Energy Efficient Scheduling of Parallelizable Jobs

Kyle Fox* Sungjin Im† Benjamin Moseley‡

Abstract

In this paper, we consider scheduling parallelizable jobs in the non-clairvoyant speed scaling setting to minimize the objective of weighted flow time plus energy. Previously, strong lower bounds were shown on this model in the unweighted setting even when the algorithm is given a constant amount of resource augmentation over the optimal solution. However, these lower bounds were given only for certain families of algorithms that do not recognize the parallelizability of alive jobs. In this work, we circumvent previous lower bounds shown and give a *scalable* algorithm under the natural assumption that the algorithm can know the current parallelizability of a job. When a general power function is considered, this is also the *first* algorithm that has a constant competitive ratio for the problem using any amount of resource augmentation.

1 Introduction

Energy aware job scheduling has recently received a significant amount of attention in scheduling theory literature [5–10, 12, 14–16, 18–20, 28, 29, 33–35]. When scheduling jobs, the scheduler must decide which resources (for example, machines or processors) to allocate to which jobs over time. In an energy aware scheduling environment, the scheduler must also decide how to use resources in order to minimize the energy consumed by the system. For example, the scheduler may turn off machines over time or the scheduler may dynamically scale the speed of a processor. The scheduler's goal is to minimize the energy consumption while optimizing the quality of service with respect to the jobs (or the clients who submitted the jobs) receive. These two objectives are naturally competing. By using less energy the scheduler will have less resources available to process the jobs. The scheduler in this setting must determine the appropriate balance between energy consumption and the quality of service delivered.

The dominant energy aware scheduling model considered in previous literature is an online speed scaling setting. In an *online* setting, the scheduler only becomes aware of a job once it arrives in the system, and the scheduler makes decisions over time without knowing the jobs that are to arrive in the future. Online models accurately capture most of the scheduling problems faced in the real world, because generally the scheduler will not be aware of a job until it arrives to the system. The model can be made even more realistic by assuming the scheduler is *non-clairvoyant*; it does not know the processing time of the jobs. In the *speed scaling* model, the processors available to the scheduler can dynamically change their speed over time. This model captures the fact that many systems today have the ability to change the processor frequency over time such as the technology used in AMD's PowerNow! [1] and Intel's SpeedStep [2]. Such technology has become ever more important in mobil computing where battery power consumption is a high priority [38] and in large scale server farms that use a very costly amount of energy [38].

Perhaps the most popular online scheduling objective without considering energy is minimizing the total flow time. The *flow time* of a job is the amount of time that passes between the job's release and its completion. By minimizing the total flow time, the server focuses on optimizing the average quality of service delivered to the jobs. In the online speed scaling setting, most previous literature considered minimizing the total flow time plus the energy consumed in the system. This has a natural interpretation; a system designer may decide that it is worth spending one unit of energy to save β units of flow time. In this case, the system designer would want to minimize the total flow time plus β multiplied by the energy. By scaling the units of time and energy, we may assume that $\beta = 1$ and consider minimizing the total flow time plus energy. Work on this objective was initiated by studying processors whose power consumption obeys a

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. kylefox2@illinois.edu. Portions of this work were done while the author was visiting Google Research. Research supported in part by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100.

†Department of Computer Science, Duke University, Durham NC 27708-0129. sungjin@cs.duke.edu. This work was done while the author was at the University of Illinois. Partially supported by NSF grant CCF-1016684.

‡Toyota Technological Institute at Chicago, Chicago, IL 60637. moseley@ttic.edu. This work was done while the author was at the University of Illinois. Partially supported by NSF grant CCF-1016684.

generalization of the cube root rule observed in standard CMOS based processors [13, 38]. For example, if the processor runs at speed s then the power consumed is s^α; the cubed root rule is observed when $\alpha = 3$. This model was rigorously studied [6, 7, 9, 12, 14, 16, 33, 35], until a generalization was introduced by Bansal et al. [8]. In the generalization, there is a convex function P where $P(s)$ is the power consumed when running the processor at speed s. This is known as the arbitrary power function model. In either model, the total energy consumed by the system is the integral of power over all time.

1.1 Old and new scheduling models. In the online speed scaling setting the scheduler has two policies:

> **Job selection:** The scheduler must decide which job should be scheduled at each moment in time.
>
> **Energy policy:** The scheduler must decide how fast to run the processor(s) at each moment in time.

When considering scheduling jobs on a single machine with speed scaling, the general approach is to use a known algorithm that performs well on a single machine without speed scaling and couple the algorithm with a natural speed scaling policy introduced in [4]. Essentially, this speed scaling policy runs the machine at speed s such that $P(s)$ is proportional to the number of unsatisfied jobs. This creates a natural balance between the energy objective and the flow time objective, because the increase in the flow time objective at each moment in time is equal to the number of unsatisfied jobs. At each point in time, the two objectives will increase roughly at the same rate.

Recently, scheduling speed scaling jobs online on multiple processors or machines has been addressed. When addressing a multiple processor setting there are two models which one can consider. The first is when each job can be assigned to at most one machine at each moment in time. This model naturally captures the settings faced in multiple machine environments such as server farms. It is known that when processors have fixed speeds and energy is not a part of the objective that no algorithm can be $O(1)$-competitive for the objective of total flow time, even if we do not assume non-clairvoyance [37]. This lower bound carries over to the speed scaling setting with arbitrary power functions. Due to this lower bound, most previous work has turned to using a *resource augmentation* analysis [31]. Here the algorithm is given extra resources over the adversary and then the competitive ratio is bounded. We say that an algorithm is b-speed c-competitive if the algorithm with b-speed achieves an objective value at most c times the optimal value with unit speed. In the fixed speed setting, we mean each of the processors runs b times faster than the optimal solution's processors. In the speed scaling setting, when the algorithm runs a processor at power $P(s)$ the algorithm can process jobs at speed $b \cdot s$. The goal of this form of analysis is to find a $(1+\varepsilon)$-speed $O(f(\varepsilon))$-competitive algorithm for any constant $\varepsilon > 0$ where f is some function of only ε. That is, an algorithm which is $O(1)$-competitive while using a minimum amount of extra resources over the adversary. Such an algorithm is called scalable. For problems with strong lower bounds on the competitive ratio, a scalable algorithm is the best result that one can show using worst case analysis.

The works of [35, 36] consider the case where each machine is identical. When machines are not identical, the problem becomes significantly more challenging. It was not until recently that scalable clairvoyant and non-clairovyant algorithms were found for the heterogeneous machine settings [27, 28].

