Speed Scaling to Manage Temperature

Nikhil Bansal[1] and Kirk Pruhs[2]

[1] IBM T. J. Watson Research Center nikhil@us.ibm.com
[2] Computer Science Department, University of Pittsburgh, kirk@cs.pitt.edu *

Abstract. We consider speed scaling algorithms to minimize device temperature subject to the constraint that every task finishes by its deadline. We assume that the device cools according to Fourier's law. We show that the optimal offline algorithm proposed in [18] for minimizing total energy (that we call YDS) is an $O(1)$-approximation with respect to temperature. Tangentially, we observe that the energy optimality of YDS is an elegant consequence of the well known KKT optimality conditions. Two online algorithms, AVR and Optimal Available, were proposed in [18] in the context of energy management. It was shown that these algorithms were $O(1)$-competitive with respect to energy in [18] and [2]. Here we show these algorithms are not $O(1)$-competitive with respect to temperature. This demonstratively illustrates the observation from practice that power management techniques that are effective for managing energy may not be effective for managing temperature. We show that the most intuitive temperature management algorithm, running at such a speed so that the temperature is constant, is surprisingly not $O(1)$-competitive with respect to temperature. In contrast, we show that the online algorithm BKP, proposed in [2], is $O(1)$-competitive with respect to temperature. This is the first $O(1)$-competitiveness analysis with respect to temperature for an online algorithm.

1 Introduction

In May Intel abruptly announced that it had scrapped the development of two new computer chips (code named Tejas and Jayhawk) for desktops/servers in order to rush to the marketplace a more efficient chip technology more than a year ahead of schedule. Analysts said the move showed how eager the world's largest chip maker was to cut back on the heat its chips generate. Intel's method of cranking up chip speed was beginning to require expensive and noisy cooling systems for computers [17]. (Current estimates are that cooling solutions are rising at $1 to $3 per watt of heat dissipated [15].) This demonstrates that exponentially growing device power consumption, the power densities in microprocessors have doubled every three years [15], has reached the critical point

* Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, and CCF-0448196.

where power management, and in particular temperature management, is one of the main criteria driving device design.

There is an extensive literature on power management in computing devices, see for example [4, 12, 16]. However, the vast majority of this literature focuses on power management to conserve energy, and not on power management to reduce temperature. Temperature and energy are different physical entities with quite different properties. Bluntly, if the processor in your mobile device exceeds its energy bound, your battery is exhausted; If your processor exceeds it thermal threshold, the processor dies. Power management schemes for conserving energy focus on reducing cumulative power, while power management schemes for reducing temperature must focus more on instantaneous power. Therefore power management schemes designed to conserve energy may not perform as well when the goal is to reduce temperature. In fact, many low-power techniques are reported to have little or no effect on temperature [15]. Temperature aware design is therefore a distinct, albeit related, area of study to energy aware design [15].

Both in academic research and practice, dynamic voltage/frequency/speed scaling is the dominant technique for power management. Speed scaling involves dynamically changing the speed of the processor. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. Some modern processors already are able to sense their own temperature, and thus can be slowed down or shut down so the processor temperature will stay below its thermal threshold [15]. In this paper we study speed scaling strategies to manage temperature.

1.1 Problem Formulation

We need to model the cooling behavior of a device. Cooling is a complex phenomenon that can not be modeled completely accurately by any simple model [14]. Still we require some first order simple approximation. We assume that the rate of cooling follows Fourier's Law, which states that the rate of cooling is proportional to the difference in temperature between the object and the ambient environment temperature. We assume that the environment has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. A first order approximation for rate of change T' of the temperature T is then $T' = aP - bT$, where P is the supplied power, and a, b are constants [14].

We make the standard assumption that if the processor is run at speed s, then the power consumption is $P(s) = s^\alpha$ for some constant $\alpha > 1$ [4, 2, 11, 1]. For CMOS based devices, which will likely remain the dominant technology for the near term future, the well known cube-root rule is that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$, i.e., the power is proportional to the cube of the speed [4].

The input is a collection of tasks, where each task i has a release time r_i when it arrives into the system, an amount of work w_i that must be performed to complete the task, and a deadline d_i for completing this work. In the online version of the problem, the scheduler learns about a task only at its release time; at this time, the scheduler also learns the exact work requirement and the

deadline of the task. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks [5]. A schedule specifies which task to run at each time, and at what speed that task should be run. The work performed on a job is the integral over time of the speed that this job is run at that time. The schedule must be feasible, that is, w_i units of work must be performed on task i during the time interval $[r_i, d_i]$. This is always possible since the processor can run at any speed. Preemption is allowed, that is, the scheduler may suspend a task and then later restart the task from the point of suspension.

