)$ , is the number of partitions of a set of size  $\alpha$  into  $j$  subsets. Therefore, we obtain the equation

$$\sum_{j=1}^{\alpha} S(\alpha, j) = B_\alpha. \quad (2)$$

It is a well-known recurrence relation of the Stirling numbers of the second kind that, for each  $1 < j < \alpha$ ,

$$S(\alpha, j) = jS(\alpha - 1, j) + S(\alpha - 1, j - 1). \quad (3)$$

This recurrence relation can be easily verified. For non-integer-valued  $\alpha > 1$ , we define  $B_\alpha := B_{\lceil \alpha \rceil}$ .

## 1.4 Contributions

For technical reasons, we assume in the following that  $m \geq \alpha$ . However, since  $\alpha$  is a small constant and the number of processors  $m$  part of the input, this covers all relevant cases. Our main result is about turning single processor algorithms into multiprocessor algorithms. Specifically, we show in Section 2 that any  $\beta$ -approximation algorithm for SS yields a randomized  $\beta B_\alpha$ -approximation algorithm for SSM, whereas the running time stays the same. Analogously, any  $\beta$ -competitive online algorithm for SS yields a randomized  $\beta B_\alpha$ -competitive online algorithm for SSM. To obtain this result, we simply assign the jobs uniformly at random to the processors, apply the single processor algorithm for SS separately to each processor, and then use a ball-into-bins interpretation of this process for the analysis. A similar strategy was used in the previous multiprocessor algorithms [20, 3], but they both used a deterministic round-robin strategy to dispatch the jobs. Assigning jobs uniformly at random to processors is a natural job dispatching strategy, and has for instance been used by Chekuri et al. [14] to obtain a constant competitive online algorithm for non-speed-scaled multiprocessor scheduling with weighted flow time and speed augmentation. Using the method of conditional expectations, we even obtain a derandomized version in Section 3. Specifically, we show that any  $\beta$ -approximation algorithm for SSM with migration yields a (deterministic)  $\beta B_\alpha$ -approximation algorithm for SSM, whereas we need to add  $O(n^{\alpha+4})$  to the running time. We obtain this result by deterministically removing migration in a multiprocessor schedule.

An interesting fact is that these results hold for any type of delay cost functions such as deadlines and flow time, but also weighted flow time [9] and many other reasonable delay cost functions not yet considered in literature. Moreover, they hold analogously for unit size jobs such that an improved algorithm for SS with unit size jobs also yields an improved algorithm for SSM with unit size jobs. Finally, it is worth mentioning that these results also hold for nonclairvoyant jobs, but not for the bounded speed model. The following list only summarizes the most important implications, but it can be straightforward extended to special cases and generalizations:

- Algorithm **YDS** [23] yields a randomized  $B_\alpha$ -approximation algorithm for SSM with deadlines, which gives the first constant factor approximation algorithm for arbitrary release times, deadlines, and job sizes. Moreover, this significantly improves the approximation guarantee for unit size jobs from [3] (e.g. for  $\alpha = 2$  and  $\alpha = 3$ , we improve 1024 and 110592 to 2 and 5, respectively). Finally, since there is a nontrivial extension of **YDS** that solves SSM with deadlines and migration in polynomial time [1], the

derandomization result given above implies that there is even a deterministic  $B_\alpha$ -approximation algorithm for SSM with deadlines.

- Algorithm **BKP** [7] yields a randomized  $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha B_\alpha$ -competitive online algorithm for SSM with deadlines, which also gives the first constant competitive online algorithm for arbitrary release times, deadlines, and job sizes.
- Algorithm **BCP** [6] yields a randomized online algorithm for SSM with flow time with competitive ratio  $(2 + \epsilon)B_\alpha$ . Note that even if we combine the round-robin approach of Lam et al. [20] with the improved single processor algorithm in [6], our randomized approach is still significantly better for small  $\alpha$  (e.g. we improve the competitive ratio by the factor  $2\eta_\epsilon/B_\alpha$ , which is  $\approx 7.63$  and  $\approx 6.63$  for  $\alpha = 2$  and  $\alpha = 3$ , respectively). However, their approach is deterministic.

**Example.** Before starting with the formal analysis, we consider a simple example that shows why assigning the jobs uniformly at random works. Assume that we are in the deadline case and that we have  $m$  unit size jobs with equal release times 0 and deadlines 1. Clearly, an optimal schedule  $\sigma^*$  assigns each of the  $m$  jobs to one of the  $m$  processors. In this case, since each processor needs to run at speed 1 during time interval  $[0, 1]$  to process one unit size job, the total energy consumption of all processors is  $m$  as well. On the other hand, if we assign the jobs uniformly at random to the processors, then it might happen that multiple jobs are assigned to a single processor. However, in this case, we can increase the speed of this processor in order to complete all these jobs during time interval  $[0, 1]$ , yielding a random schedule  $\sigma'$ . Specifically, the speed  $s$  is then exactly the number of assigned jobs, which results in energy consumption  $s^\alpha$ . We illustrate this in Figure 1. The question is how the resulting expected total power consumption relates to the optimal energy consumption  $m$ ? We show in this paper that it is at most  $B_\alpha m$ , and we extend this to the general case of arbitrary job sizes, release times, and delay cost functions.

