Scalably Scheduling Power-Heterogeneous Processors[*]

Anupam Gupta[†] Ravishankar Krishnaswamy[†] Kirk Pruhs[‡]

Abstract

We show that a natural online algorithm for scheduling jobs on a heterogeneous multiprocessor, with arbitrary power functions, is scalable for the objective function of weighted flow plus energy.

1 Introduction

Many prominent computer architects believe that architectures consisting of heterogeneous processors/cores, such as the STI Cell processor, will be the dominant architectural design in the future [8, 13, 12, 17, 18]. The main advantage of a heterogeneous architecture, relative to an architecture of identical processors, is that it allows for the inclusion of processors whose design is specialized for particular types of jobs, and for jobs to be assigned to a processor best suited for that job. Most notably, it is envisioned that these heterogeneous architectures will consist of a small number of high-power high-performance processors for critical jobs, and a larger number of lower-power lower-performance processors for less critical jobs. Naturally, the lower-power processors would be more energy efficient in terms of the computation performed per unit of energy expended, and would generate less heat per unit of computation. For a given area and power budget, heterogeneous designs can give significantly better performance for standard workloads [8, 17]; Emulations in [12] suggest a figure of 40% better performance, and emulations in [18] suggest a figure of 67% better performance. Moreover, even processors that were designed to be homogeneous, are increasingly likely to be heterogeneous at run time [8]: the dominant underlying cause is the increasing variability in the fabrication process as the feature size is scaled down (although run time faults will also play a role). Since manufacturing yields would be unacceptably low if every processor/core was required to be perfect, and since there would be significant performance loss from derating the entire chip to the functioning of the least functional processor (which is what would be required in order to attain processor homogeneity), some processor heterogeneity seems inevitable in chips with many processors/cores.

The position paper [8] identifies three fundamental challenges in scheduling heterogeneous multiprocessors: (1) the OS must discover the status of each processor, (2) the OS must discover the resource demand of each job, and (3) given this information about processors and jobs, the OS must match jobs to processors as well as possible. In this paper, we address this third fundamental

[*]A preliminary version of this paper appeared in ICALP 2010

[†]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Supported in part by NSF award CCF-0729022 and an Alfred P. Sloan Fellowship.

[‡]Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA. Supported in part by NSF grants CNS-0325353, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

challenge. In particular, we assume that different jobs are of differing importance, and we study how to assign these jobs to processors of varying power and varying energy efficiency, so as to achieve the best possible trade-off between energy and performance.

Formally, we assume that a collection of jobs arrive in an online fashion over time. When a job j arrives in the system, the system is able to discover a *size* $p_j \in \mathbb{R}_{>0}$, as well as a *importance/weight* $w_j \in \mathbb{R}_{>0}$, for that job. The importance w_j specifies an upper bound on the amount of energy that the system is allowed to invest in running j to reduce j's flow by one unit of time (assuming that this energy investment in j doesn't decrease the flow of other jobs)—hence jobs with high weight are more important, since higher investments of energy are permissible to justify a fixed reduction in flow. Furthermore, we assume that the system knows the allowable speeds for each processor, and the system also knows the power used when each processor is run at its set of allowable speeds. We make no real restrictions on the allowable speeds, or on the power used for these speeds.[1] The online scheduler has three component policies:

Job Selection: Determines which job to run on each processor at any time.

Speed Scaling: Determines the speed of each processor at each time.

Assignment: When a new job arrives, it determines the processor to which this new job is assigned.

The objective we consider is that of *weighted flow plus energy*. The rationale for this objective function is that the optimal schedule under this objective gives the best possible weighted flow for the energy invested, and increasing the energy investment will not lead to a corresponding reduction in weighted flow (intuitively, it is not possible to speed up a collection of jobs with an investment of energy proportional to these jobs' importance).

We consider the following natural online algorithm that essentially adopts the job selection and speed scaling algorithms from the uniprocessor algorithm in [5], and then greedily assigns the jobs based on these policies.

Job Selection: Highest Density First (HDF)

Speed Scaling: The speed is set so that the power is the fractional weight of the unfinished jobs.

Assignment: A new job is assigned to the processor that results in the least increase in the projected future weighted flow, assuming the adopted speed scaling and job selection policies, and ignoring the possibility of jobs arriving in the future.

Our main result is then:

Theorem 1.1 *This online algorithm is scalable for scheduling jobs on a heterogeneous multiprocessor with arbitrary power functions to minimize the objective function of weighted flow plus energy.*

In this context, *scalable* means that if the adversary can run processor i at speed s and power $P(s)$, the online algorithm is allowed to run the processor at speed $(1+\epsilon)s$ and power $P(s)$, and then for all inputs, the online cost is bounded by $O(f(\epsilon))$ times the optimal cost. Intuitively, a scalable algorithm can handle almost the same load as optimal; for further elaboration, see [20, 19]. Theorem 1.1 extends theorems showing similar results for weighted flow plus energy on a uniprocessor [5, 2],

[1]So the processors may or may not be speed scalable, the speeds may be continuous or discrete or a mixture, the static power may or may not be negligible, the dynamic power may or may not satisfy the cube root rule, etc.

and for weighted flow on a multiprocessor without power considerations [9]. As scheduling on identical processors with the objective of total flow, and scheduling on a uniprocessor with the objective of weighted flow, are special cases of our problem, constant competitiveness is not possible without some sort of resource augmentation [16, 3].

Our analysis is an amortized local-competitiveness argument. As is usually the case with such arguments, the main technical hurdle is to discover the "right" potential function. The most natural straw-man potential function to try is the sum over all processors of the single processor potential function used in [5]. While one can prove constant competitiveness with this potential in some special cases (e.g. where for each processor the allowable speeds are the non-negative reals, and the power satisfies the cube-root rule), one can not prove constant competitiveness for general power functions with this potential function. The reason for this is that the uniprocessor potential function from [5] is not sufficiently accurate. Specifically, one can construct configurations where the adversary has finished all jobs, and where the potential is much higher than the remaining online cost. This did not mess up the analysis in [5] because to finish all these jobs by this time the adversary would have had to run very fast in the past, wasting a lot of energy, which could then be used to pay for this unnecessarily high potential. But since we consider multiple processors, the adversary may have no jobs left on a particular processor simply because it assigned these jobs to a different processor, and there may not be a corresponding unnecessarily high adversarial cost that can be used to pay for this unnecessarily high potential.

Thus, the main technical contribution in this paper is a seemingly more accurate potential function expressing the additional cost required to finish one collection of jobs compared to another collection of jobs. Our potential function is arguably more transparent than the one used in [5], and we expect that this potential function will find future application in the analysis of other power management algorithms.

In section 3, we show that a similar online algorithm is $O(1/\epsilon)$-competitive with $(1+\epsilon)$-speedup for *unweighted* flow plus energy. We also remark that when the power functions $P_i(s)$ are restricted to be of the form s^{α_i}, our algorithms give a $O(\alpha^2)$-competitive algorithm (with no resource augmentation needed) for the problem of minimizing weighted flow plus energy, and an $O(\alpha)$-competitive algorithm for minimizing the unweighted flow plus energy, where $\alpha = \max_i \alpha_i$.

