Optimal Energy Trade-off Schedules

Neal Barcelo*, Daniel Cole*, Dimitrios Letsios†, Michael Nugent*, and Kirk Pruhs*
*Department of Computer Science
University of Pittsburgh, Pittsburgh, Pennsylvania 15260
Email: ncb30, dcc20, mnugent, kirk@cs.pitt.edu
†IBISC, University of Evry, France
LIP6, University Pierre et Marie Curie, France
Email: dimitris.letsios@ibisc.univ-evry.fr

***Abstract*—We consider scheduling tasks that arrive over time on a speed scalable processor. At each time a schedule specifies a job to be run and the speed at which the processor is run. Processors are generally less energy efficient at higher speeds. We seek to understand the structure of schedules that optimally trade-off the energy used by the processor with a common scheduling quality of service measure, fractional weighted delay. We assume that there is some user defined parameter β specifying the user's desired additive trade-off between energy efficiency and quality of service. We prove that the optimal energy trade-off schedule is essentially unique, and has a simple structure. Thus it is easy to check the optimality of a schedule. We further prove that the optimal energy trade-off schedule changes continuously as a function of the parameter β. Thus it is possible to compute the optimal energy trade-off schedule using a natural homotopic optimization algorithm. We further show that multiplicative trade-off schedules have fewer desirable properties.**

***Keywords*-scheduling; algorithms; energy; power; speed scaling;**

1. Introduction

It seems to be a corollary to the laws of physics that, all else being equal, higher performance devices are necessarily less energy efficient than lower performance devices. For example, a Ferrari sports car is less energy efficient than a Toyota Prius, and it is not possible with current technology to get Ferrari performance with Prius fuel efficiency. Conceptually there is an energy efficiency spectrum of design points, with high performance and low energy efficiency on one end, and low performance and high energy efficiency on the other end. Early in the design process a car designer has to select a sweet spot on this spectrum as a design goal. There is seldom a uniquely defined sweet spot that is best for all situations, which is why both sports cars and economy cars are produced. Traditionally computer chip designers invariably picked sweet spots near the high performance end of the spectrum. But about a decade ago, due in large part to the exponentiality of Moore's law finally kicking in, there was a relatively abrupt shift in the design sweet spot for major chip manufacturers towards the middle of the energy efficiency spectrum. As with cars, there is not a universal sweet spot for computer chips that is best for all situations. There are situations, for example when there are critical tasks to be done, when performance is more important than energy efficiency, and there are situations, for example when there are no critical tasks to be done, when energy efficiency may be more important than performance. However, unlike for cars, it is technologically and economically feasible to design computer chips with multiple operational modes, each at a different point on the performance and energy efficiency spectrum. The most obvious examples are speed scalable processors, which we will focus on here, and heterogeneous multiprocessors. Thus a naturally arising broad algorithmic question is how to find the right operational mode that is the sweet spot on the energy efficiency spectrum for the tasks at hand.

Our initial motivation was to extend work done in [1]. [1] considered the problem of scheduling tasks that arrive over time on a single speed scalable processor. A schedule specifies, at each time, a task to be run at that time, and the speed at which to run the processor. A user defined parameter, β, specifies the relative importance of energy versus performance (for example, the operating systems on most laptops allow the user to specify some desired trade-off between battery life and performance). The objective is to find a schedule, $\mathcal{S}$, that minimizes some quality of service objective $\mathcal{Q}$ plus β times the energy used by the processor. This schedule $\mathcal{S}$ is the optimal energy trade-off schedule in the sense that: No schedule can have better quality of service given the current investment of energy used by $\mathcal{S}$, and an additional investment of 1 unit of energy is insufficient to improve the quality of service by more than β. While this optimization problem is natural enough that it may find application someplace, the main motivation for considering this algorithmic problem is to obtain a better understanding of optimal energy trade-off schedules.

[1] assumed that all jobs were of the same size, and considered the quality of service measure of total delay. [1] further assumed that the processor could be run at any nonnegative real speed s, and that the power used by the processor was of the form s^α for some constant $\alpha > 1$. By formulating the problem as a convex program and applying the Karush-Kuhn-Tucker (KKT) optimality conditions, [1]

showed that the unique optimal schedule had a simple structure, and was easily recognizable. This revealed seemingly counter-intuitive properties of the optimal energy trade-off schedule. For example, as β becomes larger, and thus energy becomes more important relative to quality of service, the energy invested in some jobs counter-intuitively increases; hence, most natural properties of jobs in the optimal trade-off schedule are not necessarily monotone in the cost of energy. However, [1] showed that the time where each bit of work is executed is a continuous function of β. Using this insight, [1] then developed an efficient algorithm to compute the optimal trade-off schedule.

Here we interpret the algorithm given in [1] as a homotopic optimization algorithm. The setting for homotopic optimization is a collection of optimization problems with identical feasible regions, but with different objectives. Let $\mathcal{Q}$ be one objective, the scheduling objective in our setting, and let $\mathcal{E}$ be another objective, the energy objective in our setting. Assume that when the objective is $\mathcal{Q}$, it is easy to solve the optimization problem, but when the objective is $\mathcal{Q} + \mathcal{E}$, the optimization is not easy to solve. This is the case in the speed scaling setting because if energy is not part of the objective, the processor should always be run at maximum speed. Intuitively the homotopic optimization method solves the optimization problem, with objective $\mathcal{Q}+\mathcal{E}$, by maintaining the optimal solution with the objective $\mathcal{Q} + \beta\mathcal{E}$ as β continuously increases from 0 to 1. A highly desirable, if not strictly necessary condition, to develop a homotopic algorithm is that in some sense the optimal solution should change continuously as a function of β so that the knowledge of the optimal solution for a particular β is useful to obtain the optimal solution for $\beta + \epsilon$ for small ϵ. Thus a natural way to animate a homotopic optimization algorithm is to have a slider specifying β, such that moving the slider causes the optimal schedule for the objective $\mathcal{Q} + \beta\mathcal{E}$ to be displayed. A Mathematica based animation is reported on in [1], and a more portable Java based animation by Yori Zwols can be found at [2].

[1] showed that if the quality of service measure is total delay, and jobs have arbitrary sizes, then the optimal energy trade-off schedule can change radically in response to a small change in β. Thus there is seemingly no hope of extending the results in [1] when the quality of service objective is total delay and jobs have arbitrary sizes. Because of this, we instead consider fractional delay. The fractional delay of a job weights the delay according to the fraction of the job remaining. For example, if a job is 75% complete at a particular time, then the job's fractional delay is increasing at a rate of only $1/4$ the job's normal (integral) delay. If each job is assigned a relative importance, i.e., a weight, the weighted fractional delay of a job is the weight of the job times the job's fractional delay and the weighted fractional delay of a schedule is the sum, over all jobs, of the weighted fractional delays. Fractional delay is the "right" measure if the benefit that a client gets from a partially completed task is proportional to the fraction of the task that is completed.

Our optimization problem now becomes finding the schedule that minimizes the total weighted fractional delay plus β times the energy used by the processor. Again this can be viewed as an optimal trade-off schedule in the sense that: No schedule can have better weighted fractional delay given the current investment of energy, and any additional investment of one unit of energy can not improve the weighted fractional delay by more than β.

Besides changing the quality of service objective, we extend the general setting considered in [1] in three ways. First, we assume that jobs can have arbitrary sizes, as one would expect to be the case on a general purpose computational device. Second, we assume that each task has an associated importance, the previously mentioned weight of a job, derived from either some higher level application or from the user. Third, we assume that the allowable speeds, and the relationship between the speed and the power, of the processor are essentially arbitrary. Note that because we have changed our quality of service measure from delay to fractional delay, [1] is no longer a special case of our setting.

Our strategy was to try to follow the same line of inquiry as [1]. We were largely, though not completely, successful. We believe that the points of failure raise interesting future research questions, and the differences between the results obtained here from those in [1] are instructive. We first model the problem as a convex program, and then apply the KKT conditions to derive necessary and sufficient conditions for optimality of a schedule. We find that a feasible schedule is optimal if and only if three conditions hold:

- There is a linearly decreasing hypopower function associated with each job that maps a time to a hypopower, where the slope of the hypopower function is proportional to the density of the job, and inversely proportional to β. We coin the term hypopower to refer to the rate of change of power with respect to speed as we are not aware of any standard name for this quantity.
- At all times, the job that is run is the job whose hypopower function is largest at that time.
- If a job is run, it is run at the hypopower specified by the hypopower function.

