Noname manuscript No.
(will be inserted by the editor)

Multiprocessor Speed Scaling for Jobs with Arbitrary Sizes and Deadlines

Paul C. Bell, Prudence W.H. Wong

Received: date / Accepted: date

Abstract In this paper we study energy efficient deadline scheduling on multiprocessors in which the processors consumes power at a rate of s^α when running at speed s, where $\alpha \geq 2$. The problem is to dispatch jobs to processors and determine the speed and jobs to run for each processor so as to complete all jobs by their deadlines using the minimum energy. The problem has been well studied for the single processor case. For the multiprocessor setting, constant competitive online algorithms for special cases of unit size jobs or arbitrary size jobs with agreeable deadlines have been proposed by Albers et al. [6]. A randomized algorithm has been proposed for jobs of arbitrary sizes and arbitrary deadlines by Greiner et al. [22]. We propose a deterministic online algorithm for the general setting and show that it is $O(\log^\alpha P)$-competitive, where P is the ratio of the maximum and minimum job size.
Keywords: Online algorithms; Dynamic speed scaling; Competitive analysis; Multiprocessor scheduling; Deadline scheduling.

1 Introduction

Energy efficient deadline scheduling. Energy consumption has become an important concern in the design of modern processors, not only for battery-operated mobile devices with single processors but also for server farms or

This work is partially supported by EPSRC Grant EP/E028276/1. A preliminary version appeared in Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation, 2011, pp. 27–36.

P. C. Bell
Corresponding Author, Department of Computer Science, Loughborough University, Loughborough, LE11 3TU, UK, p.bell@lboro.ac.uk

P. W. H. Wong
Department of Computer Science, University of Liverpool, Ashton Building, Ashton St, Liverpool, L69 3BX, UK, pwong@liverpool.ac.uk

laptops with multi-core processors. A popular technology to reduce energy usage is *dynamic speed scaling* (see e.g., [6, 13, 18, 33]) where the processor can vary its speed dynamically. The power consumption is modelled by s^α when the processor runs at speed s, where α is typically 2 or 3 [17,31]. Running a job slower saves energy, yet it takes longer to finish the job. The challenge arises from the conflicting objectives of providing good "quality of service" (QoS) and conserving energy. Deadline feasibility is a common QoS measure for job scheduling. Jobs with arbitrary sizes and deadlines arrive at unpredictable times and they are to be run on some processor. Preemption is allowed with no penalty.

The theoretical study of speed scaling was initiated by Yao et al. [33]. They studied deadline scheduling on a single processor in which jobs with arbitrary sizes and deadlines arrive online and the aim is to finish all jobs by their deadlines using the minimum amount of energy. The decision at any time is to determine which job to run and at what speed. They gave an optimal offline algorithm and a simple online algorithm AVR which is $2^{\alpha-1}\alpha^\alpha$-competitive and they also proposed an online algorithm OA. Bansal, Kimbrel and Pruhs [13] later showed that OA is α^α-competitive. They also gave a $2(\alpha/(\alpha-1))^\alpha e^\alpha$-competitive algorithm, which is called the BKP algorithm and is better than OA when $\alpha > 5$. The result is further improved to $4^\alpha/(2\sqrt{e\alpha})$-competitive by the qOA algorithm [12].

The problem of energy efficient scheduling has also been studied for other QoS measures. The problem of minimizing flow time and energy has attracted a lot of attention [5, 10, 11, 14, 22, 23, 29, 30] while other objectives including lateness, stretch and rejection have also been considered [9, 19, 21, 32]. Energy efficient scheduling has also been extended to the setting with sleep states [3, 20, 24, 26, 27]. The literature also contains results on other aspects of energy efficient scheduling, see [1, 2, 25].

Energy efficient multiprocessor scheduling. The problem of energy efficient deadline scheduling becomes NP-hard in the multiprocessor setting, even when all the jobs have the same arrival times and deadlines. In the multiprocessor setting, in addition to determining processor speeds, a job dispatching algorithm is required to assign jobs to processors. Albers et al. [6] have extended the study to the multiprocessor setting in which preemption is allowed but migration is not allowed[1]. They study the special cases of unit-size jobs or jobs with agreeable deadlines (jobs arriving earlier have earlier deadlines). If jobs have unit-size and agreeable deadlines, Round Robin (RR) is optimal. For the case of unit-sized jobs with arbitrary deadlines or arbitrary-sized jobs with agreeable deadlines, they gave an $\alpha^\alpha 2^{4\alpha}$-competitive algorithm. Their algorithm, called Classified Round Robin (CRR) first classifies jobs according to the density of the job (the ratio of the job size to the duration between arrival and deadline), and then schedules jobs in each class independently using RR. All jobs (of different classes) dispatched on a processor are run at a

[1] Some work has considered the scenarios where migration is allowed [4, 7, 16] or preemption is not allowed [8].

speed determined by AVR. The case for jobs of arbitrary sizes and arbitrary deadlines is left as an open question.

