Speed Scaling for Energy and Performance with Instantaneous Parallelism

Hongyang Sun[1], Yuxiong He[2], and Wen-Jing Hsu[1]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
{sunh0007, hsu}@ntu.edu.sg
[2] Microsoft Research, Redmond, WA, USA
yuxhe@microsoft.com

Abstract. We consider energy-performance tradeoff for scheduling parallel jobs on multiprocessors using dynamic speed scaling. The objective is to minimize the sum of energy consumption and certain performance metric, including makespan and total flow time. We focus on designing algorithms that are aware of the jobs' instantaneous parallelism but not their characteristics in the future. For total flow time plus energy, it is known that any algorithm that does not rely on instantaneous parallelism is $\Omega(\ln^{1/\alpha} P)$-competitive, where P is the total number of processors. In this paper, we demonstrate the benefits of knowing instantaneous parallelism by presenting an $O(1)$-competitive algorithm. In the case of makespan plus energy, which is considered in the literature for the first time, we present an $O(\ln^{1-1/\alpha} P)$-competitive algorithm for batched jobs consisting of fully-parallel and sequential phases. We show that this algorithm is asymptotically optimal by providing a matching lower bound.

Keywords: Energy, Flow time, Instantaneous parallelism, Makespan, Multiprocessors, Parallel jobs, IP-clairvoyant, Speed scaling

1 Introduction

Energy consumption has become a key consideration in the design of modern high-performance computer systems. One popular approach to controlling energy is by dynamically scaling the speeds of the processors, or *dynamic speed scaling* [5, 9]. Since the seminal paper by Yao, Demers and Shenker [16], most researchers have assumed the power function of s^α when a processor runs at speed s, where $\alpha > 1$ is the power parameter. As this power function is strictly convex, the energy consumption when executing a job can be significantly reduced by slowing down the processor speed at the expense of the job's performance. Thus, how to optimally tradeoff energy and performance has become an active research topic. (See [10, 1] for two surveys of the field.)

We study energy-performance tradeoff for scheduling parallel jobs on multiprocessors. A scheduling algorithm needs to have both a *processor allocation policy*, which decides the number of processors allocated to each job at any time, and a *speed scaling policy*, which decides the speeds of the allocated processors. We assume that the parallel jobs under consideration have time-varying parallelism over different phases of execution [8, 7, 14]. This poses additional challenges compared to the speed scaling problem for sequential jobs. Our goal is to minimize sum of energy consumption and some performance metric, which in this paper includes either total flow time or makespan for a set of jobs. The *flow time* of a job is the duration between its release and completion, and the *total flow time* for a set of jobs is the sum of flow time of all jobs. The *makespan* is the completion time of the last completed job. Both metrics are widely used in scheduling literature. Although energy and flow time (or makespan) have different units, optimizing a combination of the two can be justified by looking at both objectives from a unified point of view in terms of economic costs.

Since Albers and Fujiwara [2] initiated minimizing total flow time plus energy, many results (e.g., [4, 11, 3, 6, 7, 14]) have been obtained under different online settings. Some results assume that the scheduling algorithm is *clairvoyant*, that is, it gains complete knowledge of a job, such as its total work, immediately upon the job's arrival; the other results are based on a more practical *non-clairvoyant* setting, where the scheduler knows nothing about the job. Most of these results, however, only concern scheduling sequential jobs on a single processor, and to the best of our knowledge, no previous work has considered makespan plus energy. The closest results to ours are by Chan, Edmonds and Pruhs [7], and Sun, Cao and Hsu [14], who studied non-clairvoyant scheduling for parallel jobs on multiprocessors to minimize total flow time plus energy. In both work, it is shown that any non-clairvoyant algorithm that allocates processors of uniform speed to a job will perform poorly, or $\Omega(P^{(\alpha-1)/\alpha^2})$-competitive, where P is the total number of processors. Intuitively, any non-clairvoyant algorithm may allocate a "wrong" number of processors to a job compared to its parallelism, thus either incurs excessive energy waste or causes severe execution delay. It turns out that non-uniform speed scaling can alleviate the problem. A lower bound of $\Omega(\log^{1/\alpha} P)$ has been shown in this case for any non-clairvoyant algorithm that allocates processors of different speeds to a job [7, 15].

In this paper, we consider a setting that lies in between clairvoyant and non-clairvoyant settings. In particular, a scheduling algorithm is allowed to know the available parallelism of a job at the immediate next step, or the *instantaneous parallelism* (IP). Any characteristic of the job in the future, such as its remaining parallelism and work, is still unknown. Hence, we call such algorithms *IP-clairvoyant*. In many parallel systems using centralized task queue or thread pool, instantaneous parallelism is simply the number of ready tasks in the queue or the number of ready threads in the pool, which is information practically available to the scheduler. Our contributions include the following algorithmic results that use instantaneous parallelism to schedule jobs:

- We present an $O(1)$-competitive algorithm with respect to total flow time plus energy. This significantly improves upon any non-clairvoyant algorithm and is the first $O(1)$-competitive algorithm for multiprocessor speed scaling on parallel jobs.
- We present an $O(\ln^{1-1/\alpha} P)$-competitive algorithm with respect to makespan plus energy for batched parallel jobs consisting of sequential and fully-parallel

phases. We also give a matching lower bound of $\Omega(\ln^{1-1/\alpha} P)$ for any IP-clairvoyant algorithm.