We then show that as a consequence of these conditions, the optimal schedule is unique and that to specify an optimal schedule, it is sufficient to specify the value of each hypopower function at the release time of the corresponding job. It is perhaps somewhat surprising that these conditions are so structurally different that the optimality conditions given in [1] for unit jobs when the quality of service objective is total delay. In that setting, [1] showed that in an optimal schedule the jobs could be partitioned into partitions of consecutively released jobs by cutting at points in time when a completion time coincided with the release of the next job; then a necessary and sufficient condition for

optimality of a feasible schedule is that each job that didn't delay subsequent jobs is run at a power ρ that would be optimal if it was the only job to be scheduled, and every other job that wasn't last in its partition is run at a power ρ more than the subsequent job. So the resulting natural way to specify an optimal schedule is to give the relationship between the completion time of each job and the release of the subsequent jobs, and to give the power for all jobs where these two times coincide. So when the quality of service objective is total (integer) delay, it seems most natural to reason about the power of the jobs, and when the quality of service objective is fractional delay, it seems most natural to reason about the hypopower of jobs, because the optimality conditions are linear in these quantities.

We then show that the initial values of hypopower functions change continuously as a function of β. (It is clear that the time when a particular bit of work is done does not change continuously as a function of β, as in [1].) This allows us to develop an efficient homotopic algorithm for a modest number of jobs. We also report on a Java-based animation of this algorithm, based on an animation by Yuri Zwols [2]. Unfortunately, we are not able to give an efficient homotopic algorithm for a large number of jobs because while the starting hypopowers change continuously, we do not know how to efficiently compute the direction and rate of change in the starting hypopowers as β changes infinitesimally. To give some evidence that this might be a challenging problem, we show that the most obvious mathematical programming formulation of this problem is not convex.

Another commonly studied trade-off objective is to minimize the product of the quality of service and energy objectives. In section 5, we consider the objective $\mathcal{Q}^\sigma \times \mathcal{E}$, where $\mathcal{Q}$ is fractional flow, $\mathcal{E}$ is energy, and σ is a parameter representing the relative importance of the scheduling objective and energy. We give the following results:

- Perhaps counter-intuitively, we show that if the static power is zero, then the optimal solution is to either always go as fast as possible or to always go as slow as possible.
- We show that locally optimal product trade-off schedules have a nice structure similar to globally optimal sum trade-off schedules, but local optimality does not imply global optimality.
- We show that, for a fixed instance, the set of schedules that are optimal for the product objective (for any σ) are (a generally strict) subset of the schedules that are optimal for the sum objective (for any β).
- We show that the optimal product trade-off schedule may be a discontinuous function of σ. Thus there is unlikely to be a homotopic algorithm to compute optimal product trade-off schedules.

We therefore conclude that for the purposes of reasoning theoretically about optimal energy trade-off schedules, the sum trade-off objective is probably preferable to the product trade-off objective, as it has many more mathematically desirable properties.

1.1. Related Results

The results in [1], as presented, assume an objective of total delay subject to a constraint on the total energy used, although it is straight-forward to see that the same approach works when the objective is a linear combination of total delay and energy. [3] introduced the idea of considering an objective that is a linear combination of energy and a scheduling quality of service objective into the literature. [3] gave a dynamic programming based algorithm to compute the optimal energy trade-off schedule for unit work jobs. To the best of our knowledge there is no other algorithmic work directly related to the results in this paper.

[4] showed that a natural online algorithm is 2-competitive for the objective of a linear combination of energy and weighted fractional delay. [4], [5] showed that a natural online algorithm is 2-competitive for the objective of a linear combination of energy and total (unweighted) (integer) delay. Previously, [3], [6], [7], [8], [9], [10], [11] gave online algorithms with competitive analyses in the case that the power function was of the form s^α.

[12] first introduced the energy-delay product as a metric, and its use has been prevalent since then.

2. PRELIMINARIES

The input consists of n jobs, where job i has release time r_i, work p_i, and a weight $w_i > 0$. The density of a job is its weight divided by work, that is w_i/p_i. A schedule is defined by specifying, for each time, a job to be run and a speed at which to run the processor. Preemption is allowed, that is, a job may be suspended and later restarted from the point of suspension. A schedule may only specify a job i to be run at time t if i has been released by t, i.e., $t \geq r_i$. Job i is completed once p_i units of work have been performed on i. Speed is the rate at which work is completed, thus if job i, of work p_i, is run at constant speed s until completion, job i will complete p_i/s time units after being started. Without loss of generality, we assume that no two jobs are released at the same time. If this is not the case, then we can create a new input that spaces out the releases of jobs released at the same time by $\epsilon > 0$, where jobs are then released in order of non-increasing w_i/p_i. An optimal schedule for this new input is optimal for the original input as the optimal schedule for the original input works on each job for at least time ϵ and prioritizes jobs by non-increasing w_i/p_i.

For job i, the delay (also called flow), F_i, is i's completion time minus its release time and the weighted delay is $w_i F_i$. The delay of a schedule is $\sum_{i=1}^n F_i$ and the weighted delay is $\sum_{i=1}^n w_i F_i$. If $p_i(t)$ work is performed on job i at time t then a $p_i(t)/p_i$ fraction of job i was delayed $t - r_i$ time units

before being completed, thus the total fractional delay of job i is $\int_{r_i}^{\infty} p_i(t)(t-r_i)/p_i\,dt$ and the fractional weighted delay of job i is $\int_{r_i}^{\infty} w_i p_i(t)(t-r_i)/p_i\,dt$. The fractional delay and fractional weighted delay of a schedule are the sum, over all jobs, of the fractional delay and fractional weighted delay respectively. The objective we consider is the energy used by the schedule plus the fractional weighted delay of the schedule.

We adopt the speed scaling model, originally proposed in [4], that essentially allows the speed to power function to be arbitrary. In particular the power function may have a maximum power, and hence maximum speed. For our setting, [4] showed that results that hold for power functions that are continuous, differentiable, and convex from speed/power 0 to the maximum speed/power, will hold for any power function meeting the less strict constraints of being piecewise continuous, differentiable, and integrable. Thus, without loss of generality, we may assume the power function is continuous, differentiable, and convex from speed/power 0 to the maximum speed/power. We define the power function as $P(S)$, where S is a speed. Energy is power integrated over time, $\int_t P(S)dt$. The static power is equal to $P(0)$, which is the power used by the processor even if it is idling.

We use the term hypopower to refer to the derivative of power with respect to speed. A particular hypopower function that will be important to our discussions is the derivative of the processor's power function, $P(S)$, with respect to speed, which will be denoted as $P'(S)$. The functional inverse of $P'(S)$ will be denoted as $P^*(x)$, where x is a hypopower and $P^*(x)$ is speed. It's important to keep in mind that the definition of hypopower is completely unrelated to the function $P'(S)$ in the same way that the definition of power is completely unrelated to the function $P(S)$.

In the context of schedules we will need to specify how speed, power, and hypopower change over time, thus we define $S(t,A)$, $P(t,A)$, and $P'(t,A)$, as the speed, power, and hypopower respectively, of schedule A at time t, such that $P(t,A) = P(S(t,A))$ and $P'(t,A) = P'(S(t,A))$. If A is understood, then we drop A. Further, we denote the speed, power, and hypopower experienced by a job i run at a time t in a schedule A as $S_i(t,A)$, $P_i(t,A)$, and $P'_i(t,A)$ respectively, such that $P_i(t,A) = P(S_i(t,A))$ and $P'_i(t,A) = P'(S_i(t,A))$. Again, we drop A if A is understood.

3. Characterizing the Optimal Schedule

In this section we characterize the optimal schedule for the objective of fractional flow plus energy. We start, in section 3.1, by giving a time-indexed convex programming formulation of the problem and then in section 3.2 we use the well known KKT conditions to derive properties that are necessary and sufficient for optimality. In section 3.3 we show that these properties imply that there is only a single optimal schedule. Finally, in section 3.4, we give a simple algorithm that uses the necessary and sufficient properties to decide whether or not a schedule is optimal in $O(n^2)$ time.