Energy is power integrated over time. In the energy problem, the goal is to find a feasible schedule that minimizes the total energy used. In the temperature problem, the goal is to find a feasible schedule that minimizes the the maximum temperature that the device reaches over the whole history of the schedule. That is, we want to determine the least thermal threshold that will allow us to complete these tasks. We assume that the initial temperature of the device is the ambient temperature, although this assumption is not crucial.

A schedule is c-competitive for a particular objective function if the value of that objective function on the schedule is at most c times the value of the objective function on the optimal schedule. A scheduling algorithm is c-competitive if its output is c-competitive for all instances.

1.2 Results

Theoretical investigations into speed scaling techniques to manage energy were initiated in [18]. They showed that there is a simple greedy offline algorithm, which we call YDS, that generates the feasible schedule (for all α) that uses minimum energy. We describe the algorithm YDS in section 2. [18] also proposed two online algorithms Average Rate (AVR), and Optimal Available (OA). The algorithm Average Rate (AVR) runs each task i at a rate of $w_i/(d_i - r_i)$. The algorithm Optimal Available (OA) at any point of time schedules the unfinished work optimally (say using YDS) under the assumption that no more tasks will arrive. AVR and OA were proved $O(1)$-competitive with respect to energy in [18] and [2], respectively. In [2] another online algorithm, which we call BKP, was proposed, and shown to be $O(1)$-competitive with respect to energy. The motivation for introducing BKP was that it has a lower competitive ratio, with respect to energy, than do AVR or OA when α is large. We postpone a description of the BKP algorithm until section 4.

Theoretical investigations into speed scaling techniques to manage temperature were initiated in [2]. In [2] it was shown that in principle the offline problem can be solved in polynomial time (modulo numerical stability issues) using the Ellipsoid algorithm. The Ellipsoid algorithm is useful for establishing theoretical results, but it is practically inefficient for moderate sized problems. Due to the complicated nature of Fourier's law, it seems unlikely that one can easily compute the optimal temperature schedule.

In section 2 we show that while the YDS schedule may not be optimal for temperature, it is an $O(1)$-approximate schedule with respect to temperature. Thus, this constructively shows that there are schedules that are $O(1)$-approximate with respect to both of the dual criteria: temperature and energy. In some sense, this is the best result possible, as even for some 1 job instances it is not to difficult to see that there does not exist a schedule that is $(1+\epsilon)$-approximate for both energy and temperature (see Lemma 5 for details).

In section 3 we show that that online algorithms OA and AVR, proposed in [18] in the context of energy management, are not $O(1)$-competitive with respect to temperature. Recall that there algorithms are $O(1)$-competitive with respect to energy. This demonstratively illustrates the observation from practice that power management techniques that are effective for managing energy may not be effective for temperature. We also show that an intuitive temperature management algorithm, running at such a speed so that the temperature is constant, is surprisingly not $O(1)$-competitive with respect to temperature.

In section 4 we give our main result. We show that the online speed scaling algorithm, that we call BKP, proposed in [2] in the context of energy management, is $O(1)$-competitive with respect to temperature. This is the first $O(1)$-competitiveness analysis with respect to temperature for an online algorithm. Another way to interpret this result is that if BKP exceeds the thermal threshold T_{max} of the device then every other feasible schedule causes the device to reach a temperature of $\Omega(T_{max})$. Our temperature analysis of the online algorithm BKP compares BKP to YDS, and thus our temperature analysis of YDS is also necessary component to our temperature analysis of BKP.

At first inspection it seems difficult to reason about temperature because of the exponential decay nature of Fourier's law. All of our temperature analyses use the observation that the maximum temperature can be approximated within a constant by a times the maximum energy used over an interval of length $\Theta(1/b)$. We believe that this observation nicely explains the different natures of energy and temperature management, greatly simplifies the task of reasoning about temperature, and will surely be useful in future investigations in algorithmic issues in temperature management.