For total flow time plus energy, the improved result of our IP-clairvoyant algorithm over any non-clairvoyant algorithm comes from the fact that the knowledge of instantaneous parallelism enables a scheduling algorithm to allocate a "right" number of processors to a job at any time. This ensures no energy waste while at the same time guaranteeing sufficient execution rate for the jobs. Moreover, our IP-clairvoyant algorithm only requires allocating uniform-speed processors to a job, thus may have better feasibility in practice.

In addition, compared to minimizing total flow time plus energy, where the common practice is to set the power proportionally to the number of active jobs [4, 11, 3, 6] at any time, we show that the optimal strategy for minimizing makespan plus energy is to set the power consumption at a constant level, or precisely $\frac{1}{\alpha-1}$ at any time, where α is the power parameter.

2 Models and Objective Functions

We consider a set $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ of n jobs to be scheduled on P processors. Adopting the notations in [8, 7], we assume that each job $J_i \in \mathcal{J}$ contains k_i phases $\langle J_i^1, J_i^2, \cdots, J_i^{k_i}\rangle$, and each phase J_i^k has an amount of *work* w_i^k and a speedup function Γ_i^k. Unlike [8, 7], which assumed that each phase admits an arbitrary non-decreasing and sub-linear speedup, we consider the case where each phase has a *linear* speedup function up to a certain parallelism $h_i^k \geq 1$. Suppose that at any time t, job J_i is in its k-th phase and is allocated $a_i(t)$ processors of speed $s_i(t)$. Then, only $\bar{a}_i(t) = \min\{a_i(t), h_i^k\}$ processors are effectively utilized, and the speedup or the execution rate of the job at time t is given by $\Gamma_i^k(t) = \bar{a}_i(t)s_i(t)$. The *span* l_i^k of phase J_i^k, which is a convenient parameter representing the time to execute the phase with h_i^k or more processors of unit speed, is given by $l_i^k = w_i^k/h_i^k$. We say that phase J_i^k is *fully-parallel* if $h_i^k = \infty$ and it is *sequential* if $h_i^k = 1$. Moreover, if job J_i consists of only sequential and fully-parallel phases, we call it (Par-Seq)* job [13]. Finally, for each job J_i, we define its *total work* to be $w(J_i) = \sum_{k=1}^{k_i} w_i^k$ and define its *total span* to be $l(J_i) = \sum_{k=1}^{k_i} l_i^k$.

At any time t, a scheduling algorithm needs to specify the number $a_i(t)$ of processors allocated to each job J_i, as well as the speed $s_i(t)$ of the allocated processors. The algorithm is said to be *IP-clairvoyant* if it is only aware of the *instantaneous parallelism* (IP) of the job, i.e., h_i^k if job J_i is in phase J_i^k at time t. Any characteristic of the job in the future, including the remaining work of the phase and the existence of any subsequent phase is not available. We require that the total processor allocations cannot be more than the total number of processors at any time, i.e., $\sum_{i=1}^{n} a_i(t) \leq P$. Let r_i and c_i denote the *release time* and *completion time* of job J_i, respectively. If all jobs arrive in a *batch*, then their release times are all assumed to be 0. Otherwise, we assume without loss of generality that the first released job arrives at time 0. We require that any phase of a job cannot be executed unless all its preceding phases have been completed, i.e., $r_i = c_i^0 \leq c_i^1 \leq \cdots \leq c_i^{k_i} = c_i$, and $\int_{c_i^{k-1}}^{c_i^k} \Gamma_i^k(t)dt = w_i^k$ for all $1 \leq k \leq k_i$, where c_i^k denotes the completion time of phase J_i^k.

The *flow time* f_i of job J_i is the duration between its release and completion, i.e., $f_i = c_i - r_i$. The *total flow time* $F(\mathcal{J})$ of all jobs in $\mathcal{J}$ is given by $F(\mathcal{J}) = \sum_{i=1}^n f_i$, and the makespan $M(\mathcal{J})$ is the completion time of the last completed job, i.e., $M(\mathcal{J}) = \max_{i=1,\cdots,n} c_i$. Job J_i is said to be *active* at time t if it is released but not completed at t, i.e., $r_i \leq t \leq c_i$. An alternative expression for total flow time is $F(\mathcal{J}) = \int_0^\infty n_t dt$, where n_t is the number of active jobs at time t. For each processor at a particular time, its power consumption is given by s^α if it runs at speed s, where $\alpha > 1$ is the power parameter. Let $u_i(t)$ denote the power consumed by job J_i at time t, i.e., $u_i(t) = a_i(t)s_i(t)^\alpha$. The energy consumption e_i of the job is then given by $e_i = \int_0^\infty u_i(t)dt$, and the total energy consumption $E(\mathcal{J})$ of the job set is $E(\mathcal{J}) = \sum_{i=1}^n e_i$, or alternatively $E(\mathcal{J}) = \int_0^\infty u_t dt$, where $u_t = \sum_{i=1}^n u_i(t)$ denotes the total power consumption of all jobs at time t. We consider total flow time plus energy $G(\mathcal{J}) = F(\mathcal{J}) + E(\mathcal{J})$ and makespan plus energy $H(\mathcal{J}) = M(\mathcal{J}) + E(\mathcal{J})$ of the job set, and use *competitive analysis* to bound $G(\mathcal{J})$ or $H(\mathcal{J})$ by comparing them with the performances of the optimal offline algorithms, denoted by $G^*(\mathcal{J})$ and $H^*(\mathcal{J})$ respectively.