Greiner, Nonner and Souza [22] have shown that any β-competitive algorithm for a single processor yields a randomized βB_α-competitive algorithm, where B_α is the αth Bell number [15] and this result holds for jobs of arbitrary size and arbitrary deadlines. This means that the existing algorithms [12,13,33] for single processors lead to randomized online algorithms in the multiprocessor setting. Yet it is still an open question to have a competitive deterministic algorithm for the general case of jobs with arbitrary sizes and arbitrary deadlines.

Our contribution. In this paper we study the generalized problems in the multiprocessor setting where jobs have arbitrary sizes and arbitrary deadlines and give a deterministic online algorithm. We first show that the Classified Round Robin algorithm (CRR) [6] does not scale well when jobs have arbitrary sizes and deadlines. The competitive ratio is at least $m^{\alpha-1}$, where m is the number of processors. We then consider a natural extension of CRR and propose a non-migratory deterministic job dispatching algorithm, called *Dual-Classified Round Robin* (DCRR), which classifies jobs in terms of both density and sizes. We show that DCRR coupled with AVR is $2^{4\alpha}(\log^\alpha P + \alpha^\alpha 2^{\alpha-1})$-competitive where P is the ratio between the maximum and minimum job size. Note that the competitive ratio is independent of m and holds even against an optimal migratory offline algorithm.

Roughly speaking, to analyze the performance of DCRR, we round the density and size of a job to the boundaries that define the classes, and show that the performance on the general set is no more than a constant factor of that on such a "nice" job set. This idea is similar to the proof in [6], which rounds only the density of the jobs. We further show that for a nice job set, the classification of DCRR means that the jobs in the same class satisfy the property of agreeable deadlines, making the analysis easier. We are then able to show that the competitive ratio of DCRR depends on the number of classes, which is related to $\log P$.

Organization of the paper. The rest of the paper is organized as follows. In Section 2, we define the problem and give some preliminary results. In Section 3, we review an existing algorithm CRR and show that it does not work well for jobs of arbitrary sizes and deadlines. In Section 4, we describe and analyze our algorithm DCRR. Finally, we conclude in Section 5.

2 Preliminaries

We are to schedule a set of jobs onto m processors M_0, M_1, $\cdots$, M_{m-1}. Preemption is allowed without penalty but migration is not allowed. The speed of each processor can be varied. When running at speed s, a processor processes s units of work and consumes s^α units of energy in each time unit, where $\alpha \geq 2$.

We denote the release time, deadline and size of a job j as $r(j)$, $d(j)$, and $w(j)$, respectively. The *span* of job j is $\text{span}(j) = d(j) - r(j)$ and the *density* $\text{den}(j) = \frac{w(j)}{d(j)-r(j)}$. A job j is called *active* at time t if $r(j) \leq t \leq d(j)$.

The problem is to dispatch the jobs to processors, and for each processor, to determine which job and at what speed to run at any time. The objective is to complete all jobs by their deadlines using the minimum energy.

Consider any job set $\mathcal{J}$. For any algorithm A, we overload the symbol $A(\mathcal{J})$ to mean both the schedule of A on $\mathcal{J}$ and the energy required by the schedule. Let OPT_1 and OPT_m denote the optimal schedule on a single processor and m processors, respectively. In [6], it has been shown that $\text{OPT}_1(\mathcal{J})/m^{\alpha-1} \leq \text{OPT}_m(\mathcal{J})$. We further lower bound the value $\text{OPT}_m(\mathcal{J})$. At any time t, the speed of AVR on a processor is the sum of the densities of all active jobs at t scheduled on this processor. It has been shown in [33] that $\text{AVR}_1(\mathcal{J}) \leq \alpha^\alpha 2^{\alpha-1}\text{OPT}_1(\mathcal{J})$, implying $\text{AVR}_1(\mathcal{J}) \leq \alpha^\alpha 2^{\alpha-1} m^{\alpha-1}\text{OPT}_m(\mathcal{J})$. Let $\text{MIN}(\mathcal{J})$ be the minimum energy to run each job of $\mathcal{J}$ independently of other jobs, i.e., $\text{MIN}(\mathcal{J}) = \sum_{j\in\mathcal{J}} \text{den}^\alpha(j)\text{span}(j)$. Then, we have $\text{MIN}(\mathcal{J}) \leq \text{OPT}_m(\mathcal{J})$. We summarize these bounds on $\text{OPT}_m(\mathcal{J})$ in the following lemma.