The paper [18] did not contain a proof that the YDS algorithm produces the most energy efficient feasible schedule. And to the best of our knowledge, no such proof has appeared in the literature. We show in section 5 that the correctness of YDS is an elegant consequence of the well-known KKT optimality conditions. This illustrates the utility of the KKT optimality conditions in power management problems. As another example, KKT optimality conditions can be used to simply some proofs in [13].

1.3 Further Related Results

The YDS schedule also minimizes the maximum speed, over all times in the schedule, that the processor runs at that time [18]. The BKP online algorithm is optimally (among deterministic online algorithms) e-competitive for this objective function [2].

[10] studies online speed scaling algorithms to minimize energy usage in a setting where the device also has a sleep state, and gives an offline polynomial-time 2-approximation algorithm. [10] also gives an $O(1)$-competitive online algorithm, which uses as a subroutine an algorithm for pure dynamic speed scaling. These results have been extended to the case of multiple slow-down states in [1].

[13] give an efficient algorithm for the problem of minimizing the average flow time of a collection of dynamically released equi-work processes subject to the constraint that a fixed amount of energy is available.

2 YDS Temperature

Our goal is to show that the energy optimal YDS schedule produced by the YDS algorithm is 20-competitive with respect to temperature. We first start with a description of the YDS algorithm. We then show in Lemma 1 that the optimal maximum temperature of a schedule is within a factor of 4 of the a times the maximum energy expended of an interval of length $c = \frac{\ln 2}{b}$. We call such an interval a c-interval. Thus it is sufficient to show that YDS is 5-competitive with respect to the maximum energy expended over any c-interval.

YDS Algorithm: Let $w(t_1, t_2)$ denote the work that has release time $\geq t_1$ and has deadline $\leq t_2$. The *intensity* of the interval $[t_1, t_2]$ is then $w(t_1, t_2)/(t_2 - t_1)$. The YDS algorithm repeats the following steps: Let $[t_1, t_2]$ be the maximum intensity interval. The processor will run at speed $w(t_1, t_2)/(t_2 - t_1)$ during $[t_1, t_2]$. Then the instance is modified as if the times $[t_1, t_2]$ didn't exist. That is, all deadlines $d_i > t_1$ are reduced to $\max(t_1, d_i - (t_2 - t_1))$, and all release times $r_i > t_1$ are reduced to $\max(t_1, r_i - (t_2 - t_1))$, and the process is repeated. Given the speed as a function of time as determined by this procedure, YDS always runs the released, unfinished task with the earliest deadline.

Note that the YDS schedule has the property that each task is run at a fixed rate when the task is run. This rate is fixed with respect to time, but may be different for different tasks.

Lemma 1. *For any schedule, if E denotes a times the maximum energy in a c-interval, then $E/2 \leq T_{max} \leq 2E$.*

Proof. Let $P(t)$ be the power at time t. We rewrite Fourier's Law as

$$d(e^{bt}T) = ae^{bt}P(t)dt \tag{1}$$

Let $k = cb = \ln 2$. Define T_{max} to be the maximum temperature reached for the schedule, and E be defined to be a times the maximum energy used any c-interval. We first show that T_{max} is at most twice E. Suppose that temperature T_{max} is achieved at time t_0. Then integrating equation 1 from $t_0 - k/b = t_0 - c$ to t_0, we get $T(t_0)e^{bt_0} - T(t_0 - k/b)e^{bt_0 - k} = a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$. As e^{bt} is increasing in t, the term $\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$ is at most $e^{bt_0}\int_{t_0-k/b}^{t_0} P(t)dt$. Thus, $T(t_0) \leq$

$T(t_0 - k/b)e^{-k} + a\int_{t_0-k/b}^{t_0} P(t)dt$. As $T(t_0 - k/b) \leq T(t_0) = T_{max}$, it follows that $T_{max}(1 - e^{-k}) \leq a\int_{t_0-k/b}^{t_0} P(t)dt \leq E$, and hence $T_{max} \leq \frac{E}{1-e^{-k}} = 2E$, as $k = \ln 2$.

We now want to show T_{max} is at least half of E. Let $[t_0 - k/b, t_0]$ be a c-interval where the maximum amount of energy is used. Integrating equation 1 gives $T(t_0)e^{bt_0} - T(t_0 - k/b)e^{bt_0-k} = a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt$. Using the fact that $T(t_0 - k/b) \geq 0$, and that e^{bt} is an increasing function of t, and the definition of E, it follows that $T(t_0)e^{bt_0} \geq a\int_{t_0-k/b}^{t_0} e^{bt}P(t)dt \geq ae^{bt_0-k}\int_{t_0-k/b}^{t_0} P(t)dt = e^{bt_0-k}E$. Thus, $T_{max} \geq T(t_0) \geq e^{-k}E = E/2$.