3.1. Convex Programming Formulation

We now give a convex programming formulation of the problem. Let T be some sufficiently large time bound such that all jobs finish by time T in the optimal schedule. Define the variable $S(t)$ as the speed of the schedule at time t and the variable $p_j(t)$ to be the work performed on job j at time $t \geq r_j$. The problem of minimizing a linear combination of fractional weighted flow plus energy for a set of jobs with release times and arbitrary work requirements can thus be expressed as the following convex program:

$$\min \sum_{j=1}^{n} \sum_{t \geq r_j}^{T} \frac{w_j p_j(t)}{p_j}(t - r_j) + \beta \sum_{t=1}^{T} P(S(t))$$

Subject to

$$\sum_{t=r_j}^{T} p_j(t) = p_j \qquad j \in [1,n] \quad [\text{dual } \alpha_j] \qquad (1)$$

$$\sum_{j:r_j \leq t} p_j(t) = S(t) \qquad t \in [1,T] \quad [\text{dual } \delta(t)] \qquad (2)$$

$$p_j(t) \geq 0 \qquad t \geq r_j, j \in [1,n] \quad [\text{dual } \gamma_j(t)] \qquad (3)$$

The convexity of the program follows from the fact that, with the exception of $\sum P(S(t))$, the objective and constraints are linear functions of variables. $\sum P(S(t))$ is the non-negative sum of convex functions and thus is convex.

The objective has two terms, the right term is β times the energy used by the schedule. The left term is the sum over all jobs of the fractional weighted flow of the job. Constraint (1) ensures that every job is finished, constraint (2) ensures that the speed at each time is equal to the work performed at that time, and constraint (3) ensures that the work variables are non-negative. Constraints (2) and (3) ensure that the variables for speed are non-negative. To apply the KKT conditions in the next section, we associate the dual variables $\alpha_j, \delta(t)$, and $\gamma_j(t)$ with constraints, (1), (2), and (3) respectively.

3.2. KKT Conditions

The Karush-Kuhn-Tucker (KKT) conditions provide necessary and sufficient conditions to establish the optimality of feasible solutions to certain convex programs. We now describe the KKT conditions in general. Consider the following convex program,

$$\min f_0(x)$$

Subject to

$$f_i(x) \leq 0 \qquad i = 1,\ldots,n \qquad (\lambda_i)$$
$$g_j(x) = 0 \qquad j = 1,\ldots,m \qquad (\alpha_j)$$

Assume that all f_i and g_j are differentiable and that there exists a feasible solution to this program. The variable γ_i is the dual (Lagrangian multiplier) associated with the function $f_i(x)$ and similarly for α_j with $g_j(x)$. Given that all constraints are differentiable linear functions and that the objective is differentiable and convex, the KKT conditions state that necessary and sufficient conditions for optimality are

$$f_i(x) \leq 0 \qquad i = 1, \ldots, n \tag{4}$$
$$\lambda_i \geq 0 \qquad i = 1, \ldots, n \tag{5}$$
$$\lambda_i f_i(x) = 0 \qquad i = 1, \ldots, n \tag{6}$$
$$g_j(x) = 0 \qquad j = 1, \ldots, m \tag{7}$$

$$\nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) + \sum_{j=1}^{m} \alpha_j \nabla g_j(x) = 0 \tag{8}$$

Here $\nabla f_i(x)$ and $\nabla g_j(x)$ are the gradients of $f_i(x)$ and $g_j(x)$ respectively. Condition 6 is called complementary slackness.

Before applying the KKT conditions, we define the *hypopower function* of job i in schedule A as,

$$Q_i(t, A) = q_i(A) - \frac{w_i}{\beta p_i}(t - r_i) \tag{9}$$

where $q_i(A)$ is any constant such that $Q_i(t, A)$ satisfies the condition that at all times t such that i is run in A, the hypopower at which i is run is $Q_i(t, A)$. If there is no such function (no $q_i(A)$) satisfying this condition, then i does not have an associated hypopower function in schedule A. Note that regardless of the speed to power function, $P(S)$, if the function $Q_i(t, A)$ exists (it may not), then the hypopower function of i is a linearly decreasing function of time with slope $\frac{-w_i}{\beta p_i}$. We refer to the constant $q_i(A)$ as i's *initial hypopower*. We drop A if the schedule is understood.

Because the hypopower function of i gives the hypopower of i, the hypopower function implies a speed function and a power function for i, specifically, $S_i(t, A) = P^*(Q_i(t, A))$ and $P_i(t, A) = P(P^*(Q_i(t, A)))$. Figure 2 gives a visual representation of the hypopower functions for the optimal schedule when $P(S) = S^3$ for an instance consisting of three jobs: with release times 0, 60, and 200, sizes 45, 35, and 25 and weights 0.1875, 0.25, and 0.125 respectively. Figure 1 shows the speed functions that correspond to the three hypopower functions of figure 2.

Lemma 1 states that a feasible schedule is optimal if and only if

- The hypopower of all jobs are defined by the hypopower function given in equation 9.
- At all times, the job that is run is the job whose hypopower function is largest at that time.
- If a job is run, it is run at the hypopower specified by the hypopower function.

Figure 1. A three job instance viewed as speed functions.

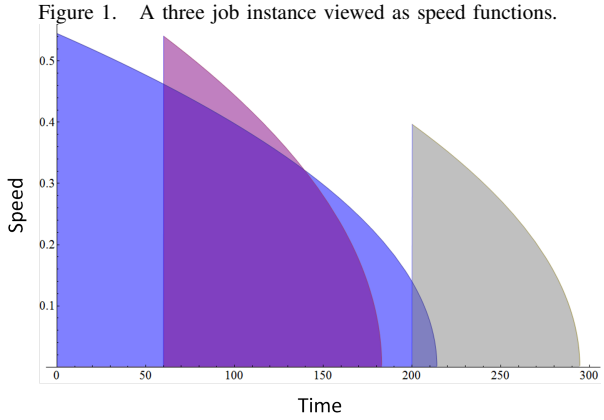
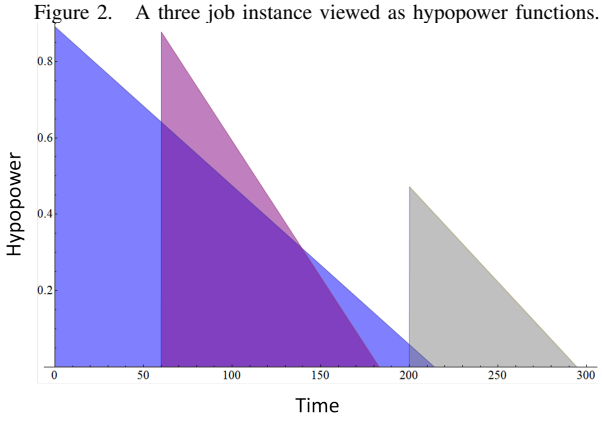


Figure 2. A three job instance viewed as hypopower functions.



Before we prove lemma 1, consider a couple of implications of these conditions.

First, an optimal schedule can be thought of as the upper envelope of the hypopower functions of all jobs. The set of times when the upper envelope is i's hypopower function are exactly the times during which i is run. Further, because the value of t that satisfies $Q_i(t) = Q_j(t)$, for any jobs i and j, is equal to the value of t that satisfies $S_i(t) = S_j(t)$, the upper envelope of the speed functions also defines an optimal schedule. The integral of the upper envelope of the speed functions equals the total work of all jobs and the integral of i's speed function, over the times when i's speed function defines the upper envelope, equals exactly i's total work. See figures 1 and 2 as examples of schedules defined by speed and hypopower functions respectively.

Second, because i's speed function is on the upper envelope of the speed functions if and only if i's hypopower function is on the upper envelope of the hypopower functions, the area under i's speed function, while i's speed function is on the upper envelope, can increase (or decrease) if and only if the area under i's hypopower function, while i's hypopower function is on the upper envelope, increases (or decreases). Thus, unless we need to calculate the specific

work done on a job, it is generally easier to work with the hypopower functions of jobs rather than the speed functions of jobs because a job's hypopower functions is linear in time, while the speed function generally is not. Thus, unless explicitly stated otherwise, we will think of any schedule as n hypopower functions or equivalently as n initial hypopower values. Note that such a representation does not guarantee optimality, but does allow full description of any optimal schedule.

Lemma 1. *A primal feasible solution to the convex program is optimal if and only if, at all times t, for all jobs j, if $p_j(t) > 0$, then $P'(S(t)) = Q_j(t)$, and if $p_j(t) = 0$ then $P'(S(t)) \geq Q_j(t)$.*

Proof: We prove the lemma by showing that the KKT conditions are exactly the three conditions of the lemma: feasibility, the hypopower of A is $Q_j(t, A)$ whenever j is run, and $Q_j(t, A)$ is a lower bound on the hypopower of the schedule if j is not run.

First note that equations 4 and 7 of the KKT conditions are simply constraining all optimal solutions to be feasible, thus we need only show that the remaining two properties are exactly equations 5, 6, and 8.