Lemma 1 ([6], [33]) *Consider any job set $\mathcal{J}$.* (a) $\text{OPT}_1(\mathcal{J})/m^{\alpha-1} \leq \text{OPT}_m(\mathcal{J})$. (b) (i) $\text{MIN}(\mathcal{J}) \leq \text{OPT}_m(\mathcal{J})$*;* (ii) $\text{AVR}_1(\mathcal{J}) \leq \alpha^\alpha 2^{\alpha-1} m^{\alpha-1}\text{OPT}_m(\mathcal{J})$.

3 Classified Round Robin (CRR)

In this section, we review the algorithm CRR which is $\alpha^\alpha 2^{4\alpha}$-competitive for the special case in which jobs are of unit size, or jobs are of arbitrary sizes but agreeable deadlines [6]. We show that CRR is no longer constant competitive when the jobs have arbitrary sizes and arbitrary deadlines.

Let Δ be the maximum density of the jobs in $\mathcal{J}$. CRR classifies jobs with density Δ into density-class-0, and jobs with density in $[\Delta/2^k, \Delta/2^{k-1})$ into density-class-k, for some positive integer k. Jobs within each class are dispatched to processors by round-robin independently. For each processor, the speed is the sum of the densities of the unfinished jobs dispatched to that processor (i.e., AVR) and the processor processes these jobs by splitting the speed equally among them.

The following theorem shows a lower bound for CRR when jobs are of arbitrary sizes and deadlines. Figure 1 shows the CRR schedule and the optimal schedule for the adversary.

Theorem 1 *For arbitrary size jobs with arbitrary deadlines, CRR has a competitive ratio of at least $m^{\alpha-1}$.*

Proof Let $\epsilon > 0$ be a small positive value and $k > 0$ be an arbitrary large value. Given m processors, define a job set $\mathcal{J}$ of m^2 jobs such that for any $1 \leq i \leq m^2$, the release time of job j_i is $i\epsilon$. For all jobs j_i with $i \bmod m \neq 0$, we set the span of the job to be ϵ. For all jobs j_i with $i \bmod m = 0$, we set the

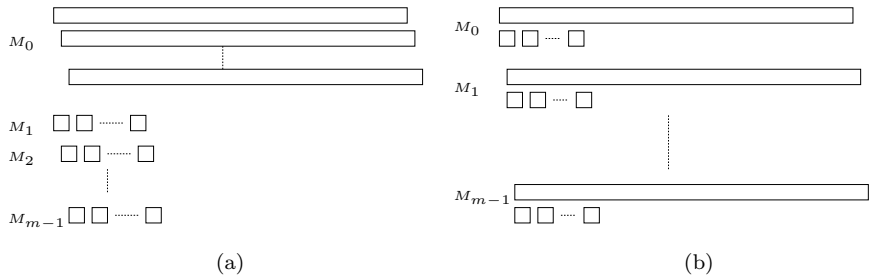

Fig. 1 In the adversary, all jobs have density 1. The span and size of the m large jobs is k and the $m^2 - m$ small jobs is ϵ. (a) CRR schedules all the large jobs to processor M_0 and m small jobs to each of $M_1, \cdots, M_{m-1}$. (b) The optimal schedule dispatches one large job and $m - 1$ small jobs to each processor.

span of the jobs to be k. We further set the sizes of all jobs to be the same as their span, in other words, all jobs have density 1.

Algorithm CRR classifies all m^2 jobs into the same class C_0 since they have the same density and dispatches jobs according to round robin by their release time. Thus the first processor receives the m jobs of large span k and large size k. The energy used by the first processor is therefore km^α as $\epsilon \to 0$ and the energy of the remaining processors approaches 0.

On the other hand, we can dispatch one large span job and $m-1$ small span jobs to each processor. As ϵ tends to 0, the energy used by each processor is k and the total energy of the schedule is km. Therefore, the competitive ratio of CRR is at least $m^{\alpha-1}$. □

We note that even if we classify jobs according to their sizes, such a classification plus round robin still does not perform well. We give a similar adversary with m^2 jobs of the same size, m of them having a small span (thus large density) and the rest with very large span. One job of small span arrives followed by $m - 1$ large span jobs and this repeats for m times. Then CRR assigns all the small span jobs to the same processor, dominating the energy used by the algorithm. The optimal offline algorithm can dispatch one small span job to each processor, distributing the energy used much better and thus the same lower bound can be obtained.