Note that YDS is not optimal for minimizing the maximum temperature, or for minimizing the total energy in a c-interval. The fact that YDS is not optimal for temperature can be seen on 1 job instances where the in the optimal temperature schedule must run at a speed that follows some non-constant Euler curve [2] (see Lemma 5 for more details). The fact that YDS is not optimal for minimizing energy used in a c-interval can be seen from the following instance. Consider for example, an instance with 2 tasks with work 1 each, both of them arrive at time 0 and have deadlines $c/2$ and $3c/2$ respectively.

For the rest of this section we only consider the objective of minimizing the maximum energy in any c-interval. This will culminate in Lemma 3, which states that the YDS schedule is 5 approximate with respect to this objective.

We first require some preliminary definitions and observations. Let I denote a problem instance. Let $YDS(I)$ be the YDS schedule on input I. Let $Y(I)$ denote the maximum energy used in any c-interval in $YDS(I)$. Let $Opt(I)$ denote the optimum value of the maximum energy used in any c-interval, over all feasible schedules. Let $X(I)$ denote a c-interval in $YDS(I)$ that uses energy $Y(I)$. Let $r_0 = (\epsilon Y(I)/c)^{1/\alpha}$. The value of ϵ will be fixed later. Call a task in I, *slow* , if it runs at rate strictly less than r_0 in $YDS(I)$. This notion is well defined because a task runs at constant rate in the YDS schedule. The rest of the tasks are called *fast*. Define I' to consist of exactly the fast tasks in I. Let $r'(t)$ denote the rate at time t in $YDS(I')$, and $r(t)$ denote the rate at time t in $YDS(I)$.

Define an *island* to be a maximal interval of time where $r'(t) > 0$. The following two claims easily follow from the nature of the YDS algorithm.

Claim. For all times t, $r'(t) = r(t)$ if $r \geq r_0$ and $r'(t) = 0$ otherwise.

Claim. Each task in the instance I' is totally contained inside an island. That is, for each task $i \in I'$, the interval $[r_i, d_i]$ does not overlap with any time where $r'(t) = 0$.

Thus, we can view the instance I' as a collection of disjoint intervals (islands), and each task is contained in an island. For an instance $\tilde{I}$ we define its support to be $(\max_{i\in\tilde{I}} d_i - \min_{i\in\tilde{I}} r_i)$, that is, all task in an instance are contained in an interval of length equal to the support of the instance. By the energy optimality of YDS, we trivially have that,

Claim. For an instance $\tilde{I}$ that has support no more than c, $Y(\tilde{I}) = Opt(\tilde{I})$.

As most of the energy in $X(I)$ is contained is fast tasks, we have that

Lemma 2. *Let I and I' be as defined above. Then $(1-\epsilon)Y(I) \leq Y(I') \leq Y(I)$.*

We omit a formal proof of Lemma 2 due to space constraints.

We now sketch the proof that YDS is 5-competitive with respect to minimizing the maximum energy used over any c-interval.

Lemma 3. *For any instance I, $Opt(I) \geq \min(\epsilon Y(I)/2, (1-\epsilon)Y(I)/3)$. Choosing $\epsilon = 2/5$, it follows that $Opt(I) \geq Y(I)/5$.*

Proof. As it is clear that $Opt(I') \leq Opt(I)$, it suffices to show that $Opt(I') \geq \min(\epsilon Y(I)/2, (1-\epsilon)Y(I)/3)$.

Consider an island G of I' and let t be the length of G. At the YDS schedule for I' has rate at least r_0 during G, the total energy consumed by YDS and hence by any schedule for G is at least tr_0^α. If $t \geq c$, then, by an averaging argument, for any schedule for G, there is some c-interval that has energy at least $r_0^\alpha t/\lceil t/c \rceil$ which is at least $(1/2)cr_0^\alpha = \epsilon Y(I)/2$. Thus, $Opt(I') \geq \epsilon Y(I)/2$ if $t \geq c$.