## 2 Turning single processor algorithms into multiprocessor algorithms

In this section, we prove the following theorem:

**Theorem 1.** *Any  $\beta$ -approximation algorithm for SS yields a randomized approx-*

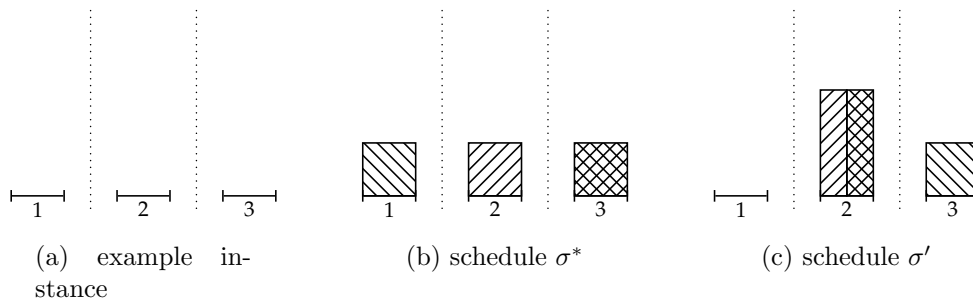


Figure 1: Subfigure (a) visualized an instance with three processors, which are horizontally ordered, and three unit size jobs with release times 0 and deadlines 1. The interval at each processor is  $[0, 1]$ . No jobs are scheduled in this subfigure yet, but observe that Subfigure (b) depicts an optimal schedule  $\sigma^*$  for this instance. Finally, Subfigure (c) depicts an example schedule  $\sigma'$  for the case that we assign the jobs randomly to the processors. In this schedule, two jobs unfortunately have been assigned to processor 2, and hence we need to double the speed in order to sequentially process these jobs.

*imation algorithm for SSM with approximation ratio*

$$\beta \sum_{j=1}^{\alpha} S(\alpha, j) \frac{m!}{m^j (m-j)!},$$

*whereas the asymptotic running time stays the same, and this analogously holds for online algorithms.*

Combining Theorem 1 with Equation (2) shows that the approximation ratio of the claimed algorithm for SSM converges to  $B_\alpha$  from below as  $m \rightarrow \infty$ . But if  $m$  is small, then the approximation ratio is even much better than the  $\alpha$ th Bell number. For instance, the approximation ratio is  $\beta 3/2$  for  $\alpha = m = 2$ .

To prove Theorem 1, we will show in this section how to turn an algorithm for SS, say  $\mathcal{A}$ , into an algorithm for SSM, denoted  $\mathcal{A}'$ . Formally, algorithm  $\mathcal{A}'$  is defined as follows:

---

 $\mathcal{A}'(J, m, \mathcal{A})$ 

---

1. Generate a random job assignment  $A : J \rightarrow \{1, \dots, m\}$  by assigning the jobs  $J$  uniformly at random to the processors, and let  $J_r^A := \{j \mid A(j) = r\}$  be the jobs assigned by  $A$  to processor  $r$ .
  2. For each processor  $r = 1, \dots, m$ , use the single processor algorithm  $\mathcal{A}$  to schedule the jobs  $J_r^A$  on processor  $r$ .
  3. Return the resulting multiprocessor schedule  $\sigma'$ .
- 

Note that this conversion works for online algorithms as well. Hence, the following lemma, which immediately implies Theorem 1, applies to the online case as well:

**Lemma 2.** *If  $\mathcal{A}$  is a  $\beta$ -approximation algorithm for SS, then  $\mathcal{A}'$  is a randomized approximation algorithm for SSM with approximation ratio*

$$\beta \sum_{j=1}^{\alpha} S(\alpha, j) \frac{m!}{m^j (m-j)!}.$$

The goal of the remainder of this section is to prove Lemma 2. To this end, we will first show in Subsection 2.1 how to transform schedules using a given job assignment. We apply this in Subsection 2.2 to finally analyse algorithm  $\mathcal{A}'$ .

## 2.1 Transforming schedules using job assignments

Consider a multiprocessor schedule  $\sigma$  and a job assignment  $A : J \rightarrow \{1, \dots, m\}$ . We can combine  $\sigma$  and  $A$  to a multiprocessor schedule  $\sigma_A$  as follows: Let  $t_0 = 0 < t_1 < \dots < t_q$  be a sequence of times such that  $J[0, t_q] = J$  and, for each  $1 \leq i \leq q$ , every processor processes at most one job during time interval  $[t_{i-1}, t_i]$  in  $\sigma$ , and if a job is processed, then this job is not completed before  $t_i$ . Recall here that  $J[0, t_q] = \{j \mid 0 \leq r_j < d_j \leq t_q\}$ . Therefore, by the convexity of the objective function, we may assume that each processor uses a single speed in each of these time intervals. For each  $1 \leq i \leq q$  and each job  $j \in J$ , let  $s_{ij}$  be the processing speed of job  $j$  in time interval  $[t_{i-1}, t_i]$ . If a job  $j$  is not processed during this time interval by some processor, then define  $s_{ij} := 0$ . Hence, there are at most  $m$  jobs  $j$  with  $s_{ij} \neq 0$ . We conclude that

$$\text{energy}(\sigma) = \sum_{i=1}^q (t_i - t_{i-1}) \sum_{j \in J} s_{ij}^\alpha.$$

Note that we do not require that  $q$  is polynomially bounded. Now, for each  $1 \leq r \leq m$  and  $1 \leq i \leq q$ , we set the speed of processor  $r$  during time interval  $[t_{i-1}, t_i)$  in  $\sigma_A$  to

$$\sum_{j \in J_r^A} s_{ij},$$

where  $J_r^A := \{j \mid A(j) = r\}$  are the jobs assigned by  $A$  to processor  $r$ . This speed ensures that processor  $r$  can sequentially process the jobs  $J_r^A$  during time interval  $[t_{i-1}, t_i)$  such that, for each such job  $j \in J_r^A$ , the volume  $(t_i - t_{i-1})s_{ij}$  of its total size  $p_j$  is processed. This already yields the final schedule  $\sigma_A$  with

$$\text{energy}(\sigma_A) = \sum_{i=1}^q (t_i - t_{i-1}) \sum_{r=1}^m \left( \sum_{j \in J_r^A} s_{ij} \right)^\alpha.$$

We are especially interested in  $\sigma_A$  for the random job assignment  $A$  from algorithm  $\mathcal{A}'$ . Before analyzing this schedule, we need some additional definitions.