3 Total Flow Time plus Energy

3.1 Preliminaries

We first derive a lower bound on the total flow time plus energy, which will help conveniently bound the performance of an online algorithm through indirect comparing with the optimal.

Lemma 1. *The optimal total flow time plus energy of a job set $\mathcal{J}$ satisfies $G^*(\mathcal{J}) \geq G_1^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^n \sum_{k=1}^{k_i} \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$.*

Proof. Consider any phase J_i^k of any job $J_i \in \mathcal{J}$. The optimal scheduler will allocate a fixed number, say a_i, processors of the same speed, say s_i, to the phase throughout its execution. This is because, by the convexity of the power function, if different numbers of processors or different speeds are used, then averaging the processor numbers or the speeds will result in the same execution rate hence flow time but less energy consumption [16]. Moreover, we have $a_i \leq h_i^k$, since allocating more processors to a phase than its parallelism incurs more energy without improving flow time. The flow time plus energy introduced by the execution of J_i^k is then given by $\frac{w_i^k}{a_i s_i} + \frac{w_i^k}{a_i s_i} \cdot a_i s_i^\alpha = w_i^k \left(\frac{1}{a_i s_i} + s_i^{\alpha-1}\right) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{a_i^{1-1/\alpha}} \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$. Extending this lower bound to all phases of all jobs proves the lemma. □

We now describe an *amortized local competitiveness argument* [4] to prove the competitive ratio of our online scheduling algorithm. We first define some notations. For any job set $\mathcal{J}$ at time t, let $\frac{dG_A(\mathcal{J}(t))}{dt}$ and $\frac{dG^*(\mathcal{J}^*(t))}{dt}$ denote the rates of change for the total flow time plus energy under an online scheduler A and the optimal offline scheduler, respectively. Apparently, we have $\frac{dG_A(\mathcal{J}(t))}{dt} = n_t + u_t$, and $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + u_t^*$, where n_t^* and u_t^* denote the number of active jobs and power consumption under the optimal at time t. Moreover, let $\frac{dG_1^*(\mathcal{J}(t))}{dt}$ denote the rate of change for the lower bound given in Lemma 1 with respect to the execution of the job set under online algorithm A at time t. Lastly, we need to define a potential function $\Phi(t)$ associated with the status of the job set at any time t under both the online algorithm and the optimal. Then, we can similarly define $\frac{d\Phi(t)}{dt}$ to be the rate of change for the potential function at t. The following lemma shows that the competitive ratio of algorithm A can be obtained by bounding the performance of A at any time t with respect to the optimal scheduler through these rates of change.

Lemma 2. *Suppose that an online algorithm* A *schedules a set* $\mathcal{J}$ *of jobs. Then* A *is* $(c_1 + c_2)$*-competitive with respect to total flow time plus energy, if given a potential function* $\Phi(t)$*, the execution of the job set under* A *satisfies*

- *Boundary condition:* $\Phi(0) \leq 0$ *and* $\Phi(\infty) \geq 0$*;*
- *Arrival condition:* $\Phi(t)$ *does not increase when a new job arrives;*
- *Completion condition:* $\Phi(t)$ *does not increase when a job completes;*
- *Running condition:* $\frac{dG_A(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \leq c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dG_1^*(\mathcal{J}(t))}{dt}$.

Proof. Let T denote the set of time instances when a job arrives or completes under either the online algorithm A or the optimal. Integrating the running condition over time, we get $G_A(\mathcal{J}) + \Phi(\infty) - \Phi(0) + \sum_{t \in T} (\Phi(t^-) - \Phi(t^+)) \leq c_1 \cdot G^*(\mathcal{J}) + c_2 \cdot G_1^*(\mathcal{J})$, where t^- and t^+ denote the time right before and after t. Now, applying boundary, arrival and completion conditions to the above inequality, we get $G_A(\mathcal{J}) \leq c_1 \cdot G^*(\mathcal{J}) + c_2 \cdot G_1^*(\mathcal{J})$. Since $G_1^*(\mathcal{J})$ is a lower bound on the total flow time plus energy of job set $\mathcal{J}$ according to Lemma 1, the performance of algorithm A satisfies $G_A(\mathcal{J}) \leq (c_1 + c_2) \cdot G^*(\mathcal{J})$. □

3.2 U-CEQ and Performance

We present and analyze a IP-clairvoyant algorithm U-Ceq (Uniform Conservative Equi-partitioning), which is shown in Algorithm 1. U-Ceq uses a conservative version of the well-known Equi (Equi-partitioning) algorithm [8], which at any time t divides the total number P of processors equally among the n_t active jobs. However, U-Ceq makes sure that no job is allocated more processors than its instantaneous parallelism, which essentially avoids any waste of processor cycle hence energy consumption. Moreover, the speed of all allocated processors in U-Ceq is set in a uniform manner, and is therefore more feasible to implement in practice than the best known non-clairvoyant algorithms that rely on non-uniform speed scaling [7, 14].