1.1 Related Results

Let us first consider previous work for the case of a single processor, with unbounded speed, and a polynomially bounded power function $P(s) = s^{\alpha}$. [21] gave an efficient offline algorithm to find the schedule that minimizes average flow subject to a constraint on the amount of energy used, in the case that jobs have unit work. [1] introduced the objective of flow plus energy and gave a constant competitive algorithm for this objective in the case of unit work jobs. [6] gave a constant competitive algorithm for the objective of weighted flow plus energy. The competitive ratio was improved by [15] for the unweighted case using a potential function specifically tailored to integer flow. [4] extended the results of [6] to the bounded speed model, and [10] gave a *nonclairvoyant* algorithm that is $O(1)$-competitive.

Still for a single processor, dropping the assumptions of unbounded speed and polynomially-bounded power functions, [5] gave a 3-competitive algorithm for the objective of unweighted flow plus energy, and a 2-competitive algorithm for fractional weighted flow plus energy, both in the uniprocessor case for a large class of power functions. The former analysis was subsequently improved by [2] to show 2-competitiveness, along with a matching lower bound.

Now for multiple processors: [14] considered the setting of multiple *homogeneous* processors, where the allowable speeds range between zero and some upper bound, and the power function is polynomial in this range. They gave an algorithm that uses a variant of round-robin for the assignment policy, and job selection and speed scaling policies from [6], and showed that this algorithm is scalable for the objective of (unweighted) flow plus energy. Subsequently, [11] showed that a randomized machine selection algorithm is scalable for weighted flow plus energy (and even more general objective functions) in the setting of polynomial power functions. Both these algorithms provide non-migratory schedules and compare their costs with optimal solutions which could even be migratory. In comparison, as mentioned above, for the case of polynomial power functions, our techniques can give a deterministic constant-competitive online algorithm for non-migratory weighted flow time plus energy. (Details appear in the final version.)

In non-power-aware settings, the paper most relevant to this work is that of [9], which gives a scalable online algorithm for minimizing weighted flow on unrelated processors. Their setting is even more demanding, since they allow the processing requirement of the job to be processor dependent (which captures a type of heterogeneity that is orthogonal to the performance energy-efficiency heterogeneity that we consider in this paper). Our algorithm is based on the same general intuition as theirs: they assign each new job to the processor that would result in the least increment in future weighted flow (assuming HDF is used for job selection), and show that this online algorithm is scalable using an amortized local competitiveness argument. However, it is unclear how to directly extend their potential function to our power-aware setting; we had success only in the case that each processor had allowable speed-power combinations lying in $\{(0,0),(s_i,P_i)\}$.

1.2 Preliminaries

1.2.1 Scheduling Basics.

We consider only non-migratory schedules, which means that no job can ever run on one processor, and later run on some other processor. In general, migration is undesirable as the overhead can be significant. We assume that preemption is allowed, that is, that jobs may be suspended, and restarted later from the point of suspension. It is clear that if preemption is not allowed, bounded competitiveness is not obtainable. The speed is the rate at which work is completed; a job j with size p_j run at a constant speed s completes in $\frac{p_j}{s}$ seconds. A job is completed when all of its work has been processed. The flow of a job is the completion time of the job minus the release time of the job. The weighted flow of a job is the weight of the job times the flow of the job. For a $t \geq r_j$, let $p_j(t)$ be the remaining unprocessed work on job j at time t. The fractional weight of job j at this time is $w_j \frac{p_j(t)}{p_j}$. The fractional weighted flow of a job is the integral over times between the job's release time and its completion time of its fractional weight at that time. The density of a job is its weight divided by its size. The job selection policy Highest Density First (HDF) always runs the job of highest density. The inverse density of a job is its size divided by its weight.

1.2.2 Power Functions.

The power function for processor i is denoted by $P_i(s)$, and specifies the power used when processor is run at speed s. We essentially allow any reasonable power function. However, we do require the following minimal conditions on each power function, which we adopt from [5]. We assume that the allowable speeds are a countable collection of disjoint subintervals of $[0,\infty)$. We assume that all the intervals, except possibly the rightmost interval, are closed on both ends. The rightmost interval may be open on the right if the power $P_i(s)$ approaches infinity as the speed s approaches the rightmost endpoint of that interval. We assume that P_i is non-negative, and P_i is continuous and differentiable on all but countably many points. We assume that either there is a maximum allowable speed T, or that the limit inferior of $P_i(s)/s$ as s approaches infinity is not zero (if this condition doesn't hold then, then the optimal speed scaling policy is to run at infinite speed). Using transformations specified in [5], we may assume without loss of generality that the power functions satisfy the following properties: P is continuous and differentiable, $P(0) = 0$, P is strictly increasing, P is strictly convex, and P is unbounded. We use Q_i to denote P_i^{-1}; i.e., $Q_i(y)$ gives us the speed that we can run processor i at, if we specify a limit of y.

1.2.3 Local Competitiveness and Potential Functions.

Finally, let us quickly review amortized local competitiveness analysis on a single processor. Consider an objective G. Let $G_A(t)$ be the increase in the objective in the schedule for algorithm A at time t. So when G is fractional weighted flow plus energy, $G_A(t)$ is $P_A^t + w_A^t$, where P_A^t is the power for A at time t and w_A^t is the fractional weight of the unfinished jobs for A at time t. Let OPT be the offline adversary that optimizes G. A is locally c-competitive if for all times t, if $G_A(t) \le c \cdot G_{OPT}(t)$. To prove A is $(c + d)$-competitive using an amortized local competitiveness argument, it suffices to give a potential function $\Phi(t)$ such that the following conditions hold (see for example [19]).

Boundary condition: Φ is zero before any job is released and Φ is non-negative after all jobs are finished.

Completion condition: Φ does not increase due to completions by either A or OPT.

Arrival condition: Φ does not increase more than $d \cdot OPT$ due to job arrivals.

Running condition: At any time t when no job arrives or is completed,

$$G_A(t) + \frac{d\Phi(t)}{dt} \le c \cdot G_{OPT}(t) \tag{1}$$

The sufficiency of these conditions for proving $(c+d)$-competitiveness follows from integrating them over time.

2 Weighted Flow

Our goal in this section is to prove Theorem 1.1. We first show that the online algorithm is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for the objective of *fractional* weighted flow plus energy. Theorem 1.1 then follows since HDF is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for fixed processor speeds [7] for the objective of (integer) weighted flow.

Let OPT be some optimal schedule minimizing fractional weighted flow. Let $w_{a,i}^t(q)$ denote the total fractional weight of jobs in processor i's queue that have an inverse density of at least q. Let $w_{a,i}^t := w_{a,i}^t(0)$ be the total fractional weight of unfinished jobs in the queue. Let $w_a^t := \sum_i w_{a,i}^t$ be the total fractional weight of unfinished jobs in all queues. Let $w_{o,i}^t(q)$, $w_{o,i}^t$, and w_o^t be similarly defined for OPT. When the time instant being considered is clear, we drop the superscript of t from all variables.