Let  $\mathbb{R}_+^n := \{x \in \mathbb{R}^n \mid x \geq 0\}$ . We say that a vector  $x \in \mathbb{R}_+^n$  is *constant* if  $x_i = x_j$ , for each pair  $1 \leq i \leq j \leq n$ . Let  $S_m$  be the  $m$ -dimensional Euclidean sphere with unit radius, and let  $S_m^+ := \{x \in S_m \mid x \geq 0\}$  be the positive part of this sphere, which is compact in the topological sense, i.e., since we consider  $\mathbb{R}^m$  for a fixed  $m$ , it is bounded and closed. Let  $\mathcal{P}_n := \mathcal{P}(J)$  be the set of all subsets of jobs, and abbreviate  $\mathcal{P} = \mathcal{P}_n$  for the special case  $n = m$ . For a subset  $\Gamma \in \mathcal{P}_n$ , let

$$\phi_n(\Gamma) := \Pr [J_r^A = \Gamma]$$

be the probability that exactly the jobs  $\Gamma$  are assigned to some processor  $r$  (note that all processors are identical), and abbreviate  $\phi(\Gamma) = \phi_n(\Gamma)$  for the special case  $n = m$ . It clearly holds that

$$\phi_n(\Gamma) = \left(\frac{1}{m}\right)^{|\Gamma|} \left(1 - \frac{1}{m}\right)^{n-|\Gamma|} = \frac{\binom{m-1}{m}^n}{(m-1)^{|\Gamma|}}.$$

Using this, define the function  $\Phi_n : \mathbb{R}_+^n \setminus \{0\} \rightarrow \mathbb{R}$  as

$$\Phi_n(x) := \frac{m \sum_{\Gamma \in \mathcal{P}_n} \phi_n(\Gamma) (\sum_{j \in \Gamma} x_j)^\alpha}{\sum_{j=1}^n x_j^\alpha}.$$

We abbreviate here the constant vector  $(0, \dots, 0)$  with  $0$ , and we remove this vector from the domain of  $\Phi$  to avoid division by zero. Analogously, abbreviate  $\Phi = \Phi_n$  for the special case  $n = m$ . Observe that  $\Phi$  is *scaling-invariant*, i.e.,  $\Phi(\lambda x) = \lambda \Phi(x)$ , for any  $\lambda > 0$  and  $x \in \mathbb{R}_+^n \setminus \{0\}$ . We illustrate this in Figure 2. The function  $\Phi$  is motivated by the following lemma:

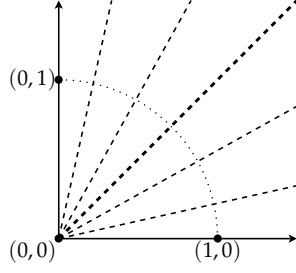


Figure 2: By the scaling-invariance of the function  $\Phi$ , the dashed lines represent some contour lines of  $\Phi$  for  $m = 2$ , where the thickened dashed line moreover represents the constant vectors, and the curved line represents the positive part of the Euclidean sphere in two dimension  $S_2^+$ .

**Lemma 3.** *Let  $A$  the job assignment from algorithm  $\mathcal{A}'$  that assigns the jobs uniformly at random to the processors. Then, for any schedule  $\sigma$ , we have*

$$\text{delay}(\sigma_A) \leq \text{delay}(\sigma) \text{ and } \mathbb{E}[\text{energy}(\sigma_A)] \leq \max_x \Phi(x) \text{energy}(\sigma).$$

*Proof.* We clearly have that  $\text{delay}(\sigma_A) \leq \text{delay}(\sigma)$ , since a job can only be completed earlier in  $\sigma_A$  than in  $\sigma$ , and we consider monotonously increasing delay cost functions. Thus, we only have to show that  $\mathbb{E}[\text{energy}(\sigma_A)] \leq \max_x \Phi(x) \text{energy}(\sigma)$ . To this end, for each  $1 \leq i \leq q$  and  $1 \leq r \leq m$ , let

$$\text{energy}_{ir}(\sigma_A) := (t_i - t_{i-1}) \left( \sum_{j \in J_r^A} s_{ij} \right)^\alpha$$

be the energy cost of processor  $r$  in time interval  $[t_{i-1}, t_i]$  in schedule  $\sigma_A$ , and let

$$\text{energy}_i(\sigma_A) := \sum_{r=1}^m \text{energy}_{ir}(\sigma_A)$$

be the energy cost of all processors in this time interval. Analogously, observe that

$$\text{energy}_i(\sigma) := (t_i - t_{i-1}) \sum_{j \in J} s_{ij}^\alpha. \quad (4)$$

is the energy cost of all processors in this time interval in schedule  $\sigma$ . Consider now a fixed  $1 \leq i \leq q$ . Since all jobs are uniformly assigned at random to the processors in  $A$ , all random variables  $\text{energy}_{ir}(\sigma_A)$ ,  $1 \leq r \leq m$ , have the same expected value, which is

$$\mathbb{E}[\text{energy}_{ir}(\sigma_A)] = (t_i - t_{i-1}) \sum_{\Gamma \in \mathcal{P}_n} \phi_n(\Gamma) \left( \sum_{j \in \Gamma} s_{ij} \right)^\alpha.$$