Because $q_j = P'(S(t^*)) + \frac{w_j}{\beta p_j}(t^* - r_j)$ for any time t^* such that j is run at t^*, it is sufficient to show the rest of the lemma for $P'(t) = P'(t^*) - \frac{w_j}{\beta p_j}(t - t^*)$ as this is equivalent to $Q_j(t)$. We start by computing the gradient of equation 8 of the KKT conditions. We then consider two cases for any job j: when j is running and when j is not running.

First consider equation 8 of the KKT conditions by first taking the partial derivative with respect to $S(t)$:

$$\beta P'(S(t)) - \delta(t) = 0 \text{ or equivalently } \delta(t) = \beta P'(t) \quad (10)$$

and $p_j(t)$:

$$\alpha_j - \gamma_j(t) + \delta(t) + (t - r_j)\frac{w_j}{p_j} = 0$$

or equivalently,

$$\alpha_j = \gamma_j(t) - \delta(t) - (t - r_j)\frac{w_j}{p_j} \quad (11)$$

We can then plug equation 10 into equation 11 to get

$$\alpha_j = \gamma_j(t) - \beta P'(t) - (t - r_j)\frac{w_j}{p_j} \quad (12)$$

Thus, whatever the value of α_j, it is constant for job j at all times $t \geq r_j$.

For the first case, consider any job j, and time, t^*, when j is run. Call the speed of the schedule at t^*, $S(t^*)$. By equation 12,

$$\alpha_j = \gamma_j(t^*) - \beta P'(t^*) - (t^* - r_j)\frac{w_j}{p_j} \quad (13)$$

Now consider any other time t during which j is run. Again by equation 12 we have,

$$\alpha_j = \gamma_j(t) - \beta P'(t) - (t - r_j)\frac{w_j}{p_j} \quad (14)$$

Equating equations 13 and 14 for α_j and solving for $\beta P'(t)$ gives us,

$$\beta P'(t) = -\gamma_j(t^*) + \gamma_j(t) + \beta P'(t^*) - (t - t^*)\frac{w_j}{p_j} \quad (15)$$

However, applying complementary slackness (6) to constraint 3 of our convex program, we get that $\gamma_j(t)(-p_j(t)) = 0$ and $\gamma_j(t^*)(-p_j(t^*)) = 0$. However, because we know that at both t^* and t, job j is run, both $p_j(t) > 0$ and $p_j(t^*) > 0$, thus it must be that $\gamma_j(t) = 0$ and $\gamma_j(t^*) = 0$, thus equation 15 becomes,

$$\beta P'(t) = \beta P'(t^*) - (t - t^*)\frac{w_j}{p_j}$$

or equivalently,

$$P'(t) = P'(t^*) - \frac{w_j}{\beta p_j}(t - t^*)$$

Thus we have the second condition of our lemma. Lastly, note that equation 5 of the KKT conditions is satisfied by $\gamma_j(t) = 0$, and it must be that $\gamma_j(t) = 0$ in order to satisfy complementary slackness (6) in the case that job j is run at time t.

For the second case, consider any time, t', such that job j is not run. We apply equation 12 to get

$$\alpha_j = \gamma_j(t') - \beta P'(t') - (t' - r_j)\frac{w_j}{p_j}$$

Because α_j is constant whether j is run or not, we then set this equal to equation 13 and solve for $P'(t')$ to get,

$$\begin{aligned} P'(t') &= -\frac{\gamma_j(t^*)}{\beta} + \frac{\gamma_j(t')}{\beta} + P'(t^*) - (t' - t^*)\frac{w_j}{\beta p_j} \\ &= \frac{\gamma_j(t')}{\beta} + P'(t^*) - (t' - t^*)\frac{w_j}{\beta p_j} \\ &\geq P'(t^*) - (t' - t^*)\frac{w_j}{\beta p_j} \end{aligned}$$

The equality follows from the fact that $\gamma_j(t^*) = 0$ and the inequality follows by the fact that $\beta \geq 0$ and by equation 5 of the KKT conditions which requires that $\gamma_j(t') \geq 0$. Thus we have the third condition of our lemma. Lastly, note that because $p_j(t') = 0$ when job j is not run at t', complementary slackness (6) is thus always satisfied in this case. ■

The job selection policy highest density first (HDF), schedules, at time t, the unfinished job i, with the largest density, which is defined to be w_i/p_i. By using a standard exchange argument one can show that HDF is the optimal job selection policy for weighted fractional delay. Thus we expect that any feasible schedule meeting the conditions of Lemma 1 schedules jobs in HDF order. To see why this is

indeed true, consider that the slope of $Q_i(t)$ is dependent only on β and i's density. Thus when $Q_i(t)$ and $Q_j(t)$ intersect, the less dense job will have a larger hypopower at all times after the intersection. Lemma 1 then implies that, in the optimal schedule, the denser of i and j must have been completed prior to the intersection of $Q_i(t)$ and $Q_j(t)$. Because this holds for all jobs, if $Q_i(t)$ is on the upper envelope at time t, then because $Q_i(t) \leq Q_j(t)$ for all j released by t, all released jobs with density larger than i's density must have been completed by t. However, this is the definition of HDF, thus the conditions of Lemma 1 imply HDF job selection.

3.3. The Optimal Schedule is Unique

In this section, we show that the optimal schedule is unique. We do this by examining the upper envelope of the hypopower functions for two purportedly optimal schedules, A and B, and consider the set of jobs H such that i is in H if the initial hypopower of i in A is strictly smaller than the initial hypopower of i in B. We show that area under the hypopower upper envelope, over all times a schedule is running any job in H, is larger for schedule B than schedule A. This implies the same for the speed functions, that is, the total work done on jobs in H is different in A and B, a contradiction to both schedules being optimal.

Lemma 2. *The optimal schedule is unique.*

Proof: We will prove this lemma by contradiction. Specifically, assume there are two optimal schedules, A, and B. We will convert A into B by changing the jobs one at a time. If the set H consists of all jobs i such that $q_i(A) < q_i(B)$, then we show that every time we change a job in H, the total work done on jobs in H either goes up or stays the same. And every time we change a job not in H the total work done on jobs in H either goes up or stays the same. Finally, we show that the work done on jobs in H goes up at least once, thus A does less work on jobs in H than B, a contradiction to both schedules being optimal. Instead of looking at work directly we will look at area under the hypopower curves. Proving this quantity increases implies the same for the area under the corresponding speed functions, thus completing the contradiction.

Assume A and B are each represented by n q_i values, thus schedule A can be thought of as $Q(t, A) = max_i \{Q_i(t, A)\}$ and B as $Q(t, B) = max_i \{Q_i(t, B)\}$. For any set of jobs S, define $Q(t, S \in A)$ as 0 if $max_i \{Q_i(t, A)\} > max_{i \in S} \{Q_i(t, A)\}$ and $max_{i \in S} \{Q_i(t, A)\}$ otherwise. We can similarly define $Q(t, S \in B)$. In other words, these functions are the subset of the upper envelope where some job in S is the running job. Without loss of generality, assume A has at least one job, j, such that $q_j(A) < q_j(B)$. Call the set H, all jobs, i, such that $q_i(A) < q_i(B)$ and the set L, all jobs, i, such that $q_i(A) > q_i(B)$. We can convert A into B by setting, one at a time, $q_k(A)$ to $q_k(B)$, for each job k. Consider what happens when we do this for job arbitrary job k:

If $k \in H$, then $q_k(A) < q_k(B)$. Thus $Q_k(t, A)$ increases at all times t, but $Q_j(t, A)$, for all $j \neq k$ and time t, does not decrease. These facts imply that the area under $Q(t, H \in A)$ either increases or stays the same. Further, if prior to increasing $q_k(A)$, it is the case that $Q_k(t, A) = Q(t, H \in A)$ for any t, then the area under $Q(t, H \in A)$ strictly increases.

If $k \in L$, then $q_k(A) > q_k(B)$. Thus $Q_k(t, A)$ decreases at all times t, but $Q_j(t, A)$, for all $j \neq k$ and time t, does not decrease. These facts also imply that the area under $Q(t, H \in A)$ either increases or stays the same.

We still need at least a single increase in the area under $Q(t, H \in A)$. However, recall that we are guaranteed to have some job j such that such that $q_j(A) < q_j(B)$. Thus if we convert A to B by starting with j, it must be that $Q_j(t, A) = Q(t, H \in A)$ for at least one time t, else A does no work on j, a contradiction to the optimality of A. ■

3.4. Checking a Schedule for Optimality

We conclude section 3 by giving an algorithm to check the optimality of a schedule in time $O(n^2)$. The algorithm takes as input the initial hypopower for each job j. If the input schedule is not in this form, then $q_i = P_i'(t^*) + (w_i/\beta p_i)(t^* - r_i)$ for any time t^* when j is run in the input schedule. If the resulting hypopower functions are not optimal then the input schedule is not optimal. If the resulting hypopower functions are optimal, then determining if the input schedule is optimal reduces to the problem of deciding if the input schedule is identical to the schedule produced by the upper envelope of the resulting hypopower functions.

