Competitive Non-migratory Scheduling for Flow Time and Energy

Tak-Wah Lam
Department of Computer Science
University of Hong Kong
twlam@cs.hku.hk

Lap-Kei Lee
Department of Computer Science
University of Hong Kong
lklee@cs.hku.hk

Isaac K. K. To
Department of Computer Science
University of Liverpool, UK
isaacto@liv.ac.uk

Prudence W. H. Wong *
Department of Computer Science
University of Liverpool, UK
pwong@liv.ac.uk

ABSTRACT

Energy usage has been an important concern in recent research on online scheduling. In this paper we extend the study of the tradeoff between flow time and energy from the single-processor setting [8, 6] to the multi-processor setting. Our main result is an analysis of a simple non-migratory online algorithm called CRR (classified round robin) on $m \geq 2$ processors, showing that its flow time plus energy is within $O(1)$ times of the optimal non-migratory offline algorithm, when the maximum allowable speed is slightly relaxed. This result still holds even if the comparison is made against the optimal migratory offline algorithm (the competitive ratio increases by a factor of 2.5). As a special case, our work also contributes to the traditional online flow-time scheduling. Specifically, for minimizing flow time only, CRR can yield a competitive ratio one or even arbitrarily smaller than one, when using sufficiently faster processors. Prior to our work, similar result is only known for online algorithms that needs migration [21, 23], while the best non-migratory result can achieve an $O(1)$ competitive ratio [14].

The above result stems from an interesting observation that there always exists some optimal migratory schedule $\mathcal{S}$ that can be converted (in an offline sense) to a non-migratory schedule $\mathcal{S}'$ with a moderate increase in flow time plus energy. More importantly, this non-migratory schedule always dispatches jobs in the same way as CRR.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

*This research is partly supported by EPSRC Grant EP/E028276/1.

SPAA'08, June 14–16, 2008, Munich, Germany.

General Terms

Algorithms, Performance, Theory

Keywords

Online scheduling algorithms, competitive analysis, dynamic speed scaling, energy minimization

1. INTRODUCTION

Energy consumption has become a key issue in the design of modern processors. This is essential not only for battery-operated mobile devices with single processors but also for server farms or laptops with multi-core processors. A popular technology to reduce energy usage is *dynamic speed scaling* (see, e.g., [9, 15, 24, 28]) where the processor can vary its speed dynamically. Running a job at a slower speed is more energy efficient, yet it takes longer time and may affect the performance. In the past few years, a lot of effort has been devoted to revisiting classical scheduling problems with dynamic speed scaling and energy concern taken into consideration (e.g., [29, 7, 1, 8, 11, 2, 10, 26, 16]; see [17] for a survey). The challenge basically arises from the conflicting objectives of providing good "quality of service" (QoS) and conserving energy.

One commonly used QoS measurement for scheduling jobs on a processor is the total flow time (or equivalently, average response time). Here, jobs with arbitrary size are released at unpredictable times and the flow time of a job is the time elapsed since it arrives until it is completed. When energy is not a concern, the objective of a scheduler is simply to minimize the total flow time of all jobs. The study of energy-efficient scheduling was initiated by Yao, Demers and Shenker [29]. They considered deadline scheduling on a model where the processor can run at any speed between 0 and ∞, and incurs an energy of s^α per unit time when running at speed s, where $\alpha \geq 2$ (typically 2 or 3 [9, 22]). This model, which we call infinite speed model, also paves the way for studying scheduling that minimizes both the flow time and energy. In particular, Pruhs et al. [27] studied offline scheduling for minimizing the total flow time on a single processor with a given amount of energy. They gave a polynomial time optimal algorithm for the special case when jobs are of unit size. However, this problem does not admit

any constant competitive online algorithm even if jobs are of unit size [8].

Flow time and energy. To better understand the trade-off between flow time and energy, Albers and Fujiwara [1] proposed combining the dual objectives into a single objective of minimizing the sum of total flow time and energy. The intuition is that, from an economic viewpoint, it can be assumed that users are willing to pay a certain units (say, ρ units) of energy to reduce one unit of flow time. By changing the units of time and energy, one can further assume that $\rho = 1$ and thus would like to optimize total flow time plus energy. Albers and Fujiwara presented an online algorithm that is $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$-competitive for jobs of unit size. This result was recently improved by Bansal et al. [8], who gave a 4-competitive algorithm for jobs of unit size. They also considered the case for jobs with arbitrary size and weight, and presented an $O(1)$-competitive online algorithm for minimizing weighted flow time plus energy (precisely, the competitive ratio is $\mu_\epsilon\gamma_1$, where ϵ is any positive constant, $\mu_\epsilon = \max\{(1+1/\epsilon), (1+\epsilon)^\alpha\}$, and $\gamma_1 < 2\alpha/\ln\alpha$).

The infinite speed model has provided a convenient model for studying power management, yet it is not realistic to assume infinite speed. Recently, Chan et al. [11] introduced the bounded speed model, where the speed can be scaled between 0 and some maximum T. Bansal et al. [6] successfully adapted the previous results on minimizing flow time plus energy to this model. For jobs of unit size and unit weight, they gave a 4-competitive online algorithm. For jobs of arbitrary size and weight, they gave a $(\mu_\epsilon\gamma_2)$-competitive algorithm that uses a processor with maximum speed $(1+\epsilon)T$ for any $\epsilon > 0$, where $\gamma_2 = (2 + o(1))\alpha/\ln\alpha$.

Multiprocessor scheduling. All the results above are about single-processor scheduling. In the older days, when energy was not a concern, flow time scheduling on multiple processors running at fixed speed was an interesting problem by itself (e.g., [4, 5, 19, 25, 13, 14]). In this setting, jobs remain sequential in nature and cannot be executed by more than one processor in parallel. Different online algorithms like SRPT and IMD that are $\Theta(\log P)$-competitive have been proposed respectively under the migratory and the non-migratory model, where P is the ratio of the maximum job size to the minimum job size. Recently, IMD is further shown to be $O(1+\epsilon)$-competitive, when using processors $(1+\epsilon)$ times faster [14]; if migration is allowed, SRPT can achieve a competitive ratio one or even smaller, when using processors two or more times faster [21, 23].

The only previous work on multi-processors taking flow time and energy into consideration was by Bunde [10], which is about an offline approximation algorithm for jobs of unit size. The literature also contains some multi-processor results on optimizing other classical objectives together with energy in the infinite speed model. Pruhs et al. [26] and Bunde [10] both studied offline algorithms for the makespan objective. Albers et al. [2] studied online algorithms for scheduling jobs with restricted deadlines.

Our work on flow time plus energy. We extend the study of online scheduling for minimizing flow time plus energy to the setting of $m \geq 2$ processors. This extension is not only of theoretical interest. Modern processors utilize speed scaling as well as multi-core technology (dual-core and quad-core are getting common); and a multi-core processor is essentially a pool of processors. To make the work more meaningful, we aim at schedules without migrating jobs among processors. In practice, migrating jobs requires overheads and is avoided in many applications.

To save energy in a multiprocessor setting, we want to balance the load of the processors so as to avoid running any processors at high speed. It is natural to consider some kind of round-robin strategy to dispatch jobs. Typical examples include IMD for flow time scheduling [4] and CRR for energy-efficient deadline scheduling in the infinite speed model [2]. The main contribution of this paper is a non-trivial analysis of CRR (classified round robin) for optimizing flow time plus energy. Unlike [2], we apply CRR according to job size rather than job density. Specifically, when a job arrives, CRR dispatches it immediately based on the following notion of *classes* of job sizes. Consider some $\lambda > 0$. A job is said to be in class k if its size is in the range $((1+\lambda)^{k-1}, (1+\lambda)^k]$. Jobs of the same class are dispatched to the m processors using a round-robin strategy (i.e., the i-th job of a class will be dispatched to processor $(i \bmod m)$). It is worth-mentioning that IMD is slightly more complicated than CRR as it dispatches a job to the processor with the smallest accumulated load of the corresponding class.