Another possible parallel scheduling model is when jobs can be scheduled on more than one processor at each moment in time. That is, jobs can run in parallel on multiple processors. The challenge in this setting is that there can be different degrees of parallelizability, even within phases of a single job. That is, each job consists of a set of phases that are to be processed sequentially. According to the phase a job is in, the job may be considerably sped-up when assigned to multiple processors, might not get sped-up, or something in between. The phases of a job can lead to different amounts of parallelism, because one phase may require a lot of computation that can be easily parallelized, while another phase may require I/O that cannot be sped-up with extra computational power.

The model given above is known as the arbitrary speed-up curves model, so named because each phase of a job will have its own arbitrary speed-up curve/function that specifies its parallizability. Scheduling in this model was originally studied when energy was not taken into consideration and the processor ran at a fixed speed. In the arbitrary speed-up curves model, all pervious work has focused on non-clairvoyant algorithms. In [22], Edmonds gave a $(2+\varepsilon)$-speed $O(1)$-competitive algorithm for any fixed $\varepsilon > 0$. The algorithm considered was a round robin like algorithm known as Equipartition. In a breakthrough result, Edmonds and Pruhs [25] introduced the algorithm LAPS and showed that it is scalable. Since its introduction, the algorithm LAPS has proven to be very useful in many other scheduling settings [11, 16, 17, 19, 24, 26, 27].

Prior work shows that scheduling in the speed-up curves model becomes considerably more challenging when speed scaling is introduced. Chan *et al.* [17] show that an $O(\log m)$-competitive algorithm exists if $P(s) = s^\alpha$ where m is the number of available processors. Their algorithm makes the assumption that jobs do not have side effects. The assumption allows their algorithm to process multiple copies of the same job simultaneously; if a job has side effects then multiple copies running at the same

time could interfere with each other. Unfortunately, this assumption is not valid in many systems, and Chan *et al.* argue that no non-clairvoyant randomized algorithm can be $\Omega(m^{(\alpha-1)/\alpha})$-competitive when $P(s) = s^\alpha$ if jobs do have side effects [17]. This lower bound implies no algorithm can be $O(1)$-speed $O(1)$-competitive in the arbitrary function case when jobs can have side effects. They also argue that an $\Omega(\log^{1/\alpha} m)$ lower bound exists on the competitive ratio even when jobs do not have side effects [17].

Their work suggests that scheduling with speed scalable processors when jobs are parallizable is essentially closed. No algorithm can have a small competitive ratio even with resource augmentation in the arbitrary power function case. In the case where $P(s) = s^\alpha$ and jobs do not have side effects, there is an algorithm that achieves roughly the best possible competitive ratio. In the case where jobs have side effects, no algorithm can have a small competitive ratio even when $P(s) = s^\alpha$. It appears that adding both parallelism and the ability to speed scale processors makes the problem much harder and there is no hope for an online algorithm with a constant competitive ratio using a small amount of resource augmentation.

However, previous work on the arbitrary speed-up curves problem such as [17] rely on a crucial assumption; the scheduler is never aware of the parallizability of a phase of a job. That is, the online algorithm is in the 'dark' about how parallel a phase is, yet an optimal solution being compared against may rely on this information. We argue that this assumption is perhaps too strong. Indeed, if some phase of a job uses multiple threads or processes, then one can determine how many there are. There is a vast amount of literature on how to measure the parallel performance during runtime [3, 21, 32, 39, 44]. Further, there are many systems where the parallelism of incoming jobs is given a priori to the scheduler either specified by the user or based on estimates calculated from previous job traces [40–42]. We suggest that giving the scheduler knowledge about the parallizability of job phases still accurately models the real world while allowing for the existence of competitive scheduling algorithms. Indeed, in [43] it was shown that if the scheduler is aware of the instantaneous paralizability of a job and a job is always fully parallel up to a certain number of processors, then their algorithm is $O(1)$ competitive when the power function is s^α.

1.2 Our results and contributions. In this paper we consider a non-clairvoyent arbitrary speed-up curves model where processors can be dynamically speed scaled. Unlike previous work such as [17], we assume that each algorithm knows how parallel each phase of any job is, and unlike [43], we allow *arbitrary* speedup curves that may not be fully parallel up to a point. This information need not be given to the algorithm initially; although, it could be. Instead, the algorithm only requires knowledge of the parallizability of a job phase at the point in time where the algorithm decides to process that phase. The algorithm does not learn the total number of phases in any job or the total processing needed for a job until the job's completion. Thus, the algorithm is non-clairvoyant. In this model, we can show the following result and its corollary, circumventing the strong lower bounds in the more restrictive model mentioned above.

THEOREM 1.1. *In the arbitrary speed-up curves setting where processors can be speed scaled with an arbitrary convex power function, there is a $(1+\varepsilon)$-speed $O(1/\varepsilon^2)$-competitive algorithm for minimizing weighted flow time plus energy for any $\varepsilon > 0$.*

COROLLARY 1.1. *In the arbitrary speed-up curves setting where processors can be speed scaled with a power function of the form $P(s) = s^\alpha$, there is a $O((1+\varepsilon)^\alpha/\varepsilon^2)$-competitive algorithm for minimizing weighted flow time plus energy for any $\varepsilon > 0$.*

Note that no resource augmentation is needed in the above corollary. The corollary immediately follows because when $P(s) = s^\alpha$, we can simulate resource augmentation by running the processors a factor of $(1+\varepsilon)$ faster. This process increases the energy cost by a factor of $(1+\varepsilon)^\alpha$.

We note that our results apply to the more general *weighted* case, where each job is given a non-negative weight and the total flow time is calculated by summing the flow time of individual jobs multiplied by their weights. This is essentially the best positive result one can hope for because this problem generalizes minimizing total flow time in the identical machine setting where each job can be scheduled on at most one machine at once and, as mentioned, there are strong lower bounds for any online algorithm for this problem. This is also the first positive result in the speed-up curves setting with power for the cases where the power function P is an arbitrary convex function.

We feel our work shows that scheduling with speed scalable processors with parallelizable jobs is not as hard as previous research suggests. Another contribution is in our analysis. All previous works considering the speed-up curves model and the speed scaling model use a potential function analysis to bound the performance of an online algorithm. The works on the speed-up curves model first use a reduction from the general parallel setting to a setting where each phase of a job is either fully parallelizable or sequential. We cannot use this reduction for our model as our algorithm's behavior changes when the parallelizability of jobs changes. Due to this, we directly analyze the general setting and develop a new potential function that captures the parallelizability of

different job phases. Our potential function shows how to circumvent the reductions used in previous work and we hope the underlying ideas will be of use in future studies of the speed-up curve model.