Corollary 1 *For arbitrary size jobs with arbitrary deadlines, modifying CRR to classify on the jobs sizes leads to a competitive ratio of at least* $m^{\alpha-1}$.

4 Dual-Classified Round Robin (DCRR)

4.1 The Algorithm

We now describe our algorithm DCRR (Dual-Classified Round Robin). In addition to classifying jobs into density classes, DCRR also classifies jobs according to sizes. Let Γ be the maximum job size of a job set $\mathcal{J}$. Jobs with

size in $(\Gamma/2^{h+1}, \Gamma/2^h]$ are classified into size-class-h, for some integer $h \geq 0$ (note the difference from the definition of density-classes). We then define the set $C_{k,h}$ to be the set of jobs in density-class-k and size-class-h. For simplicity, we assume that Δ and Γ are known in advance.[2] With the definition of $C_{k,h}$, DCRR dispatches jobs in the same $C_{k,h}$ in a round robin manner, independent of other classes. Then all jobs (of different classes) dispatched to the same processor are run using a speed determined by AVR (see Algorithm 1).

Algorithm 1 Algorithm DCRR

Let Δ and Γ be (respectively) the maximum density and maximum size of all jobs.

Classification: A job is classified into $C_{k,h}$ if its density is in $[\Delta/2^k, \Delta/2^{k-1})$ and its size is in $(\Gamma/2^{h+1}, \Gamma/2^h]$.

Job dispatching: Jobs of the same class $C_{k,h}$ are dispatched (upon their arrival) to the m processors using a round-robin strategy, i.e., the i-th job in $C_{k,h}$ is dispatched to processor-(i mod m), and different classes are handled independently.

Speed running: The speed of each processor is determined by AVR on the jobs dispatched to that processor and the speed is split equally among these jobs (note that this gives a feasible schedule).

Let us consider how DCRR acts upon the example we saw in Figure 1. In Figure 1(a), algorithm CRR dispatches all jobs in the same density class and thus the m jobs with long span went to the same processor. However, we see that DCRR would dispatch those m (long span) jobs into one class and the remaining $m^2 - m$ (short span) jobs in a separate class due to their differing sizes. Algorithm DCRR applies Round-Robin to these classes independently leading to the situation of Figure 1(b) which is optimal in terms of energy.

4.2 Framework of the Analysis and Nice Job Sets.

To analyze the performance of DCRR, we transform job set $\mathcal{J}$ to a *nice* job set $\mathcal{J}^*$ (to be defined) and show that such a transformation only increases the energy usage modestly. Furthermore, we show that for a nice job set $\mathcal{J}^*$, we can bound $\text{DCRR}(\mathcal{J}^*)$ by $\text{OPT}_m(\mathcal{J}^*)$ and in turn by $\text{OPT}_m(\mathcal{J})$. Then we can establish the competitive ratio of DCRR.

A job set $\mathcal{J}^*$ is said to be a *nice job set* if every job j^* in $\mathcal{J}^*$ satisfies the following properties.

- The density $\text{den}(j^*) = \Delta/2^k$, for some positive integer k.
- The size $w(j^*) = \Gamma/2^h$, for some positive integer h.

[2] If Δ and Γ are not known in advance, the class definition could be modified slightly. Specifically, the first job which arrives will define the initial density and size classes Δ' and Γ'. New jobs may have larger sizes or density than these Δ' and Γ' and thus we may have classes with a negative index, but the analysis can be seen to still hold and increasing the competitive ratio by at most a factor of 2, see [6] for further details.

Given a job set $\mathcal{J}$, we transform each job $j \in \mathcal{J}$ into a job j^* as follows. Suppose j is in class $C_{k,h}$.

- We set the release time of j^* to be the same as j, i.e., $r(j^*) = r(j)$.
- We round up the size of j to the maximum in the class $C_{k,h}$, i.e., $w(j^*) = \Gamma/2^h$. Then, we have $w(j) \leq w(j^*) \leq 2w(j)$.
- We round down the density of j to the minimum in the class $C_{k,h}$, i.e., $\text{den}(j^*) = \Delta/2^k$. Then, we have $\text{den}(j)/2 \leq \text{den}(j^*) \leq \text{den}(j)$.
- Effectively, we set the deadline $d(j^*) = r(j^*) + (\frac{\Gamma}{2^h} \cdot \frac{2^k}{\Delta})$.