By linearity of expectation, this yields that

$$\mathbb{E}[\text{energy}_i(\sigma_A)] = \sum_{r=1}^m \mathbb{E}[\text{energy}_{ir}(\sigma_A)] = m(t_i - t_{i-1}) \sum_{\Gamma \in \mathcal{P}_n} \phi_n(\Gamma) \left( \sum_{j \in \Gamma} s_{ij} \right)^\alpha,$$

which implies with Equation (4) that  $\mathbb{E}[\text{energy}_i(\sigma_A)] / \text{energy}_i(\sigma) = \Phi_n(s)$ , where  $s$  is an  $n$ -dimensional vector that contains the speeds  $s_{ij}$ ,  $j \in J$ , in an arbitrary order (assume w.l.o.g. that  $s \neq 0$ ). Let now  $s'$  be an arbitrary  $m$ -dimensional subvector of  $s$  that contains all non-zero speeds, but possibly also some zeros. Such a vector exists, since it follows from the definition of the sequence  $t_0 < t_1 < \dots < t_q$  that there are at most  $m$  non-zero speeds. Observe that  $\Phi(s') = \Phi_n(s)$ . Thus, we obtain that

$$\frac{\mathbb{E}[\text{energy}_i(\sigma_A)]}{\text{energy}_i(\sigma)} \leq \max_x \Phi(x).$$

Using this, linearity of expectation implies that

$$\begin{aligned} \mathbb{E}[\text{energy}(\sigma_A)] &= \sum_{i=1}^q \mathbb{E}[\text{energy}_i(\sigma_A)] \leq \max_x \Phi(x) \sum_{i=1}^q \text{energy}_i(\sigma) \\ &= \max_x \Phi(x) \text{energy}(\sigma), \end{aligned}$$

which completes the proof of the lemma.  $\square$

## 2.2 Bounding $\max_x \Phi(x)$

**Lemma 4.** *For an integer-valued  $\alpha$ , the vectors that maximize the function  $\Phi$  are the constant vectors.*

*Proof.* First, recall that the positive part of the euclidean sphere  $S_m^+$  is compact. Consequently, the continuous function  $\Phi$  has a global maximum in  $S_m^+$ . Moreover, since  $\Phi$  is scaling-invariant as illustrated in Figure 2, we conclude that  $\Phi$  has a global maximum although it holds that its domain  $\mathbb{R}_+^m \setminus \{0\}$  is unbounded, and hence not compact. There are two possibilities,  $\Phi$  takes its maximum either in the interior of its domain, which is  $\{x \in \mathbb{R}_+^m \mid \nexists i : x_i = 0\}$ , or at the boundary of its domain, which is  $\{x \in \mathbb{R}_+^m \mid \exists i : x_i = 0\} \setminus \{0\}$ . First, assume for contradiction that there is an  $x$  at the boundary that maximizes  $\Phi$  with  $x_i = 0$  for some  $1 \leq i \leq m$ . But it is then easy to see that  $\Phi(x)$  increases if we set  $x_i$  to 1, which results in a contradiction. Hence, we only have to consider the interior of the domain of  $\Phi$ .

For contradiction, assume now that there is a non-constant  $x \in \mathbb{R}_+^m$  which yields an extremal value. Since  $\Phi$  is differentiable, it is a well-known fact that all partial

derivatives of  $\Phi$  in  $x$  must be 0. Moreover, since  $x$  is non-constant, and hence, at least two dimensions in  $x$  must differ, we can assume w.l.o.g. that  $x_1 > x_2 \geq 1$ , since we can otherwise simply scale this vector and permute the dimensions. Using the quotient rule, we can compute the partial derivative of  $\Phi$  in  $x$  and dimension 1 as

$$\begin{aligned} \frac{\partial \Phi(x)}{\partial x_1} &= c \left[ \left( \frac{\partial}{\partial x_1} \sum_{\Gamma \in \mathcal{P}} \phi(\Gamma) \left( \sum_{j \in \Gamma} x_j \right)^\alpha \right) \left( \sum_{j=1}^m x_j^\alpha \right) \right. \\ &\quad \left. - \left( \sum_{\Gamma \in \mathcal{P}} \phi(\Gamma) \left( \sum_{j \in \Gamma} x_j \right)^\alpha \right) \alpha x_1^{\alpha-1} \right], \end{aligned} \quad (5)$$

where

$$c = \frac{m}{\left( \sum_{j=1}^m x_j^\alpha \right)^2},$$

and let

$$Z_1 := \frac{1}{\alpha x_1^{\alpha-1}} \frac{\partial}{\partial x_1} \sum_{\Gamma \in \mathcal{P}} \phi(\Gamma) \left( \sum_{j \in \Gamma} x_j \right)^\alpha.$$

Hence, the partial derivative (5) is 0 if and only if  $Z_1 = \Phi(x)$ . Using the fact that  $\phi(\Gamma) = \phi(\Gamma')/(m-1)$ , for each pair of subsets  $\Gamma' \subset \Gamma \subseteq J$  with  $|\Gamma| = |\Gamma'| + 1$ , and the binomial formula, we can rewrite  $Z_1$  as

$$\begin{aligned} Z_1 &= \frac{1}{x_1^{\alpha-1}} \sum_{\Gamma \in \mathcal{P}: 1 \in \Gamma} \phi(\Gamma) \left( \sum_{j \in \Gamma} x_j \right)^{\alpha-1} \\ &= \frac{1}{x_1^{\alpha-1}} \sum_{\Gamma \in \mathcal{P}: 1, 2 \notin \Gamma} \left[ \frac{\phi(\Gamma)}{m-1} \left( x_1 + \sum_{j \in \Gamma} x_j \right)^{\alpha-1} \right. \\ &\quad \left. + \frac{\phi(\Gamma)}{(m-1)^2} \left( x_1 + x_2 + \sum_{j \in \Gamma} x_j \right)^{\alpha-1} \right] \\ &= \frac{1}{x_1^{\alpha-1}} \sum_{\Gamma \in \mathcal{P}: 1, 2 \notin \Gamma} \frac{\phi(\Gamma)}{m-1} \sum_{i=0}^{\alpha-1} \binom{\alpha-1}{i} \left( x_1^i + \frac{(x_1+x_2)^i}{m-1} \right) \left( \sum_{j \in \Gamma} x_j \right)^{\alpha-1-i} \\ &= \sum_{i=0}^{\alpha-1} b_i \frac{x_1^i + \frac{(x_1+x_2)^i}{m-1}}{x_1^{\alpha-1}}, \end{aligned} \quad (6)$$