We assume that once OPT has assigned a job to some processor, it runs the BCP algorithm [5] for job selection and speed scaling—i.e., it sets the speed of the i^{th} processor to $Q_i(w_{o,i})$, and hence the i^{th} processor uses power $W_{o,i}$, and uses HDF for job selection. We can make such an assumption because the results of [5] show that the fractional weighted flow plus energy of the schedule output by this algorithm is within a factor of two of optimal. Therefore, the only real difference between OPT and the online algorithm is the assignment policy.

2.1 The Assignment Policy

To better understand the online algorithm's assignment policy, define the "shadow potential" for processor i at time t to be

$$\widehat{\Phi}_{a,i}(t) = \int_{q=0}^{\infty} \int_{x=0}^{w^t_{a,i}(q)} \frac{x}{Q_i(x)}\, dx\, dq \tag{2}$$

The shadow potential captures (up to a constant factor) the total fractional weighted flow to serve the current set of jobs if no jobs arrive in the future. Based on this, the online algorithm's assignment policy can alternatively be described as follows:

Assignment Policy. When a new job with size p_j and weight w_j arrives at time t, the assignment policy assigns it to a processor which would cause the smallest increase in the shadow potential; i.e. a processor minimizing

$$\begin{aligned} &\int_{q=0}^{d_j} \int_{x=0}^{w^t_{a,i}(q)+w_j} \frac{x}{Q_i(x)}\, dx\, dq - \int_{q=0}^{d_j} \int_{x=0}^{w^t_{a,i}(q)} \frac{x}{Q_i(x)}\, dx\, dq \\ = \; &\int_{q=0}^{d_j} \int_{x=w^t_{a,i}(q)}^{w^t_{a,i}(q)+w_j} \frac{x}{Q_i(x)}\, dx\, dq \end{aligned}$$

2.2 Amortized Local Competitiveness Analysis

We apply a local competitiveness argument as described in subsection 1.2. Because the online algorithm is using the BCP algorithm on each processor, the power for the online algorithm is $\sum_i P_i(Q_i(w_{a,i})) = w_a$. Thus $G_A = 2w_a$. Similarly, since OPT is using BCP on each processor $G_{\text{OPT}} = 2w_o$.

2.2.1 Defining the potential function

For processor i, define the potential

$$\Phi_i(t) = \frac{2}{\epsilon} \int_{q=0}^{\infty} \int_{x=0}^{(w^t_{a,i}(q)-w^t_{o,i}(q))_+} \frac{x}{Q_i(x)}\, dx\, dq \tag{3}$$

Here $(\cdot)_+ = \max(\cdot, 0)$. The global potential is then defined to be $\Phi(t) = \sum_i \Phi_i(t)$. Firstly, we observe that the function $x/Q_i(x)$ is increasing and subadditive. Then, the following lemma will be useful subsequently, the proof of which will appear in the full version of the paper.

Lemma 2.1 *Let g be any increasing subadditive function with $g(0) \geq 0$, and $w_a, w_o, w_j \in \mathbb{R}_{\geq 0}$. Then,*

$$\int_{x=w_a}^{w_a+w_j} g(x)\, dx - \int_{x=(w_a-w_o-w_j)_+}^{(w_a-w_o)_+} g(x)\, dx \leq 2 \int_{x=0}^{w_j} g(w_o + x)\, dx$$

That the boundary and completion conditions are satisfied are obvious. In Lemma 2.2 we prove that the arrival condition holds, and in Lemma 2.3 we prove that the running condition holds.

Lemma 2.2 *The arrival condition holds with $d = \frac{4}{\epsilon}$.*
Proof: Consider a new job j with processing time p_j, weight w_j and inverse density $d_j = p_j/w_j$, which the algorithm assigns to processor 1 while the optimal solution assigns it to processor 2. Observe that $\int_{q=0}^{d_j} \int_{x=w_{o,2}(q)}^{w_{o,2}(q)+w_j} \frac{x}{Q_2(x)}\, dx\, dq$ is the increase in OPT's fractional weighted flow due to this new job j. Thus our goal is to prove that the increase in the potential due to job j's arrival is at most this amount. The change in the potential $\Delta\Phi$ is:

$$\frac{2}{\epsilon} \int_{q=0}^{d_j} \left(\int_{x=(w_{a,1}(q)-w_{o,1}(q))_+}^{(w_{a,1}(q)-w_{o,1}(q)+w_j)_+} \frac{x}{Q_1(x)}\, dx - \int_{x=(w_{a,2}(q)-w_{o,2}(q)-w_j)_+}^{(w_{a,2}(q)-w_{o,2}(q))_+} \frac{x}{Q_2(x)}\, dx \right) dq$$

Now, since $x/Q_1(x)$ is an increasing function we have that

$$\int_{x=(w_{a,1}(q)-w_{o,1}(q))_+}^{(w_{a,1}(q)-w_{o,1}(q)+w_j)_+} \frac{x}{Q_1(x)}\, dx \le \int_{x=w_{a,1}(q)}^{w_{a,1}(q)+w_j} \frac{x}{Q_1(x)}\, dx$$

and hence the change of potential can be bounded by

$$\frac{2}{\epsilon} \int_{q=0}^{d_j} \left(\int_{x=w_{a,1}(q)}^{w_{a,1}(q)+w_j} \frac{x}{Q_1(x)}\, dx - \int_{x=(w_{a,2}(q)-w_{o,2}(q)-w_j)_+}^{(w_{a,2}(q)-w_{o,2}(q))_+} \frac{x}{Q_2(x)}\, dx \right) dq$$

Since we assigned the job to processor 1, we know that

$$\int_{q=0}^{d_j} \int_{x=w_{a,1}(q)}^{w_{a,1}(q)+w_j} \frac{x}{Q_1(x)}\, dx\, dq \le \int_{q=0}^{d_j} \int_{x=w_{a,2}(q)}^{w_{a,2}(q)+w_j} \frac{x}{Q_2(x)}\, dx\, dq$$

Therefore, the change in potential is at most

$$\Delta\Phi \le \frac{2}{\epsilon} \int_{q=0}^{d_j} \left(\int_{x=w_{a,2}(q)}^{w_{a,2}(q)+w_j} \frac{x}{Q_2(x)}\, dx - \int_{x=(w_{a,2}(q)-w_{o,2}(q)-w_j)_+}^{(w_{a,2}(q)-w_{o,2}(q))_+} \frac{x}{Q_2(x)}\, dx \right) dq$$

Applying Lemma 2.1, we get:

$$\Delta\Phi \le \left(2 \cdot \frac{2}{\epsilon}\right) \int_{q=0}^{d_j} \int_{x=w_{o,2}(q)}^{w_{o,2}(q)+w_j} \frac{x}{Q_2(x)}\, dx\, dq$$