In other words, job densities only decrease and sizes only increase. The following lemma relates the optimal schedule for $\mathcal{J}$ and $\mathcal{J}^*$, as well as the DCRR schedule for $\mathcal{J}$ and $\mathcal{J}^*$. The implication of the lemma is that we can focus on analyzing the performance of DCRR on nice job set $\mathcal{J}^*$.

Lemma 2 *For any job set $\mathcal{J}$ and its corresponding nice job set $\mathcal{J}^*$, we have* (a) $2^\alpha \text{OPT}_m(\mathcal{J}) \geq \text{OPT}_m(\mathcal{J}^*)$*;* (b) $\text{DCRR}(\mathcal{J}) \leq 2^\alpha \text{DCRR}(\mathcal{J}^*)$.

Proof (a) We construct from $\text{OPT}_m(\mathcal{J})$ a feasible schedule S for $\mathcal{J}^*$, and show that this increases the energy slightly. The dispatching of S follows the dispatching of $\text{OPT}_m(\mathcal{J})$. For any processor, at any time t, S runs at double the speed that $\text{OPT}_m(\mathcal{J})$ does. S is feasible for $\mathcal{J}^*$ because $w(j^*) \leq 2w(j)$ and $\text{span}(j) \leq \text{span}(j^*)$, the latter implies that whenever j is run, it is within the span of j^*. Because of the double speed, $S = 2^\alpha \text{OPT}_m(\mathcal{J})$. As S is a feasible schedule for $\mathcal{J}^*$, we have $S \geq \text{OPT}_m(\mathcal{J}^*)$. Then the statement follows.

(b) First we notice that a job j and its corresponding j^* belong to the same class. The release time of j^* is also kept the same as j. Therefore, j^* will be dispatched to the same processor as j. In the schedule of $\text{DCRR}(\mathcal{J}^*)$, at any time when the job j^* is active, it contributes $\text{den}(j^*)$ to the speed of that processor. If we consider a schedule S' that runs double the speed at any time and on any processor as $\text{AVR}(\mathcal{J}^*)$ does, the job j^* contributes $2 \times \text{den}(j^*)$ to the speed. As for energy usage, $S' = 2^\alpha \text{DCRR}(\mathcal{J}^*)$. On the other hand, in $\text{DCRR}(\mathcal{J})$, at any time that j is active, it contributes $\text{den}(j)$ to the speed of that processor. Since $2\,\text{den}(j^*) \geq \text{den}(j)$ and $\text{span}(j^*) \geq \text{span}(j)$, for any processor, the schedule S' runs at least the same speed as $\text{DCRR}(\mathcal{J})$, and probably higher. Therefore, $S' \geq \text{DCRR}(\mathcal{J})$, and the statement follows. □

4.3 Analysis of DCRR

With Lemma 2, the analysis of DCRR on a general job set $\mathcal{J}$ can be done via the analysis of DCRR on $\mathcal{J}^*$. Recall that we defined $\text{MIN}(\mathcal{J})$ to be the minimum energy to run each job of $\mathcal{J}$ independently of other jobs, i.e., $\text{MIN}(\mathcal{J}) = \sum_{j \in \mathcal{J}} \text{den}^\alpha(j)\text{span}(j)$. First, we show in Lemma 3 a property about how DCRR dispatches jobs in a class to the m processors. Then, in Lemma 4, we relate the sum of energy usage of AVR on jobs DCRR dispatched to each machine with $\text{MIN}(\mathcal{J})$ and $\text{AVR}_1(\mathcal{J})$. Finally, together with Lemma 2, we can then conclude in Theorem 2 the competitive ratio of DCRR.

The following is a modification to a lemma from [6]. Since all spans within a class are identical, they have agreeable deadlines and a similar proof follows as is shown in [6] which we include here for completeness.