The most non-trivial result in this paper is that CRR always admits a non-migratory schedule S whose flow time plus energy is within a constant factor of the optimal migratory offline schedule. In an online setting, we do not know how to compute this S, in particular how each individual processor schedules jobs in S. Yet we can approximate S by using the online algorithm BPS [8, 6] separately for each processor. Since BPS is $O(1)$-competitive for minimizing flow time plus energy in a single processor [8, 6], CRR plus BPS would give a competitive result for the multiprocessor setting. Below is a summary of the results based on the bounded speed model (where T denotes the maximum speed). Our analysis can also be applied to the infinite speed model, but it is of less interest. In addition to the constants $\mu_\epsilon, \gamma_1, \gamma_2$ used in [8, 6], it is convenient to define a constant $\eta_\epsilon = (1+\epsilon)^\alpha[(1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\epsilon)/\epsilon^2]$.

- Against the optimal non-migratory schedule: For any $\epsilon > 0$, CRR-BPS can be $(2\eta_\epsilon\mu_\epsilon\gamma_2)$-competitive for minimizing flow time plus energy, when the maximum allowable speed is relaxed to $(1+\epsilon)^3T$. E.g., if $\alpha = 2$ and $\epsilon = 0.6$, the competitive ratio is at most $27\mu_\epsilon\gamma_2$.
- Against the optimal migratory schedule: The competitive ratio becomes $5\eta_\epsilon\mu_\epsilon\gamma_2$.

Implication for flow-time scheduling. Our work also contributes to the study of traditional flow-time scheduling, which assumes fixed-speed processors. We adapt the above result to show that for minimizing flow time only, CRR (plus SRPT for individual processor) would give a non-migratory online algorithm which, when compared with the optimal migratory algorithm, has a competitive ratio of one or even any constant arbitrarily smaller than one, when using sufficiently fast processors. Prior to our work, similar result is known only for online algorithms that need migration; in particular, McCullough and Torng [21] showed that for $m \geq 2$ processors, SRPT is s-speed $\frac{1}{s}$-competitive for flow time, where $s \geq 2 - \frac{1}{m}$. Our non-migratory result is less efficient. More precisely, for any $s \geq 1$, it is s-speed $\frac{5(\sqrt{s}+1)}{(\sqrt{s}-1)^2}$-competitive. E.g., if $s = 64$, the competitive ratio is 0.92.

This paper is about online algorithms, yet the key technique is an offline transformation. Given an optimal migratory (or non-migratory) offline schedule $\mathcal{O}$, we show how to construct a schedule S that follows CRR to dispatch jobs with the total flow time plus energy increasing by a constant factor only. Note that S takes advantage of $\mathcal{O}$ to determine how to schedule jobs and scale the speed within each processor.

1.1 Definitions, notations, and a simple lower bound

Given a job set $\mathcal{J}$, we want to schedule $\mathcal{J}$ on a pool of $m \geq 2$ processors. Note that jobs are sequential in nature and cannot be executed by more than one processor in parallel. All processors are identical and a job can be executed in any processor. Preemption is allowed and a preempted job can be resumed at the point of preemption. We differentiate two types of schedules: a migratory schedule can move partially-executed jobs from one processor to another processor without any penalty, and a non-migratory schedule cannot.

We use $r(j)$ and $p(j)$ to denote respectively the release time and work requirement (or size) of job j. We let $p(\mathcal{J}) = \sum_{j\in\mathcal{J}} p(j)$ be the total size of $\mathcal{J}$. The time required to complete job j using a processor with fixed speed s is $p(j)/s$.

With respect to a schedule $\mathcal{S}$ of a job set $\mathcal{J}$, we use the notations *max-speed*($\mathcal{S}$), $E(\mathcal{S})$ and $F(\mathcal{S})$ for the maximum speed, energy usage, and total flow time of $\mathcal{S}$, respectively. Note that $F(\mathcal{S})$ is the sum, over all jobs, of the time since a job is released until it is completed, or equivalently, the integration over time of the number of unfinished jobs. As processor speed can vary dynamically and the time to execute a job j is not necessarily equal to $p(j)$, we define the *execution time* of job j to be the flow time minus waiting time of j, and define the total execution time of $\mathcal{S}$ to be the sum of execution time over all jobs in $\mathcal{S}$.

It is convenient to define $G(\mathcal{S}) = F(\mathcal{S}) + E(\mathcal{S})$. The following lemma shows a lower bound on $G(\mathcal{S})$ which depends on $p(\mathcal{J})$, irrelevant of the number of processors.

LEMMA 1. *For any m-processor schedule $\mathcal{S}$ for a job set $\mathcal{J}$, $G(\mathcal{S}) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} p(\mathcal{J})$.*

PROOF. Suppose that a job in $\mathcal{S}$ has flow time t. The energy usage for j is minimized if j is run at constant speed $p(j)/t$ throughout, and it is at least $(p(j)/t)^\alpha t = p(j)^\alpha/t^{\alpha-1}$. Since $t + p(j)^\alpha/t^{\alpha-1}$ is minimized when $t = (\alpha-1)^{1/\alpha}p(j)$, we have $t + p(j)^\alpha/t^{\alpha-1} \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} p(j)$. Summing over all jobs, we obtain the desired lower bound. □

2. THE ONLINE ALGORITHM

This section presents the formal definition of the online algorithm CRR-BPS, which produces a non-migratory schedule for $m \geq 2$ processors. In the following sections, we will show that, when compared with the optimal non-migratory or migratory schedule, this algorithm is $O(1)$-competitive for flow time plus energy, when the maximum allowable speed is slightly relaxed.

Consider any $\lambda > 0$. Recall that a job is said to be in class k if its size is in the range $((1+\lambda)^{k-1}, (1+\lambda)^k]$. In a *CRR($\lambda$)-dispatching* schedule, jobs of the same class are dispatched upon their arrival (ties are broken using job ids) to the m processors using a round-robin strategy, and different classes are handled independently. Jobs once dispatched to a processor will be processed there entirely; thus a CRR(λ)-dispatching schedule is non-migratory in nature. For notational ease, when the value of λ is clear in the context, we will simply use the term CRR-dispatching.

The intuition of using a CRR-dispatching schedule comes from a new offline result that there is a CRR-dispatching schedule such that the total flow time plus energy is $O(1)$ times that of the optimal (non-migratory or migratory) offline schedule and the maximum allowable speed is only slightly higher. Details are stated in Theorem 2 below (Sections 3, 4 and 5 are devoted to proving this theorem).

THEOREM 2. *Given a job set $\mathcal{J}$, let $\mathcal{N}^*$ be an optimal non-migratory schedule of $\mathcal{J}$, and let $\mathcal{S}^*$ be an optimal migratory schedule of $\mathcal{J}$. Then for any $\lambda, \epsilon > 0$,*

i. there is a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ such that $G(\mathcal{S}) \leq 2(1+\lambda)^\alpha((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon)G(\mathcal{N}^*)$, and max-speed($\mathcal{S}$) $\leq (1+\lambda)(1+\epsilon) \times$ max-speed($\mathcal{N}^*$); and

ii. there is a CRR(λ)-dispatching schedule $\mathcal{S}'$ for $\mathcal{J}$ such that $G(\mathcal{S}') \leq 5(1+\lambda)^\alpha((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon)G(\mathcal{S}^*)$, and max-speed($\mathcal{S}'$) $\leq (1+\lambda)(1+\epsilon) \times$ max-speed($\mathcal{S}^*$).

Theorem 2 naturally suggests an online algorithm that first dispatches jobs using a CRR(λ)-dispatching policy, and then schedules jobs in each processor independently and in a way that is competitive in the single-processor setting. For the latter we make use of the single-processor algorithm BPS [8, 6]. Below we review the algorithm BPS (the definition is for reference only, in this paper we only need to know the performance of BPS), and define the multi-processor algorithm CRR$_\lambda$-BPS$_\epsilon$, where $\lambda, \epsilon > 0$ are constants.