2 Preliminaries

2.1 Formal problem definition. We first describe the arbitrary speed-up curves model [23] and then extend the model to the speed-scaling setting.

Arbitrary speed-up curves: In this model there are m uniform parallel machines/processors. A job i may consist of multiple phases. Let $\hat{\mu}_i$ denote the number of phases of job i. In the μ_{th} phase ($1 \leq \mu \leq \hat{\mu}_i$), job i has work p_i^μ that that needs to be completed, and is associated with a speed-up curve Γ_i^μ that specifies the parallelizability of job i in the phase. The phases of job i must be processed sequentially. In other words, job i's $(\mu+1)_{th}$ phase immediately starts when the p_i^μ amount of work is done for job i in its μ_{th} phase. Job i is considered complete when $p_i^{\hat{\mu}_i}$ work is done in its $\hat{\mu}_i$th phase. A speed-up curve $\Gamma : \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing and continuous function where $\mathbb{R}^+$ denotes the set of non-negative real numbers. If a job is in a phase with speed-up curve Γ, then it can be processed (have work done) at a rate of $\Gamma(h)$ when using h machines. It is assumed that $\Gamma(h)/h$ is non-increasing, so that the processing efficiency per machine does not increase as more machine are used. That is, a job's parallelizability satisfies Brent's Thoerem. All properties of job i can be described in a compact way as $\langle((\Gamma_i^1, p_i^1), (\Gamma_i^2, p_i^2), ..., (\Gamma_i^{\hat{\mu}_i}, p_i^{\hat{\mu}_i})\rangle$ along with its release time r_i and weight w_i.

At each point in time t, a feasible schedule in the arbitrary speed-up curve model allocates the m available machines to the currently alive/unsatisfied jobs. To formally describe a feasible schedule and its scheduling objective, we need to define more notation. Let n denote the number of jobs that are released over time. Consider any schedule A and let $q^A(t)$ denote the set of alive jobs at time t (the jobs that are released no later than time t and are not completed by time t). The schedule can be formally described as n functions $x_i^A(t) : [0, \infty) \to [0, m]$; we will not include A in the notation when it is clear in the context. Here $x_i(t)$ denotes the number of machines that are used for job i at time t. It must be that $\sum_{i \in q^A(t)} x_i(t) \leq m$, so that no more than m machines are used. Note that a fractional number of processors can be assigned to each job. If we wish to work in a model where only an integral number of processors can be assigned to a job, we can reduce to the fractional case. Details about the reduction including commentary on other assumptions in the model can be found in Appendix A.

We assume that the online scheduler has access to the speed-up curve for the phase each alive job is in as well as the job's weight. As mentioned in Section 1, this assumption is different from the non-clairvoyant scheduler that is assumed in [17, 23], and is key to our scalable algorithm. However, the scheduler does not know job i's remaining size in its phase, nor does it know the remaining phases' speed-up curves. In other words, the scheduler becomes aware of job i only when it is released at time r_i. The scheduler also becomes aware of job i's weight w_i upon its release. Further, it knows the speed-up curve $\Gamma_i^{\mu'}(t)$ for all $\mu' \leq \mu$ where μ is job i's current phase. In retrospect, it knows $p_i^{\mu'}$, $\mu' \leq \mu - 1$, but not p_i^μ.

Let C_i^μ denote the completion time of the μ_{th} phase of job i. For notional convenience, let $C_i^0 := r_i$. Given $x_i(t)$, the time C_i^μ, $1 \leq \mu \leq \hat{\mu}_i$ is formally defined as the earliest time τ such that

$$\int_{t=C_i^{\mu-1}}^{\tau} \Gamma_i^\mu(x_i(t))\mathtt{dt} \geq p_i^\mu.$$

The completion time C_i is defined as the earliest time when all phases of job i are completed, i.e. $C_i := C_i^{\hat{\mu}_i}$. The weighted flow time (or response time) of job i is defined as $w_i \cdot (C_i - r_i)$ and is a weighted measurement of how long job i waits to be completed since its release time. The total (or equivalently average) weighted flow time of schedule A is $\sum_{i \in [n]} w_i(C_i - r_i)$.

Extension to dynamic speed scaling: In the speed scaling setting, each machine can be run faster or slower over time by changing the power it consumes. In the arbitrary speed up curves setting where all machines are uniform, we are given a power function $P : \mathbb{R}^+ \to \mathbb{R}^+$ where $P(s)$ is the power used by a machine when running at speed s. The power function P is assumed to be non-decreasing and convex. It can be further assumed without loss of generality that P is continuous as shown in [8]. Throughout this paper, we will be concerned with the inverse $Q : \mathbb{R}^+ \to \mathbb{R}^+$ of power function P. In words, $Q(E)$ specifies the speed achieved when the processor is given power E. We will refer to Q as the speed function. One can assume without loss of generality that Q is concave, continuous, and $Q(0) = 0$ [8].

In this extension, a feasible schedule consists of $\{x_i(t)\}_{i=1}^n$, which is defined above, and powers $\{E_i(t)\}_{i=1}^n$ that specify the power consumed by job i at time t. Again, we may optionally assume each $x_i(t)$ is integral using a reduction given in Appendix A. Note that we are assuming that all machines that are processing the same job i consumes the same power. This assumption is justified by the following fact. Consider any infinitesimal interval $[t, t + \mathtt{dt})$, and assume that an energy budget E is given that can be used for job i during the interval. Then the maximum speed at which job i is processed is achieved by distributing the given energy equally among the $\{x_i(t)\}_{i=1}^n$ machines that are assigned to job i, since

Q is concave. Given $\{x_i(t)\}_{i=1}^n$ and $\{E_i(t)\}_{i=1}^n$, job i is processed at a rate of $\Gamma_i^{\mu_i(t)}(x_i(t)) \cdot Q(E_i(t)/x_i(t))$ at time t, where $\mu_i(t)$ is the phase that job i is in at time t. The completion time C_i^μ of the μ_{th} phase of job i is redefined accordingly. That is, the time C_i^μ, $1 \leq \mu \leq \hat{\mu}_i$ is foamily defined as the earliest time τ such that

$$\int_{t=C_i^{\mu-1}}^{\tau} \Gamma_i^\mu(x_i(t))Q(E_i(t)/x_i(t))\mathtt{dt} \geq p_i^\mu,$$

and the completion time of job i, $C_i := C_i^{\hat{\mu}_i}$.