where the coefficients  $b_0, b_1, \dots, b_{\alpha-1} > 0$  are defined as

$$b_i := \binom{\alpha-1}{i} \sum_{\Gamma \in \mathcal{P}: 1, 2 \notin \Gamma} \frac{\phi(\Gamma)}{m-1} \left( \sum_{j \in \Gamma} x_j \right)^{\alpha-1-i},$$

for  $0 \leq i \leq \alpha-1$ . Moreover, we can analogously define  $Z_2$  with respect to

dimension 2, which we can also rewrite as

$$Z_2 = \sum_{i=0}^{\alpha-1} b_i \frac{x_2^i + \frac{(x_1+x_2)^i}{m-1}}{x_2^{\alpha-1}}, \quad (7)$$

and it holds that  $Z_2 = \Phi(x)$  as well. Consequently, we obtain that  $Z_1 = Z_2$ . Finally, if  $x_1 > x_2 \geq 1$ , then, for each  $0 \leq i \leq \alpha - 1$ , we have

$$\frac{x_2^i + \frac{(x_1+x_2)^i}{m-1}}{x_2^{\alpha-1}} > \frac{x_1^i + \frac{(x_1+x_2)^i}{m-1}}{x_1^{\alpha-1}}.$$

By combining this with Equations (6) and (7), it follows that  $Z_1 < Z_2$ , which contradicts  $Z_1 = Z_2$ .  $\square$

**Lemma 5.** *For an integer-valued  $\alpha$  with  $\alpha \leq m$ ,*

$$\max_x \Phi(x) = \sum_{j=1}^{\alpha} S(\alpha, j) \frac{m!}{m^j (m-j)!}.$$

*Proof.* We know from Lemma 4 that if  $x$  maximizes  $\Phi$ , then  $x$  is constant. Hence, using the scaling-invariance of  $\Phi$ , we can choose  $x = (1, \dots, 1)$ . In this case,

$$\begin{aligned} \Phi(x) &= \sum_{i=0}^m \binom{m}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{m-i} i^\alpha \\ &= \left(\frac{m-1}{m}\right)^m \sum_{i=0}^m \binom{m}{i} \frac{i^\alpha}{(m-1)^i}. \end{aligned} \quad (8)$$

It is worth mentioning that although there is a close relation, we cannot simply apply Dobinski's formula (1) to upper bound (8). Instead, we inductively construct a sequence of functions  $f_0, f_1, \dots, f_\alpha$ . Let  $f_0(t) := (1+t)^m$ , and, for each  $1 \leq i \leq \alpha$ , let

$$f_i(t) := t \frac{df_{i-1}(t)}{dt}.$$

For each  $1 \leq i \leq \alpha$ , note there is a sequence of coefficients  $c(i, 1), c(i, 2), \dots, c(i, i)$  which are independent of  $t$  such that we can write

$$f_i(t) = \sum_{j=1}^i c(i, j) \frac{m!}{(m-j)!} (1+t)^{m-j} t^j. \quad (9)$$

By the definition of the functions  $f_1, \dots, f_\alpha$ , it is easy to verify that, for any  $1 \leq i \leq \alpha$  and  $1 < j < i$ , the recurrence relation

$$c(i, j) = jc(i-1, j) + c(i-1, j-1)$$

is satisfied. In combination with the fact that  $c(i, 1) = c(i, i) = 1$ , for any  $1 \leq i \leq \alpha$ , we conclude with the recurrence relation of the Stirling numbers of the second kind in Equation (3) that  $c(\alpha, i) = S(\alpha, i)$ , for each  $1 \leq i \leq \alpha$ . On the other hand, by the binomial formula,

$$f_0(t) = \sum_{i=0}^m \binom{m}{i} t^i,$$

which yields with the definition of the function  $f_1, f_2, \dots, f_\alpha$  that

$$f_\alpha(t) = \sum_{i=1}^m \binom{m}{i} i^\alpha t^i.$$

Therefore, by Equations (8) and (9),

$$\begin{aligned} \Phi(x) &= \left(\frac{m-1}{m}\right)^m f_\alpha\left(\frac{1}{m-1}\right) \\ &= \left(\frac{m-1}{m}\right)^m \sum_{j=1}^{\alpha} c(\alpha, j) \frac{m!}{(m-j)!} \left(1 + \frac{1}{m-1}\right)^{m-j} \left(\frac{1}{m-1}\right)^j \\ &= \sum_{j=1}^{\alpha} S(\alpha, j) \frac{m!}{m^j (m-j)!}, \end{aligned}$$

which proves the claim.  $\square$

*Proof of Lemma 2.* The correctness of algorithm  $\mathcal{A}'$  is obvious. Moreover, since  $\mathcal{A}$  is an approximation algorithm, it has polynomial running time. On the other, for any monotonously increasing polynomial  $p(x)$  of degree at least 1, it holds that

$$\sum_{r=1}^m p(|J_r^A|) \leq p\left(\sum_{r=1}^m |J_r^A|\right) = p(n),$$

which shows that algorithm  $\mathcal{A}'$  inherits the asymptotic running time of algorithm  $\mathcal{A}$ . Hence, it remains to show that algorithm  $\mathcal{A}'$  has the claimed approximation ratio. To this end, consider the multiprocessor schedule  $\sigma'$  computed by algorithm  $\mathcal{A}'$ , the random job assignment  $A$  from algorithm  $\mathcal{A}'$ , and an optimal multiprocessor schedule  $\sigma^*$  with  $\text{cost}(\sigma^*) = \text{OPT}$ . Note that  $\sigma'$  and  $\sigma_A^*$  process the same set of jobs  $J_r^A$  on each processor  $r$ , where  $\sigma_A^*$  is defined in Subsection 2.1. Let then  $\text{cost}_r(\sigma')$  and  $\text{cost}_r(\sigma_A^*)$  be the cost of processor  $r$  in  $\sigma'$  and  $\sigma_A^*$ , respectively, i.e., the energy cost of this processor plus the sum of delay costs of the jobs  $J_r^A$ . Consequently, since the single processor  $\beta$ -approximation algorithm  $\mathcal{A}$  is applied

separately to each processor in  $\mathcal{A}'$ , we have that

$$\mathbb{E}[\text{cost}_r(\sigma') \mid A] \leq \beta \mathbb{E}[\text{cost}_r(\sigma_A^*) \mid A].$$