Because the hypopower functions are linear, when two hypopower functions intersect, the function defined by a job of lower density will be strictly larger at all times after the intersection. Thus any hypopower function is involved in at most 1 crossing on the upper envelope with a lower density job, specifically the earliest crossing with a lower density job. Thus there are most n such crossings, and it is not too hard to see that for each hypopower function, $Q_i(t)$, we can find, in linear time, the earliest time, if it exists, that $Q_i(t)$ crosses some $Q_j(t)$ such that $w_i/p_i > w_j/p_j$ and the crossing time is at least $max\{r_i, r_j\}$. Thus one can, in $O(n^2)$ time, compute the upper envelope of the hypopower functions.

4. Applying the Homotopic Approach

The main idea of the homotopic approach is to start with a schedule we can easily compute for some β' and slowly change β', calculating the new optimal schedule each time we do so, until $\beta' = \beta$. This seemingly requires that the optimal schedules for infinitesimally different β's must be closely related, so to find the new optimal schedule, when

we change β' to $\beta' + \epsilon$, we only have to examine schedules that are close to the previously computed optimal schedule for β'. In section 4.1 we discuss how to easily find an initial optimal schedule that can be used as the starting point for a homotopic algorithm. In section 4.2 we prove that the initial hypopowers change continuously as a function of β. (Although, perhaps somewhat counter-intuitively, the initial hypopowers are not monotone in β.) This then allows us to obtain an efficient homotopic algorithm for computing the optimal energy trade-off schedule for a small number of jobs by simply searching over schedules with nearly identical initial hypopowers. In section 4.3 we report on a Java-based animation of this algorithm.

4.1. Finding an Initial Optimal Schedule

If the processor has a maximum speed, then initially $\beta = 0$, and the optimal schedule always runs at the maximum speed if there are unfinished jobs, and uses HDF to determine which job to run. If the processor does not have a maximum speed, we choose β small enough such that every job is completed before any other jobs are released. For each job j, we can calculate a β_j such that it completes before any other job is released, and then take β to be the minimum over all β_j.

4.2. The Optimal Schedule is a Continuous Function of β

In this section we show that the initial hypopowers are a continuous function of β. To this end, we first define the initial hypopower of job j, as a function of β, as $q_j(\beta)$ and likewise the hypopower function for j, as a function of β, as $Q_j(t, \beta)$.

We show in Lemma 3 that the optimal schedule changes continuously as a function of β. By Lemma 1, the optimal schedule can be described as set of n initial hypopowers, thus we show Lemma 3 by showing that these initial hypopowers are continuous functions of β. We do this by showing that if there is some non-empty set of jobs whose initial hypopowers are increasing discontinuously at β, then for some small increase in β, the total work done by the optimal schedule on this set of jobs increases. However, this is a contradiction to optimality. If the initial hypopowers are decreasing continuously, then the same method can be used for a small decrease in β.

Lemma 3. *The initial hypopowers are a continuous function of β.*

Proof: We show the lemma by showing that, in the optimal schedule, for all jobs j, the value $q_j(\beta)$ is a continuous function of β. That is, assume that there is at least one $q_j(\beta)$ value that is not continuous in β. More precisely, $q_j(\beta)$ is *discontinuously increasing* at $\beta > 0$ if there exists constants $c_1, c_2 > 0$ such that for all $\epsilon \in (0, c_2)$, $q_j(\beta + \epsilon) \geq q_j(\beta) + c_1$. Likewise, $q_j(\beta)$ is *discontinuously decreasing* at $\beta > 0$ if there exists constants $c_1, c_2 > 0$ such that for all $\epsilon \in (0, c_2)$, $q_j(\beta - \epsilon) \geq q_j(\beta) + c_1$.

We start by assuming the following claim:

Claim 1. *For any job j with discontinuously increasing $q_j(\beta)$, there exists some $\epsilon' \in (0, c_2)$ such that the following two properties hold:*

1) *For all times t, $Q_j(t, \beta') > Q_j(t, \beta)$ for all $\beta' \in (\beta, \beta + \epsilon']$*
2) *For all jobs i such that $q_i(\beta)$ is not discontinuously increasing at β, define t_c as the solution of $Q_j(t_c, \beta) = Q_i(t_c, \beta)$ and t'_c as the solution of $Q_j(t'_c, \beta') = Q_i(t'_c, \beta')$, then for all $\beta' \in (\beta, \beta + \epsilon']$:*
   - *If $w_i/p_i > w_j/p_j$ then $t_c > t'_c$ else*
   - *If $w_i/p_i < w_j/p_j$ then $t_c < t'_c$*

Likewise if $q_j(\beta)$ is discontinuously decreasing, then the same facts hold except $\beta' \in [\beta - \epsilon', \beta)$.

The proof of the lemma is now the same as for Lemma 2: we convert from one schedule into another by changing jobs one at a time and show that there is some set of jobs such that the total work increases. For the sake of contradiction, assume that there is at least one job such that the job's $q(\beta)$ is discontinuous at some value of β. There are two cases, if $q(\beta)$ is discontinuously increasing and if $q(\beta)$ is discontinuously decreasing.

If there is at least one discontinuously increasing $q(\beta)$ function, consider the smallest $\beta = \beta_1$ such that there are some set of jobs, H with $q(\beta)$ functions that are discontinuously increasing at β_1. By Claim 1, there exists some $\epsilon' > 0$, such that the properties of Claim 1 hold for all jobs in H. We now convert the optimal schedule at β_1 to the optimal schedule at any $\beta' \in (\beta_1, \beta_1 + \epsilon']$ following the proof of Lemma 2 with the main difference being that for some jobs $i \notin H$, it may be that q_i increases, however Claim 1 ensures that changing them does not decrease the work done on any job in H.

If there is at least one discontinuously decreasing $q(\beta)$ function, we follow the same method except we start from the largest such β_1.

All that remains is to show Claim 1.

First note that if we find an ϵ'_1 satisfying the first property and an ϵ'_2 satisfying the second property, then $\epsilon' = min\{\epsilon'_1, \epsilon'_2\}$ will satisfy both properties. Thus we can find an ϵ' value separately for each property.

We start by showing that if $q_j(\beta)$ is discontinuously increasing at β, the first property holds. In other words, we want to find an ϵ' such that the following holds for any $\beta' \in (\beta, \beta + \epsilon']$:

$$q_j(\beta) - \frac{w_j}{p_j \beta}(t - r_j) < q_j(\beta') - \frac{w_j}{p_j \beta'}(t - r_j)$$

As ϵ' is constrained to be less than c_2 and by definition

of the discontinuity of $q_j(\beta)$, we have that,

$$q_j(\beta) + c_1 - \frac{w_j}{p_j\beta'}(t - r_j) < q_j(\beta') - \frac{w_j}{p_j\beta'}(t - r_j)$$

Thus it is sufficient to show

$$q_j(\beta) - \frac{w_j}{p_j\beta}(t - r_j) < q_j(\beta) + c_1 - \frac{w_j}{p_j\beta'}(t - r_j)$$

Or equivalently,

$$-\frac{w_j}{p_j\beta}(t - r_j) < c_1 - \frac{w_j}{p_j\beta'}(t - r_j) \tag{16}$$

However, this is clearly true for all $\beta' > \beta$ as $c_1 > 0$, thus any $\epsilon' \in (0, c_2)$ will satisfy the inequality. If instead $q_j(\beta)$ discontinuously increasing, and we require $\beta' \in [\beta - \epsilon', \beta)$, we can re-arrange inequality 16 to get

$$\frac{w_j}{p_j}(t - r_j)\left(\frac{1}{\beta'} - \frac{1}{\beta}\right) < c_1 \tag{17}$$

Because $(w_j/p_j)(t - r_j)\,(1/\beta' - 1/\beta)$ is a decreasing function of β', if we can find a single $\beta' = \beta - \epsilon'$ such that 17 holds then we are done. However, consider that for $\epsilon' = 0$,

$$\frac{w_j}{p_j}(t - r_j)\left(\frac{1}{\beta - \epsilon'} - \frac{1}{\beta}\right) < c_1$$

holds. Thus, because $(w_j/p_j)(t - r_j)\,(1/(\beta - \epsilon') - 1/\beta)$ is continuous in ϵ' and $c_1 > 0$, there must be some $\epsilon' > 0$ for which this holds.