□

Lemma 2.3 *The running condition holds with constant $c = 1 + \frac{1}{\epsilon}$.*
Proof: Let us consider an infinitesimally small interval $[t, t + dt)$ during which no jobs arrive and analyze the change in the potential $\Phi(t)$. Since $\Phi(t) = \sum_i \Phi_i(t)$, we can do this on a per-processor basis. Fix a single processor i, and time t. Let $w_i(q) := (w_{a,i}(q) - w_{o,i}(q))_+$, and $w_i := (w_{a,i} - w_{o,i})_+$. Let q_a and q_o denote the inverse densities of the jobs being executed on processor i by the algorithm and optimal solution respectively (which are the densest jobs in their respective queues, since both run HDF). Define $s_a = Q_i(w_{a,i})$ and $s_o = Q_i(w_{o,i})$. Since we assumed that OPT uses the BCP algorithm on each processor, OPT runs processor i at speed s_o. Since the online algorithm is also

using BCP, but has $(1+\epsilon)$-speed augmentation, the online algorithms runs the processor at speed $(1+\epsilon)s_a$. Hence the fractional weight of the job the online algorithm works on decreases at a rate of $s_a(1+\epsilon)/q_a$. Therefore, the quantity $w_{a,i}(q)$ drops by $s_a\,dt(1+\epsilon)/q_a$ for $q \in [0, q_a]$. Likewise, $w_{o,i}(q)$ drops by $s_o\,dt/q_o$ for $q \in [0, q_o]$ due to the optimal algorithm working on its densest job. We consider several different cases based on the values of $q_o, q_a, w_{o,i}$, and $w_{a,i}$ and establish bounds on $d\Phi_i(t)/dt$; Recall the definition of $\Phi_i(t)$ from equation (3):

$$\Phi_i(t) = \frac{2}{\epsilon}\int_{q=0}^{\infty}\int_{x=0}^{(w^t_{a,i}(q)-w^t_{o,i}(q))_+} \frac{x}{Q_i(x)}\,dx\,dq$$

Case (1): $w_{a,i} < w_{o,i}$: The only possible increase in potential function occurs due to the decrease in $w_{o,i}(q)$, which happens for values of $q \in [0, q_o]$. But for q's in this range, $w_{a,i}(q) \le w_{a,i}$ and $w_{o,i}(q) = w_{o,i}$. Thus the inner integral is empty, resulting in no increase in potential. The running condition then holds since $w_{a,i} < w_{o,i}$.

Case (2): $w_{a,i} > w_{o,i}$: To quantify the change in potential due to the online algorithm working, observe that for any $q \in [0, q_a]$, the inner integral of Φ_i decreases by

$$\int_{x=0}^{w_i(q)} \frac{x}{Q_i(x)}\,dx - \int_{x=0}^{w_i(q)-(1+\epsilon)\frac{s_a\,dt}{q_a}} \frac{x}{Q_i(x)}\,dx = \frac{w_i(q)}{Q_i(w_i(q))}(1+\epsilon)\frac{s_a\,dt}{q_a}$$

Here, we have used the fact that dt is infinitisemally small to get the above equality. Hence, the total drop in Φ_i due to the online algorithm's processing is

$$\begin{aligned}\frac{2}{\epsilon}\int_{q=0}^{q_a} \frac{w_i(q)}{Q_i(w_i(q))}(1+\epsilon)\frac{s_a\,dt}{q_a}\,dq \quad &\ge \quad \frac{2}{\epsilon}\int_{q=0}^{q_a} \frac{w_i}{Q_i(w_i)}(1+\epsilon)\frac{s_a\,dt}{q_a}\,dq \\ &= \quad \frac{2}{\epsilon}\frac{w_i}{Q_i(w_i)}(1+\epsilon)s_a\,dt\end{aligned}$$

Here, the first inequality holds because $x/Q_i(x)$ is a non-decreasing function, and for all $q \in [0, q_a]$, we have $w_{a,i}(q) = w_{a,i}$ and $w_{o,i}(q) \le w_{o,i}$ and hence $w_i(q) \ge w_i$.

Now to quantify the increase in the potential due to the optimal algorithm working: observe that for $q \in [0, q_o]$, the inner integral of Φ_i increases by at most

$$\int_{x=w_i(q)}^{w_i(q)+\frac{s_o\,dt}{q_o}} \frac{x}{Q_i(x)}\,dx = \frac{w_i(q)}{Q_i(w_i(q))}\frac{s_o\,dt}{q_o}$$

Again notice that we have used that fact that here dt is an infinitesimal period of time that in the limit is zero. Hence the total increase in Φ_i due to the optimal algorithm's processing is at most

$$\frac{2}{\epsilon}\int_{q=0}^{q_o} \frac{w_i(q)}{Q_i(w_i(q))}\frac{s_o\,dt}{q_o}\,dq \le \frac{2}{\epsilon}\int_{q=0}^{q_o} \frac{w_i}{Q_i(w_i)}\frac{s_o\,dt}{q_o}\,dq = \frac{2}{\epsilon}\frac{w_i}{Q_i(w_i)}s_o\,dt.$$

Again here, the first inequality holds because $x/Q_i(x)$ is a non-decreasing function, and for all $q \in [0, q_o]$, we have $w_{a,i}(q) \le w_{a,i}$ and $w_{o,i}(q) = w_{o,i}$ and hence $w_i(q) \le w_i$.

Putting the two together, the overall increase in $\Phi_i(t)$ can be bounded by

$$\begin{aligned}\frac{d\Phi_i(t)}{dt} &\leq \frac{2}{\epsilon}\frac{w_{a,i}-w_{o,i}}{Q_i(w_{a,i}-w_{o,i})}\left[-(1+\epsilon)s_a+s_o\right]\\ &= \frac{2}{\epsilon}(w_{a,i}-w_{o,i})\frac{[-(1+\epsilon)Q_i(w_{a,i})+Q_i(w_{o,i})]}{Q_i(w_{a,i}-w_{o,i})}\\ &\leq -\frac{2}{\epsilon}\epsilon(w_{a,i}-w_{o,i}) = -2(w_{a,i}-w_{o,i})\end{aligned}$$

It is now easy to verify that by plugging this bound on $\frac{d\Phi_i(t)}{dt}$ into the running condition that one gets a valid inequality.

Case (3): $w_{a,i} = w_{o,i}$: In this case, let us just consider the increase due to OPT working. The inner integral in the potential function starts off from zero (since $w_{a,i} - w_{o,i} = 0$) and potentially (in the worst case) could increase to

$$\int_0^{\frac{s_o\,dt}{q_o}} \frac{x}{Q_i(x)}\,dx$$

(since $w_{o,i}$ drops by $s_o\,dt/q_o$ and $w_{a,i}$ cannot increase). However, since $x/Q_i(x)$ is a monotone non-decreasing function, this is at most

$$\int_0^{\frac{s_o\,dt}{q_o}} \frac{w_{o,i}}{Q_i(w_{o,i})}\,dx = \frac{s_o\,dt}{q_o}\frac{w_{o,i}}{Q_i(w_{o,i})}$$

Therefore, the total increase in the potential $\Phi_i(t)$ can be bounded by

$$\frac{2}{\epsilon}\int_{q=0}^{q_o}\frac{w_{o,i}}{Q_i(w_{o,i})}\frac{s_o\,dt}{q_o}\,dq = \frac{2}{\epsilon}s_o\,dt\frac{w_{o,i}}{Q_i(w_{o,i})} = \frac{2}{\epsilon}w_{o,i}\,dt$$

It is now easy to verify that by plugging this bound on $\frac{d\Phi_i(t)}{dt}$ into the running condition, and using the fact that $w_{a,i} = w_{o,i}$, one gets a valid inequality. □

3 Algorithm for Unweighted Flow

In this section, we give an immediate assignment based scheduling policy and show that it is $O(1/\epsilon)$-competitive against a non-migratory adversary for the objective of *unweighted* flow plus energy, assuming the online algorithm has resource augmentation of $(1+\epsilon)$ in speed. Note that this result has a better competitiveness than the result for weighted flow from Section 2, but holds only for the unweighted case.