It follows that

$$\begin{aligned} \mathbb{E}[\text{cost}_r(\sigma')] &= \mathbb{E}[\mathbb{E}[\text{cost}_r(\sigma') \mid A]] \leq \beta \mathbb{E}[\mathbb{E}[\text{cost}_r(\sigma_A^*) \mid A]] \\ &= \mathbb{E}[\text{cost}_r(\sigma_A^*)]. \end{aligned}$$

Therefore, using linearity of expectation, we have that

$$\begin{aligned} \mathbb{E}[\text{cost}(\sigma')] &= \sum_{r=1}^m \mathbb{E}[\text{cost}_r(\sigma')] \leq \beta \sum_{r=1}^m \mathbb{E}[\text{cost}_r(\sigma_A^*)] = \beta \mathbb{E}[\text{cost}(\sigma_A^*)] \\ &\leq \beta \max_x \Phi(x) \text{OPT}, \end{aligned}$$

whereas the second line is due to Lemma 3. The claim of the lemma follows by combining this with Lemma 5.  $\square$

We only considered integer-valued  $\alpha \leq m$  so far. However, for any  $x$  in the domain of  $\Phi$ , it is easy to see that  $\Phi(x)$  increases monotonously for increasing  $\alpha$ . Therefore, since we define  $B_\alpha := B_{\lceil \alpha \rceil}$  for arbitrary  $\alpha > 0$ , all arguments in this subsection hold for any  $\alpha > 1$  with  $\alpha \leq m$  as well.

### 3 Removing migration in multiprocessor schedules

In this section, we prove the following theorem:

**Theorem 6.** *Any  $\beta$ -approximation algorithm for SSM with migration yields a (deterministic)  $\beta B_\alpha$ -approximation algorithm for SSM, whereas we need to add  $O(n^{\alpha+4})$  to the running time.*

As for Theorem 1, we prove Theorem 6 by converting an algorithm  $\mathcal{A}$  into an algorithm  $\mathcal{A}'$  for SSM, but in this section,  $\mathcal{A}$  is an algorithm for SSM with migration. To this end, we show how to remove migration in a multiprocessor schedule by derandomizing the transformation presented in Subsection 2.1. Note that this transformation works as well if the input schedule  $\sigma$  is a multiprocessor schedule with migration, and Lemma 3 holds analogously. However, in this case, the final schedule  $\sigma_A$  will be a multiprocessor schedule without migration, which we exploit in the following.

In Subsection 2.1, we did not require that the used value  $q$  is polynomially bounded. However, it clearly holds for a multiprocessor schedule  $\sigma$  returned by some polynomial time algorithm  $\mathcal{A}$  that  $q$  is polynomially bounded. Moreover, assume in this section that  $q = O(n)$ , and hence, we can construct  $\sigma_A$  from  $\sigma$  and  $A$  in polynomial time  $O(nm)$ . Otherwise, the conversion of  $\mathcal{A}$  into  $\mathcal{A}'$  works as well, but it is harder to bound the additional polynomial running time of  $\mathcal{A}'$ .

We call a job assignment  $A$  *partial* if only a subset of the jobs  $J^A \subset J$  is assigned to some processors. For such an assignment, let  $\bar{A}$  be the random job assignment where all remaining jobs  $J \setminus J^A$  are uniformly at random assigned to the  $m$  processors. Using this additional definition, algorithm  $\mathcal{A}'$  is defined as follows:

---

$\mathcal{A}'(J, m, \mathcal{A})$

---

1. Using algorithm  $\mathcal{A}$ , compute an optimal schedule  $\sigma$  with migration for the jobs  $J$  on  $m$  processors.
2. Set  $A$  to the empty job assignment with  $J^A = \emptyset$ .
3. While  $J^A \neq J$ :
  - (a) Select an arbitrary job  $j' \in J \setminus J^A$ .
  - (b) For each processor  $r = 1, \dots, m$ , set  $A_r$  to the extension of  $A$  which assigns the jobs  $J^A \cup \{j'\}$ , where  $A_r(j) = A(j)$ , for each  $j \in J^A$ , and  $A_r(j') = r$ .
  - (c) Compute the expected values

$$\mathbb{E} [\text{energy}(\sigma_{\bar{A}_1})], \mathbb{E} [\text{energy}(\sigma_{\bar{A}_2})], \dots, \mathbb{E} [\text{energy}(\sigma_{\bar{A}_m})],$$

and set  $A$  to the extension  $A_r$  with minimum  $\mathbb{E} [\text{energy}(\sigma_{\bar{A}_r})]$ .

4. Return the multiprocessor schedule  $\sigma' \leftarrow \sigma_A$ .
- 

To see that the critical Step 3c runs in polynomial time, we need the following lemma, which is proven in the end of this subsection:

**Lemma 7.** *Let  $\sigma$  be a multiprocessor schedule and  $A$  a partial job assignment. Then, we can compute the expected value  $\mathbb{E} [\text{energy}(\sigma_{\bar{A}})]$  in polynomial time  $O(n^{\alpha+2})$ .*

Theorem 6 immediately follows from the following lemma:

**Lemma 8.** *If  $\mathcal{A}$  is a  $\beta$ -approximation algorithm for SSM with migration, then  $\mathcal{A}'$*

is a  $\beta B_\alpha$ -approximation algorithm for SSM. Moreover, the running time of  $\mathcal{A}'$  is the running time of  $\mathcal{A}$  plus  $O(n^{\alpha+4})$ .