> **Algorithm BPS.** At any time t, run the job with the smallest size at speed $w_a(t)^{1/\alpha}$, where $w_a(t)$ is the sum of the remaining fraction (i.e., remaining work divided by original work) of all jobs. If $w_a(t)^{1/\alpha}$ exceeds the maximum allowable speed T (if any), just use the maximum speed T.
>
> **Algorithm BPS$_\epsilon$.** At any time t, select the job with the smallest size to execute, and the speed to be used is $(1+\epsilon)$ times the current speed of a simulated BPS schedule on the jobs. Note that the maximum allowable speed is relaxed to $(1+\epsilon)T$.
>
> **Algorithm CRR$_\lambda$-BPS$_\epsilon$.** Jobs are dispatched to the m processors with the CRR(λ)-dispatching policy. Jobs in each processor are scheduled independently using BPS$_\epsilon$.

It is known that BPS$_\epsilon$ performs well in minimizing flow time plus energy for a single processor in the infinite speed model [8] as well as the bounded speed model [6]. Recall that the constants μ_ϵ, γ_1 and γ_2 are defined as $\max\{(1+1/\epsilon), (1+\epsilon)^\alpha\}$, $2\alpha/\ln\alpha$, and $(2+o(1))\alpha/\ln\alpha$, respectively.

LEMMA 3. *[6, 8] Consider any $\epsilon > 0$. For minimizing flow time plus energy on a single processor, BPS$_\epsilon$ is $\mu_\epsilon\gamma_1$-competitive in the infinite speed model, and $\mu_\epsilon\gamma_2$-competitive in the bounded speed model, when the maximum speed is relaxed to $(1+\epsilon)T$.*

Analysis of CRR-BPS. With Theorem 2 and Lemma 3, we can easily derive the performance of CRR-BPS against the optimal non-migratory or migratory algorithm. In addition to the constants $\mu_\epsilon, \gamma_1, \gamma_2$, we define $\eta_\epsilon = (1+\epsilon)^\alpha[(1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\epsilon)/\epsilon^2]$.

COROLLARY 4. *For any $\epsilon > 0$, the performance of CRR$_\epsilon$-BPS$_\epsilon$ for minimizing flow time plus energy is as follows.*

i. Against a non-migratory optimal schedule: CRR$_\epsilon$-BPS$_\epsilon$ is $(2\eta_\epsilon\mu_\epsilon\gamma_1)$-competitive in the infinite speed model, and $(2\eta_\epsilon\mu_\epsilon\gamma_2)$-competitive in the bounded speed model, if the maximum speed is relaxed to $(1+\epsilon)^3T$.

ii. Against a migratory optimal schedule: CRR$_\epsilon$-BPS$_\epsilon$ is $(5\eta_\epsilon\mu_\epsilon\gamma_1)$-competitive in the infinite speed model and $(5\eta_\epsilon\mu_\epsilon\gamma_2)$-competitive in the bounded speed model, if the maximum speed is relaxed to $(1+\epsilon)^3T$.

PROOF. Let $\mathcal{N}^*$ be the optimal non-migratory schedule of a job set $\mathcal{J}$. By Theorem 2, there exists a CRR(ϵ)-dispatching schedule $\mathcal{N}_1$ for $\mathcal{J}$ such that $G(\mathcal{N}_1) \le 2\eta_\epsilon G(\mathcal{N}^*)$ and *max-speed*$(\mathcal{N}_1) \le (1+\epsilon)^2$*max-speed*$(\mathcal{N}^*)$. Let $\mathcal{N}$ be the schedule produced by CRR$_\epsilon$-BPS$_\epsilon$ for $\mathcal{J}$. Applying Lemma 3 to individual processors, we conclude that in the bounded speed model, $G(\mathcal{N}) \le \mu_\epsilon\gamma_2 G(\mathcal{N}_1) \le \eta_\epsilon\mu_\epsilon\gamma_2 G(\mathcal{N}^*)$ and *max-speed*$(\mathcal{N}) \le (1+\epsilon)^3$*max-speed*$(\mathcal{N}^*)$. The analysis of infinite speed model and the migratory case is similar. □

3. RESTRICTED BUT USEFUL OPTIMAL SCHEDULES

The remaining three sections are devoted to proving Theorem 2. In essence, for any $\lambda > 0$, we want to construct a CRR(λ)-dispatching schedule from an optimal schedule with a mild increase in flow time plus energy and in maximum speed. In this section, we introduce two notions to restrict the possible optimal (non-migratory or migratory) schedules so as to ease the construction.

- A job sequence $\mathcal{J}$ is said to be *power-of-*$(1+\lambda)$ if every job in $\mathcal{J}$ has size $(1+\lambda)^k$ for some k.
- For any job sequence $\mathcal{J}$ and schedule $\mathcal{S}$, we say that $\mathcal{S}$ is *immediate-start* if every job starts at exactly its release time in $\mathcal{J}$.

The rest of this section shows that it suffices to focus on job sequences that are power-of-$(1+\lambda)$ and optimal schedules that are immediate-start. See Lemmas 5 and 6 below. The former is relatively easy to observe as we can exploit a slightly higher maximum speed. The latter is, however, non-trivial or perhaps counter-intuitive.

LEMMA 5. *Given a job sequence $\mathcal{J}$, we construct another job sequence $\mathcal{J}_1$ by rounding up the size of every job in $\mathcal{J}$ to the nearest power of $(1+\lambda)$. Then* (i) *any schedule $\mathcal{S}$ for $\mathcal{J}$ defines a schedule $\mathcal{S}_1$ for $\mathcal{J}_1$ such that $G(\mathcal{S}_1) \le (1+\lambda)^\alpha G(\mathcal{S})$ and max-speed$(\mathcal{S}_1) \le (1+\lambda)$max-speed$(\mathcal{S})$; and* (ii) *any schedule $\mathcal{S}_1$ for $\mathcal{J}_1$ defines a schedule $\mathcal{S}$ for $\mathcal{J}$ with $G(\mathcal{S}) \le G(\mathcal{S}_1)$ and max-speed$(\mathcal{S})$ = max-speed$(\mathcal{S}_1)$.*

PROOF. (i) $\mathcal{S}$ naturally defines a schedule $\mathcal{S}_1$ for $\mathcal{J}_1$ as follows. Whenever $\mathcal{S}$ runs a job j at speed s, $\mathcal{S}_1$ runs j at speed $s_1 = s \times (1+\lambda)^{\lceil \log_{1+\lambda} p(j) \rceil}/p(j)$. Note that $s_1 \le (1+\lambda)s$, $E(\mathcal{S}_1) \le (1+\lambda)^\alpha E(\mathcal{S})$, and $F(\mathcal{S}_1) = F(\mathcal{S})$. (ii) is obvious as we can apply $\mathcal{S}_1$ to schedule $\mathcal{J}$ with extra idle time. □

LEMMA 6. *Consider a power-of-$(1+\lambda)$ job sequence $\mathcal{J}_1$ and an optimal schedule $\mathcal{O}_1$ for $\mathcal{J}_1$. We can convert them to a power-of-$(1+\lambda)$ job sequence $\mathcal{J}_2$ and an immediate-start, optimal schedule $\mathcal{O}_2$ for $\mathcal{J}_2$ such that any CRR(λ)-dispatching schedule $\mathcal{S}_2$ for $\mathcal{J}_2$ defines a CRR(λ)-dispatching schedule $\mathcal{S}_1$ for $\mathcal{J}_1$, and if $G(\mathcal{S}_2) \le \beta G(\mathcal{O}_2)$ for some $\beta \ge 1$, then $G(\mathcal{S}_1) \le \beta G(\mathcal{O}_1)$.*

PROOF. We first construct $\mathcal{O}_2$ from $\mathcal{J}_1$ and $\mathcal{O}_1$. Consider any two jobs of the same size in $\mathcal{J}_1$. To construct $\mathcal{O}_2$ from $\mathcal{O}_1$, each time we consider all jobs in $\mathcal{J}_1$ of a particular size, and swap their schedules so that their release times and start times in $\mathcal{O}_2$ are in the same order. That is, for all i, the job with the i-th smallest release time will take up the schedule of the job with the i-th smallest start time; note that the i-th smallest start time can never be earlier than the i-th smallest release time. Thus, $\mathcal{O}_2$ is also a valid schedule for $\mathcal{J}_1$.