We begin by giving intuition behind our algorithm, which is again similar to that for the weighted case. Let OPT be some optimal schedule. However, for the rest of the section, we assume that on a single machine, the optimal scheduling algorithm for minimizing sum of flow times plus energy on a single machine is that of Andrew et al.[2] which sets the power at any time to be $Q(n)$ when there are n unfinished jobs, and processes jobs according to SRPT. Since we know that this ALW algorithm [2] is 2-competitive against the optimal schedule on a single processor, we will imagine that, once OPT has assigned a job to some processor, it uses the ALW algorithm on each processor. Likewise, once our assignment policy assigns a job to some processor, our algorithm also runs the ALW algorithm on each processor. Therefore, just like the weighted case, the crux of our algorithm is in designing a good assignment policy, and arguing that it is $O(1)$-competitive even though our algorithm and OPT may schedule a new job on different processors with completely different power functions.

3.1 Algorithm

Our algorithm works as follows: Each processor maintains a queue of jobs that have currently been assigned to it. At some time instant t, for any processor i, let $n^t_{a,i}(q)$ denote the number of jobs in processor i's queue that have a remaining processing time of at least q. Let $n^t_{a,i}$ denote the total number of unfinished jobs in the queue. Also, let us define the *shadow potential* for processor i at this time t as

$$\widehat{\Phi}_{a,i}(t) = \int_{q=0}^{\infty} \sum_{j=1}^{n^t_{a,i}(q)} \frac{j}{Q_i(j)}\, dq \tag{4}$$

Note that the shadow potential $\widehat{\Phi}_{a,i}(t)$ is the total future cost of the online algorithm (up to a constant factor) assuming no jobs arrive after this time instant, and the online algorithm runs the ALW algorithm on all processors (i.e., the job selection is SRPT, and the processor is run at a speed of $Q_i(n^t_{a,i})$). Now our algorithm is the following:

When a new job arrives, the assignment policy assigns it to a processor which would cause the smallest increase in the "shadow potential"; i.e., a processor minimizing

$$\int_{q=0}^{p} \sum_{j=1}^{n^t_{a,i}(q)+1} \frac{j}{Q_i(j)}\, dq - \int_{q=0}^{p} \sum_{j=1}^{n^t_{a,i}(q)} \frac{j}{Q_i(j)}\, dq = \int_{q=0}^{p} \frac{(n^t_{a,i}(q)+1)}{Q_i(n^t_{a,i}(q)+1)}\, dq$$

The job selection on each processor is SRPT (Shortest Remaining Processing Time), and we set the power of processor i at time t to $n^t_{a,i}$. Once the job is assigned to a processor, it is never migrated.

3.2 The Amortized Local-Competitive Analysis

We again employ a potential function based analysis, similar to the one in Section 2.

3.2.1 The Potential Function.

We now describe our potential function Φ. For time t and processor i, recall the definitions $n^t_{a,i}$ and $n^t_{a,i}(q)$ given above; analogously define $n^t_{o,i}$ as the number of unfinished jobs assigned to processor i by the optimal solution at time t, and $n^t_{o,i}(q)$ to be the number of these jobs with remaining processing time at least q. Henceforth, we will drop the superscript t from these terms whenever the time instant t is clear from the context.

Now, we define the global potential function to be $\Phi(t) = \sum_i \Phi_i(t)$, where $\Phi_i(t)$ is the potential for processor i defined as:

$$\Phi_i(t) = \frac{4}{\epsilon} \int_{q=0}^{\infty} \sum_{j=1}^{(n^t_{a,i}(q)-n^t_{o,i}(q))_+} j/Q_i(j)\, dq \tag{5}$$

Recall that $(x)_+ = \max(x, 0)$, and $Q_i = P_i^{-1}$. Notice that if the optimal solution has no jobs remaining on processor i at time t, we get $\Phi_i(t)$ is (within a constant off) simply $\widehat{\Phi}_{a,i}(t)$.

3.2.2 Proving the Arrival Condition.

We now show that the increase in the potential Φ is bounded (up to a constant factor) by the increase in the future optimal cost when a new job arrives. Suppose a new job of size p arrives at time t, and suppose the online algorithm assigns it to processor 1 while the optimal solution assigns it to processor 2. Then Φ_1 increases since $n_{a,1}(q)$ goes up by 1 for all $q \in [0, p]$, Φ_2 could decrease due to $n_{o,2}(q)$ dropping by 1 for all $q \in [0, p]$, and Φ_i (for $i \notin \{1, 2\}$) does not change.

Let us first assume that $n_{a,i}(q) \geq n_{o,i}(q)$ for all $q \in [0, p]$ and for $i \in \{1, 2\}$; we will show below how to remove this assumption. Under this assumption, the total change in potential Φ is

$$\frac{4}{\epsilon} \int_{q=0}^{p} \left(\frac{n_{a,1}(q) - n_{o,1}(q) + 1}{Q_1(n_{a,1}(q) - n_{o,1}(q) + 1)} - \frac{n_{a,2}(q) - n_{o,2}(q)}{Q_2(n_{a,2}(q) - n_{o,2}(q))} \right) dq$$

But since $x/Q(x)$ is increasing this is less than

$$\frac{4}{\epsilon} \int_{q=0}^{p} \left(\frac{n_{a,1}(q) + 1}{Q_1(n_{a,1}(q) + 1)} - \frac{n_{a,2}(q) - n_{o,2}(q)}{Q_2(n_{a,2}(q) - n_{o,2}(q))} \right) dq \tag{6}$$

By the greedy choice of processor 1 (instead of 2), this is less than

$$\frac{4}{\epsilon} \int_{q=0}^{p} \left(\frac{n_{a,2}(q) + 1}{Q_2(n_{a,2}(q) + 1)} - \frac{n_{a,2}(q) - n_{o,2}(q)}{Q_2(n_{a,2}(q) - n_{o,2}(q))} \right) dq \tag{7}$$

Now, since $x/Q(x)$ is subadditive this is less than $\frac{4}{\epsilon} \int_{q=0}^{p} \frac{n_{o,2}(q)+1}{Q_2(n_{o,2}(q)+1)}\, dq$, which in turn is (within a factor of $\frac{4}{\epsilon}$) precisely the *increase in the future cost* incurred by the optimal solution, since we had assumed that OPT also runs the ALW algorithm on its processors.