We can see that each job J_i at any time t under U-Ceq consumes power $u_i(t) = \frac{1}{\alpha-1}$. Therefore, the overall power consumption is $u_t = \frac{n_t}{\alpha-1}$, which has been a common practice to minimize total flow time plus energy in the speed scaling literature (see, e.g., [4, 11, 3, 6]). The intuition is that an efficient online

Algorithm 1 U-Ceq

```
1: At any time $t$, allocate $a_i(t) = \min\{h_i^k, P/n_t\}$ processors to each active job
   $J_i$, where $h_i^k$ is the instantaneous parallelism of $J_i$ at time $t$.
2: set the speed of the allocated processors to $s_i(t) = \left(\frac{1}{(\alpha-1)a_i(t)}\right)^{1/\alpha}$.
```

algorithm should balance total flow time and energy. Since the rate of increase for the total flow time at time t is the number of active jobs n_t, having proportional increase for the energy consumption provides a good balance.

At time t, when job J_i is in its k-th phase, we say that it is *satisfied* if its processor allocation is exactly the instantaneous parallelism, i.e., $a_i(t) = h_i^k$. Otherwise, the job is *deprived* if $a_i(t) < h_i^k$. Let $\mathcal{J}_S(t)$ and $\mathcal{J}_D(t)$ denote the set of satisfied and the set of deprived jobs at time t, respectively. For convenience, we let $n_t^S = |\mathcal{J}_S(t)|$ and $n_t^D = |\mathcal{J}_D(t)|$. Apparently, we have $n_t = n_t^S + n_t^D$. Moreover, we define $x_t = n_t^D/n_t$ to be the *deprived ratio*. Since there is no energy waste, we will show that the execution rate for each job J_i, given by $\Gamma_i^k(t) = \frac{a_i(t)^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}}$, is sufficient to ensure the competitive performance of U-Ceq.

To apply the amortized local competitiveness argument shown in Lemma 2, we adopt the potential function by Lam et al. [11] in the analysis of online speed scaling algorithm for sequential jobs. Let $n_t(z)$ denote the number of active jobs whose remaining work is at least z at time t under U-Ceq, and let $n_t^*(z)$ denote the number of active jobs whose remaining work is at least z under the optimal. The potential function is defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) - n_t(z)^{1-1/\alpha} n_t^*(z) \right] dz, \qquad (1)$$

where $\eta = \frac{\eta'}{P^{1-1/\alpha}}$ and $\eta' = \frac{2\alpha^2}{(\alpha-1)^{1-1/\alpha}}$. We also need to apply the following lemma in our proof, which gives us an important inequality.

Lemma 3. $n_t^{1-1/\alpha} s_j^* \le \frac{\lambda P^{1-1/\alpha}}{\alpha} \left(s_j^*\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)} P^{1/\alpha}} n_t$ *for any* $n_t, s_j^* \ge 0$ *and* $P, \lambda > 0$.

Proof. The lemma is a direct result of Young's Inequality, which has been previously applied in [4, 11, 7, 6]. It is formally stated as follows. If f is a continuous and strictly increasing function on $[0, c]$ with $c > 0$, $f(0) = 0$, $a \in [0, c]$ and $b \in [0, f(c)]$, then $ab \le \int_0^a f(x)dx + \int_0^b f^{-1}(x)dx$, where f^{-1} is the inverse function of f. In this case, by setting $f(x) = \lambda P^{1-1/\alpha} x^{\alpha-1}$, $a = s_j^*$ and $b = n_t^{1-1/\alpha}$, the lemma can be implied. □

Theorem 1. U-Ceq *is $O(1)$-competitive with respect to total flow time plus energy for any set of parallel jobs.*

Proof. We will show that, with the potential function defined in Eq. (1), the execution of any job set under U-CEQ (UC for short) satisfies boundary, arrival and completion conditions shown in Lemma 2, as well as the running condition $\frac{dG_{UC}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \leq c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + c_2 \cdot \frac{dG_1^*(\mathcal{J}(t))}{dt}$, where $c_1 = \max\{\frac{2\alpha^2}{\alpha-1}, 2^\alpha\alpha\}$ and $c_2 = 2\alpha$. Since both c_1 and c_2 are constants with respect to P, the theorem is proved. We now examine each of these conditions in the following.

- *Boundary condition*: At time 0, no job exists. The terms $n_t(z)$ and $n_t^*(z)$ are both 0 for all z. Therefore, we have $\Phi(0) = 0$. At time ∞, all jobs have completed, so again we have $\Phi(\infty) = 0$. Hence, the boundary condition holds.

- *Arrival condition*: Suppose that a new job with work w arrives at time t. Let t^- and t^+ denote the time right before and after t. Thus, we have $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq w$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > w$, and similarly $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w$. For convenience, we define $\phi_t(z) = \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha}\right) - n_t(z)^{1-1/\alpha} n_t^*(z)$. It is obvious that for $z > w$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w$, we can get $\phi_{t^+}(z) - \phi_{t^-}(z) = n_{t^-}^*(z)\left(n_{t^-}(z)^{1-1/\alpha} - (n_{t^-}(z)+1)^{1-1/\alpha}\right) \leq 0$. Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t^+}(z)dz \leq \eta \int_0^\infty \phi_{t^-}(z)dz = \Phi(t^-)$, and the arrival condition holds.