If all the islands have length no more than c, then consider the islands that intersect $X(I)$. If some such island G has energy at least $(1-\epsilon)Y(I)/3$, the result follows by Claim 2. In the case that all islands that intersect $X(I)$ have energy less than $(1-\epsilon)Y/3$. By Lemma 2 we know that in $YDS(I')$ the total energy during $X(I)$ is at least $(1-\epsilon)Y(I)$. As at most two islands can lie partially in $X(I)$, at least $(1-\epsilon)Y(I)/3$ energy is in islands that are totally contained inside $X(I)$, and hence the result follows by Claim 2.

The following theorem is then a direct consequence of Lemmas 1 and 3.

Theorem 1. *The energy optimal algorithm YDS is 20-competitive with respect to maximum temperature.*

3 Online Algorithms that are not $O(1)$-competitive

We show that AVR, OA and the constant temperature algorithm are not $O(1)$-competitive.

Lemma 4. *The online algorithms AVR, OA are not $O(1)$-competitive with respect to temperature. More precisely, the competitive ratio of these algorithms must depend on either the number of jobs, or the cooling rate b.*

Proof. We use a variation of an instance from [18]. Let $r_i = 1 - 1/i$, $w_i = 1/i$ and $d_i = 1$ for $1 \leq i \leq n$. For instances with a common deadline, as is the case here, AVR and OA behave identically. Let $c = 1/n$. The YDS schedule runs jobs at a constant rate of 1. Using Lemma 1 it is sufficient to show that there is some c-interval where the energy used by OA and AVR is $\omega(c)$. In particular, during the c-interval $[1-1/n, 1]$ it is the case that AVR and OA are running at a rate of $\Omega(\log n)$, and hence the energy used during this c-interval is $\Omega(c \log^\alpha n)$.

In the reasonable case that the thermal threshold T_{max} of the device is known, the most obvious temperature management strategy is to run at a rate that leaves the temperature fixed at T_{max}. We call such a strategy $O(1)$-competitive if on any instance I on which this constant temperature algorithm missed a deadline, every feasible schedule causes a temperature of $\Omega(T_{max})$.

Lemma 5. *The speed scaling algorithm that runs at such a speed that the temperature remains constant at the thermal threshold T_{max} is not $O(1)$-competitive.*

Proof. Suppose at time 0 a task with work x (which will be specified later) and deadline ϵ arrives. We will think of ϵ as going to 0. Suppose that the temperature at time 0 is 0. We choose x such that it is equal to the maximum work that the adversary can get done by time ϵ while keeping the temperature below T_{max}/k. Using equations from [2] for the case that $\alpha = 3$, which probably are too involved to repeat here, $x = \Theta((\frac{bT_{max}}{ka})^{1/3}\epsilon^{2/3})$. The crucial fact in the term above is that the maximum work that the adversary can do depends on ϵ as $\epsilon^{2/3}$. On the other hand, the constant temperature algorithm at temperature T_{max} has power $P = bT_{max}/a$ and hence speed $(bT_{max}/a)^{1/3}$ and work $\Theta((bT_{max}/a)^{1/3}\epsilon)$, which depends linearly on ϵ. Thus, for any constant k, the ratio of the work completed the adversary over the work completed by online goes to infinity as ϵ goes to 0.

4 The Online BKP Algorithm

In this section we first introduce some notation, then state the BKP algorithm, and then show in Theorem 2 that BKP is $O(1)$-competitive with respect to temperature. Theorem 2 compares the maximum energy used in a c-interval by BKP to the maximum energy used in a c-interval by YDS. Thus our analysis of YDS is a component of our analysis of BKP. It was shown in [2] that BKP always produces a feasible schedule. We assume without loss of generality that all release times and deadlines are integers.

Let $w(t, t_1, t_2)$ denote amount of work that has arrived by time t, that has release time $\geq t_1$ and deadline $\leq t_2$. Let $k(t)$ be the maximum over all $t' > t$ of $(w(t, et-(e-1)t', t'))/(e(t'-t))$. Note that $w(t, t_1, t_2)$ and $k(t)$ may be computed by an online algorithm at time t.

BKP Algorithm Description: At time t, work at rate $ek(t)$ on the unfinished job with the earliest deadline.

Theorem 2. *The online algorithm BKP is $20e^{\alpha}2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^{\alpha}+1)$-competitive with respect to temperature.*

Proof. By Theorem 1, it is sufficient to show that BKP uses at most factor of $e^{\alpha}2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^{\alpha}+1)$ times as much energy over any c-interval as does YDS. Again let Y be the maximum energy used by YDS in any c-interval.