Now we show that if $q_j(\beta)$ is discontinuously increasing at β, then the second property holds. First we give the explicit definition of t_c and t'_c:

$$t_c = \frac{(q_j(\beta) - q_i(\beta))\beta + \frac{w_j r_j}{p_j} - \frac{w_i r_i}{p_i}}{\frac{w_j}{p_j} - \frac{w_i}{p_i}} \tag{18}$$

Likewise, solving for t'_c gives

$$t'_c = \frac{(q_j(\beta') - q_i(\beta'))\beta' + \frac{w_j r_j}{p_j} - \frac{w_i r_i}{p_i}}{\frac{w_j}{p_j} - \frac{w_i}{p_i}} \tag{19}$$

Now we would like to show that $t_c > t'_c$ if $w_i/p_i > w_j/p_j$ and $t_c < t'_c$ if $w_i/p_i < w_j/p_j$. Equivalently, $t'_c - t_c < 0$ and $t'_c - t_c > 0$ respectively. Solving directly for $t'_c - t_c$ using equations 18 and 19 we get

$$t'_c - t_c = \frac{(q_j(\beta') - q_i(\beta'))\beta' - (q_j(\beta) - q_i(\beta))\beta}{\frac{w_j}{p_j} - \frac{w_i}{p_i}}$$

Note that if $w_i/p_i > w_j/p_j$, then $w_j/p_j - w_i/p_i < 0$ and if $w_i/p_i < w_j/p_j$ then $w_j/p_j - w_i/p_i > 0$, thus for both cases it is sufficient to show that

$$(q_j(\beta') - q_i(\beta'))\beta' - (q_j(\beta) - q_i(\beta))\beta > 0$$

By definition of job j and the valid range of β', $q_j(\beta') \geq q_j(\beta) + c_1$, thus giving us

$$(q_j(\beta') - q_i(\beta'))\beta' - (q_j(\beta) - q_i(\beta))\beta \geq$$
$$(q_j(\beta) + c_1 - q_i(\beta'))\beta' - (q_j(\beta) - q_i(\beta))\beta$$

Or equivalently,

$$(q_j(\beta') - q_i(\beta'))\beta' - (q_j(\beta) - q_i(\beta))\beta \geq$$
$$c_1\beta' + q_j(\beta)(\beta' - \beta) + q_i(\beta)\beta - q_i(\beta')\beta' \tag{20}$$

In the case that $q_j(\beta)$ is discontinuously increasing, we have that $\beta' > \beta$, which implies that $q_j(\beta)(\beta' - \beta) > 0$ and $c_1\beta' > c_1\beta$. Thus it is sufficient to show

$$c_1\beta + q_i(\beta)\beta - q_i(\beta')\beta' > 0 \tag{21}$$

If $q_i(\beta)$ is continuous at β, the limit of $q_i(\beta)\beta - q_i(\beta')\beta'$ as β' goes to β is 0. If $q_i(\beta)\beta - q_i(\beta')\beta'$ goes to 0 from larger than 0, then inequality 21 holds. Because $c_1\beta$ is strictly larger than 0, if $q_i(\beta)\beta - q_i(\beta')\beta'$ goes to 0 from less than 0, then we can make $q_i(\beta)\beta - q_i(\beta')\beta'$ arbitrarily close to 0, in a continuous manner, as we decrease β'. Thus, there must be some $\epsilon' > 0$ such that $-(q_i(\beta)\beta - q_i(\beta + \epsilon'')) < c_1\beta$ for all $\epsilon'' < \epsilon'$.

If however $q_i(\beta)$ is discontinuously decreasing at β, then by the first property, there is some small ϵ_1 such that for $\beta_1 \in (\beta, \beta + \epsilon_1]$, $Q_i(t, \beta_1) < Q_i(t, \beta)$ for all t. This implies that $q_i(\beta)\beta - q_i(\beta')\beta' > 0$ for $\beta' \in (\beta, \beta + \epsilon_1]$, thus inequality 21 holds for any $\epsilon' < \epsilon_1$.

Finally consider the case that $q_i(\beta)$ is discontinuously decreasing at β and thus we want $\beta' \in [\beta - \epsilon, \beta)$. The only difference is that $c_1\beta' \geq c_1(\beta - \epsilon') = c_1\beta - \epsilon' c_1$ and $q_j(\beta)(\beta' - \beta) \geq -q_j(\beta)\epsilon'$. Applying these to equation 20, we need to show the following for all β',

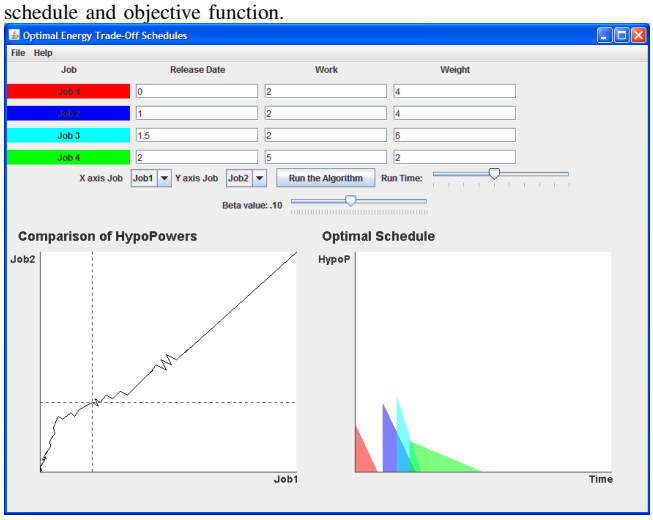
$$c_1\beta + q_i(\beta)\beta - q_i(\beta')\beta' - \epsilon' c_1 - q_j(\beta)\epsilon' > 0 \tag{22}$$

However, consider that both $-\epsilon' c_1$ and $-q_j(\beta)\epsilon'$ are continuous and go to 0 as ϵ' goes to 0. In the case that $q_i(\beta)$ is continuous, $q_i(\beta)\beta - q_i(\beta')\beta' - \epsilon' c_1 - q_j(\beta)\epsilon'$ is thus continuous, has a limit of 0 at $\epsilon' = 0$, and we can therefore use the same reasoning as in the previous continuous case. If $q_i(\beta)$ is discontinuous, it must be discontinuously increasing in β or in other words, discontinuously decreasing as we lower β. Thus there exists some ϵ_1 such that for all $\beta' \in [\beta - \epsilon_1, \beta)$, $q_i(\beta)\beta - q_i(\beta')\beta' \geq c_3 > 0$, but this just makes the left hand side of inequality 22 larger than if $q_i(\beta)$ was continuous. ■

4.3. Implementation

We built a java-based application that, using the homotopic approach presented in this paper, allows one to view the evolution of the optimal schedule as a function of β. Our application allows the user to view the optimal schedule of up to four jobs. Moving a slider changes the value of β in real time, thus allowing the user to see how the optimal schedule evolves over several values of β in the range of 0 to 1. The application displays the optimal schedule as hypopower over time, and also displays how the initial hypopowers of any two jobs change as a function of β. Figure 3 shows the application running for a 4 job instance

Figure 3. The interface of our implementation of the homotopic algorithm. Moving the slider back and forth changes the value of β and thus the schedule and objective function.



that can be seen at the top of the figure. The implementation can be found at http://www.cs.pitt.edu/~kirk/erg2. It is our hope that our animation implementation will be of use in gaining intuition on how to attack some of the open questions presented in the next section.

The primary difficulty in developing this animation was reasonably bounding the number of potential solutions that the program examined to find the next set of initial hypopowers given the current set of initial hypopowers and some small change in β. It is not clear how to upper bound the change in the initial hypopowers as a function of the change in β. This necessitated that the program search a large neighborhood to find the new optimal schedule, which severely restricted the granularity of candidate hypopowers since if we consider m possible values for each initial hypopower, the running time of exhaustive search is $\Omega(m^n)$. To achieve a reasonable degree of accuracy within a reasonable period of time, we set the initial granularity to be rather course, and then recursively searched the most promising subregion. Thus there may be small inaccuracies and flickers in the displayed schedule. We believe that in the default setting that these inaccuracies are negligible. We provide a slider which allows the user to adjust the relative importance between run time and accuracy.