- *Completion condition*: When a job completes under either U-CEQ or the optimal, $\Phi(t)$ is unchanged since $n(t)$ or $n^*(t)$ is unchanged for all $z > 0$. Hence, the completion condition holds.

- *Running condition*: At any time t, suppose that the optimal offline scheduler sets the speed of the j-th processor to s_j^*, where $j = 1, \cdots, P$. We have $\frac{dG_{UC}(\mathcal{J}(t))}{dt} = n_t + u_t = \frac{\alpha}{\alpha-1} n_t$ and $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + u_t^* = n_t^* + \sum_{j=1}^P \left(s_j^*\right)^\alpha$. To bound the rate of change $\frac{dG_1^*(\mathcal{J}(t))}{dt}$, which only depends on the portions of the jobs executed under U-CEQ at t, we focus on the set $J_S(t)$ of satisfied jobs. Since each job $J_i \in J_S(t)$ has processor allocation $a_i(t) = h_i^k$, we can get $\frac{dG_1^*(\mathcal{J}(t))}{dt} \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{J_i \in \mathcal{J}_S(t)} \frac{\Gamma_i^k(t)}{\left(h_i^k\right)^{1-1/\alpha}} = \frac{\alpha}{\alpha-1} |\mathcal{J}_S(t)| = \frac{\alpha}{\alpha-1}(1-x_t)n_t$. We now focus on finding an upper bound for $\frac{d\Phi(t)}{dt}$. In this case, we consider the set $\mathcal{J}_D(t)$ of deprived jobs, which in the worst case may have the most remaining work. In addition, each job $J_i \in J_D(t)$ has processor allocation $a_i(t) = P/n_t$. The rate of change for the potential function can then be shown to satisfy

$$\frac{d\Phi(t)}{dt} \leq \frac{\eta'}{P^{1-1/\alpha}} \left(-\sum_{i=1}^{n_t^D} i^{1-1/\alpha} \Gamma_i^k(t) + n_t^{1-1/\alpha} \sum_{j=1}^{P} s_j^* \right) + \frac{\eta'}{P^{1-1/\alpha}} \left(n_t^* \sum_{i=1}^{n_t} \left(i^{1-1/\alpha} - (i-1)^{1-1/\alpha} \right) \Gamma_i^k(t) \right). \tag{2}$$

More details on the above derivation can be found in the full version of this paper [15]. Now, to simplify Inequality (2), we have $\sum_{i=1}^{n_t}\left(i^{1-1/\alpha} - (i-1)^{1-1/\alpha}\right) = n_t^{1-1/\alpha}$ and by approximating summation with integral, we get $\sum_{i=1}^{n_t^D} i^{1-1/\alpha} \geq \int_0^{n_t^D} i^{1-1/\alpha} di = \frac{(n_t^D)^{2-1/\alpha}}{2-1/\alpha} \geq \frac{x_t^2 n_t^{2-1/\alpha}}{2}$. According to Lemma 3, we also have

$n_t^{1-1/\alpha}\sum_{j=1}^P s_j^* \le \frac{\lambda P^{1-1/\alpha}}{\alpha}\sum_{j=1}^P \left(s_j^*\right)^\alpha + \frac{(1-1/\alpha)P^{1-1/\alpha}}{\lambda^{1/(\alpha-1)}}n_t$, where λ is any positive constant. Finally, we have $\Gamma_i^k(t) = \frac{P^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}n_t^{1-1/\alpha}}$ for each job $J_i \in \mathcal{J}_D(t)$. Thus, we get $\frac{d\Phi(t)}{dt} \le \eta'\left(-\frac{x_t^2}{2(\alpha-1)^{1/\alpha}}n_t + \frac{\lambda}{\alpha}\sum_{j=1}^P \left(s_j^*\right)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}}n_t + \frac{n_t^*}{(\alpha-1)^{1/\alpha}}\right)$. Set $\lambda = 2^{\alpha-1}(\alpha-1)^{1-1/\alpha}$ and substitute various rates of change as well as c_1, c_2 into the running condition, we can verify that it holds for all values of x_t. □

4 Makespan plus Energy

4.1 Performance of the Optimal

We first show that as far as minimizing makespan plus energy for batched jobs, the optimal (online/offline) strategy maintains a constant total power $\frac{1}{\alpha-1}$ at any time. This corresponds to the *power equality property* shown in [12], which applies to any optimal offline algorithm for the makespan minimization problem with an energy budget.

Lemma 4. *For any schedule* A *on a set* $\mathcal{J}$ *of batched jobs, there exists a schedule* B *that executes* $\mathcal{J}$ *with a constant total power* $\frac{1}{\alpha-1}$ *at any time, and performs no worse than* A *with respect to makespan plus energy, i.e.,* $H_B(\mathcal{J}) \le H_A(\mathcal{J})$.