We focus on the objective of minimizing the total weighted flow time plus the total energy consumed,

$$\sum_{i=1}^n w_i(C_i - r_i) + \sum_{i=1}^n \int_{t=0}^{\infty} E_i(t)\mathtt{dt}.$$

In sum, in this paper we are interested in the problem of finding a non-clairvoyant scheduler, that can be specified by $\{x_i(t)\}_{i=1}^n$ and $\{E_i(t)\}_{i=1}^n$, that has access to the speed-up curve of the current phase of each job. The goal is to minimize the total flow time and total power consumption.

3 Algorithm Definition

We present our algorithm for scheduling in the speedup curve model with energy which we call Weighted Latest Arrival Processor Sharing with Energy (WLAPS+E). For the definition of WLAPS+E, we make the following technical yet reasonable assumption regarding Γ and Q.

ASSUMPTION 3.1. *Consider any speed-up curve $\Gamma : \mathbb{R}^+ \to \mathbb{R}^+$ corresponding to some phase of a job. Then for the given speed function $Q : \mathbb{R}^+ \to \mathbb{R}^+$ and for any positive real numbers h and w, there exists $0 < h^* \leq h$ that maximizes*

$$\Upsilon_w(h^*) = \Gamma(h^*) \cdot Q(\frac{w}{h^*}).$$

Further, such h^* can be efficiently computed.

In words, the function $\Upsilon_w(h)$ is the speed achieved by distributing power w equally among h processors. Detailed justification of this assumption can be found in Appendix A. We also note that our algorithm may assign a fractional number of processor to each job. Recall that the case were the algorithm can assign only assign an integral number of processors to each job can be reduced to the fractional case. Again, details appear in Appendix A.

Given a positive real number h, let $y_i^\mu(h)$ be the number of processors less than h that yield the greatest speed for job i in phase μ when using w_i units of power. In other words,

$$y_i^\mu(h) = \operatorname*{argmax}_{0<h'\leq h} \Gamma_i^\mu(h') \cdot Q\left(\frac{w_i}{h'}\right).$$

Recall that we assume $y_i^\mu(h)$ is well defined and efficiently computable (Assumption 3.1). Let $g_i^\mu(h)$ be the highest speed achievable for job i in phase μ when using at most h processors and power w_i.

$$g_i^\mu(h) = \max_{0<h'\leq h} \Gamma_i^\mu(h') \cdot Q\left(\frac{w_i}{h'}\right).$$

Let ε be an arbitrary real number such that $0 < \varepsilon \leq 1/6$. Informally, WLAPS+E shares processors proportionally by weight among the most recently arriving jobs so that the weight of all scheduled jobs is an ε fraction of the total weight in the system. Then, for each job i, WLAPS+E scales back the amount of processors given to i in order to maximize the amount of processing done with w_i units of power.

More formally, let $q^A(t)$ be the set of jobs alive for WLAPS+E at time t and let $W(t) = \sum_{i\in q^A(t)} w_i$. Let $q_i^A(t)$ be the set of jobs alive for WLAPS+E at time t that arrive earlier than job i, including job i where ties are broken arbitrarily but consistently. Let $W_i(t) = \sum_{j\in q_i^A(t)} w_j$. Let $h_i(t) = w_i m/(\varepsilon W(t))$ if $W_i(t) - w_i \geq (1-\varepsilon)W(t)$, let $h_i(t) = 0$ if $W_i(t) < (1-\varepsilon)W(t)$, and let $h_i(t) = (W_i(t) - (1-\varepsilon)W(t))m/(\varepsilon W(t))$ otherwise. Let $\mu_i^A(t)$ be the phase job i is in at time t under WLAPS+E's schedule. For each of job i, WLAPS+E processes i on $y_i^{\mu_i^A(t)}(h_i(t))$ processors using w_i units of power.

Our algorithm is similar to LAPS in that it schedules late arriving jobs, and each of those jobs is allowed to use up to the same number of machines proportioned by job weight. However, our algorithm actually leaves some processors inactive in favor of more energy efficiency. It is somewhat surprising that even in the speed scaling setting, a scalable algorithm can be obtained by putting the same quota on the maximum number of machines that each late arriving job gets.

We will prove the following theorem.

THEOREM 3.1. *For any $0 < \varepsilon \leq 1/6$, WLAPS+E is $(1+6\varepsilon)$-speed, $\frac{5}{\varepsilon^2}$-competitive.*

4 Analysis of WLAPS+E

Here, we analyze WLAPS+E in order to prove Theorem 3.1. Set an input instance I, and consider running the optimal schedule for I using unit speed alongside WLAPS+E using speed $1+6\varepsilon$. For any job i, let C_i^A be the completion time for i under WLAPS+E. For any time $t \geq r_i$, let $A_i(t) = w_i(\min\{C_i^A, t\} - r_i)$ be the accumulated weighted flow time of job i at time t and let $A_i = A_i(C_i^A)$. Let $A(t)$ denote the total energy that WLAPS+E has consumed by time t plus the total accumulated weighted flow time in WLAPS+E's schedule. Finally, let $A = A(\infty)$ be the total weighted flow time plus energy consumption for WLAPS+E. Let $\mathrm{OPT}_i(t)$, OPT_i,

$\text{OPT}(t)$ and OPT be defined similarly for the optimal schedule.

We use a potential function analysis to prove our theorem. We give a potential function $\Phi(t)$ that is almost everywhere differentiable such that $\Phi(0) = \Phi(\infty) = 0$. We will bound the continuous and discrete increases to $A(t)+\Phi(t)$ by a function of OPT. Potential functions are the dominant form of analysis used for analyzing algorithms in the speed scaling setting. See [30] for a recent tutorial on potential functions. Our potential function is somewhat similar to the one used by Edmonds and Pruhs [25] for the formulation of this problem *without* energy, but we have modified the terms in the potential function to account for job weights and our use of the maximum in selecting how many processors a job should use. A significant difference between our potential function and those used in the past is that our potential function takes the functions Γ into consideration. Previous work on the speed up curve model used a reduction from the general setting to a setting where there are only two possible speed up functions $\Gamma(h) = h$ or $\Gamma(h) = 1$. We cannot use this reduction because our algorithm uses the parallelizability of a job phase to make scheduling decisions. By taking the speed up functions into consideration, our potential function captures the full generality of the problem.

