



Speed Scaling to Manage Temperature.

Leon Atkins, Guillaume Aupy, Daniel Cole, Kirk Pruhs

► **To cite this version:**

Leon Atkins, Guillaume Aupy, Daniel Cole, Kirk Pruhs. Speed Scaling to Manage Temperature.. TAPAS - 1st International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems, 2011, Rome, Italy. pp.9-20, 2011, <10.1007/978-3-642-19754-3_4>. <hal-00786200>

HAL Id: hal-00786200

<https://hal.inria.fr/hal-00786200>

Submitted on 3 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Speed Scaling to Manage Temperature

Leon Atkins¹, Guillaume Aupy^{2,*}, Daniel Cole³, and Kirk Pruhs^{4,**}

¹ Department of Computer Science, University of Bristol,
atkins@compsci.bristol.ac.uk

² LIP, ENS Lyon, guillaume.aupy@ens-lyon.fr

³ Computer Science Department, University of Pittsburgh, dcc20@cs.pitt.edu

⁴ Computer Science Department, University of Pittsburgh, kirk@cs.pitt.edu

Abstract. We consider the speed scaling problem where the quality of service objective is deadline feasibility and the power objective is temperature. In the case of batched jobs, we give a simple algorithm to compute the optimal schedule. For general instances, we give a new online algorithm, and obtain an upper bound on the competitive ratio of this algorithm that is an order of magnitude better than the best previously known bound upper bound on the competitive ratio for this problem.

1 Introduction

Speed scaling technology allows the clock speed and/or voltage on a chip to be lowered so that the device runs slower and uses less power [1]. Current desktop, server, laptop, and mobile class processors from the major manufacturers such as AMD and Intel incorporate speed scaling technology. Further these manufacturers produce associated software, such as AMD's PowerNow and Intel's SpeedStep, to manage this technology. With this technology, the operating system needs both a scheduling policy to determine which job to run at each point in time, as well as a speed scaling policy to determine the speed of the processor at that time. The resulting optimization problems have dual objectives, a quality of service objective (e.g. how long jobs have to wait to be completed), as well as a power related objective (e.g. minimizing energy or minimizing maximum temperature). These objectives tend to be in opposition as the more power that is used, generally the better the quality of service that can be provided.

The theoretical study of such dual objective scheduling and speed scaling optimization problems was initiated in [2]. [2] studied the problem where the quality of service objective was a deadline feasibility constraint, that is, each job has to be finished by a specified deadline, and the power objective was to minimize to total energy used. Since [2] there have been a few tens of speed scaling papers in the theoretical computer science literature [3] (and probably hundreds of papers in the general computer science literature). Almost all of

* Guillaume Aupy was supported in part by the ANR StochaGrid.

** Kirk Pruhs was supported in part by NSF grant CCF-0830558, and an IBM Faculty Award.

the theoretical speed scaling papers have focused on energy management. We believe that the main reason for the focus on energy, instead of temperature, is mathematical; it seems to be much easier to reason about the mathematical properties of energy than it is to reason about the mathematical properties of temperature. From a technological perspective, temperature management is at least on par with energy management in terms of practical importance.

Energy and temperature are intuitively positively correlated. That is, running at a high power generally leads to both high temperatures and high energy use. It is therefore tempting to presume that a good energy management policy will also be a good temperature management policy. Unfortunately, the first theoretical paper on speed scaling for temperature management [4] showed that some algorithms that were proved to be good for energy management in [2], can be quite bad for temperature management. The reason for this is the somewhat subtle difference between energy and temperature.

To understand this, we need to quickly review the relationship between speed, power, and energy. The well-known cube-root for CMOS-based processor states that the dynamic power used by a processor is roughly proportional to the speed of the processor cubed [5]. Energy is power integrated over time. Cooling is a complex phenomenon that is difficult to model accurately. [4] suggested assuming that all heat is lost via conduction, and that the ambient temperature is constant. This is a not completely unrealistic assumption, as the purpose of fans within computers is to remove heat via conduction, and the purpose of air conditioning is to maintain a constant ambient temperature. Newton's law of cooling states that the rate of cooling is proportional to the difference in temperature between the device and the ambient environment. This gives rise to the following differential equation describing the temperature T of a device as a function of time t :

$$\frac{dT(t)}{dt} = aP(t) - bT(t) \quad (1)$$

That is the rate of increase in temperature is proportional to the power $P(t)$ used by the device at time t , and the rate of decrease in temperature due to cooling is proportional to the temperature (assuming that the temperature scale is translated so the ambient temperature is zero). It can be assumed without loss of generality that $a = 1$. The device specific constant b , called the *cooling parameter*, describes how easily the device loses heat through conduction [4]. For example, all else being equal, the cooling parameter would be higher for devices with high surface area than for devices with low surface area. [4] showed that the maximum temperature that a device reaches is approximately the maximum energy used over any time period of length $1/b$. So a schedule that for some period of time of length $1/b$ used an excessive amount of power could still be a near optimal schedule in terms of energy (if the aggregate energy used during this time interval is small relative to the total energy used) but might reach a much higher temperature than is necessary to achieve a certain quality of service.

In this paper we consider some algorithmic speed scaling problems where the power objective is temperature management. Our high level goal is to develop

techniques and insights that allow mathematical researchers to more cleanly and effectively reason about temperature in the context of optimization.

We adopt much of the framework considered in [2] and [4], which we now review, along with the most closely related results in the literature.