Proof. For any schedule A on a set $\mathcal{J}$ of batched jobs, consider an interval Δt during which the speeds of all processors, denoted as $(s_1, s_2, \cdots, s_P)$, remain unchanged. The makespan plus energy of A incurred by executing this portion of the job set is given by $H_A = \Delta t(1+u)$, where $u = \sum_{j=1}^P s_j^\alpha$ is the power consumption of all processors during Δt. We now construct schedule B such that it executes the same portion of the job set by running the j-th processor at speed $k \cdot s_j$, where $k = \left(\frac{1}{(\alpha-1)u}\right)^{1/\alpha}$. This portion will then finish under schedule B in $\frac{\Delta t}{k}$ time, and the power consumption at any time in this interval is given by $\frac{1}{\alpha-1}$. The makespan plus energy of B incurred by executing the same portion of the job set is $H_B = \frac{\Delta t}{k}(1+\frac{1}{\alpha-1}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\Delta t u^{1/\alpha}$. Since $\frac{1+u}{u^{1/\alpha}}$ is minimized when $u = \frac{1}{\alpha-1}$, we have $\frac{H_A}{H_B} = \frac{(\alpha-1)^{1-1/\alpha}}{\alpha} \cdot \frac{1+u}{u^{1/\alpha}} \ge 1$, i.e., $H_A \ge H_B$. Extending this argument to all such intervals in schedule A proves the lemma. □

Compared to total flow time plus energy, where the completion time of each job contributes to the overall objective function, makespan for a set of jobs is the completion time of the last job. The other jobs only contribute to the energy consumption part of the objective, thus can be slowed down to consume less energy, which eventually results in better overall performance. Based on this observation as well as Lemma 4, we derive the performance of the optimal offline scheduler for any batched (Par-Seq)* job set in the following lemma.

Lemma 5. *The optimal makespan plus energy for any batched set* $\mathcal{J}$ *of* (Par-Seq)* *jobs satisfies* $H^*(\mathcal{J}) \ge \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \max\{\frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}}, \left(\sum_{i=1}^n l(J_i)^\alpha\right)^{1/\alpha}\}$.

Proof. Given any job $J_i \in \mathcal{J}$, define $J_{i,P}$ to be a job with a single fully-parallel phase of the same work as J_i, and define $J_{i,S}$ to be a job with a single sequential phase of the same span as J_i. Moreover, we define $\mathcal{J}_P = \{J_{i,P} : J_i \in \mathcal{J}\}$ and $\mathcal{J}_S = \{J_{i,S} : J_i \in \mathcal{J}\}$. Clearly, the optimal makespan plus energy for $\mathcal{J}_P$ and $\mathcal{J}_S$ will be no worse than that for $\mathcal{J}$, i.e., $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_P)$ and $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_S)$, since the optimal schedule for $\mathcal{J}$ is a valid schedule for $\mathcal{J}_P$ and $\mathcal{J}_S$.

For job set $\mathcal{J}_P$, the optimal scheduler can execute the jobs in any order since all jobs are fully-parallel in this case. Moreover, by the convexity of the power function, all P processors are run with constant speed s. According to Lemma 4, we have $Ps^\alpha = \frac{1}{\alpha-1}$, hence $s = \left(\frac{1}{(\alpha-1)P}\right)^{1/\alpha}$. The makespan plus energy is therefore $H^*(\mathcal{J}_P) = \frac{\sum_{i=1}^n w(J_i)}{Ps}(1 + Ps^\alpha) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}}$.

For job set $\mathcal{J}_S$, the optimal can execute each job on a single processor with constant speed. Moreover, all jobs are completed simultaneously, since otherwise jobs completed earlier can be slowed down to save energy without affecting makespan. Let s_i denote the speed by the optimal for job $J_{i,S}$, so $\frac{l(J_1)}{s_1} = \frac{l(J_2)}{s_2} = \cdots = \frac{l(J_n)}{s_n}$, and $\sum_{i=1}^n s_i^\alpha = \frac{1}{\alpha-1}$ according to Lemma 4. Therefore, we have $s_i = \frac{1}{(\alpha-1)^{1/\alpha}} \cdot \frac{l(J_i)}{\left(\sum_{i=1}^n l(J_i)^\alpha\right)^{1/\alpha}}$ for $i = 1, 2, \cdots, n$. The makespan plus energy is $H^*(\mathcal{J}_S) = \frac{l(J_1)}{s_1} + \frac{l(J_1)}{s_1}\left(\sum_{i=1}^n s_i^\alpha\right) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\left(\sum_{i=1}^n l(J_i)^\alpha\right)^{1/\alpha}$. □

4.2 P-FIRST and Performance

We now present and analyze a IP-clairvoyant algorithm P-First (Parallel-First) for any batched set $\mathcal{J}$ of (Par-Seq)* jobs. As shown in Algorithm 2, P-First will first execute the fully-parallel phases of any job whenever possible, and then executes the sequential phases of all jobs at the same rate.