Next, we modify $\mathcal{J}_1$ to $\mathcal{J}_2$, by replacing the release time of each job j with its start time in $\mathcal{O}_2$. Note that the release time of j can only be delayed (and never gets advanced). Any schedule for $\mathcal{J}_2$ (including $\mathcal{O}_2$) is also a valid schedule for $\mathcal{J}_1$.

By construction, $\mathcal{O}_2$ is an immediate-start schedule for $\mathcal{J}_2$. Next, we analyze the relationship between $\mathcal{O}_1$ and $\mathcal{O}_2$. To ease the discussion, we add a subscript $\mathcal{J}$ to the notations F, E, and G to denote that the job set under concern is $\mathcal{J}$.

$\mathcal{O}_1$ and $\mathcal{O}_2$ incur the same flow time plus energy for $\mathcal{J}_1$. Since $\mathcal{O}_1$ and $\mathcal{O}_2$ use the same speed at any time, $E_{\mathcal{J}_1}(\mathcal{O}_1) = E_{\mathcal{J}_1}(\mathcal{O}_2)$. Furthermore, at any time, $\mathcal{O}_1$ completes a job if and only if $\mathcal{O}_2$ completes a (possibly different) job, and thus $\mathcal{O}_1$ and $\mathcal{O}_2$ always have the same number of unfinished jobs. This means that $F_{\mathcal{J}_1}(\mathcal{O}_1) = F_{\mathcal{J}_1}(\mathcal{O}_2)$ and $G_{\mathcal{J}_1}(\mathcal{O}_1) = G_{\mathcal{J}_1}(\mathcal{O}_2)$.

$\mathcal{O}_2$ is optimal for $\mathcal{J}_2$ (in terms of flow time plus energy). Suppose on the contrary that there is a schedule $\mathcal{O}'$ for $\mathcal{J}_2$ with $G_{\mathcal{J}_2}(\mathcal{O}') < G_{\mathcal{J}_2}(\mathcal{O}_2)$. Any schedule of $\mathcal{J}_2$, including $\mathcal{O}'$ and $\mathcal{O}_2$, is also a valid schedule for $\mathcal{J}_1$. Note that $E_{\mathcal{J}_1}(\mathcal{O}') = E_{\mathcal{J}_2}(\mathcal{O}')$, and $F_{\mathcal{J}_1}(\mathcal{O}') = F_{\mathcal{J}_2}(\mathcal{O}')+d$, where d is the total delay of release times of all jobs in $\mathcal{J}_2$ (when comparing with $\mathcal{J}_1$). Therefore, $G_{\mathcal{J}_1}(\mathcal{O}') = G_{\mathcal{J}_2}(\mathcal{O}') + d$, and similarly for $\mathcal{O}_2$. Thus, if $G_{\mathcal{J}_2}(\mathcal{O}') < G_{\mathcal{J}_2}(\mathcal{O}_2)$, then

$$\begin{aligned} G_{\mathcal{J}_1}(\mathcal{O}') &= G_{\mathcal{J}_2}(\mathcal{O}') + d \\ &< G_{\mathcal{J}_2}(\mathcal{O}_2) + d = G_{\mathcal{J}_1}(\mathcal{O}_2) = G_{\mathcal{J}_1}(\mathcal{O}_1) \ . \end{aligned}$$

This contradicts the optimality of $\mathcal{O}_1$ for $\mathcal{J}_1$.

CRR-dispatching schedules preserve performance. Consider any CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}_2$ satisfying $G_{\mathcal{J}_2}(\mathcal{S}) \le \beta G_{\mathcal{J}_2}(\mathcal{O}_2)$, for some $\beta \ge 1$. By definition, jobs of the same class are also of same size and have the same order of release times in $\mathcal{J}_1$ and $\mathcal{J}_2$. Therefore, $\mathcal{S}$ is also an CRR(λ)-dispatching schedule for $\mathcal{J}_1$. For total flow time plus energy,

$$\begin{aligned} G_{\mathcal{J}_1}(\mathcal{S}) &= G_{\mathcal{J}_2}(\mathcal{S}) + d \le \beta G_{\mathcal{J}_2}(\mathcal{O}_2) + d \\ &\le \beta(G_{\mathcal{J}_2}(\mathcal{O}_2) + d) = \beta G_{\mathcal{J}_1}(\mathcal{O}_2) = \beta G_{\mathcal{J}_1}(\mathcal{O}_1) \ . \end{aligned}$$

Thus the lemma follows. □

In the rest of this paper, we exploit the fact that an optimal schedule runs a job at the same speed throughout its lifespan. This is due to the convexity of the power function s^α. Also, without loss of generality, at any time an optimal

schedule never runs a job at speed less than the global critical speed, defined as $1/(\alpha-1)^{1/\alpha}$ [1], and the maximum speed T is at least the global critical speed (see Appendix for justifications).

4. COMPARING AGAINST NON-MIGRATORY SCHEDULES

This section shows how to construct a CRR-dispatching schedule from the optimal non-migratory schedule, provided that it is immediate-start. The main result is stated in the following lemma.

LEMMA 7. *Given a power-of-$(1+\lambda)$ job sequence $\mathcal{J}$ with an optimal non-migratory schedule $\mathcal{N}^*$ that is immediate-start, we can construct a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ such that $G(\mathcal{S}) \le 2((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon)$ $G(\mathcal{N}^*)$, and max-speed$(\mathcal{S}) \le (1+\epsilon) \times$ max-speed$(\mathcal{N}^*)$.*

The above lemma, together with Lemma 5 about power-of-$(1+\lambda)$ jobs and Lemma 6 about immediate schedules, would immediately give a way to construct CRR-dispatching schedules from optimal non-migratory schedules, with the desired flow time and energy as stated in Theorem 2(i). Details are omitted.

Below we describe a property of immediate-start, optimal non-migratory schedules, and then present an algorithm for constructing CRR-dispatching schedules. The analysis of its flow time and energy and the proof of Lemma 7 are further divided into three subsections.

First of all, we observe that any immediate-start, optimal non-migratory schedule $\mathcal{N}^*$ for a power-of-$(1+\lambda)$ job sequence $\mathcal{J}$ satisfies the property that at any time, for each job size, there are at most m jobs started but not completed. This is because jobs of the same size must work in a First-Come-First-Serve manner, otherwise we can shuffle the execution order to First-Come-First-Serve and reduce the total flow time, so the schedule is not optimal.

PROPERTY 1. *Consider any power-of-$(1+\lambda)$ job sequence $\mathcal{J}$, and any immediate-start, optimal non-migratory schedule for $\mathcal{J}$ on $m \ge 2$ processors. At any time, for each job size, there are at most m jobs started but not completed.*

Given the schedule $\mathcal{N}^*$ for $\mathcal{J}$, the algorithm below constructs a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ with a mild increase in flow time and energy. The ordering of job execution in $\mathcal{S}$ could be very different from $\mathcal{N}^*$. Roughly speaking, $\mathcal{S}$ only makes reference to the speed used by $\mathcal{N}^*$. Recall that in $\mathcal{N}^*$, a job is run at the same speed throughout its lifespan. For any job, $\mathcal{S}$ determines its speed as the average of a certain subset of $2m$ jobs of the same size. These $2m$ jobs are chosen according to the release times. Details are as follows. Note that the processors are numbered from 0 to $m-1$.

Algorithm 1. The construction runs in multiple rounds, from the smallest job size to the largest. Let $\mathcal{S}_0$ denote the intermediate schedule, which is initially empty and eventually becomes $\mathcal{S}$. We modify $\mathcal{S}_0$ in each round to include more jobs. In the round for size p, suppose that $\mathcal{J}$ contains n jobs $\{j_1, j_2, \ldots, j_n\}$ of size p, arranged in increasing order of release times. It is convenient to define $j_{n+1} = j_1$, $j_{n+2} = j_2$, etc. For $i = 1$ to n, let x_i be the average speed in $\mathcal{N}^*$ of the fastest m jobs among the following $2m$ jobs: $j_i, j_{i+1}, \ldots, j_{i+2m-1}$. We modify $\mathcal{S}_0$ by adding a schedule for j_i in processor ($i \bmod m$): it can start as early as at its release time, runs at constant speed $(1+\epsilon)x_i$, and occupies the earliest possible times, while avoiding times already committed to earlier jobs for processor ($i \bmod m$).