Now suppose $n_{a,1}(q) < n_{o,1}(q)$ for some $q \in [0, p]$. There is no increase in the inner sum of Φ_1 for such values of q, and hence we can trivially upper bound this zero increase by $\frac{4}{\epsilon} \int_{q=0}^{p} \frac{n_{a,1}(q)+1}{Q_1(n_{a,1}(q)+1)} \leq \frac{4}{\epsilon} \int_{q=0}^{p} \frac{n_{a,2}(q)+1}{Q_2(n_{a,2}(q)+1)}$. And to discharge the assumption for processor 2, note that if $n_{a,2}(q) < n_{o,2}(q)$ for some q, there is no decrease in the inner sum of Φ_2 for this value of q, but in this case we can simply use $\frac{n_{a,2}(q)+1}{Q_1(n_{a,2}(q)+1)} \leq \frac{n_{o,2}(q)+1}{Q_1(n_{o,2}(q)+1)}$ for such values of q. Therefore, we get the same bound of

$$\frac{4}{\epsilon} \int_{q=0}^{p} \frac{n_{o,2}(q) + 1}{Q_2(n_{o,2}(q) + 1)}\, dq$$

on the increase in all cases, thus proving the following lemma.

Lemma 3.1 *The arrival condition holds for the unweighted case with* $d = \frac{4}{\epsilon}$.

3.2.3 Proving the Running Condition.

In this section, our goal is to analyze the change in Φ in an infinitesimally small time interval $[t, t + dt)$ and compare $d\Phi/dt$ to $d\mathsf{Alg}/dt$ and $d\mathrm{OPT}/dt$. We do this on a per-processor basis; let us focus on processor i at time t. Recall (after dropping the t superscripts) the definitions of

$n_{a,i}(q), n_{o,i}(q), n_{a,i}$ and $n_{o,i}$ from above, and define $n_i(q) := (n_{a,i}(q) - n_{o,i}(q))_+$. Finally, let q_a and q_o denote the remaining sizes of the jobs being worked on by the algorithm and optimal solution respectively at time t; recall that both of them use SRPT for job selection. Define $s_a = Q_i(n_{a,i})$ and $s_o = Q_i(n_{o,i})$ to be the (unaugmented) speeds of processor i according to the online algorithm and the optimal algorithm respectively—though, since we assume resource augmentation, our processor runs at speed $(1+\epsilon)Q_i(n_{a,i})$. Hence $n_{a,i}(q)$ drops by 1 for $q \in (q_a - (1+\epsilon)s_a\, dt, q_a]$ and $n_{o,i}(q)$ drops by 1 for $q \in (q_o - s_o\, dt, q_o]$ for the optimal algorithm. Let $I_a := (q_a - (1+\epsilon)s_a\, dt, q_a]$ and $I_o := (q_o - s_o\, dt, q_o]$ denote these two intervals. Let us consider some cases:

Case (1): $n_{a,i} < n_{o,i}$: The increase in potential function may occur due to $n_{o,i}(q)$ dropping by 1 in $q \in I_o$. However, since $n_{a,i} < n_{o,i}$, it follows that $n_{a,i}(q) \le n_{a,i} < n_{o,i} = n_{o,i}(q)$ for all $q \in I_o$; the equality follows from OPT using SRPT. Consequently, even with $n_{o,i}(q)$ dropping by 1, $n_{a,i}(q) - n_{o,i}(q) \le 0$ and there is no increase in potential, or equivalently $d\Phi_i(t)/dt \le 0$. Hence, in this case, $4n_{a,i} + d\Phi_i(t)/dt \le 4n_{o,i}$.

Case (2a): $n_{a,i} \ge n_{o,i}$**, and** $q_a < q_o$: For $q \in I_a$, the inner summation of Φ_i drops by $\frac{n_{a,i}(q)-n_{o,i}(q)}{Q_i(n_{a,i}(q)-n_{o,i}(q))}$, since $n_{a,i}(q)$ decreases by 1. Moreover, $n_{a,i}(q) = n_{a,i}$ and $n_{o,i}(q) = n_{o,i}$, because both Alg and OPT run SRPT, and we're considering $q \le q_a < q_o$. For $q \in I_o$, the inner summation of Φ_i increases by $\frac{n_{a,i}(q)-(n_{o,i}(q)-1)}{Q_i(n_{a,i}(q)-(n_{o,i}(q)-1))}$. However, we have $n_{a,i}(q) \le n_{a,i} - 1$ and $n_{o,i}(q) = n_{o,i}$ because $q_a < q_o$, and $n_{o,i}(q) = n_{o,i}$ because OPT runs SRPT. Therefore the increase is at most $\frac{n_{a,i}-n_{o,i}}{Q_i(n_{a,i}-n_{o,i})}$ for all $q \in I_o$. Combining these two, we get

$$\begin{aligned}\frac{d\Phi_i(t)}{dt} &\le \frac{4}{\epsilon}\frac{n_{a,i}-n_{o,i}}{Q_i(n_{a,i}-n_{o,i})}\big[-(1+\epsilon)s_a + s_o\big]\\ &= \frac{4}{\epsilon}\big(n_{a,i}-n_{o,i}\big)\frac{\big[-(1+\epsilon)Q_i(n_{a,i}) + Q_i(n_{o,i})\big]}{Q_i(n_{a,i}-n_{o,i})}\\ &\le \frac{4}{\epsilon}\big(n_{a,i}-n_{o,i}\big)\frac{-\epsilon\, Q_i(n_{a,i})}{Q_i(n_{a,i}-n_{o,i})} \le -4(n_{a,i}-n_{o,i}),\end{aligned}$$

where we repeatedly use that $Q_i(\cdot)$ is a non-decreasing function. This implies that $4n_{a,i}+d\Phi_i(t)/dt \le 4n_{o,i}$.

Case (2b): $n_{a,i} \ge n_{o,i}$**, and** $q_a > q_o$: In this case, for $q \in I_o$, the inner summation of Φ_i increases by $\frac{n_{a,i}(q)-(n_{o,i}(q)-1)}{Q_i(n_{a,i}(q)-(n_{o,i}(q)-1))}$. Also, we have $n_{a,i}(q) = n_{a,i}$ and $n_{o,i}(q) = n_{o,i}$, because $q \le q_o < q_a$ and both algorithms run SRPT. Therefore the overall increase in potential function can be bounded by $\frac{4}{\epsilon}\frac{n_{a,i}-n_{o,i}+1}{Q_i(n_{a,i}-n_{o,i}+1)}\, s_o\, dt$. Moreover, for $q \in I_a$, the inner summation of Φ_i drops by $\frac{n_{a,i}(q)-n_{o,i}(q)}{Q_i(n_{a,i}(q)-n_{o,i}(q))}$. Also, $n_{a,i}(q) = n_{a,i}$ and $n_{o,i}(q) \le n_{o,i} - 1$, because $q_o < q_a$ and there was a job of remaining size q_o among the optimal solution's active jobs. Thus the potential function drops by at least $\frac{4}{\epsilon}\frac{n_{a,i}-n_{o,i}+1}{Q_i(n_{a,i}-n_{o,i}+1)}\,(1+\epsilon)s_a\, dt$, since $x/Q_i(x)$ is a non-decreasing function. Combining these terms,

$$\begin{aligned}\frac{d\Phi_i(t)}{dt} &\le \frac{4}{\epsilon}\frac{n_{a,i}-n_{o,i}+1}{Q_i(n_{a,i}-n_{o,i}+1)}\big[-(1+\epsilon)s_a + s_o\big]\\ &= \frac{4}{\epsilon}\big(n_{a,i}-n_{o,i}+1\big)\frac{[-(1+\epsilon)Q_i(n_{a,i}) + Q_i(n_{o,i})]}{Q_i(n_{a,i}-n_{o,i}+1)}\\ &\le -\frac{4}{\epsilon}\epsilon(n_{a,i}-n_{o,i}+1) \le -4(n_{a,i}-n_{o,i})\end{aligned}$$

In the above, we have used the fact that $n_{o,i} \geq 1$, and consequently, $Q_i(n_{a,i}) \geq Q_i(n_{a,i} - n_{o,i} + 1)$. Therefore, in this case too we get $4n_{a,i} + d\Phi_i(t)/dt \leq 4n_{o,i}$.