Algorithm 2 P-First

```
1: if there is at least one active job in fully-parallel phase at any time $t$ then
2:     execute any such job on $P$ processors, each with speed $\left(\frac{1}{(\alpha-1)P}\right)^{1/\alpha}$.
3: else
4:     execute all $n_t$ active jobs on $P' = \min\{n_t, P\}$ processors by equally sharing
       the processors among the jobs; each processor runs at speed $\left(\frac{1}{(\alpha-1)P'}\right)^{1/\alpha}$.
```

P-First ensures that the overall energy consumption $E(\mathcal{J})$ and the makespan $M(\mathcal{J})$ of job set $\mathcal{J}$ satisfies $E(\mathcal{J}) = \frac{1}{\alpha-1}M(\mathcal{J})$, since at any time t, the total power is given by $u_t = \frac{1}{\alpha-1}$, and $E(\mathcal{J}) = \int_0^{M(\mathcal{J})} u_t dt$. The makespan plus energy of the job set thus satisfies $H(\mathcal{J}) = E(\mathcal{J}) + M(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$. The performance of P-First is shown in the following theorem.

Theorem 2. P-First *is* $O(\ln^{1-1/\alpha} P)$*-competitive with respect to makespan plus energy for any set of batched* (Par-Seq)* *jobs, where* P *is the total number of processors.*

Proof. Since the makespan plus energy of job set $\mathcal{J}$ scheduled by P-First satisfies $H(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$, we only focus on the makespan $M(\mathcal{J})$ by bounding separately the time $M'(\mathcal{J})$ when all P processors are utilized and the time $M''(\mathcal{J})$ when less than P processors are utilized. Obviously, we have $M(\mathcal{J}) = M'(\mathcal{J}) + M''(\mathcal{J})$.

According to P-First, the execution rate when all P processors are utilized is given by $\frac{P^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}}$. The total work completed in this case is upper bounded by $\sum_{i=1}^{n} w(J_i)$. Hence, we have $M'(\mathcal{J}) \le (\alpha-1)^{1/\alpha}\frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}}$. We now bound $M''(\mathcal{J})$ when less than P processors are used, which only occurs while P-First executes sequential phases. Since all jobs are batch released, the number of active jobs monotonically decreases. Let T denote the first time when the number of active jobs drops below P, and let $m = n_T$. Therefore, we have $m < P$. For each of the m active job J_i at time T, let $\bar{l}_i$ denote the remaining span of the job. Rename the jobs such that $\bar{l}_1 \le \bar{l}_2 \le \cdots \le \bar{l}_m$. Since P-First executes the sequential phases of all jobs at the same speed, the sequential phases of the m jobs will complete exactly in the above order. Define $\bar{l}_0 = 0$, then we have $M''(\mathcal{J}) = \sum_{i=1}^{m} \frac{\bar{l}_i - \bar{l}_{i-1}}{\left(\frac{1}{(\alpha-1)(m-i+1)}\right)^{1/\alpha}} = (\alpha-1)^{1/\alpha}\sum_{i=1}^{m}\left((m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}\right)\bar{l}_i$. For convenience, define $c_i = (m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}$ for $1 \le i \le m$, and we can get $c_i \le \frac{1}{(m-i+1)^{1-1/\alpha}}$. Let $R = \sum_{i=1}^{m} \bar{l}_i^{\alpha}$, and subject to this condition and the ordering of $\bar{l}_i$, $\sum_{i=1}^{m} c_i \cdot \bar{l}_i$ is maximized when $\bar{l}_i = R^{1/\alpha} \cdot \frac{c_i^{\frac{1}{\alpha-1}}}{\left(\sum_{i=1}^{m} c_i^{\frac{\alpha}{\alpha-1}}\right)^{1/\alpha}}$. Hence, we have $M''(\mathcal{J}) \le (\alpha-1)^{1/\alpha}R^{1/\alpha}\left(\sum_{i=1}^{m} c_i^{\frac{\alpha}{\alpha-1}}\right)^{1-1/\alpha} \le (\alpha-1)^{1/\alpha}R^{1/\alpha}H_m^{1-1/\alpha}$, where $H_m = 1 + 1/2 + \cdots + 1/m$ denotes the m-th harmonic number.

The makespan plus energy of the job set scheduled under P-First thus satisfies $H(\mathcal{J}) \le \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}\left(\frac{\sum_{i=1}^{n} w(J_i)}{P^{1-1/\alpha}} + R^{1/\alpha}H_m^{1-1/\alpha}\right)$. Since it is obvious that $\sum_{i=1}^{n} l(J_i)^{\alpha} \ge \sum_{i=1}^{m} \bar{l}_i^{\alpha} = R$, comparing the performance of P-First with that of the optimal in Lemma 5, we have $H(\mathcal{J}) \le (1+H_m^{1-1/\alpha})\cdot H^*(\mathcal{J}) = O(\ln^{1-1/\alpha} P)\cdot H^*(\mathcal{J})$, as $m < P$ and it is well-known that $H_m = O(\ln m)$. □

From the proof of Theorem 2, we observe that the competitive ratio of P-First is dominated by the execution of sequential phases of the (Par-Seq)* jobs. Without knowing the jobs' future work, the optimal strategy for any online algorithm does seem to execute their sequential phases at the same rate. In the following theorem, we confirm this intuition by proving a matching lower bound for any IP-clairvoyant algorithm using sequential jobs only. It also implies that P-First is asymptotically optimal with respect to makespan plus energy.