Lemma 3 *For any time t, DCRR assigns to each processor at most $\lceil C_{k,h}(t)/m \rceil$ jobs in $\mathcal{J}^*$ from $C_{k,h}(t)$, where $C_{k,h}(t)$ is the set of jobs from $C_{k,h}$ active at time t.*

Proof Recall that within any class $C_{k,h}$ all jobs have agreeable deadlines since we are using nice job set $\mathcal{J}^*$. Since DCRR dispatches each job class using Round Robin independently, they will be dispatched according to non-decreasing release time and thus every m-th job goes to a particular fixed processor. This implies each processor receives at most $\lceil C_{k,h}(t)/m \rceil$ jobs. □

Let $\mathcal{J}_i^*$ be the subset of $\mathcal{J}^*$ that is dispatched to processor i by DCRR. Then $\mathrm{DCRR}(\mathcal{J}^*) = \sum_{1 \le i \le m} \mathrm{AVR}_1(\mathcal{J}_i^*)$. We now relate $\sum_{1 \le i \le m} \mathrm{AVR}_1(\mathcal{J}_i^*)$ with $\mathrm{MIN}(\mathcal{J}^*)$ and $\mathrm{AVR}_1(\mathcal{J}^*)$.

Lemma 4 *For any nice job set $\mathcal{J}^*$, the following inequality holds*

$$\sum_{1 \le i \le m} \mathrm{AVR}_1(\mathcal{J}_i^*) \le 2^{2\alpha}((\log^\alpha P^*)\,\mathrm{MIN}(\mathcal{J}^*) + \mathrm{AVR}_1(\mathcal{J}^*)/m^{\alpha-1})$$

where $P^* = \frac{\max\{w(j) | j \in \mathcal{J}^*\}}{\min\{w(j) | j \in \mathcal{J}^*\}}$.

Proof We adapt the proof of CRR in [6]. Let $C_{k,h,i}(t)$ for $1 \le i \le m$ be the set of jobs from class $C_{k,h}$ *assigned to processor* i active at time t dispatched by DCRR. Let $s_i(t)$ denote the speed of the average rate AVR algorithm on processor i at time t. Since the speed of AVR is the sum of densities of all active jobs at each time point, we see that:

$$s_i(t) = \sum_{k \ge 0} \sum_{h \ge 0} |C_{k,h,i}(t)| \frac{\Delta}{2^k}. \tag{1}$$

Running jobs according to the *Earliest Deadline First* policy yields a feasible schedule. Let $s(t)$ denote the speed of the AVR algorithm for the whole job set $\mathcal{J}^*$ on a single machine. Then $s(t) = \sum_{k \ge 0} \sum_{h \ge 0} |C_{k,h}(t)| \Delta / 2^k$.

Fix a time $t \ge 0$ and a processor $1 \le i \le m$. Let K_1 be the set of job class indices (k,h) such that $|C_{k,h,i}(t)| = 1$ and K_2 be the indices (k,h) such that $|C_{k,h,i}(t)| \ge 2$. Define $k_1 = \min\{k \mid (k,h) \in K_1\}$ for some $h \ge 0$ and $P^* = \frac{\max\{w(j) | j \in \mathcal{J}^*\}}{\min\{w(j) | j \in \mathcal{J}^*\}}$. The analysis for K_2 is similar to [6] while that for K_1 is different. Note that $\sum_{(k,h) \in K_1} \frac{\Delta}{2^k} \le (\log P^*) \sum_{k \ge k_1} \frac{\Delta}{2^k} = (\log P^*) \frac{\Delta}{2^{k_1 - 1}}$. Using

Equation (1) and Lemma 3, we see that

$$\begin{aligned} s_i(t) &= \sum_{(k,h)\in K_1} \frac{\Delta}{2^k} + \sum_{(k,h)\in K_2} |C_{k,h,i}(t)| \frac{\Delta}{2^k} \\ &\leq (\log P^*) \frac{\Delta}{2^{k_1-1}} + \sum_{(k,h)\in K_2} \left\lceil \frac{|C_{k,h}(t)|}{m} \right\rceil \frac{\Delta}{2^k} \qquad \text{(by Lemma 3)} \\ &\leq (\log P^*) \frac{\Delta}{2^{k_1-1}} + \sum_{(k,h)\in K_2} \frac{2|C_{k,h}(t)|}{m} \frac{\Delta}{2^k} \\ &\leq 4 \cdot \max\left\{ (\log P^*) \frac{\Delta}{2^{k_1}}, \frac{s(t)}{m} \right\} \qquad (2) \end{aligned}$$

We shall integrate $s_i(t)^\alpha$ first over all t when the first term of Equation (2) is dominating to give an upper bound on required energy of:

$$(4\log(P^*))^\alpha \sum_{k\geq 0} \sum_{h\geq 0} |C_{k,h} \cap \mathcal{J}_i^*| \left(\frac{\Delta}{2^k}\right)^\alpha \left(2^{k-h} \frac{\Gamma}{\Delta}\right) \quad .$$

The integration sums over all classes and for each class over the time that the first term dominates. Note that the duration of such time is bounded by the span of the jobs in the corresponding class, which is $2^{k-h}\frac{\Gamma}{\Delta}$.