Let $p_i^{A\mu}(t)$ and $p_i^{O\mu}(t)$ be the remaining processing time for phase μ of job i at time t for WLAPS+E and the optimal schedule respectively. Let $z_i^{\mu}(t) = \max\{p_i^{A\mu}(t) - p_i^{O\mu}(t), 0\}$. The potential function is composed of one term for each job-phase pair.

$$\Phi_i^{\mu}(t) = \frac{w_i z_i^{\mu}(t)}{g_i^{\mu}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)}.$$

Our potential function is defined as

$$\Phi(t) = \frac{1}{\varepsilon^2} \sum_{i \in q^A(t), \mu} \Phi_i^{\mu}(t).$$

We now focus on changes made to $A(t) + \Phi(t)$ that occur over time. Note that job arrivals have no effect on $A(t) + \Phi(t)$. Further, the completion of jobs by WLAPS+E or the optimal schedule do not cause any increase in $A(t)+\Phi(t)$. This is because when the optimal solution completes a job there is no effect on the potential function. When the algorithm completes a job i the terms corresponding to job i are removed from the potential, and since these terms equal 0, this removal does not affect the potential. For any other job j, the quantity $W_j(t)$ may decrease, but this can only decrease the potential since g_j^{μ} is non-decreasing for any μ. Our entire analysis can focus on the continuous changes to $A(t) + \Phi(t)$.

We first summarize simple facts regarding Γ_i^u, Q and g_i^u in the following section. The continuous changes of $\Phi(t)$ due to WLAPS+E and the optimal scheduler's processing are addressed in Sections 4.2 and 4.3. The changes are aggregated in Section 4.4, yielding Theorem 3.1.

4.1 Simple observations. In this section, we make several simple observations that will be useful in our analysis. We will implicitly use the following observation.

PROPOSITION 4.1. *Consider any continuous and concave function $f : \mathbb{R}^+ \to \mathbb{R}^+$ with $f(0) \geq 0$. Then for any $b \geq a > 0$, we have that $f(a)/a \geq f(b)/b$.*

Proof. By definition of concave functions, it follows that $f(tx+(1-t)y) \geq tf(x)+(1-t)f(y)$ for any real x, y and $t \in [0,1]$. Consider setting $x = 0$, $y = b$ and $t = 1 - \frac{a}{b}$. Then we have $f(a) \geq (1 - \frac{a}{b})f(0) + \frac{a}{b}f(b) \geq \frac{a}{b}f(b)$.

The above proposition immediately gives the following property regarding Q and Γ.

PROPOSITION 4.2. *Let a, b be any reals such that $b \geq a > 0$. Then $Q(b)/Q(a) \leq b/a$. Further for any job i and μ, it holds that $\Gamma_i^{\mu}(b)/\Gamma_i^{\mu}(a) \leq b/a$.*

4.2 Changes in $\Phi(t)$ due to the optimal solution. Fix a time t. We begin by considering the optimal schedule's contribution to $\frac{d}{dt}\Phi(t)$. For each job i, let $\mu_i^O(t)$ be the current phase of job i for the optimal schedule at time t. Let P_i be the number of processors the optimal schedule assigns to i at the current time t and let E_i be the total power used by the optimal schedule to process i at the same time t. We will place each job i into one of four categories and then bound the contribution to $\frac{d}{dt}\Phi(t)$ by jobs in each category. The types of jobs that fit into each category are as follows:

Job Categories
A: $P_i > \frac{w_i m}{(1+2\varepsilon)\varepsilon W_i(t)}$ and $E_i/P_i > \frac{(1+2\varepsilon)\varepsilon W_i(t)}{m}$
B: $P_i > \frac{w_i m}{(1+2\varepsilon)\varepsilon W_i(t)}$ and $E_i/P_i \leq \frac{(1+2\varepsilon)\varepsilon W_i(t)}{m}$
C: $P_i \leq \frac{w_i m}{(1+2\varepsilon)\varepsilon W_i(t)}$ and $E_i > w_i$
D: $P_i \leq \frac{w_i m}{(1+2\varepsilon)\varepsilon W_i(t)}$ and $E_i \leq w_i$

Note z_i^{μ} is the only factor of Φ_i^{μ} affected by continuous processing from WLAPS+E or the optimal schedule. The optimal schedule may affect that factor by at most the rate it decreases $p_i^{O\mu}$. We may assume the optimal schedule is giving each processor an equal amount of power (E_i/P_i), because Q is concave. Therefore, the optimal schedule's contribution to $\frac{d}{dt}\Phi_i^{\mu}(t)$ is at most

$$\frac{w_i \Gamma_i^{\mu_i^O(t)}(P_i)\, Q\left(E_i/P_i\right)}{g_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)}.$$

Categories A and B: Suppose job i is in category A or B. By definition of the function $g_i^{\mu_i^O(t)}(\cdot)$, first we observe that,

$$\Gamma_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right) Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right) \le g_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right).$$

Using this fact we have the following. The second inequality follows by Proposition 4.2 and the assumption that job i is in category A or B and therefore $P_i > w_i m/((1+2\varepsilon)\varepsilon W_i(t))$.

$$\begin{aligned}
&\frac{w_i \Gamma_i^{\mu_i^O(t)}(P_i)\, Q(E_i/P_i)}{g_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)} \\
&\le w_i \cdot \frac{\Gamma_i^{\mu_i^O(t)}(P_i)}{\Gamma_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)} \cdot \frac{Q(E_i/P_i)}{Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right)} \\
&\le \frac{(1+2\varepsilon)\varepsilon W_i(t) P_i}{m} \cdot \frac{Q(E_i/P_i)}{Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right)}
\end{aligned}$$

If job i is in category A, then we have the following. The first inequality follows from Proposition 4.2 and the assumption that job i is in category A and therefore $E_i/P_i > (1+2\varepsilon)\varepsilon W_i(t)/m$.

$$\begin{aligned}
&\frac{(1+2\varepsilon)\varepsilon W_i(t) P_i}{m} \cdot \frac{Q(E_i/P_i)}{Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right)} \\
&\le \frac{(1+2\varepsilon)\varepsilon W_i(t) P_i}{m} \cdot \frac{E_i m}{(1+2\varepsilon)\varepsilon W_i(t) P_i} \\
&\le E_i.
\end{aligned}$$

The total contribution to $\frac{d}{dt}\Phi(t)$ by category A jobs is at most $\frac{1}{\varepsilon^2}\frac{d}{dt}\text{OPT}(t)$, because $\sum_{i\in q^O(t)} E_i$ is the total power used by the optimal solution at time t.