Case (2c): $n_{a,i} \geq n_{o,i}$, and $q_a = q_o$: Since $n_{a,i} \geq n_{o,i}$, and Q_i is an increasing function, $s_a \geq s_o$ and thus $I_o \subset I_a$. For q in the interval I_o, the term $n_{o,i}(q)$ drops by 1 *and* the term $n_{a,i}(q)$ drops by 1, and therefore there is no change in $n_{a,i}(q) - n_{o,i}(q)$. For $q \in I_a \setminus I_o$, the inner summation for Φ_i drops by $\frac{n_{a,i}(q)-n_{o,i}(q)}{Q_i(n_{a,i}(q)-n_{o,i}(q))}$. Also, $n_{a,i}(q) = n_{a,i}$ and $n_{o,i}(q) = n_{o,i}$, and the decrease in potential function is $\frac{4}{\epsilon}((1+\epsilon)s_a\,dt - s_o\,dt)\,\frac{n_{a,i}-n_{o,i}}{Q_i(n_{a,i}-n_{o,i})}$. But the analysis in Case (2a) implies that $4n_{a,i} + d\Phi_i(t)/dt \leq 4n_{o,i}$ in this case as well.

Summing over all i, we get

Lemma 3.2 *The running condition holds for the unweighted case with constant* 4. *At any time t when there are no job arrivals, we have*

Combining Lemmas 3.1 and 3.2 with the standard potential function argument indicated in Section 1.2, we get the following theorem.

Theorem 3.3 *There is a $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive immediate-assignment algorithm to minimize the total flow plus energy on heterogeneous processors with arbitrary power functions.*

Acknowledgments: We thank Sangyeun Cho and Bruce Childers for helpful discussions about heterogeneous multicore processors, and also Srivatsan Narayanan for several useful discussions.

References

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

[2] 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.

[3] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA*, pages 1238–1244, 2009.

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

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

[6] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4), 2009.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

[8] Fred A. Bower, Daniel J. Sorin, and Landon P. Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. *IEEE Micro*, 28(3):17–25, 2008.

[9] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *STOC*, pages 679–684, 2009.

[10] 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.

[11] Gero Greiner, Tim Nonner, and Alexander Souza. The bell is ringing in speed-scaled multiprocessor scheduling. In *SPAA '09*, pages 11–18, 2009.

[12] Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *International conference on parallel architectures and compilation techniques*, pages 23–32. ACM, 2006.

[13] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. *SIGARCH Computer Architecture News*, 32(2):64, 2004.

[14] 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.

[15] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *European Symposium on Algorithms*, pages 647–659, 2008.

[16] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and Systems Sciences*, 73(6):875–891, 2007.

[17] Rick Merritt. CPU designers debate multi-core future. *EE Times*, February 2008.

[18] Tomer Y. Morad, Uri C. Weiser, Avinoam Kolodny, Mateo Valero, and Eduard Ayguade. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *IEEE Computer Architecture Letters*, 5(1):4, 2006.

[19] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[20] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook on Scheduling.* CRC Press, 2004.

[21] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.

A Estimating the Future Cost of BCP

Suppose we have a set of n jobs, with weight w_j and processing time p_j for job j $(1 \leq j \leq n)$ such that $p_1/w_1 \leq p_2/w_2 \leq \ldots \leq p_n/w_n$. Let $d_j = p_j/w_j$ denote the inverse density of a job j. We now explain how we get the estimate of the future cost of this configuration when we run the BCP algorithm, i.e. HDF at a speed of $Q(w^t)$, where w^t is the total fractional weight of unfinished jobs at time t.

Firstly, observe that by virtue of our algorithm running HDF, it schedules job 1 followed by job 2, etc. Also, as long as the algorithm is running job 1, it runs the processor at speed $Q(W_{\geq 2}+\widetilde{w_1}(t))$, where $W_{\geq 2} := w_2+w_3+\dots w_n$ and $\widetilde{w_1}(t)$ is the fractional weight of job 1 remaining. Secondly, since our algorithm always uses power equal to the fractional weight remaining, the rate of increase of the objective function at any time t is simply $2w^t$. Therefore, the following equations immediately follow:

$$\begin{aligned}
G_A(t) &= \frac{dA}{dt} = 2\big(W_{\geq 2} + \widetilde{w_1}(t)\big) \\
\frac{d\widetilde{w_1}(t)}{dt} &= -\Big(\frac{w_1}{p_1}\Big) Q\Big(W_{\geq 2} + \widetilde{w_1}(t)\Big) \\
\Rightarrow \frac{dA}{d\widetilde{w_1}(t)} &= -2\Big(\frac{p_1}{w_1}\Big)\frac{W_{\geq 2} + \widetilde{w_1}(t)}{Q\big(W_{\geq 2} + \widetilde{w_1}(t)\big)} \\
\Rightarrow A_1 &= -2\int_{x=W_{\geq 2}+w_1}^{W_{\geq 2}} d_1 \frac{W_{\geq 2}+x}{Q\big(W_{\geq 2}+x\big)}\, dx
\end{aligned}$$

That is, the total cost incurred while job 1 is being scheduled is

$$2\int_{x=W_{\geq 2}}^{W_{\geq 2}+w_1} d_1 \frac{W_{\geq 2}+x}{Q\big(W_{\geq 2}+x\big)}\, dx = 2\int_{q=0}^{d_1}\int_{x=W_{\geq 2}}^{W_{\geq 2}+w_1} \frac{W_{\geq 2}+x}{Q\big(W_{\geq 2}+x\big)}\, dx\, dq$$

Similarly, while any job i is being scheduled, we can use the same arguments as above to show that the total fractional flow incurred is

$$2\int_{x=W_{\geq(i+1)}}^{W_{\geq(i+1)}+w_i} d_i \frac{W_{\geq(i+1)}+x}{Q\big(W_{\geq(i+1)}+x\big)}\, dx = 2\int_{q=0}^{d_i}\int_{x=W_{\geq(i+1)}}^{W_{\geq(i+1)}+w_i} \frac{W_{\geq(i+1)}+x}{Q\big(W_{\geq(i+1)}+x\big)}\, dx\, dq$$

Summing over i, the total fractional flow incurred by our algorithm is

$$2\sum_{i=1}^{n}\int_{q=0}^{d_i}\int_{x=W_{\geq(i+1)}}^{W_{\geq(i+1)}+w_i} \frac{W_{\geq(i+1)}+x}{Q\big(W_{\geq(i+1)}+x\big)}\, dx\, dq$$

Rearranging the terms, it is not hard to see (given $d_1 \leq d_2 \leq \dots \leq d_n$) that this is equal to

$$2\int_{q=0}^{\infty}\int_{x=0}^{w(q)} \frac{x}{Q(x)}\, dx\, dq$$

where $w(q)$ is the total weight of jobs with inverse density at least q.