5. Product Objectives

In this section we consider objectives of the form $\mathcal{Q}^\sigma \times \mathcal{E}$, where $\mathcal{Q}$ is a scheduling objective, $\mathcal{E}$ is energy, and σ is a parameter representing the relative importance of the scheduling objective versus energy. In Lemma 4, we show that (perhaps counter-intuitively) this product objective is not particularly interesting from a theoretical perspective if the static power is zero. We do this by showing that if the power function is of the form $P(s) = s^\alpha$, then the optimal solution is to either always go as fast as possible or to always go as slow as possible.

When the scheduling objective $\mathcal{Q}$ is integer flow, we are unable to characterize the optimal scheduling for the objective $\mathcal{Q}^\sigma \times \mathcal{E}$ for the same reasons that we are unable to characterize the optimal schedules for the additive objective with integral flow. We therefore only consider fractional flow. The next issue that arises is during what time period does one count the static power's contribution to energy. Perhaps the most natural assumption might be to only count static power when the processor is running jobs. But this has various mathematical issues, such as it then becomes difficult to write a reasonable mathematical program. Thus we assume that there is a specified time period T, add a constraint that all jobs much be finished by time T, and assume that static power accumulates during exactly the period T. So the energy arising from static power is $P(0)T$. We characterize the optimal schedule in Lemma 5. In Lemma 6 and Lemma 7 we show that, for a fixed instance, the schedules that are optimal for the product objective $\mathcal{Q}^\sigma \times \mathcal{E}$ (for any σ) are a (generally strict) subset of the schedules that are optimal for the sum objective $\mathcal{Q}+\beta\mathcal{E}$ (for any β). We then show in Lemma 8 that it is unlikely that one will be able to compute an optimal schedule for $\mathcal{Q}^\sigma \times \mathcal{E}$ using a homotopic approach as the optimal schedule may be a discontinuous function of σ.

Lemma 4. *Assume that the power function is $P(s) = s^\alpha$ and the scheduling objective $\mathcal{Q}$ is either fractional flow or integral flow. Then for every instance, the optimal schedule for the objective $\mathcal{Q}^\sigma \times \mathcal{E}$ either always runs the processor as slowly as possible, or always runs the processor as fast as possible.*

Proof: We give the proof for integral flow; the proof for fractional flow is similar. Consider an instance with a single job with work 1. When running at constant speed s, the time to finish the job is $1/s$ and the energy used is $s^\alpha/s = s^{\alpha-1}$, which yields an objective value of $\frac{s^{\alpha-1}}{s^\sigma} = s^{\alpha-\sigma-1}$. Thus, if $\alpha - \sigma - 1 > 0$, the objective is minimized at 0 by running as slow as possible, while if $\alpha - \sigma - 1 < 0$, the objective is minimized again at 0 by running as fast as possible. Finally, if $\alpha - \sigma - 1 = 0$, any schedule that runs the job at constant speed is optimal so running either as fast or as slow as possible is optimal. In any unweighted, unit work n job instance, if $\alpha - \sigma - 1 \leq 0$, the schedule that runs jobs as fast as possible is still optimal, and if $\alpha - \sigma - 1 > 0$ the schedule that runs jobs as slow as possible is also still optimal, since the increased flow at every time only increases the objective by a factor of $O(n^{2\sigma})$. It is straightforward to see that this to extends when jobs have weights and arbitrary work requirements since in this case the objective would, at worst, be multiplied by an additional value dependent on the input, but not the schedule. ■

The problem of minimizing $\mathcal{Q}^\sigma \times \mathcal{E}$ over a time period T

can be expressed as the following mathematical program:

$$\min \left(\sum_{j=1}^{n} \sum_{t \geq r_j}^{T} \frac{w_j p_j(t)}{p_j} (t - r_j) \right)^{\sigma} \left(\sum_{t=1}^{T} P(S(t)) \right)$$

Subject to:

$$\sum_{t=r_j}^{T} p_j(t) = p_j \qquad j \in [1,n] \quad [\text{dual } \alpha_j] \qquad (23)$$

$$\sum_{j:r_j \leq t} p_j(t) = S(t) \qquad t \in [1,T] \quad [\text{dual } \delta(t)] \qquad (24)$$

$$p_j(t) \geq 0 \qquad t \geq r_j, j \in [1,n] \quad [\text{dual } \gamma_j(t)] \qquad (25)$$

Note that these constraints are the same as (1), (2), and (3) from section 3.1. Unlike there, however, this mathematical program is not convex, and so the KKT conditions (section 3.2) provide only necessary conditions for optimality.

Before applying the KKT conditions, we define a new hypopower function of job i in schedule A as,

$$\widetilde{Q}_i(t, A) = \tilde{q}_i(A) - \frac{\mathcal{E}(\mathcal{A})}{\mathcal{Q}(\mathcal{A})} \frac{\sigma w_i}{p_i}(t - r_i) \qquad (26)$$

where $\mathcal{E}(\mathcal{A})$ is the total energy used by schedule A, $\mathcal{Q}(\mathcal{A})$ is the total fractional flow incurred by schedule A, and $\tilde{q}_i(A)$ is any constant such that $\widetilde{Q}_i(t, A)$ satisfies the condition that at all times t such that i is run in A, the hypopower at which i is run is $\widetilde{Q}_i(t, A)$. Note that (26) is very similar to the hypopower function defined by equation (9), and the same qualifications apply. The only difference is that the slope is now $\frac{\mathcal{E}(\mathcal{A})}{\mathcal{Q}(\mathcal{A})} \frac{\sigma w_i}{p_i}$, which is a function of σ as well as the total energy and total flow used by the entire schedule.

Lemma 5 states that if a feasible schedule is optimal, the following must be true:

- The hypopower of all jobs are defined by the hypopower function given in equation 26.
- At all times, the job that is run is the job whose hypopower function is largest at that time.
- If a job is run, it is run at the hypopower specified by the hypopower function.

Lemma 5. *If a solution to the mathematical program is optimal then at all times t, for all jobs j, if $p_j(t) > 0$, then $P'(S(t)) = \widetilde{Q}_j(t)$, and if $p_j(t) = 0$ then $P'(S(t)) \geq \widetilde{Q}_j(t)$.*

Proof: This proof is almost identical to the proof of lemma 1, so only the differences are highlighted, with the main difference being the difference in objective functions. In regards to the different objective, first note that minimizing the objective $\mathcal{Q}^{\sigma} \times \mathcal{E}$ is equivalent to minimizing $\sigma \log(\mathcal{Q}) + \log(\mathcal{E})$. Thus, when we take the partial derivative with respect to $S(t)$ we obtain

$$\frac{P'(S(t))}{\sum_{t'=0}^{T} P(S(t'))} - \delta(t) = 0$$

or equivalently,

$$\delta(t) = \frac{P'(S(t))}{\mathcal{E}} \qquad (27)$$

and $p_j(t)$:

$$\alpha_j - \gamma_j(t) + \delta(t) + \frac{\sigma (t - r_j) \frac{w_j}{p_j}}{\sum_{j'=1}^{n} \sum_{t' \geq r_{j'}}^{T} \frac{w_{j'} p_{j'}(t')}{p'_j} (t' - r_{j'})} = 0$$

or equivalently,

$$\alpha_j = \gamma_j(t) - \delta(t) - \frac{\sigma (t - r_j) w_j}{\mathcal{Q}\, p_j} \qquad (28)$$

Observe that (27) and (28) are almost identical to (10) and (11) from lemma 1, the only differences being the coefficients on some of the terms. The rest of the proof of this lemma follows exactly as the proof of lemma 1, only requiring the additional observation at one point that $\mathcal{E} \geq 0$. ■

Lemma 6. *For any instance, the set of all optimal schedules for the product objective (for any σ) is a subset of the set of all optimal schedules for the sum objective (for any β).*

Proof: Fix an arbitrary σ. In the optimal schedule for that σ, if it exists, each job has a linearly decreasing hypopower function with slope $\sigma \frac{\mathcal{E}}{\mathcal{Q}}$ times its density. If we set $1/\beta = \sigma \frac{\mathcal{E}}{\mathcal{Q}}$, then we have a set of identical hypopower functions for the sum objective for that β. Thus if the schedule is optimal for that σ in the product objective, it must also be optimal for β in the sum objective. Thus the optimal schedule for any σ in the product objective has a corresponding optimal schedule for some β in the sum objective. ■

Lemma 7. *There are instances where there are schedules that are optimal for the sum objective but are not optimal for the product objective for any σ.*

Proof: Consider a two job instance where the jobs have release time 0 and 60, work 11 and 7, and weight 0.15 and 1.5, respectively. The power function is $P(s) = s^3 + .01/T$, where T is sufficiently big so that finishing before T is not a tight constraint. Thus the total static energy used is 0.01. As an example, see Figure 4 with $\sigma = 1.79$ and Figure 5. For each β, Figure 4 shows the value of σ for which the optimal sum schedule is a locally optimum schedule for the produce objective. Simple calculations show that different local optimums have different objective values in the product objective. ■

Lemma 8. *The optimal schedule for the product objective is not necessarily a continuous function of σ.*

Figure 4. β vs σ for schedules satisfying the KKT conditions in a two job instance.