Now suppose i is in category B, so $E_i/P_i \le (1+2\varepsilon)\varepsilon W_i(t)/m$. Function Q is increasing, so we observe that $Q(E_i/P_i) \le Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right)$. Hence,

$$\begin{aligned}
&\frac{(1+2\varepsilon)\varepsilon W_i(t) P_i}{m} \cdot \frac{Q(E_i/P_i)}{Q\left((1+2\varepsilon)\varepsilon W_i(t)/m\right)} \\
&\le \frac{(1+2\varepsilon)\varepsilon W_i(t) P_i}{m} \\
&\le \frac{(1+2\varepsilon)\varepsilon W(t) P_i}{m}.
\end{aligned}$$

The optimal schedule has a total of m processors to work with, so the total contribution to $\frac{d}{dt}\Phi(t)$ by category B jobs is at most $(1/\varepsilon^2)\sum_{i\in q^O(t)} \frac{(1+2\varepsilon)\varepsilon W(t) P_i}{m} = (1+2\varepsilon)W(t)/\varepsilon \sum_{i\in q^O(t)} \frac{P_i}{m} \le (1+2\varepsilon)W(t)/\varepsilon$. Here $q^O(t)$ denotes the set of alive jobs at time t in the optimal schedule.

Categories C and D: Suppose job i is in category C or D. By definition of the function $g_i^{\mu_i^O(t)}(\cdot)$ and the assumption that $P_i \le w_i m/((1+2\varepsilon)\varepsilon W_i(t))$, we have that,

$$\Gamma_i^{\mu_i^O(t)}(P_i)\, Q(w_i/P_i) \le g_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)$$

Therefore,

$$\begin{aligned}
&\frac{w_i \Gamma_i^{\mu_i^O(t)}(P_i)\, Q(E_i/P_i)}{g_i^{\mu_i^O(t)}\left(w_i m/((1+2\varepsilon)\varepsilon W_i(t))\right)} \\
&\le w_i \cdot \frac{\Gamma_i^{\mu_i^O(t)}(P_i)}{\Gamma_i^{\mu_i^O(t)}(P_i)} \cdot \frac{Q(E_i/P_i)}{Q(w_i/P_i)} \\
&= w_i \cdot \frac{Q(E_i/P_i)}{Q(w_i/P_i)}.
\end{aligned}$$

Suppose i is in category C. We see the following. The first inequality follows from Proposition 4.2 and the assumption that $E_i > w_i$.

$$w_i \cdot \frac{Q(E_i/P_i)}{Q(w_i/P_i)} \le w_i \cdot \frac{E_i}{w_i} = E_i$$

The total contribution to $\frac{d}{dt}\Phi(t)$ by category C jobs is at most $\frac{1}{\varepsilon^2}\frac{d}{dt}\text{OPT}(t)$. Again, this fact is because $\sum_{i\in q^O(t)} E_i$ is the total power used by the optimal solution at time t.

Finally, suppose job i is in category D. Knowing that Q is increasing and $E_i \le w_i$ we have

$$w_i \cdot \frac{Q(E_i/P_i)}{Q(w_i/P_i)} \le w_i.$$

The total contribution to $\frac{d}{dt}\Phi(t)$ by category D jobs is at most $\frac{1}{\varepsilon^2}\frac{d}{dt}\text{OPT}(t)$. This is because $\sum_{i\in q^O(t)} w_i$ is the increase in the weighted flow objective for the optimal solution at time t.

In sum, the optimal scheduler's contribution to $\frac{d}{dt}\Phi(t)$ is at most

$$(2+\frac{1}{\varepsilon})W(t) + \frac{3}{\varepsilon^2}\frac{d}{dt}\text{OPT}(t). \tag{4.1}$$

4.3 Change in $\Phi(t)$ due to the algorithm. Now we discuss the contributions to $\frac{d}{dt}\Phi(t)$ from WLAPS+E. Recall that $q^O(t)$ is the set of jobs alive for the optimal schedule at time t. For each job $i \in q^A(t)$, we note $z_i^{\mu_i^A(t)}(t)$ is positive if $i \notin q^O(t)$. Hence $z_i^{\mu_i^A(t)}(t)$ decreases for all jobs $i \notin q^O(t)$ that WLAPS+E processes at time t. Recall we are running WLAPS+E at speed $1+6\varepsilon$.

Let $q^{A'}(t)$ be the set of jobs i where $h_i(t) = w_i m/(\varepsilon W(t))$. For every job $i \in q^{A'}(t)$, we have $W_i(t) \geq (1-\varepsilon)W(t)$.

Let Δ_q be the contribution to $\frac{d}{dt}\Phi(t)$ from WLAPS+E processing jobs in $q^{A'}(t)$. We see that Δ_q is less than,

$$
\begin{aligned}
&-\frac{(1+6\varepsilon)}{\varepsilon^2}\sum_{i\in q^{A'}(t)\setminus q^O(t)}\frac{w_i g_i^{\mu_i^A(t)}(w_i m/(\varepsilon W_i(t)))}{g_i^{\mu_i^A(t)}(w_i m/((1+2\varepsilon)\varepsilon W_i(t)))}\\
\leq&-\frac{(1+6\varepsilon)}{\varepsilon^2}\sum_{i\in q^{A'}(t)\setminus q^O(t)}\frac{w_i g_i^{\mu_i^A(t)}(w_i m/(\varepsilon W_i(t)))}{g_i^{\mu_i^A(t)}(w_i m/((1+2\varepsilon)\varepsilon(1-\varepsilon)W(t)))}\\
\leq&-\frac{(1+6\varepsilon)}{\varepsilon^2}\sum_{i\in q^{A'}(t)\setminus q^O(t)}\frac{w_i g_i^{\mu_i^A(t)}(w_i m/(\varepsilon W_i(t)))}{g_i^{\mu_i^A(t)}(w_i m/(\varepsilon W(t)))}\\
\leq&-\frac{(1+6\varepsilon)}{\varepsilon^2}\sum_{i\in q^{A'}(t)\setminus q^O(t)} w_i\\
=&-\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\sum_{i\in q^{A'}(t)\setminus q^O(t)}\frac{h_i(t)}{m}
\end{aligned}
$$

[Since $h_i(t) = w_i m/(\varepsilon W(t))$ for $i \in q^{A'}(t)$]

The second line holds since $g_i^{\mu_i^A(t)}$ is an non-decreasing function. The third line follows from the assumption that $0 < \varepsilon \leq 1/6$ in addition to $g_i^{\mu_i^A(t)}$ being non-decreasing.