B Subadditivity of x/Q(x)

Let $Q(x) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be any concave function such that $Q(0) \geq 0$ and $Q'(x) \geq 0$ for all $x \geq 0$, and let $g(x) = x/Q(x)$. Then the following facts are true about $g(\cdot)$.

(a) $g(\cdot)$ is non-decreasing. That is, $g(y) \geq g(x)$ for all $y \geq x$.

(b) $g(\cdot)$ is subadditive. That is, $g(x) + g(y) \geq g(x + y)$ for all $x, y \in \mathbb{R}_{\geq 0}$

To see why the first is true, consider x and $y = \lambda x$ for some $\lambda \geq 1$. Then, showing (a) is equivalent to showing

$$\frac{\lambda x}{Q(\lambda x)} \geq \frac{x}{Q(x)}$$

But this reduces to showing $Q(\lambda x) \leq \lambda Q(x)$ which is true because $Q(\cdot)$ is a concave function. To prove the second property, we first observe that the function $1/Q(x)$ is convex. This is because the second derivative of $1/Q(x)$ is

$$\frac{-Q(x)^2 Q''(x) + 2Q(x)Q'(x)^2}{Q(x)^4}$$

which is always non-negative for all $x \geq 0$, since $Q(x)$ is non-negative and $Q''(x)$ is non-positive for all $x \geq 0$. Therefore, since $1/Q(\cdot)$ is convex, it holds for any x, y, and $\alpha \geq 0$, $\beta \geq 0$ that

$$\frac{\frac{\alpha}{Q(x)} + \frac{\beta}{Q(y)}}{\alpha + \beta} \geq \frac{1}{Q(\frac{\alpha x + \beta y}{\alpha + \beta})}$$

Plugging in $\alpha = x$ and $\beta = y$, we get

$$\frac{\frac{x}{Q(x)} + \frac{y}{Q(y)}}{x + y} \geq \frac{1}{Q(\frac{x^2 + y^2}{x + y})}$$

which implies

$$\frac{x}{Q(x)} + \frac{y}{Q(y)} \geq \frac{x + y}{Q(\frac{x^2 + y^2}{x + y})}$$

But since $Q(\cdot)$ is non-decreasing, we have $Q(x + y) \geq Q(\frac{x^2 + y^2}{x + y})$ and hence

$$\frac{x}{Q(x)} + \frac{y}{Q(y)} \geq \frac{x + y}{Q(x + y)}$$

C Missing Proofs

Proof of Lemma 2.1: We first show that

$$\int_{x=w_a}^{w_a + w_j} g(x)\,dx - \int_{x=(w_a - w_o - w_j)_+}^{(w_a - w_o)_+} g(x)\,dx \leq \int_{x=0}^{w_j} g(w_o + w_j)\,dx$$

and then argue that $\int_{x=0}^{w_j} g(w_o + w_j)\,dx \leq 2\int_{x=0}^{w_j} g(w_o + x)\,dx$ because $g(\cdot)$ is subadditive. To this end, we consider several cases and prove the lemma. Suppose w_a is such that $w_a \geq w_o + w_j$: in this case we can discard the $(\cdot)_+$ operators on all the limits to get

$$\begin{aligned}
&\int_{x=w_a}^{w_a + w_j} g(x)\,dx \; - \int_{x=w_a - w_o - w_j}^{w_a - w_o} g(x)\,dx \\
&= \int_{x=0}^{w_j} \Big(g(w_a + x) - g(w_a - w_o - w_j + x) \Big)\,dx \; \leq \; \int_{x=0}^{w_j} g(w_o + w_j)\,dx
\end{aligned}$$

Here, the final inequality follows because $g(\cdot)$ is a subadditive function. On the other hand, suppose it is the case that $w_a \leq w_o$, then both limits $(w_a - w_o)_+$ and $(w_a - w_o - w_j)_+$ are zero, and therefore we only need to bound $\int_{x=w_a}^{w_a+w_j} g(x)\,dx$, which can be done as follows:

$$\int_{x=w_a}^{w_a+w_j} g(x)\,dx = \int_{x=0}^{w_j} g(w_a + x)\,dx \leq \int_{x=0}^{w_j} g(w_o + w_j)\,dx$$

Finally, if $w_a = w_o + \delta$ for some $\delta \in (0, w_j)$, we first observe that $\int_{x=(w_a-w_o-w_j)_+}^{(w_a-w_o)_+} g(x)dx$ simplifies to $\int_{x=0}^{\delta} g(x)dx$. Therefore, we are interested in bounding

$$\begin{aligned}\int_{x=w_a}^{w_a+w_j} g(x)\,dx \; - \int_{x=0}^{\delta} g(x)\,dx &= \int_{x=w_a}^{w_a+w_j-\delta} g(x)\,dx + \int_{x=w_a+w_j-\delta}^{w_a+w_j} g(x)\,dx \; - \int_{x=0}^{\delta} g(x)\,dx \\ &\leq (w_j - \delta)g(w_a + w_j - \delta) + \int_{x=0}^{\delta} g(w_a + w_j - \delta)\,dx \; \leq \int_{x=0}^{w_j} g(w_o + w_j)\,dx\end{aligned}$$

Here again, in the second to last inequality, we used the fact that $g(\cdot)$ is subadditive and therefore $g(w_a+w_j-\delta+x)-g(x) \leq g(w_a+w_j-\delta)$, for all values of $x \geq 0$; hence we get $\int_{x=w_a+w_j-\delta}^{w_a+w_j} g(x)\,dx - \int_{x=0}^{\delta} g(x)\,dx \leq \int_{x=0}^{\delta} g(w_a + w_j - \delta)\,dx$.

To complete the proof, we need to show that $\int_{x=0}^{w_j} g(w_o + w_j)\,dx \leq 2\int_{x=0}^{w_j} g(w_o + x)\,dx$. To see this, consider the following sequence of steps:

For any $x \in [0, w_j]$, since g is subadditive, we have

$$g(w_o + w_j - x) + g(x) \geq g(w_o + w_j)$$

Integrating both sides from $x = 0$ to w_j we get

$$\int_{x=0}^{w_j} g(w_o + w_j - x)\,dx + \int_{x=0}^{w_j} g(x)\,dx \geq \int_{x=0}^{w_j} g(w_o + w_j)\,dx$$

which is, by variable renaming, equivalent to

$$\int_{y=0}^{w_j} g(w_o + y)\,dy + \int_{x=0}^{w_j} g(x)\,dx \geq \int_{x=0}^{w_j} g(w_o + w_j)\,dx$$

But since $g(\cdot)$ is non-decreasing, we have $\int_{x=0}^{w_j} g(x)\,dx \leq \int_{x=0}^{w_j} g(w_o + x)\,dx$ and therefore

$$\int_{y=0}^{w_j} g(w_o + y)\,dy + \int_{x=0}^{w_j} g(w_o + x)\,dx \geq \int_{x=0}^{w_j} g(w_o + w_j)\,dx$$

which is what we want. □