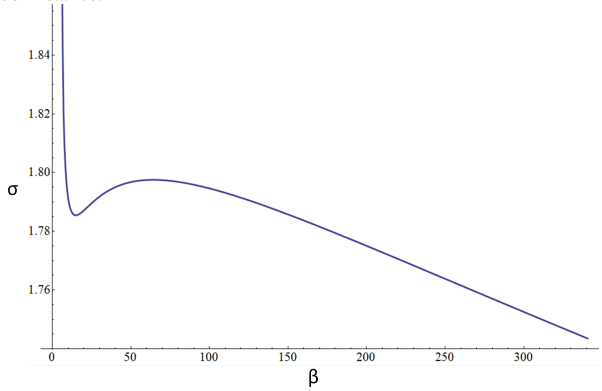
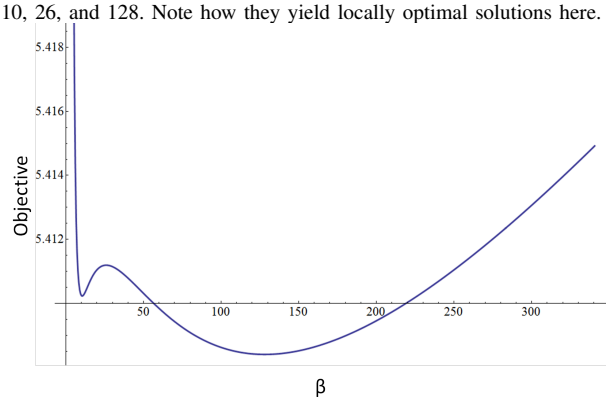


Figure 5. β vs (the log of) $\mathcal{Q}^{\sigma} \times \mathcal{E}$ for $\sigma = 1.79$ in the same instance as Figure 4. The β values that give $\sigma = 1.79$ in in Figure 4 are approximately 10, 26, and 128. Note how they yield locally optimal solutions here.



Proof: To see this, consider the example in Figure 4. Here, for all σ less than $\sigma_1 \approx 1.785$ or greater than $\sigma_2 \approx 1.80$ there is only one β that maps to it, but there are multiple β that map to all $\sigma \in [\sigma_1, \sigma_2]$. Further, in this case, for any σ, only one of the up to three β's that provide KKT condition satisfying schedules for that σ minimize the objective. This implies that the function mapping a σ to the β that shares its optimal schedule is not continuous since it is an inverse function of a continuous function that is not one-to-one. This further implies the optimal schedule does not change continuously from σ_1 to σ_2, since a discontinuous increase (or decrease) in β implies a discontinuous decrease (or increase) in the slopes of all hypopower functions, so if the initial hypopowers did not also change discontinuously, the resulting schedule after the discontinuity would do too little (or too much) work, and thus not be feasible (or optimal). ■

6. Open Questions

Our investigations have revealed some natural open questions, which we list below. We believe that it is quite plausible that the introduction of these problems into the literature might be the most lasting contribution of this paper.

Question 1: *Given the configuration of the optimal energy trade-off schedule (when the quality of service measure is weighted fractional flow), can the optimal schedule be computed in polynomial time?* In this context, a configuration is the sequence of jobs run on the processor as well as whether a job completes before, at, or after the release of the next job to be run (which is not necessarily the next job that was released). This seems to be the natural definition of a configuration, as the configuration and β uniquely defines a set of equations for which the initial hypopowers of the optimal schedule for that configuration are the sole (real) solution. [1] was able to use binary search to solve these equations. We were unable to compute the optimal schedule even knowing the optimal configuration. The main difficulty is that even when the power function is s^{α} the configuration equations yield a series of α-degree polynomial equations (in the variables hypopower, or speed, or interval length). We know of no technique to acquire a general algebraic solution to these equations. Another natural alternative approach is to create a mathematical program from the equations, and solve them using some standard optimization algorithm. However, any formulation we examined had equations relating to work that were not convex (again, in variables hypopower, or speed, or interval length). More precisely, let $p_j(\vec{x})$ be the amount of work done on job j for some variable assignment $\vec{x}$. Then for variable assignments $\vec{x}$ and $\vec{y}$, where $p_j(\vec{x}) = p_j(\vec{y}) = p_j$, there exist some situations where $p_j((\vec{x}+\vec{y})/2) > p_j$ and other situations where $p_j((\vec{x}+\vec{y})/2) < p_j$. Thus one can not seemingly use convex optimization.

Question 2: *Is there an efficient algorithm to compute the optimal energy trade-off schedule (when the quality of service measure is weighted fractional flow) for $\beta + \epsilon$ given the optimal trade-off schedule for β?* As we know how to detect when the optimal configuration changes and how to find the new optimal configuration, this is closely related to the previous open question.

Question 3: *As a function of the number of jobs n, how many times can the optimal configuration change as β changes?* [1] shows that for unit jobs and when the quality of service measure is total delay, the number of configuration changes is $O(n^2)$. We would be quite surprised if the number of configurations changes in our setting was not also polynomially bounded, although we do not know how to prove any upper bound that is a function of n. The problem is because the initial hypopowers are not monotone in β, we do not even know how to show that a particular configuration will be optimal for a contiguous collection of β's.

Question 4: *Is there a polynomial time algorithm to verify the optimality of schedule for the objective of total (integer) delay plus energy?* Using insights from [1], one can design a polynomial time algorithm that given a schedule S and

a configuration C, can verify the optimality of S among schedules in configuration C. But we do not know how to verify that a schedule is in the optimal configuration.

Question 5: *For what quality of service measures can homotopic optimization be used to compute optimal energy trade-off schedules?* It is not completely implausible that one can characterize the natural quality of service measures where this is possible.

ACKNOWLEDGMENT

Supported in part by an IBM Faculty Award, and NSF grants CCF-0830558 and 1115575. Supported partially by GDR RO, ANR TODO and a grant by the Ecole Doctorale of the University of Evry.

REFERENCES

[1] K. Pruhs, P. Uthaisombut, and G. J. Woeginger, "Getting the best response for your erg," *ACM Transactions on Algorithms*, vol. 4, no. 3, 2008.

[2] Y. Zwols, http://www.cs.mcgill.ca/~yzwols/software.html.

[3] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," *ACM Transactions on Algorithms*, vol. 3, no. 4, 2007.

[4] N. Bansal, H.-L. Chan, and K. Pruhs, "Speed scaling with an arbitrary power function," in *SODA*, C. Mathieu, Ed. SIAM, 2009, pp. 693–701.

[5] L. L. H. Andrew, A. Wierman, and A. Tang, "Optimal speed scaling under arbitrary power functions," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, pp. 39–41, 2009.

[6] N. Bansal, K. Pruhs, and C. Stein, "Speed scaling for weighted flow time," *SIAM J. Comput.*, vol. 39, no. 4, pp. 1294–1308, 2009.

[7] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong, "Speed scaling functions for flow time scheduling based on active job count," in *ESA*, ser. Lecture Notes in Computer Science, D. Halperin and K. Mehlhorn, Eds., vol. 5193. Springer, 2008, pp. 647–659.

[8] H.-L. Chan, J. Edmonds, T. W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs, "Nonclairvoyant speed scaling for flow and energy," *Algorithmica*, vol. 61, no. 3, pp. 507–517, 2011.

[9] S.-H. Chan, T. W. Lam, and L.-K. Lee, "Non-clairvoyant speed scaling for weighted flow time," in *ESA (1)*, ser. Lecture Notes in Computer Science, M. de Berg and U. Meyer, Eds., vol. 6346. Springer, 2010, pp. 23–35.

[10] S.-H. Chan, T. W. Lam, L.-K. Lee, H.-F. Ting, and P. Zhang, "Non-clairvoyant scheduling for weighted flow time and energy on speed bounded processors," *Chicago J. Theor. Comput. Sci.*, vol. 2011, 2011.

[11] H.-L. Chan, T. W. Lam, and R. Li, "Tradeoff between energy and throughput for online deadline scheduling," in *WAOA*, ser. Lecture Notes in Computer Science, K. Jansen and R. Solis-Oba, Eds., vol. 6534. Springer, 2010, pp. 59–70.

[12] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, oct 1994, pp. 8 –11.