There may be a single job j with $0 < h_j(t) < w_j m/(\varepsilon W(t))$. Let Δ_j be the contribution to $\frac{d}{dt}\Phi(t)$ due to WLAPS+E processing such a job j. We show that $\Delta_j \leq -\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\sum_{i\in\{j\}\setminus q^O(t)}\frac{h_i(t)}{m}$. Assume that $j \notin O(t)$, since the inequality trivially holds otherwise. Let $k > 1$ be such that $h_j(t) = (1/k)w_j m/(\varepsilon W(t))$. Observe by Proposition 4.2 that $g_j^{\mu_j^A(t)}((1/k)w_j m/(\varepsilon W(t))) \geq (1/k)g_j^{\mu_j^A(t)}(w_j m/(\varepsilon W(t)))$. A similar algebra to that just shown gives $\Delta_j \leq -\frac{1}{k}\cdot\frac{1+6\varepsilon}{\varepsilon^2}\cdot w_j = -\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\cdot\frac{h_j(t)}{m}$. Hence the total contribution to $\frac{d}{dt}\Phi(t)$ from WLAPS+E's processing is

(4.2)

$$
\begin{aligned}
&\Delta_q + \Delta_j\\
\leq&-\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\sum_{i\in q^A(t)\cup\{j\}\setminus q^O(t)}\frac{h_i(t)}{m}\\
\leq&-\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\Big[\sum_{i\in q^A(t)\cup\{j\}}\frac{h_i(t)}{m}-\sum_{i\in q^O(t)}\frac{h_i(t)}{m}\Big]\\
\leq&-\frac{1+6\varepsilon}{\varepsilon}\cdot W(t)\Big(1-\sum_{i\in q^O(t)}\frac{w_i}{\varepsilon W(t)}\Big)\\
=&-\frac{1+6\varepsilon}{\varepsilon}W(t)+\frac{1+6\varepsilon}{\varepsilon^2}\frac{d}{dt}\mathrm{OPT}(t)
\end{aligned}
$$

The second line holds since WLAPS+E distributes machines among jobs in such a way that $\sum_{i\in q^A(t)\cup\{j\}}\frac{h_i(t)}{m} = m$ and $h_i(t) \leq \frac{w_i m}{\varepsilon W(t)}$.

4.4 Final analysis: putting the pieces all together

In this section, we complete the analysis by aggregating all changes of $A(t)$ and $\Phi(t)$. Recall that when jobs arrive or are completed, $\Phi(t)$ does not increase. Hence, there are no positive discontinuous changes in $\Phi(t)$. Observe $\frac{d}{dt}A(t) \leq 2W(t)$ because the increase in WLAPS+E's weighted flow objective is $W(t)$ and the most energy the algorithm uses at this time is $W(t)$. By integrating over time $\frac{d}{dt}A(t)$, and (4.1) and (4.2), the contributions to $\frac{d}{dt}\Phi(t)$ from WLAPS+E and the optimal scheduler's processing, we have

$$
\begin{aligned}
A \leq&\int_{t=0}^{\infty}\frac{d}{dt}A(t)+\frac{d}{dt}\Phi(t)dt\\
\leq&\int_{t=0}^{\infty}2W(t)+(4.1)+(4.2)dt\\
\leq&\int_{t=0}^{\infty}2W(t)+(2+\frac{1}{\varepsilon})W(t)+\frac{3}{\varepsilon^2}\frac{d}{dt}\mathrm{OPT}(t)\\
&\qquad-\frac{1+6\varepsilon}{\varepsilon}W(t)+\frac{1+6\varepsilon}{\varepsilon^2}\frac{d}{dt}\mathrm{OPT}(t)dt\\
\leq&(\frac{6}{\varepsilon}+\frac{4}{\varepsilon^2})\mathrm{OPT}\leq\frac{5}{\varepsilon^2}\mathrm{OPT}\quad[\text{Since } 0<\varepsilon\leq 1/6].
\end{aligned}
$$

Here the first inequality follows, since $\Phi(0) = \Phi(\infty) = 0$, and there is no positive discontinuous change of $\Phi(t)$. This completes the proof of Theorem 3.1. By scaling ε appropriately, we obtain Theorem 1.1.

Acknowledgement. The authors would like to thank an anonymous reviewer for giving us a reference to [43].

References

[1] Amd powernow! technology. *http://www.amd.com/us/products/technologies/amd-powernow-technology/pages/amd-powernow-technology.aspx*.

[2] Intel speedstep technology. *http://www.intel.com/cd/channel/reseller/asmona/eng/203838.htm*.

[3] Kunal Agrawal, Yuxiong He, and Charles E. Leiserson. Adaptive work stealing with parallelism feedback. In *PPOPP*, pages 112–120, 2007.

[4] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[5] Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

[6] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. In *LATIN*, pages 240–251, 2008.

[7] Nikhil Bansal, Ho-Leung Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for speed bounded processors. In *ICALP (1)*, pages 409–420, 2008.

[8] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, 2009.

[9] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *ICALP (1)*, pages 144–155, 2009.

[10] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007. Preliminary version in *FOCS* 2004.

[11] Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. Better scalable algorithms for broadcast scheduling. In *ICALP*, 2010.

[12] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009. Preliminary version in *SODA* 2007.

[13] David Brooks, Pradip Bose, Stanley Schuster, Hans M. Jacobson, Prabhakar Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor V. Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[14] Ho-Leung Chan, Joseph Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Optimizing throughput and energy in online deadline scheduling. *ACM Transactions on Algorithms*, 6(1), 2009. Preliminary version in *SODA* 2007.

[15] Ho-Leung Chan, Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence W. H. Wong. Energy efficient online deadline scheduling. In *SODA*, pages 795–804, 2007.

[16] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 255–264, 2009.

[17] Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

[18] Ho-Leung Chan, Tak Wah Lam, and Rongbin Li. Tradeoff between energy and throughput for online deadline scheduling. In *WAOA*, pages 59–70, 2010.

[19] Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Nonclairvoyant speed scaling for weighted flow time. In *ESA (1)*, pages 23–35, 2010.

[20] Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Scheduling for weighted flow time and energy with rejection penalty. In *STACS*, 2011.

[21] Julita Corbalán and Jesús Labarta. Improving processor allocation through run-time measured efficiency. In *IPDPS*, page 74, 2001.

[22] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000. Preliminary version in *STOC* 1999.

[23] Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *J. Scheduling*, 6(3):231–250, 2003.

[24] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work. In *SODA*, 2011.

[25] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, pages 685–692, 2009.

[26] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA '10: 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[27] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *SODA*, pages 1242–1253, 2012.

[28] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneous processors. In *ICALP (1)*, pages 312–323, 2010.

[29] Xin Han, Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Deadline scheduling and power management for speed bounded processors. *Theor. Comput. Sci.*, 411(40-42):3587–3600, 2010. Preliminary version in *MAPSP* 2009.

[30] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42:83–97, June 2011.

[31] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. Preliminary version in *FOCS* 1995.

[32] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Commun. ACM*, 33(5):539–543, 1990.