Let $w(t_1, t_2)$ denote the work that has release time $\geq t_1$ and has deadline $\leq t_2$. Let $y(t)$ denote the rate of work of the algorithm YDS at time t. Let $q(t)$ be the maximum over all t_1 and t_2, such that $t_1 < t < t_2$, of $w(t_1, t_2)/(t_2 - t_1)$.

For purposes of analysis, we will assume BKP always runs at a rate $q(t) \geq k(t)$, even if there is no work to do. It is obvious that $q(t) \geq k(t)$ for all t. We can then assume that in the input instance it is the case that $y(t)$ amount of work arrives with release time t and deadline $t+1$. Under this transformation, the YDS schedule remains the same, and $q(t)$ will not decrease. The fact that $q(t)$ will not decrease follows from the definition of $q(t)$.

Let X be an arbitrary c-interval. Let X_-^k (respectively X_+^k) denote the k^{th}, c-interval immediately to the left (respectively right) of X. That is, the left endpoint of X_-^k is kc units to the left of X. Let the interval Z be defined to be $X \cup X_-^1 \cup X_+^1$. We now show that the energy used by BKP during X is at most $e^\alpha 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha + 1)Y$. As X is an arbitrary c-interval, this will imply the result.

We decompose the original instance as follows: Let $w_1(t) = y(t)$ if $t \in Z$ and 0 at all other times. Let $w_2(t) = y(t) - w_1(t)$ for all t. Let

$$q_1(t) = \max_{t' < t \leq t''} \frac{\sum_{x=t'}^{t''} w_1(t)}{t'' - t'} \qquad \text{and} \qquad q_2(t) = \max_{t' < t \leq t''} \frac{\sum_{x=t'}^{t''} w_2(t)}{t'' - t'}$$

Note that $q(t) \leq q_1(t) + q_2(t)$ since $y(t) = w_1(t) + w_2(t)$ for all each t. By convexity of the speed to power function $P(s)$, it follows that $q(t)^\alpha \leq (q_1(t) + q_2(t))^\alpha \leq 2^{\alpha-1}(q_1(t)^\alpha + q_2(t)^\alpha)$ and thus $\sum_{t \in X} q(t)^\alpha \leq 2^{\alpha-1}(\sum_{t \in X} q_1(t)^\alpha + \sum_{t \in X} q_2(t)^\alpha)$.

We now wish to upper bound the sum $\sum_{t \in X} q_1(t)^\alpha$. We follow the method used in [2]. Since $X \subseteq Z$, it is the case that $\sum_{t \in X} q_1(t)^\alpha \leq \sum_{t \in Z} q_1(t)^\alpha$. Let $l(t)$ be the maximum over all t_1, such that $t_1 < t$, of $\sum_{x=t_1}^{t} w_1(x)/(t - t_1)$. Similarly, let $r(t)$ be the maximum over all t_2, such that $t \leq t_2$, of $\sum_{x=t}^{t_2} w_1(x)/(t_2 - t)$. Clearly, $q(t) \leq \max(l(t), r(t))$, and hence $q(t)^\alpha \leq l(t)^\alpha + r(t)^\alpha$. We will show that both $\sum_{t=-\infty}^{\infty} l(t)^\alpha$, and $\sum_{t=-\infty}^{\infty} r(t)^\alpha$, are at most $(\frac{\alpha}{\alpha-1})^\alpha \sum_{t>0} y(t)^\alpha = (\frac{\alpha}{\alpha-1})^\alpha \sum_{t \in Z} w_1(t)^\alpha$. This implies $\sum_{t \in Z} q_1(t)^\alpha \leq 2\left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t \in Z} w_1(t)^\alpha$.

The following result was first proved by Hardy and Littlewood in [8] and later simplified by Gabriel [6]. It can also be found in [9, Theorem 393 and 394].

Fact: If $y(1), \ldots, y(n)$ are arbitrary non-negative integers, let $l(t)$ be the maximum over all v, such that $1 \leq v \leq t$, of $\sum_{k=v}^{t} y(k)/(t - v + 1)$. Let $s(y)$ be a positive increasing function of y. Let $\bar{y}(1), \ldots \bar{y}(n)$ denote the sequence of $y(i)$ in decreasing sorted order. Then $\sum_{k=1}^{n} s(l(k)) \leq \sum_{k=1}^{n} s(\sum_{j=1}^{k} \bar{y}(j)/k)$.