Integrating $s_i(t)^\alpha$ when the second term of Equation (2) is dominating gives $\left(\frac{4}{m}\right)^\alpha \mathrm{AVR}_1(\mathcal{J}^*)$. We then sum over $1 \leq i \leq m$ to give $\sum_{i=1}^m \mathrm{AVR}_1(\mathcal{J}_i^*) \leq 4^\alpha((\log^\alpha P^*)\mathrm{MIN}(\mathcal{J}^*) + m^{1-\alpha}\mathrm{AVR}_1(\mathcal{J}^*))$ as required. □

Together with Lemma 2, we can conclude the competitive ratio of DCRR in the following theorem.

Theorem 2 *For an arbitrary job set $\mathcal{J}$, the competitive ratio of algorithm DCRR is at most $2^{4\alpha}(\log^\alpha P + \alpha^\alpha 2^{\alpha-1})$, where P is the ratio between the maximum and minimum job size.*

Proof By Lemma 4, we know that:

$$\sum_{1\leq i\leq m} \mathrm{AVR}_1(\mathcal{J}_i^*) \leq 2^{2\alpha}((\log^\alpha P^*)\,\mathrm{MIN}(\mathcal{J}^*) + \mathrm{AVR}_1(\mathcal{J}^*)/m^{\alpha-1}).$$

Since $\mathrm{MIN}(\mathcal{J}^*) \leq \mathrm{OPT}_m(\mathcal{J}^*)$ and $\mathrm{AVR}_1(\mathcal{J}^*) \leq \alpha^\alpha 2^{\alpha-1} m^{\alpha-1}\mathrm{OPT}_m(\mathcal{J}^*)$ by Lemma 1 (b) (ii), we therefore conclude that

$$\mathrm{DCRR}(\mathcal{J}^*) = \sum_{1\leq i\leq m} \mathrm{AVR}_1(\mathcal{J}_i^*) \leq 2^{2\alpha}\mathrm{OPT}_m(\mathcal{J}^*)((\log^\alpha P^*) + \alpha^\alpha 2^{\alpha-1}).$$

By Lemma 2 (a) and (b), $\mathrm{DCRR}(\mathcal{J}) \leq 2^\alpha \mathrm{DCRR}(\mathcal{J}^*)$ and $\mathrm{OPT}_m(\mathcal{J}^*) \leq 2^\alpha \mathrm{OPT}_m(\mathcal{J})$. Then, we have

$$\mathrm{DCRR}(\mathcal{J}) \leq 2^{4\alpha}\mathrm{OPT}_m(\mathcal{J})((\log^\alpha P^*) + \alpha^\alpha 2^{\alpha-1}).$$

Note that from the proof of Lemma 4, $\log P^*$ is essentially the number of size classes used by DCRR which does not change under $\mathcal{J}$ or $\mathcal{J}^*$, therefore $\log P$ and $\log P^*$ can be taken to be equal and the theorem holds. □

5 Conclusion

We consider energy efficient deadline scheduling for jobs with arbitrary sizes and deadlines in the multiprocessor setting. We analyze the performance of the deterministic algorithm DCRR and show that it is $O(\log^{\alpha} P)$-competitive. On the other hand, the lower bound we obtain so far for DCRR is no more than a constant. In the proof of Theorem 2, the $\log P$ factor comes in the case K_1, yet this bound is rather loose and we believe that it can be improved.

To improve the result, one may consider how DCRR can be coupled with OA instead of AVR to improve the results. Another open question is to consider speed bounded processors [18], in which case, not all the jobs can be completed by their deadlines. The concern becomes to maximize the throughput (number of jobs completed by their deadlines) and to minimize the energy used to achieve this throughput. The problem has been considered in the single processor setting [10,18] and a two processor system [28]. It would be interesting to derive algorithms that are competitive both in throughput and energy in the multiprocessor setting.