[33] Tak Wah Lam, Lap-Kei Lee, Hing-Fung Ting, Isaac Kar-Keung To, and Prudence W. H. Wong. Sleep with guilt and work faster to minimize flow plus energy. In *ICALP (1)*, pages 665–676, 2009.

[34] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Energy efficient deadline scheduling in two processor systems. In *ISAAC*, pages 476–487, 2007.

[35] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *SPAA*, pages 256–264, 2008.

[36] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Improved multi-processor scheduling for flow time and energy. *J. Scheduling*, 15(1):105–116, 2012.

[37] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007. Preliminary version in *STOC* 1997.

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

[39] Thu D. Nguyen, Raj Vaswani, and John Zahorjan. Using runtime measured workload characteristics in parallel processor scheduling. In *JSSPP*, pages 155–174, 1996.

[40] Gerald Sabin, Matthew Lang, and P. Sadayappan. Moldable parallel job scheduling using job efficiency: An itera-

tive approach. In *JSSPP*, pages 94–114, 2006.

[41] Srividya Srinivasan, Vijay Subramani, Rajkumar Kettimuthu, Praveen Holenarsipur, and P. Sadayappan. Effective selection of partition sizes for moldable scheduling of parallel jobs. In *HiPC*, pages 174–183, 2002.

[42] Sudha Srinivasan, Sriram Krishnamoorthy, and P. Sadayappan. A robust scheduling strategy for moldable scheduling of parallel jobs. In *CLUSTER*, pages 92–99, 2003.

[43] Hongyang Sun, Yuxiong He, and Wen-Jing Hsu. Speed scaling for energy and performance with instantaneous parallelism. In *TAPAS*, pages 240–251, 2011.

[44] Nathan R. Tallent and John M. Mellor-Crummey. Effective performance measurement and analysis of multithreaded applications. In *PPOPP*, pages 229–240, 2009.

A Integral Processor Assignments and an Assumption on Γ and Q

Consider Assumption 3.1. To see why the assumption is necessary, consider $\Gamma(h) = h^\alpha$ and $Q(h) = h^\beta$ where $0 < \alpha < \beta \leq 1$. Then $\Upsilon_w(h) = (1/h)^{\beta-\alpha}$ and there exists no $0 < h^* \leq h$ that maximizes $\Upsilon_w(h^*)$. In this case, the phase of speedup curve Γ can be finished instantaneously by using an infinitesimal number of processors!

We now discuss why such a degenerate case does not occur in practice. Although a speedup curve Γ is defined over $\mathbb{R}^+$, assigning a fractional number of machines to a job is not permitted when there are only finite number of machines in practice. In fact, it would better capture reality to assume that Γ is defined only over non-negative integers. Then one can extend the domain to $\mathbb{R}^+$ by making $\Gamma(h)$ a piecewise linear function as follows. For a non-integer positive number $h = h' + \lambda$, where h' is an integer and $0 < \lambda < 1$,

$$\Gamma(h) = (1-\lambda)\Gamma(h') + \lambda\Gamma(h'+1).$$

This extension implies that during an infinitesimal interval of length $\mathtt{dt}$, the job in consideration is processed on h' processors for $(1-\lambda)\mathtt{dt}$ units of time and on $h'+1$ machines for $\lambda\mathtt{dt}$ units of time. Observe that the average number of machines used during the interval is exactly h. Then we say that the job can be scheduled integrally compatible. It is not difficult to see that one can find a schedule where all alive jobs can be scheduled integrally compatible: Consider an infinitesimal interval $[t, t+\mathtt{dt})$. Initially give $\lfloor x_i(t) \rfloor$ processors to each job i. Consider jobs in any order, and the remaining machines one by one. Each of the remaining machines is available during $[t, t+\mathtt{dt})$. Think of job i as having size $x_i(t) - \lfloor x_i(t) \rfloor$. Schedule job i on the machine in consideration, and if necessary on the next remaining machine. Each machine is used from the earliest time t to $t + \mathtt{dt}$.

Then $\Upsilon_w(\cdot)$ can be naturally defined over a fractional value $h = h' + \lambda$ as follows:

$$\Upsilon_w(h) = \begin{cases} \lambda\Gamma(1)Q(\frac{w}{\lambda}) & \text{if } 0 < h < 1 \\ (1-\lambda)\Gamma(h')Q(\frac{wh'}{h} \cdot \frac{1}{h'}) & \text{if } h \geq 1 \\ \quad +\lambda\Gamma(h'+1)Q(\frac{w(h'+1)}{h} \cdot \frac{1}{h'+1}) & \end{cases}$$

In words, during an interval $[t, t+\mathtt{dt})$, for $0 < h < 1$, we run the job on a single machine with power $\frac{w}{\lambda}$ for $\lambda\mathtt{dt}$ time steps. Observe that the average power dedicated to the job per a unit time is exactly $\lambda\frac{w}{\lambda} = w$. For $h \geq 1$, we run the job on h' machines, each with power $\frac{w}{h}$ for $(1-\lambda)\mathtt{dt}$ time steps, and on $h'+1$ machines, each with the same power $\frac{w}{h}$ for $\lambda\mathtt{dt}$ time steps. Likewise, the average power usage per a unit time is $(1-\lambda)h'\frac{w}{h} + \lambda(h'+1)\frac{w}{h} = w$. Hence the schedule suggested by the above extension of Γ and Υ preserves the power and the number of machines used in addition to being feasible.

Further, observe that the above semantics gives the same definition of $\Upsilon_w(h) = \Gamma(h) \cdot Q(\frac{w}{h})$. Indeed, when $0 < h < 1$,

$$\Gamma(h) \cdot Q(\frac{w}{h}) = \Gamma(\lambda) \cdot Q(\frac{w}{\lambda}) = \lambda\Gamma(1) \cdot Q(\frac{w}{h}).$$

Also when $h \geq 1$,

$$\Gamma(h) \cdot Q(\frac{w}{h}) = \Big((1-\lambda)\Gamma(h') + \lambda\Gamma(h'+1)\Big)Q(\frac{w}{h}).$$

Now when considering the extension of Γ defined only over non-negative integers, note that for $h \in (0, 1]$, $\Upsilon_w(h) = h\Gamma(1)Q(\frac{w}{h})$ is maximized when $h = 1$, since Q is concave and $Q(0) = 0$; see Proposition 4.2. Further, for all reasonably-behaving functions Q (e.g., differentiable), Υ_w has a maximum value over any compact set in $\mathbb{R}^+$. Hence we conclude that there always exists $0 < h^* \leq h$ such that $\Upsilon_w(h^*)$ is maximized.