We now apply this fact. We rescale time so that these times in Z are $1, \ldots, 3c$. We set $y(i)$ equal to $w_1(i)$ in the fact above. Further, we set $s(y) = y^\alpha$. Note that since $w_1(t) = 0$ for $t \notin Z$, no other work affects $l(t)$ other than the $y(t)$'s for $t \in Z$. Thus the definition of $l(t)$ in the fact is identical to the previous definition of $l(t)$. We can then conclude that $\sum_{t \in Z} l(t)^\alpha$ is maximized if for all i, $y(i) \geq y(i+1)$. In this case, $l(t) = \sum_{k=1}^{t} y(k)/t$. Thus, $\sum_{t \in Z} l(t)^\alpha \leq \sum_{t \in Z} \left(\sum_{i=1}^{t} y(i)/t\right)^\alpha$.

Hardy showed [7] as a special case of Hilbert's theorem (see [9, Theorem 326]), that $\sum_t \left(\sum_{i=1}^{t} y(i)/t\right)^\alpha \leq (\frac{\alpha}{\alpha-1})^\alpha \sum_t y(t)^\alpha$. Note that in these inequalities the $y(i)$'s may be arbitrary, and all that is required of α is that $\alpha > 1$.

Thus it follows that $\sum_{t \in Z} l(t)^\alpha \leq \left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t \in Z} w_1(t)^\alpha$. A similar analysis of $r(t)$ shows that $\sum_{t \in Z} r(t)^\alpha \leq \left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t \in Z} w_1(t)^\alpha$. Thus, $\sum_{t \in Z} q_1(t)^\alpha \leq 2\left(\frac{\alpha}{\alpha-1}\right)^\alpha \sum_{t \in Z} w_1(t)^\alpha$. Finally by the definition of $w_1(t)$ and $y(t)$, $\sum_{t \in Z} w_1(t)^\alpha = \sum_{t \in Z} y(t)^\alpha \leq 3Y$. Thus $\sum_{t \in X} q_1(t)^\alpha \leq 6\left(\frac{\alpha}{\alpha-1}\right)^\alpha Y$.

We now bound the term $\sum_{t \in X} q_2(t)^\alpha$. Note that $w_2(t) = 0$ at all times $t \in Z$. By the definition of Y, any c-interval contains at most $c(Y/c)^{1/\alpha}$ amount of work in the YDS schedule. Thus, any c-interval X_+^k or X_-^k for $k \geq 2$ contains at most $c(Y/c)^{1/\alpha}$ work in $w_2(t)$, it follows that $q_2(t) \leq (Y/c)^{1/\alpha}$. Thus

$$\sum_{t \in X} q_2(t)^\alpha \leq \sum_{t \in X} \left((\frac{Y}{c})^{1/\alpha}\right)^\alpha = \sum_{t \in X} \frac{Y}{c} = Y$$

Combining the above, we have that for any c-interval X

$$\sum_{t \in X} k(t)^\alpha \leq \sum_{t \in X} 2^{\alpha-1}(q_1(t)^\alpha + q_2(t)^\alpha) \leq 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha + 1)Y$$

Since the BKP algorithm works at rate at most $ek(t)$, the energy used during the c-interval X is at most $e^\alpha 2^{\alpha-1}(6(\frac{\alpha}{\alpha-1})^\alpha + 1)Y$.

5 Using KKT to Prove YDS Correct

We show that the energy optimality of the YDS schedule follows as a direct consequence of the well known KKT optimality conditions for convex programs. We start by stating the KKT conditions. We then show how to express the energy problem as a convex program. And then show the result of applying the KKT conditions to this convex program.

Consider a convex program

$$\begin{aligned} &\min f_0(x) \\ &\quad f_i(x) \leq 0 \qquad i = 1, \ldots, n \end{aligned}$$