References

1. Albers, S.: Algorithms for energy saving. In: S. Albers, H. Alt, S. Näher (eds.) Efficient Algorithms, *Lecture Notes in Computer Science*, vol. 5760, pp. 173–186. Springer (2009)
2. Albers, S.: Energy-efficient algorithms. Communication ACM **53**(5), 86–96 (2010)
3. Albers, S., Antoniadis, A.: Race to idle: new algorithms for speed scaling with a sleep state. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1266–1285 (2012)
4. Albers, S., Antoniadis, A., Greiner, G.: On multi-processor speed scaling with migration: extended abstract. In: Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 279–288 (2011)
5. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms **3**(4), 49 (2007)
6. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 289–298 (2007)
7. Angel, E., Bampis, E., Kacem, F., Letsios, D.: Speed scaling on parallel processors with migration. In: Proceedings of International Conference Euro-Par 2012 Parallel Processing, pp. 128–140 (2012)
8. Antoniadis, A., Huang, C.C.: Non-preemptive speed scaling. In: Proceedings of Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), pp. 249–260 (2012)
9. Bampis, E., Letsios, D., Milis, I., Zois, G.: Speed scaling for maximum lateness. In: Proceedings of Annual International Computing and Combinatorics Conference (COCOON) (2012). To appear
10. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. In: Proceedings of International Colloquium on Automata, Languages and Programming (ICALP), pp. 409–420 (2008)
11. Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 693–701 (2009)
12. Bansal, N., Chan, H.L., Pruhs, K., Rogozhnikov-Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: Proceedings of International Colloquium on Automata, Languages and Programming (ICALP), pp. 114–155 (2009)

13. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM **54**(1), 3 (2007)
14. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. SIAM J. Computing **39**(4), 1294–1308 (2009). Preliminary version appeared in Proceedings of Symposium on Discrete Algorithms (SODA), pages 805–813, 2007.
15. Becker, H.W., Riordan, J.: The arithmetic of Bell and Stirling numbers. American Journal of Mathematics **70**, 385–394 (1948)
16. Bingham, B.D., Greenstreet, M.R.: Energy optimal scheduling on multiprocessors with migration. In: Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 153–161 (2008)
17. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro **20**(6), 26–44 (2000)
18. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.W.H.: Optimizing throughput and energy in online deadline scheduling. ACM Transactions on Algorithms **6**(1), 10 (2009). Preliminary version appeared in Proceedings of Symposium on Discrete Algorithms (SODA), pages 795–804, 2007.
19. Chan, S.H., Lam, T.W., Lee, L.K.: Scheduling for weighted flow time and energy with rejection penalty. In: Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 392–403 (2011)
20. Chan, S.H., Lam, T.W., Lee, L.K., Liu, C.M., Ting, H.F.: Sleep management on multiple machines for energy and flow time. In: Proceedings of International Colloquium on Automata, Lanaguages and Programming (ICALP), pp. 219–231 (2011)
21. Cole, D., Im, S., Moseley, B., Pruhs, K.: Speed scaling for stretch plus energy. Operations Research Letters **40**(3), 180–184 (2012)
22. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 11–18 (2009)
23. Gupta, A., Krishnaswamy, R., Pruhs, K.: Scalably scheduling power-heterogeneous processors. In: Proceedings of International Colloquium on Automata, Lanaguages and Programming (ICALP), pp. 312–323 (2010)
24. Han, X., Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Deadline scheduling and power management for speed bounded processors. Theoretical Computer Science **411**(40–42), 3587–3600 (2010)
25. Irani, S., Pruhs, K.: Algorithmic problems in power management. ACM SIGACT News **32**(2), 63–76 (2005)
26. Irani, S., Shukla, S., Gupta, R.K.: Algorithms for power savings. ACM Transactions on Algorithms **3**(4), 41 (2007)
27. Lam, T.W., Lee, L.K., Ting, H.F., To, I.K.K., Wong, P.W.H.: Sleep with guilt and work faster to minimize flow plus energy. In: Proceedings of International Colloquium on Automata, Lanaguages and Programming (ICALP), pp. 665–676 (2009)
28. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Energy efficient deadline scheduling in two processor systems. In: Proceedings of the International Symposium of Algorithms and Computation (ISAAC), pp. 476–487 (2007)
29. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Speed scaling functions for flow time scheduling based on active job count. In: Proceedings of European Symposium on Algorithms (ESA), pp. 647–659 (2008)
30. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Improved multi-processor scheduling for flow time and energy. Journal of Scheduling **15**(1), 105–116 (2012). Preliminary version appeared in Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pages 256–264, 2008.
31. Mudge, T.: Power: A first-class architectural design constraint. Computer **34**(4), 52–58 (2001)
32. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. In: Proceedings of International Workshop on Approximation and Online Algorithms (WAOA), pp. 307–319 (2005)

33. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS), pp. 374–382 (1995)