*Proof.* Lemma 3 and 5 imply that  $\mathbb{E}[\text{energy}(\sigma_{\bar{A}})] \leq B_\alpha \text{energy}(\sigma)$  for the initial empty partial job assignment  $A$  with  $J^A = \emptyset$ . Now consider a fixed iteration of the loop in Step 3, and let  $A$  be the current partial job assignment. If  $\mathbb{E}[\text{energy}(\sigma_{\bar{A}})] \leq B_\alpha \text{energy}(\sigma)$  before this iteration, then, since

$$\mathbb{E}[\text{energy}(\sigma_{\bar{A}})] = \frac{1}{m} \sum_{r=1}^m \mathbb{E}[\text{energy}(\sigma_{\bar{A}_r})],$$

the selection of  $A$  in Step 3c implies that still  $\mathbb{E}[\text{cost}(\sigma_{\bar{A}})] \leq B_\alpha \text{energy}(\sigma)$  before the next iteration. Moreover, we finally have after the last iteration that  $J^A = J$ , and hence,  $A$  is then a non-partial assignment. Consequently, by the arguments above,  $\text{energy}(\sigma') \leq B_\alpha \text{energy}(\sigma)$ . In combination with the fact  $\text{delay}(\sigma') \leq \text{delay}(\sigma)$  from Lemma 3, we conclude that  $\text{cost}(\sigma') \leq B_\alpha \text{cost}(\sigma)$ . This proves the claim of the lemma, since having migration is a relaxation, and hence  $\text{cost}(\sigma) \leq \text{OPT}$ , where  $\text{OPT} = \text{cost}(\sigma^*)$  for an optimal multiprocessor schedule  $\sigma^*$  without migration. To see the running time with Lemma 7, observe that we have to compute at most  $n^2$  expected values.  $\square$

*Proof of Lemma 7.* As defined in Subsection 2.1, consider a sequence of times  $t_0 < t_1 < \dots < t_q$  with respect to  $\sigma$ , and, for each  $1 \leq i \leq q$  and job  $j \in J$ , let again  $s_{ij}$  be the processing speed of  $j$  in time interval  $[t_{i-1}, t_i)$ , where  $s_{ij} = 0$  if  $j$  is not processed by any processor in this time interval. Moreover, for each  $1 \leq r \leq m$ , let again  $J_r^A := \{j \mid A(j) = r\}$  be the jobs assigned to processor  $r$  by  $A$ . Finally, for each  $1 \leq i \leq q$  and  $1 \leq r \leq m$ , let

$$s_{ir}^A := \sum_{j \in J_r^A} s_{ij}$$

be the random variable that describes the speed of processor  $r$  in time interval  $[t_{i-1}, t_i)$  in the random schedule  $\sigma_{\bar{A}}$ . Hence, by linearity of expectation, it holds that

$$\mathbb{E}[\text{energy}(\sigma_{\bar{A}})] = \sum_{i=1}^q (t_i - t_{i-1}) \sum_{r=1}^m \mathbb{E}[(s_{ir}^A)^\alpha]. \quad (10)$$

Therefore, we need to analyze the expected value  $\mathbb{E}[(s_{ir}^A)^\alpha]$  for each  $1 \leq i \leq q$  and  $1 \leq r \leq m$ .

Consider a fixed  $1 \leq i \leq q$  and  $1 \leq r \leq m$ , and let

$$\mathcal{P}_n^r := \{\Gamma \in \mathcal{P}_n \mid \Gamma \cap J^A = J_r^A\}.$$

be the set of all possible sets of jobs which can be assigned to processor  $r$  subject to the constraint that the jobs  $J_r^A$  are definitely assigned to this processor. Moreover, let

$$a := |J_r^A| \text{ and } b := |J_r^A \cup (J \setminus J^A)|$$

be the minimal and maximal cardinality of a set  $\Gamma \in \mathcal{P}_n^r$ , respectively, and for each  $a \leq s \leq b$ , let then

$$\mathcal{P}_n^{rs} := \{\Gamma \in \mathcal{P}_n^r \mid |\Gamma| = s\}$$

be the sets in  $\mathcal{P}_n^r$  of cardinality  $s$ . For each  $\Gamma \in \mathcal{P}_n^{rs}$ , observe that

$$\phi_n^r(\Gamma) := \left(\frac{1}{m}\right)^{s-a} \left(1 - \frac{1}{m}\right)^{b-s}$$

is the probability that exactly the set of jobs  $\Gamma$  is assigned to processor  $r$  if all remaining jobs  $J \setminus J^A$  are assigned uniformly at random. Finally, for each  $\Gamma \in \mathcal{P}_n^r$ , let

$$\Omega(\Gamma) := \left\{ f \in \mathbb{N}_0^J \mid \sum_{j \in J} f(j) = \alpha \text{ and } \chi(f) \subseteq \Gamma \right\},$$

where  $\mathbb{N}_0^J$  denotes the functions that map the jobs  $J$  to  $\mathbb{N}_0$  and  $\chi(f) := \{j \mid f(j) \neq 0\}$  is the support of such a function. Now assume that given some  $\Gamma \in \mathcal{P}_n^r$ , we have some positive values  $x_j, j \in \Gamma$ . Then it holds that

$$\left( \sum_{j \in \Gamma} x_j \right)^\alpha = \sum_{f \in \Omega(\Gamma)} B_f \prod_{j \in J} x_j^{f(j)}, \quad (11)$$

for some coefficients  $B_f, f \in \Omega(\Gamma)$ . We can think of these coefficients as generalized binomial coefficients. Observe that they are independent of the used subset  $\Gamma$ . To compute such a coefficient for a function  $f \in \Omega(\Gamma)$ , consider an arbitrary ordering  $f_1, f_2, \dots, f_{|\chi(f)|}$  of the non-zero function values of  $f$ . It can be easily verified that

$$\begin{aligned} B_f &= \prod_{l=1}^{|\chi(f)|} \binom{\alpha - \sum_{z < l} f_z}{f_l} \\ &= \frac{\alpha!}{\prod_{l=1}^{|\chi(f)|} f_l!} \end{aligned}$$

Hence, since  $\alpha$  is constant, we can compute  $B_f$  in constant time.