Assume that this program is strictly feasible, that is, there is some point x where where $f_i(x) < 0$ for $i = 1, \ldots, n$. Assume that the f_i are all differentiable. Let λ_i, $i = 1, \ldots, n$ be a variable (Lagrangian multiplier) associated with the function $f_i(x)$. Then a necessary and sufficient KKT conditions for solutions x and λ to be primal and dual feasible are [3]:

$$f_i(x) \leq 0 \qquad i = 1, \ldots, n \tag{2}$$

$$\lambda_i \geq 0 \qquad i = 1, \ldots, n \tag{3}$$

$$\lambda_i f_i(x) = 0 \tag{4}$$

$$\nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) = 0 \tag{5}$$

To state the energy minimization problem as a convex program, we break time into intervals $t_0, \ldots t_m$ at release times and deadlines of the tasks. Let $J(i)$ be the tasks that can feasibly be executed during the time interval $I_i = [t_i, t_{i+1}]$, and $J^{-1}(j)$ be intervals during which task j can be feasibly executed. We introduce a variable $w_{i,j}$, for $j \in J(i)$, that represents the work done on task j during time $[t_i, t_{i+1}]$. Our (interval indexed) mathematical program P is then:

$$\min \quad E \tag{6}$$

$$w_j \leq \sum_{i \in J^{-1}(j)} w_{i,j} \qquad j = 1, \ldots, n \tag{7}$$

$$\sum_{i=1}^{m} \left(\frac{\sum_{j \in J(i)} w_{i,j}}{t_{i+1} - t_i} \right)^{\alpha} (t_{i+1} - t_i) \leq E \tag{8}$$

$$w_{i,j} \geq 0 \qquad i = 1, \ldots, m \qquad j \in J(i) \tag{9}$$

We now apply the KKT conditions to this convex program. We associate a dual variable δ_j with equation j in line 7, a dual variable β with the equation in line 8, and a dual variable $\gamma_{i,j}$ with equation i, j in line 9. We now evaluate line 5 of the KKT conditions for our convex program. Looking at the component of equation 5 corresponding to the variable E, we get that $\beta = 1$. Looking at the component of equation 5 corresponding to the variable $w_{i,j}$, we get that

$$-\delta_j + \beta p \left(\frac{\sum_{k \in J(i)}^{n} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} - \gamma_{i,j} = 0 \tag{10}$$

Consider a $w_{i,j}$ such that $w_{i,j} > 0$. We know that by complementary slackness (equation 4) that it must be the case that $\gamma_{i,j} = 0$. Hence,

$$\delta_j = \alpha \left(\frac{\sum_{k \in J(i)} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} \tag{11}$$

Hence, the interpretation of the dual variable δ_j is α times the speed at which the processor runs during interval i raised to the power of $(\alpha - 1)$. This quantity, and hence the speed of the processor, must be the same for each interval i during which task j is run. Now consider a $w_{i,j}$ such that $w_{i,j} = 0$. Rearranging equation 10 we find that

$$\gamma_{i,j} = \alpha \left(\frac{\sum_{k \in J(i)} w_{i,k}}{t_{i+1} - t_i} \right)^{\alpha - 1} - \delta_j \tag{12}$$

Then $\gamma_{i,j}$ will be non-negative if the processor is running faster during interval i than during the intervals where task j is run.

Thus we can conclude that a sufficient condition for a primal feasible solution to be optimal is that:

- For each task j, the processor runs at the same speed, call is s_j in the intervals i in which task j is run.

– And the processor runs at speed no less than s_j during intervals i, such that $j \in J(i)$, in which task j is not run.

The solution produced the the YDS algorithm clearly has these properties and hence is optimal.

References

1. J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *IEEE Syposium on Foundations of Computer Science*, 2004.
2. N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, 2004.
3. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
4. D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
5. G. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer, 1997.
6. R. M. Gabriel. An additional proof of a maximal theorem of hardy and littlewood. *Journal of London Mathematical Society*, 6:163–166, 1931.
7. G. H. Hardy. Note on a theorem of hilbert. *Math. Zeitschr.*, 6:314–317, 1920.
8. G. H. Hardy and J. Littlewood. A maximal theorem with function-theoretic applications. *Acta Mathematica*, 54:81–116, 1930.
9. G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.
10. S. Irani, , R. K. Gupta, and S. Shukla. Algorithms for Power Savings. In *ACM/SIAM Symposium on Discrete Algorithms*, 2003.
11. S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power saving states. *Trans. on Embedded Computing Sys.*, 2003. Special Issue on Power Aware Embedded Computing.
12. T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.
13. K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.
14. J. E. Sergent and A. Krum. *Thermal Management Handbook*. McGraw-Hill, 1998.
15. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, pages 2–13, 2003.
16. V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Design Automation Conference*, pages 732–737, 1998.
17. http://www.usatoday.com/tech/news/2004-05-07-intel-kills-tejas_x.htm.
18. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science*, page 374, 1995.