We claim that the schedule $\mathcal{S}$ produced by Algorithm 1 satisfies Lemma 7. In Section 4.1, we analyze the energy usage. The analysis of flow time is based on an upper bound on the execution time $\mathcal{S}$ spends on jobs of certain classes within a period of time. This upper bound is stated and proved in Section 4.2. With this upper bound, we can analyze the flow time in Section 4.3.

4.1 Speed and energy

We first note that in $\mathcal{S}$, the speed of a job is $(1+\epsilon)$ times the average speed of m jobs in $\mathcal{N}^*$, so max-speed$(\mathcal{S}) \le (1+\epsilon) \times$ max-speed$(\mathcal{N}^*)$. Next, we consider the energy.

LEMMA 8. *The energy used by $\mathcal{S}$ produced by Algorithm 1 is at most $2(1+\epsilon)^{\alpha-1}G(\mathcal{N}^*)$.*

PROOF. Consider m jobs of the same size being run at different constant speed, and let x be their average speed. Energy is a function of speed to the power of $\alpha - 1 \ge 1$, which is convex. Running a job of the same size at speed x incurs energy at most $1/m$ times the total energy for running these m jobs. If we further increase the speed to $(1+\epsilon)x$, the power increases by a factor of $(1+\epsilon)^\alpha$, and the running time decreases by a factor of $(1+\epsilon)$. Thus, the energy usage increases by a factor of $(1+\epsilon)^{\alpha-1}$. In $\mathcal{S}$, running a job at $(1+\epsilon)$ times the average speed of m jobs in $\mathcal{N}^*$ requires no more energy than $(1+\epsilon)^{\alpha-1}/m$ times the sum of the energy usage of those m jobs in $\mathcal{N}^*$.

To bound $E(\mathcal{S})$, we use a simple charging scheme: for a job j in $\mathcal{S}$, we charge to every one of the m jobs j' chosen for determining the speed of j in Algorithm 1; the amount to be charged is $1/m$ times of the energy usage of j' in $\mathcal{N}^*$. By Algorithm 1, each job can be charged by at most $2m$ jobs. Thus,

$$\begin{aligned} E(\mathcal{S}) &\le \frac{(1+\epsilon)^{\alpha-1}}{m} 2mE(\mathcal{N}^*) \\ &\le 2(1+\epsilon)^{\alpha-1}E(\mathcal{N}^*) \\ &\le 2(1+\epsilon)^{\alpha-1}G(\mathcal{N}^*) \ . \end{aligned}$$

□

4.2 Upper bound on job execution time of $\mathcal{S}$

To analyze the flow time of a job in $\mathcal{S}$, we attempt to upper bound the execution time of other jobs dispatched to the same processor during its lifespan. The lemmas below look technical, yet the key observation is quite simple—For any processor z, if we consider all jobs that $\mathcal{S}$ dispatches to z during an interval I, excluding the last two jobs of each class (size), their total execution time is at most $\ell/(1+\epsilon)$, where ℓ is the length of I.

Consider any job $h_0 \in \mathcal{J}$. Let $h_1, h_2, \ldots, h_n$ be all the jobs in $\mathcal{J}$ such that $r(h_0) \le r(h_1) \le \cdots \le r(h_n)$ and they have the same size as h_0. Suppose that $n \ge im$ for some $i \ge 2$. We focus on two sets of jobs: $\{h_0, h_1, \ldots, h_{im-1}\}$ and $\{h_0, h_m, h_{2m}, \ldots, h_{(i-2)m}\}$. The latter contains jobs dispatched to the same processor as h_0. Lemma 9 below gives an upper bound of the execution time of $\mathcal{S}$ for $\{h_0, h_m, h_{2m}, \ldots, h_{(i-2)m}\}$ with respect to $\mathcal{N}^*$. This lemma stems from the fact that $\mathcal{N}^*$ is immediate-start.

LEMMA 9. *For any job h_0 and $i \geq 2$, suppose h_{im} exists. Let t be the execution time of $\mathcal{N}^*$ for the jobs $h_0, h_1, \ldots, h_{im-1}$ during the interval $[r(h_0), r(h_{im})]$. Then in the entire schedule of $\mathcal{S}$, the total execution time of the jobs $h_0, h_m, \ldots, h_{(i-2)m}$ is at most $t/m(1+\epsilon)$.*

PROOF. Since $\mathcal{N}^*$ is immediate-start, jobs $h_0, \ldots, h_{im-1}$ each starts within the interval $[r(h_0), r(h_{im})]$. At the time $r(h_{im})$, at most m jobs among these im jobs are not yet completed (Property 1), or equivalently, $\mathcal{N}^*$ has completed at least $(i-1)m$ jobs. Let Δ denote a set of any $(i-1)m$ such completed jobs. Based on release times, we partition Δ accordingly into $i-1$ subsets Δ_0, Δ_1, ..., Δ_{i-2}, each of size exactly m. Δ_0 contains the m jobs with smallest release times in Δ, Δ_1 contains jobs with the next m smallest release times in Δ, etc.

Since Δ misses out only m jobs in $\{h_0, h_1, \ldots, h_{im-1}\}$, each Δ_u, for $u \in \{0, \ldots, i-2\}$, is a subset of the $2m$ jobs $\{h_{um}, h_{um+1}, \ldots, h_{um+2m-1}\}$. Because the speed used by $\mathcal{S}$ for h_{um} is $(1+\epsilon)$ times the average speed of the m fastest jobs in $h_{um}, h_{um+1}, \ldots, h_{um+2m-1}$ used by $\mathcal{N}^*$, which is faster than $1+\epsilon$ times the average speed of Δ_u in $\mathcal{N}^*$, it follows that the execution time of h_{um} in $\mathcal{S}$ is at most $1/m(1+\epsilon)$ times the total execution time of Δ_u in $\mathcal{N}^*$. Summing over all $u \in \{0, \ldots, i-2\}$, the execution time of $\mathcal{S}$ for $h_0, h_m, \ldots, h_{(i-2)m}$ is no more than $1/m(1+\epsilon)$ times the total execution time of Δ in $\mathcal{N}^*$. In $\mathcal{N}^*$, Δ is only executed during $[r(h_0), r(h_{im})]$, and the lemma follows. □

Below is the main result of this section (to be used for analyzing the flow time of $\mathcal{S}$ later).

LEMMA 10. *Consider any k and any time interval I of length ℓ. For jobs of size at most $(1+\lambda)^k$ that are released during I, the total execution time of any processor in $\mathcal{S}$ for these jobs is at most $\ell/(1+\epsilon) + 2(1+\lambda)^k \cdot (\alpha-1)^{1/\alpha}/\lambda(1+\epsilon)$.*

PROOF. Consider a particular $k' \leq k$. Let y be the execution time over all processors that $\mathcal{N}^*$ uses for jobs of size $(1+\lambda)^{k'}$ during the interval I. Consider a particular processor z in $\mathcal{S}$; suppose that $\mathcal{S}$ dispatches i jobs of size $(1+\lambda)^{k'}$ to processor z during I, and denote these i jobs as $\mathcal{J}' = \{h'_0, h'_m, \ldots, h'_{(i-1)m}\}$, arranged in the order of their release times. We claim that the execution time of processor z in $\mathcal{S}$ for these i jobs is at most $y/m(1+\epsilon)$ plus the execution time of $\mathcal{S}$ for the last two jobs of $\mathcal{J}'$. This is obvious if $\mathcal{J}'$ contains two or fewer jobs. It remains to consider the case when $\mathcal{J}'$ has $i \geq 3$ jobs. By Lemma 9, if t is the execution time of $\mathcal{N}^*$ for $h'_0, h'_1, \ldots, h'_{(i-1)m-1}$ during $[r(h'_0), r(h'_{(i-1)m})]$, then $\mathcal{S}$ uses no more than $t/m(1+\epsilon)$ time to execute $h'_0, h'_m, \ldots, h'_{(i-3)m}$. The claim then follows by noticing that $t \leq y$, and we only have two jobs $h'_{(i-2)m}$ and $h'_{(i-1)m}$ not being counted.