Now we are ready to compute  $\mathbb{E}[(s_{ir}^A)^\alpha]$ . Using the definitions above and Equation (11), we obtain that

$$\begin{aligned}
\mathbb{E}[(s_{ir}^A)^\alpha] &= \sum_{\Gamma \in \mathcal{P}_n^r} \phi_n^r(\Gamma) \left( \sum_{j \in \Gamma} s_{ij} \right)^\alpha \\
&= \sum_{\Gamma \in \mathcal{P}_n^r} \phi_n^r(\Gamma) \sum_{f \in \Omega(\Gamma)} B_f \prod_{j \in J} s_{ij}^{f(j)} \\
&= \sum_{f \in \Omega(J)} B_f \prod_{j \in J} s_{ij}^{f(j)} \sum_{\Gamma \in \mathcal{P}_n^r: \chi(f) \subseteq \Gamma} \phi_n^r(\Gamma). \tag{12}
\end{aligned}$$

Consider now a fixed  $f \in \Omega(J)$ , and let

$$c := |\chi(f) \cup J_r^A|$$

be the minimal cardinality of a set  $\Gamma \in \mathcal{P}_n^r$  with  $\chi(f) \subseteq \Gamma$ . Moreover, note that, for each  $s \geq c$ ,

$$|\{\Gamma \in \mathcal{P}_n^{rs} \mid \chi(f) \subseteq \Gamma\}| = \binom{b-c}{s-c}.$$

Using this, we can rewrite the last sum in Equation (12) as

$$\begin{aligned}
\sum_{\Gamma \in \mathcal{P}_n^r: \chi(f) \subseteq \Gamma} \phi_n^r(\Gamma) &= \sum_{s=c}^b \sum_{\Gamma \in \mathcal{P}_n^{rs}: \chi(f) \subseteq \Gamma} \phi_n^r(\Gamma) \\
&= \sum_{s=c}^b \binom{b-c}{s-c} \left(\frac{1}{m}\right)^{s-a} \left(1 - \frac{1}{m}\right)^{b-s} \\
&= \left(\frac{1}{m}\right)^{c-a} \sum_{s=c}^b \binom{b-c}{s-c} \left(\frac{1}{m}\right)^{s-c} \left(1 - \frac{1}{m}\right)^{b-s} \\
&= \left(\frac{1}{m}\right)^{c-a} \sum_{s=0}^{b-c} \binom{b-c}{s} \left(\frac{1}{m}\right)^s \left(1 - \frac{1}{m}\right)^{b-c-s} \\
&= \left(\frac{1}{m}\right)^{c-a}.
\end{aligned}$$

Thus, we can compute this sum in time  $O(\log n)$ . Moreover, since this sum only depends on the integer value  $c - a$ , we can initially compute all possible values of such a sum in time  $O(n \log n)$ , and store these values in an array. Finally, it clearly holds that  $|\Omega(J)| \leq n^\alpha$ . By combining these two facts with Equation (12), it follows that we can compute  $\mathbb{E}[(s_{ir}^A)^\alpha]$  in time  $O(n^\alpha)$ . Therefore, by Equation (10) and the assumption that  $m \leq n$ , we can compute  $\mathbb{E}[\text{energy}(\sigma_{\overline{A}})]$  in polynomial time  $O(n^{\alpha+2})$ , which proves the claim.  $\square$

## References

- [1] Susanne Albers. Personal communication. 2008.
- [2] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3(4), 2007.
- [3] Susanne Albers, Fabian Müller, and Swen Schmelzer. Speed scaling on parallel processors. In *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '07)*, pages 289–298, 2007.
- [4] Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *Perf. Eval. Review (to appear)*.
- [5] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. In *Proceedings of 8th Latin American Symposium (LATIN'08)*, pages 240–251, 2008.
- [6] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*.
- [7] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *In Proceedings of 45th Symposium on Foundations of Computer Science (FOCS'04)*, pages 520–529, 2004.
- [8] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.
- [9] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, pages 805–813, 2007.
- [10] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
- [11] David P. Bunde. Power-aware scheduling for makespan and flow. In *In Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '06)*, pages 190–196, 2006.
- [12] Ho-Leung Chan, Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Energy efficient online deadline scheduling. In

- Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 795–804, 2007.
- [13] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *In Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS'09)*, pages 255–264, 2009.
- [14] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 363–372, 2004.
- [15] G. Dobinski. Summierung der reihe  $\sum n^m/n!$  fuer  $m = 1, 2, 3, 4, 5, \dots$ . *Arch. Math. Physik*, 61:333–336, 1877.
- [16] H.W.Becker and John Riordan. The arithmetic of bell and stirling numbers. *Amer. J. Math.*, 70:385–394, 1934.
- [17] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
- [18] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, , and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, pages 647–659, 2008.
- [19] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Energy efficient deadline scheduling in two processor systems. In *Proceedings of 18th International Symposium on Algorithms and Computation, (ISAAC'07)*, pages 476–487, 2007.
- [20] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of the 20th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'08)*, pages 256–264, 2008.
- [21] László Lovász. *Combinatorial problems and exercises*. North-Holland, Amsterdam, 1964.
- [22] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Trans. Algorithms*, 4(3), 2008.
- [23] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *In Proceedings of 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 374–382, 1995.