Theorem 3. *Any IP-clairvoyant algorithm is* $\Omega(\ln^{1-1/\alpha} P)$*-competitive with respect to makespan plus energy, where* P *is the total number of processors.*

Proof. Consider a batched set $\mathcal{J}$ of P sequential jobs, where the i-th job has span $l(J_i) = \frac{1}{(P-i+1)^{1/\alpha}}$. Since the number of jobs is the same as the number of processors, any reasonable algorithm will assign one job to one processor. From Lemma 5, the optimal offline scheduler has makespan plus energy $H^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} H_P^{1/\alpha}$, where H_P is the P-th harmonic number. We will show that P-First performs no worse than any IP-clairvoyant algorithm A. From proof of Theorem 2, we can get $H_{PF}(\mathcal{J}) = \frac{\alpha}{\alpha-1} M(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \sum_{i=1}^{P} \left((P-i+1)^{1/\alpha} - (P-i)^{1/\alpha}\right) l(J_i) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \sum_{i=1}^{P} \frac{l(J_i)}{\alpha(P-i+1)^{1-1/\alpha}} = \frac{1}{(\alpha-1)^{1-1/\alpha}} H_P$. Comparing the performances of P-First and the optimal proves the theorem, since it is also well-known that $H_P = \Omega(\ln P)$.

To show $H_{PF}(\mathcal{J}) \leq H_A(\mathcal{J})$, we construct schedules from A to P-First in three steps without increasing the total cost. For schedule A, the adversary always assigns the i-th job to the processor that first completes $\frac{1}{(P-i+1)^{1/\alpha}}$ amount of work with ties broken arbitrarily. For convenience, we let the i-th job assigned to the i-th processor. First, we construct schedule A$'$ from A by executing each job J_i with constant speed s'_i derived by taking the average speed of processor i in A. Based on the convexity of the power function, the completion time of each job remains the same in A$'$ but the energy may be reduced. Thus, we have $H_{A'}(\mathcal{J}) \leq H_A(\mathcal{J})$. According to the adversarial strategy, the processor speeds in A$'$ satisfy $s'_1 \geq s'_2 \geq \cdots \geq s'_P$. We then construct schedule A$''$ by executing each job J_i with speed s'_P throughout its execution. Since we also have $l(J_1) < l(J_2) < \cdots < l(J_P)$, the makespan in A$''$ is still determined by job J_P and is the same as that in A$'$, but the energy may be reduced by slowing down other jobs. Thus, we have $H_{A''}(\mathcal{J}) \leq H_{A'}(\mathcal{J})$. Note that the speeds of all processors are the same in A$''$. According to Lemma 4, we can construct schedule B from A$''$ such that it consumes constant total power $\frac{1}{\alpha-1}$ at any time and $H_B(\mathcal{J}) \leq H_{A''}(\mathcal{J})$. By observing that B is identical to P-First, the proof is complete. □

5 Discussion

For the objective of makespan plus energy, we have only studied the performance of IP-clairvoyant algorithms on (Par-Seq)* jobs. How to deal with jobs with arbitrary parallelism profile and what is the performance in the non-clairvoyant setting remain interesting problems to consider. In particular, comparing the known performance ratios of IP-clairvoyant and non-clairvoyant algorithms with respect to both objective functions as shown in Table 1, we conjecture that minimizing makespan plus energy is inherently more difficult than minimizing total flow time plus energy, hence is likely to incur a much larger lower bound in the non-clairvoyant setting. Intuitively, a non-clairvoyant algorithm for makespan plus energy can potentially make mistakes not only in speed assignment, but also in processor allocation. The former mistake leads to bad performance since jobs that complete early may in fact be slowed down to save energy, and this has contributed to the lower bound of IP-clairvoyant algorithms shown in this

	Total flow time plus energy	*Makespan plus energy*
IP-clairvoyant	$O(1)$	$\Omega(\ln^{1-1/\alpha} P)$
Non-clairvoyant	$\Omega(\log^{1/\alpha} P)$	?

Table 1. Performance comparison for total flow time plus energy and makespan plus energy under IP-clairvoyant setting and non-clairvoyant setting.

paper. The situation may deteriorate further in the non-clairvoyant setting as more energy will be wasted or slower execution rate will result if a wrong number of processors is also allocated to a job.

References

[1] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[2] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *STACS*, pages 621–633, 2006.

[3] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, 2009.

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

[5] 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.

[6] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 409–420, 2009.

[7] H.-L. Chan, J. Edmonds, and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

[8] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, 1999.

[9] D. Grunwald, C. B. Morrey III, P. Levis, M. Neufeld, and K. I. Farkas. Policies for dynamic clock scheduling. In *OSDI*, pages 6–6, 2000.

[10] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[11] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, 2008.

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

[13] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *ESA*, pages 741–753, 2007.

[14] H. Sun, Y. Cao, and W.-J. Hsu. Non-clairvoyant speed scaling for batched parallel jobs on multiprocessors. In *CF*, pages 99–108, 2009.

[15] H. Sun, Y. He, and W.-J. Hsu. Energy-Efficient Multiprocessor Scheduling for Flow Time and Makespan. *CoRR abs/1010.4110*, 2010.

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