Now we sum over all $k' \leq k$ the upper bound of these flow times, i.e., $y/m(1+\epsilon)$ plus the execution time of $\mathcal{S}$ for the last 2 jobs in $\mathcal{J}'$. The sum of the first part is $\sum y/m(1+\epsilon)$. Note that $\sum y$ is the execution time of $\mathcal{N}^*$ during I, so $\sum y \leq m|I| = m\ell$, and the sum of the first part is

$$\frac{\sum y}{m(1+\epsilon)} \leq \frac{\ell}{1+\epsilon} \ .$$

The sum of the second part is over at most 2 jobs for each k'. Recall that the speed used by $\mathcal{N}^*$ is at least the global critical speed $1/(\alpha-1)^{1/\alpha}$, and the speed used by $\mathcal{S}$ is $(1+\epsilon)$ times the average of some job speeds in $\mathcal{N}^*$. Thus the speed used by $\mathcal{S}$ for any job is at least $(1+\epsilon)/(\alpha-1)^{1/\alpha}$, and the execution time of each job of size $(1+\lambda)^{k'}$ is at most $(1+\lambda)^{k'}(\alpha-1)^{1/\alpha}/(1+\epsilon)$. Summing over all k' the execution time for these jobs, we have

$$\sum_{k'=0}^{k} \frac{2(1+\lambda)^{k'}(\alpha-1)^{1/\alpha}}{1+\epsilon} < \frac{2(1+\lambda)^{k+1}(\alpha-1)^{1/\alpha}}{\lambda(1+\epsilon)} \ .$$

The lemma follows by summing the two parts. □

4.3 Flow time

In this section we show that the flow time of each job in $\mathcal{S}$ is $O(1/\epsilon)$ times of its job size, which implies that the total flow time is $O(1/\epsilon)G(\mathcal{N}^*)$. Together with Lemma 8, Lemma 7 can be proved. We first bound the flow time of a job of a particular job size in $\mathcal{S}$, making use of Lemma 10.

LEMMA 11. *In $\mathcal{S}$, the flow time of a job of size $(1+\lambda)^k$ is at most $2(2+\lambda)(1+\lambda)^k(\alpha-1)^{1/\alpha}/\lambda\epsilon$.*

PROOF. Consider a job j of size $(1+\lambda)^k$ that is scheduled on some processor z in $\mathcal{S}$. Let $r = r(j)$, and f be the flow time of j in $\mathcal{S}$, i.e., j completes at time $r+f$. To determine f, we focus on the scheduling of processor z in the intermediate schedule $\mathcal{S}_0$ immediate after Algorithm 1 has scheduled j. Note that f is due to jobs that have been executed in $\mathcal{S}$ during $[r, r+f]$. They can be partitioned into two subsets: $\mathcal{J}_1$ for jobs released at or before r, and $\mathcal{J}_2$ for jobs released during $(r, r+f]$. Let f_1 and f_2 be the contribution on f by $\mathcal{J}_1$ and $\mathcal{J}_2$, respectively, i.e., $f = f_1 + f_2$.

We first consider $\mathcal{J}_1$. Let t be the last time $< r$ such that processor z is idle right before t in $\mathcal{S}_0$. Thus all jobs executed by processor z at or after t, and hence all jobs in $\mathcal{J}_1$, must be released at or after t. By Lemma 10, the execution time of processor z for jobs in $\mathcal{J}_1$ is no more than $(r-t)/(1+\epsilon) + [2(1+\lambda)^{k+1}(\alpha-1)^{1/\alpha}/\lambda(1+\epsilon)]$. Since processor z is busy throughout $[t, r)$, the amount of execution time for jobs in $\mathcal{J}_1$ remaining at r is at most

$$\frac{r-t}{1+\epsilon} + \frac{2(1+\lambda)^{k+1}(\alpha-1)^{1/\alpha}}{\lambda(1+\epsilon)} - (r-t) \leq \frac{2(1+\lambda)^{k+1}(\alpha-1)^{1/\alpha}}{\lambda(1+\epsilon)} \ .$$

This implies $f_1 \leq 2(1+\lambda)^{k+1}(\alpha-1)^{1/\alpha}/\lambda(1+\epsilon)$.

Next we consider $\mathcal{J}_2$. Since Algorithm 1 schedules jobs from the smallest to the largest size, jobs in $\mathcal{J}_2$ are of size at most $(1+\lambda)^{k-1}$. We apply Lemma 10 to the interval $[r, r+f]$ for jobs of size at most $(1+\lambda)^{k-1}$. The execution time of processor z for jobs in $\mathcal{J}_2$, i.e., f_2, is no more than

$$\frac{f}{1+\epsilon} + \frac{2(1+\lambda)^{k}(\alpha-1)^{1/\alpha}}{\lambda(1+\epsilon)} \ .$$

Then we have

$$f = f_1 + f_2 \leq \frac{2(2+\lambda)(1+\lambda)^k(\alpha-1)^{1/\alpha}}{\lambda(1+\epsilon)} + \frac{f}{1+\epsilon} \ ,$$

immediately implying $f \leq 2(2+\lambda)(1+\lambda)^k(\alpha-1)^{1/\alpha}/\lambda\epsilon$. □

Summing over all jobs and recalling that $G(\mathcal{N}^*) \geq (\alpha/(\alpha-1)^{1-1/\alpha})p(\mathcal{J})$ (see Lemma 1), we have the following corollary. Then Lemma 7 is a direct consequence of Lemma 8 and Corollary 12.

COROLLARY 12. *The total flow time incurred by $\mathcal{S}$ produced by Algorithm 1 is at most $(2(1-1/\alpha)(2+\lambda)/\lambda\epsilon)G(\mathcal{N}^*)$.*

5. COMPARING AGAINST MIGRATORY SCHEDULES

The construction algorithm given in the previous section can be applied to handle optimal migratory schedule, except that Property 1 no longer holds for any optimal migratory schedule. Nevertheless, the following weaker property is satisfied by some (rather than all) optimal schedule, which we show later in this section.

PROPERTY 2. *For any job sequence $\mathcal{J}$, there is an optimal migratory schedule $\mathcal{S}^*$ such that at any time, there are less than $4m$ jobs of the same size started but not yet completed.*

This schedule may not be immediately-start (indeed, for most job sequences there is no immediately-start optimal schedules). However, it is easy to check that if we apply the construction of Lemma 6 to a schedule satisfying Property 2, the resulting schedule also satisfies Property 2 (since the only manipulation done in that construction is to swap the scheduling of pairs of jobs completely). As a result, we can extend Lemma 6 as follows.

LEMMA 13. *Consider a power-of-$(1+\lambda)$ job sequence $\mathcal{J}_1$ and an optimal migratory schedule $\mathcal{O}_1$ for $\mathcal{J}_1$. Then there is a power-of-$(1+\lambda)$ job sequence $\mathcal{J}_2$ and an immediate-start, optimal schedule $\mathcal{O}_2$ for $\mathcal{J}_2$ that satisfies Property 2, such that any CRR(λ)-dispatching schedule $\mathcal{S}_2$ for $\mathcal{J}_2$ defines a CRR(λ)-dispatching schedule $\mathcal{S}_1$ for $\mathcal{J}_1$, and if $G(\mathcal{S}_2) \le \beta G(\mathcal{O}_2)$ for some $\beta \ge 1$, then $G(\mathcal{S}_1) \le \beta G(\mathcal{O}_1)$.*

We can now make use of Algorithm 1 to construct a CRR-dispatching schedule. Note that the new property causes a minor change to Algorithm 1, namely, for each job j_i, we determine the speed x by considering the $5m$ jobs j_i, j_{i+1}, $\ldots$, j_{i+5m-1}, instead of the $2m$ jobs j_i, j_{i+1}, $\ldots$, j_{i+2m-1}. The following is the main result of this section, which is a straightforward adaptation of Section 4 to the migratory case based on Property 2. Details are left in the full paper.

LEMMA 14. *Given a power-of-$(1+\lambda)$ job sequence $\mathcal{J}$ with an optimal migratory schedule $\mathcal{S}^*$ that is immediate-start and satisfies Property 2, we can construct a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ with $G(\mathcal{S}) \le 5((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon)G(\mathcal{S}^*)$ and max-speed$(\mathcal{S}) \le (1+\epsilon) \times$ max-speed$(\mathcal{S}^*)$.*

The rest of this section is devoted to proving Property 2. For comparing against non-migratory schedules, Property 1 obviously holds for an arbitrary schedule. On the contrary, Property 2 is not obvious for any migratory optimal schedules. We show that Property 2 is true for a *lazy-start* optimal migratory schedule, defined as follows: Given a schedule $\mathcal{S}$, we define its "start time sequence" to be the sequence of start time of each job, sorted in the order of time. Among all optimal migratory schedules (which may or may not be immediate-start), a lazy-start optimal schedule is the one with lexicographically maximum start time sequence. Such a schedule has the following property.

LEMMA 15. *In a lazy-start optimal schedule, suppose a job j_1 is started at time t, when another job j_2 of the same size has been started, has not completed, and is not running at t. Then after t, j_1 runs whenever j_2 runs.*

PROOF. Suppose the contrary, and let t' be the first time after t that j_2 runs but not j_1. Let p_0 be the amount of work processed for j_1 during $[t, t']$ when j_2 is not running. We separate the analysis into three cases, each arriving at a contradiction.

Case 1: j_1 is not yet completed by t'. We can exchange some work of j_2 done starting from t' with those of j_1 done starting from t, without changing processor speed at any time. The start time of j_1 is thus delayed without changing the start times of other jobs or increasing the energy or flow time, so the original schedule is not lazy-start.

Case 2: j_1 is completed by t', and the amount of work processed for j_2 after t' is at most p_0. We can exchange all work of j_2 after t' with some work of j_1 starting from t. The completion times of j_1 and j_2 are exchanged, but the total flow time and energy is preserved. The start time of j_1 is delayed without changing the start times of other jobs, so the original schedule is not lazy-start.

Case 3: j_1 is completed by t', and the amount of work processed for j_2 after t' is more than p_0. These conditions imply that there must be some work processed for j_1 at other times, i.e., when both j_1 and j_2 are running. Furthermore, j_2 must be running slower than j_1 during this period, otherwise the total amount of work processed for j_2 would be larger than the size of j_1, so the two jobs cannot be of the same size. Since jobs run at constant speed in optimal schedules, the speed of j_1 is higher than the speed of j_2.

Note that j_1 lags behind j_2 at t but is ahead of j_2 at t'. So there must be a time $t_0 \in (t', t)$ such that j_1 and j_2 has been processed for the same amount of work. Exchange the scheduling of j_1 and j_2 after t_0 gives a schedule with the completion time of j_1 and j_2 exchanged, while the energy consumption and flow time remain the same. But now j_1 and j_2 are not running at constant speed, so the schedule is not optimal. □

PROOF OF PROPERTY 2. Suppose, for the sake of contradiction, that at some time, $4m$ jobs have been started but not yet completed. Consider these $4m$ jobs. When the $(m+i)$-th job j is being started, at least i jobs previously started must idle. For each such idling job j', Lemma 15 dictates that after $r(j)$, whenever j' runs, j must also be running. We say *j precedes j'*. Since there are $4m$ jobs, there are $1+2+\cdots+3m = \frac{3}{2}m(3m+1)$ such relations. So some job j_0 must be preceded by at least $\frac{3}{2}m(3m+1)/4m > m$ other jobs. After all these other jobs are released, they must all run whenever j_0 runs, contradicting that there are only m processors. The property follows. □

6. SCHEDULING FOR FIXED-SPEED PROCESSORS

In this section, we consider traditional flow time scheduling where processors always run at a fixed speed. Without loss of generality, we assume the speed is fixed at 1. We present an online algorithm CRR$_\epsilon$-SRPT, which produces a non-migratory schedule for $m \ge 2$ processors. We show that CRR$_\epsilon$-SRPT, when compared with the optimal non-migratory or migratory offline schedule for minimizing flow

time, has a competitive ratio of one or even any constant arbitrarily smaller than one, when using sufficiently fast processors.

> **Algorithm CRR$_\epsilon$-SRPT.** Jobs are dispatched to the m processors with the CRR(ϵ)-dispatching policy. Jobs in each processor are scheduled independently using SRPT: at any time, run the job with the least remaining work.

Note that the objective function is total flow time, instead of flow time plus energy. To analyze CRR$_\epsilon$-SRPT, we adapt Theorem 2 (in Section 2) as follows.

THEOREM 16. *Given a job set $\mathcal{J}$, let $\mathcal{N}^*$ be an optimal non-migratory schedule of $\mathcal{J}$, and let $\mathcal{S}^*$ be an optimal migratory schedule of $\mathcal{J}$. Then for any $\lambda, \epsilon > 0$,*

i. there is a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ such that $F(\mathcal{S}) \leq (2(2+\lambda)/\lambda\epsilon)F(\mathcal{N}^*)$, and max-speed$(\mathcal{S}) \leq (1+\lambda)(1+\epsilon) \times$ max-speed$(\mathcal{N}^*)$; and

ii. there is a CRR(λ)-dispatching schedule $\mathcal{S}'$ for $\mathcal{J}$ such that $F(\mathcal{S}) \leq (5(2+\lambda)/\lambda\epsilon)F(\mathcal{S}^*)$, and max-speed$(\mathcal{S}') \leq (1+\lambda)(1+\epsilon) \times$ max-speed$(\mathcal{S}^*)$.

To prove Theorem 16, we first show that scheduling fixed-speed processors to minimize flow time is actually a special case of scheduling variable speed processors to minimize flow time plus energy. Consider an optimal schedule for minimizing flow time plus energy when $\alpha = 2$ and $T = 1$. Recall that without loss of generality, the optimal schedule always runs at a speed at least the global critical speed, $1/(\alpha-1)^{1/\alpha} = 1$, which is also the maximum speed bound. Thus the processor runs at fixed speed $T = 1$, spending a constant amount of energy. In this particular case, to minimize the total flow time plus energy, the optimal schedule must thus minimize the total flow time. We further note that running Algorithm 1 on such a schedule produces a schedule that always uses speed $(1+\epsilon)$.

Since scheduling of fixed-speed processors is simply a special case of scheduling of variable-speed processors, all arguments in Section 4 and 5 works unchanged. It suffices to adapt those theorems and lemmas in the previous sections so that the comparison is made against the total flow time, rather than the total flow time plus energy, of the optimal schedule. For comparing against the non-migratory optimal schedule, we adapt Lemma 7 as follows.

LEMMA 17. *Given a power-of-$(1+\lambda)$ job sequence $\mathcal{J}$ with an optimal non-migratory schedule $\mathcal{N}^*$ that is immediate-start, we can construct a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ such that $F(\mathcal{S}) \leq \frac{2(2+\lambda)}{\lambda\epsilon}F(\mathcal{N}^*)$ and max-speed$(\mathcal{S}) \leq (1+\epsilon) \times$ max-speed$(\mathcal{N}^*)$.*

PROOF. We focus on the analysis about flow time in Section 4.3. Since $\alpha = 2$, Lemma 11 shows that in schedule $\mathcal{S}$, the flow time of a job of size $(1+\lambda)^k$ is at most $2(2+\lambda)(1+\lambda)^k/\lambda\epsilon$. Summing over all jobs in $\mathcal{J}$, the total flow time is $F(\mathcal{S}) \leq 2(2+\lambda)p(\mathcal{J})/\lambda\epsilon$. To relate $F(\mathcal{S})$ and $F(\mathcal{N}^*)$, we note that $\mathcal{N}^*$ always runs at speed 1, so the flow time of a job j is at least $p(j)$. Summing over all jobs, we have $F(\mathcal{N}^*) \geq p(\mathcal{J})$. Thus $F(\mathcal{S}) \leq (2(2+\lambda)/\lambda\epsilon)F(\mathcal{N}^*)$. □

For comparing against the optimal migratory schedule, we can use the same techniques to adapt Lemma 14 in Section 5. The proof is left in the full paper.

LEMMA 18. *Given a power-of-$(1+\lambda)$ job sequence $\mathcal{J}$ with an optimal migratory schedule $\mathcal{S}^*$ that is immediate-start and satisfies Property 2, we can construct a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ with $F(\mathcal{S}) \leq (5(2+\lambda)/\lambda\epsilon)F(\mathcal{S}^*)$, and max-speed$(\mathcal{S}) \leq (1+\epsilon) \times$ max-speed$(\mathcal{S}^*)$.*

Now consider any job set $\mathcal{J}_0$ (which may not be a power-of-$(1+\lambda)$ job set). The analysis in Section 3 can be adapted. Then Lemmas 17 and 18 lead to Theorem 16.

Setting $\lambda = \epsilon$ in Theorem 16 and using the fact that SRPT optimizes total flow time on a single processor, the performance of CRR$_\epsilon$-SRPT against the optimal non-migratory or migratory algorithm is easily shown as follows.

COROLLARY 19. *For any $\epsilon > 0$, CRR$_\epsilon$-SRPT achieves the following performance for minimizing total flow time.*

i. Against the non-migratory optimal schedule: given processors of speed $(1+\epsilon)^2$, CRR$_\epsilon$-SRPT is $(2(2+\epsilon)/\epsilon^2)$-competitive.

ii. Against the migratory optimal schedule: given processors of speed $(1+\epsilon)^2$, CRR$_\epsilon$-SRPT is $(5(2+\epsilon)/\epsilon^2)$-competitive.

By setting $s = (1+\epsilon)^2$, the above corollary is equivalent to that for any $s > 1$, CRR$_\epsilon$-SRPT is s-speed $\frac{2(\sqrt{s}+1)}{(\sqrt{s}-1)^2}$-competitive against the non-migratory optimal schedule, and s-speed $\frac{5(\sqrt{s}+1)}{(\sqrt{s}-1)^2}$-competitive against the migratory optimal schedule, respectively.

7. REFERENCES

[1] S. Albers, and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Alg.*, 3(4):49, 2007 .

[2] S. Albers, F. Muller, and S. Schmelzer. Speed Scaling on parallel processors. In *Proc. SPAA*, pages 289–298, 2007.

[3] J. Augustine, S. Irani, and C. Swany. Optimal power-down strategies. In *Proc. FOCS*, pages 530–539, 2004.

[4] N. Avrahami, and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proc. SPAA*, pages 11–18, 2003.

[5] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. *SIAM J. on Comput.*, 31(5):1370–1382, 2002.

[6] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. To appear in *Proc. ICALP*, 2008.

[7] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proc. FOCS*, pages 520-529, 2004.

[8] N. Bansal, K. Pruhs and C. Stein. Speed scaling for weighted flow time. In *Proc. SODA*, pages 805–813, 2007.

[9] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.D. Wellman, V. Zyuban, M. Gupta, and P.W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[10] D. P. Bunde. Power-aware scheduling for makespan and flow. In *Proc. SPAA*, pages 190–196, 2006.

[11] H. L. Chan, W. T. Chan, T. W. Lam, L. K. Lee, K. S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *Proc. SODA*, pages 795–804, 2007.

[12] H. L. Chan, T. W. Lam and K. K. To. Nonmigratory online deadline scheduling on multiprocessors. *SIAM J. on Comput.*, 34(3):669–682, 2005.

[13] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proc. STOC*, pages 84–93, 2001.

[14] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with ϵ resource augmentation. In *Proc. STOC*, pages 363–372, 2004.

[15] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *Proc. OSDI*, pages 73–86, 2000.

[16] S. Irani, R. K. Gupta, and S. Shukla. Algorithms for power savings. In *Proc. SODA*, pages 37–46, 2003.

[17] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 32(2):63–76, 2005.

[18] B. Kalyanasundaram and K. Pruhs. Eliminating migration in multi-processor scheduling. *J. Alg.*, 38:2–24, 2001.

[19] S. Leonardi, and D. Raz. Approximating total flow time on parallel machines. In *Proc. STOC*, pages 110–119, 1997.

[20] M. Li, B.J. Liu, and F.F. Yao. Min-energy voltage allocations for tree-structured tasks. In *Proc. COCOON*, pages 283–296, 2005.

[21] J. McCullough and E. Torng. SRPT optimally utilizes faster machines to minimize flow time. In *Proc. SODA*, pages 350–358, 2004.

[22] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[23] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *STOC*, pages 140–149, 1997.

[24] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. SOSP*, pages 89–102, 2001.

[25] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 15-1–15-41. CRC Press, 2004.

[26] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *Proc. WAOA*, pages 307–319, 2005.

[27] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Proc. SWAT*, pages 14–25, 2004.

[28] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. OSDI*, pages 13–23, 1994.

[29] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. FOCS*, pages 374–382, 1995.

Appendix. Global critical speed

In this appendix we justify the assumptions that at any time the optimal schedules $\mathcal{N}^*$ and $\mathcal{S}^*$ never run a job at speed less than the global critical speed $1/(\alpha-1)^{1/\alpha}$, and the maximum speed T is at least the global critical speed. The assumption stems from an observation (Lemma 20) that if necessary, a multi-processor schedule can be transformed without increasing the flow time plus energy so that it never runs a job j at speed less than the global critical speed.

LEMMA 20. *Given any m-processor schedule $\mathcal{S}$ for a job set $\mathcal{J}$, we can construct an m-processor schedule $\mathcal{S}'$ for $\mathcal{J}$ such that $\mathcal{S}'$ never runs a job at speed less than the global critical speed and $G(\mathcal{S}') \leq G(\mathcal{S})$. Moreover, $\mathcal{S}'$ needs migration if and only if $\mathcal{S}$ does; and max-speed($\mathcal{S}'$) is at most* $\max\{\textit{max-speed}(\mathcal{S}), 1/(\alpha-1)^{1/\alpha}\}$.

Albers and Fujiwara [1] observed that when scheduling a single job j on a single processor for minimizing total flow time plus energy, j should be executed at the global critical speed, i.e., $1/(\alpha-1)^{1/\alpha}$.

LEMMA 21. *[1] At any time after a job j has been run on a processor for a while, suppose that we want to further execute j for another $x > 0$ units of work and minimize the flow time plus energy incurred to this period. The optimal strategy is to let the processor always run at the global critical speed.*

PROOF OF LEMMA 20. Assume that there is a time interval I in $\mathcal{S}$ during which a processor i is running a job j below the global critical speed. If $\mathcal{S}$ needs migration, we transform $\mathcal{S}$ to a migratory schedule $\mathcal{S}_1$ of $\mathcal{J}$ such that job j is always scheduled in processor i. This can be done by swapping the schedules of processor i and other processors for different time intervals. If $\mathcal{S}$ does not need migration, job j is entirely scheduled in processor i and $\mathcal{S}_1$ is simply $\mathcal{S}$. In both cases, $G(\mathcal{S}_1) = G(\mathcal{S})$.

We can then improve $G(\mathcal{S}_1)$ by modifying the schedule of processor i as follows. Let x be the amount of work of j processed during I on processor i. First, we schedule this amount of work of j at the global critical speed. Note that the time required is shortened. Then we move the remaining schedule of j backward to fill up the time shortened. By Lemma 21, the flow time plus energy for j is preserved. Other jobs in $\mathcal{J}$ are left intact. To obtain the schedule $\mathcal{S}'$, we repeat this process to eliminate all such intervals I. □

Assumption on T. We assume that the maximum speed T is at least the global critical speed. Otherwise, any multi-processor schedule including the optimal one would always run a job at the maximum speed. It is because when running a job below the global critical speed, the slower the speed, the more total flow time plus energy is incurred. In other words, the problem is reduced to minimizing flow time alone.