Preliminaries We assume that a processor running at a speed s consumes power $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. We assume that the processor can run at any nonnegative real speed (using techniques in the literature, similar results could be obtained if one assumed a bounded speed processor or a finite number of speeds). The job environment consists of a collection of tasks, where each task i has an associated release time r_i , amount of work p_i , and a deadline d_i . An online scheduler does not learn about task i until time r_i , at which point it also learns the associated p_i and d_i . A schedule specifies for each time, a job to run, and a speed for the processor. The processor will complete s units of work in each time step when running at speed s . Preemption is allowed, which means that the processor is able to switch which job it is working on at any point without penalty. The deadline feasibility constraints are that all of the work on a job must be completed after its release time and before its deadline. [2] and subsequent follow-up papers consider the online and offline problems of minimizing energy usage subject to these deadline feasibility constraints. Like [4], we will consider the online and offline problems of minimizing the maximum temperature, subject to deadline feasibility constraints.

Related Results [2] showed that there is a greedy offline algorithm YDS to compute the energy optimal schedule. A naive YDS implementation runs in time $O(n^3)$, which is improved in [6] to $O(n^2 \log n)$. [2] suggested two online algorithms OA and AVR. OA runs at the optimal speed assuming no more jobs arrive in the future (or alternately plans to run in the future according to the YDS schedule). AVR runs each job at an even rate between its release time and deadline. In a complicated analysis, [2] showed that AVR is at most $2^{\alpha-1} \alpha^\alpha$ -competitive with respect to energy. A simpler competitive analysis of AVR, with the same bound, as well as a nearly matching lower bound on the competitive ratio for AVR can be found in [7]. [4] shows that OA is α^α -competitive with respect to energy. [4] showed how potential functions can be used to give relatively simple analyses of the energy used by an online algorithm. [8] introduces an online algorithm qOA, which runs at a constant factor q faster than OA, and shows that qOA is at most $4^\alpha / (2\sqrt{e\alpha})$ -competitive with respect to energy. When the cube root rule holds, qOA has the best known competitive ratio with respect to energy, namely 6.7. [8] also gives the best known general lower bound on the competitive ratio, for energy, of deterministic algorithms, namely $e^{\alpha-1}/\alpha$.

Turning to temperature, [4] showed that a temperature optimal schedule could be computed in polynomial time using the Ellipsoid algorithm. Note that this is much more complicated than the simple greedy algorithm, YDS, for computing an energy optimal schedule. [4] introduces an online algorithm, BKP, that is simultaneously $O(1)$ -competitive for both total energy and maximum temperature. An algorithm that is c -competitive with respect to temperature has the

property that if the thermal threshold T_{\max} of the device is exceeded, then it is not possible to feasibly schedule the jobs on a device with thermal threshold T_{\max}/c . [4] also showed that the online algorithms OA and AVR, both $O(1)$ -competitive with respect to energy, are not $O(1)$ -competitive for the objective of minimizing the maximum temperature. In contrast, [4] showed that the energy optimal YDS schedule is $O(1)$ -competitive for maximum temperature.

Besides [4], the only other theoretical speed scaling for temperature management papers that we are aware of are [9] and [12]. In [9] it is assumed that the speed scaling policy is fixed to be: if a particular thermal threshold is exceeded then the speed of the processor is scaled down by a constant factor. Presumably chips would have such a policy implemented in hardware for reasons of self-preservation. The paper then considers the problem of how to schedule unit work tasks, that generate varying amounts of heat, so as to maximize throughput. [9] shows that the offline problem is NP-hard even if all jobs are released at time 0, and gives a 2-competitive online algorithm. [12] provides an optimal algorithm for a batched release problem similar to ours but with a different objective, minimizing the makespan, and a fundamentally different thermal model.

Surveys on speed scaling can be found in [3], [10], and [11].

Our Results A common online scheduling heuristic is to partition jobs into batches as they arrive. Jobs that arrive, while jobs in the previous batch are being run, are collected in a new batch. When all jobs in the previous batch are completed, a schedule for the new batched is computed and executed. We consider the problem of how to schedule the jobs in a batch. So this batched problem is a special case of the general problem where all release times are zero.

In section 2.1, we consider the feasibility version of this batched problem. That is, the input contains a thermal threshold T_{\max} and the problem is to determine whether the jobs can be scheduled without violating deadlines or the thermal threshold. We give a relatively simple $O(n^2)$ time algorithm. This shows that temperature optimal schedules are easier to compute in the case of batched jobs. Our algorithm maintains the invariant that after the i^{th} iteration, it has computed a schedule S_i that completes the most work possible subject to the constraints that the first i deadlines are met and the temperature never exceeds T_{\max} . The main insight is that when extending S_i to S_{i+1} , one need only consider n possibilities, where each possibility corresponds to increasing the speed from immediately after one deadline before d_i until d_i in a particular way.

In section 2.2, we consider the optimization version of the batched problem. That is, the goal is to find a deadline feasible schedule that minimizes the maximum temperature T_{\max} attained. One obvious way to obtain an algorithm for this optimization problem would be to use the feasibility algorithm as a black box, and binary search over the possible maximum temperatures. This would result in an algorithm with running time $O(n^2 \log T_{\max})$. Instead we give an $O(n^2)$ time algorithm that in some sense mimics one run of the feasibility algorithm, raising T_{\max} throughout so that it is always the minimum temperature necessary to maintain feasibility.

We then move on to dealing with the general online setting. We assume that the online speed scaling algorithm knows the thermal threshold T_{\max} of the device. It is perfectly reasonable that an operating system would have knowledge of the thermal threshold of the device on which it is scheduling tasks. In section 3, we give an online algorithm A that runs at a constant speed (that is a function of the known thermal threshold) until an emergency arises, that is, it is determined that some job is in danger of missing its deadline. The speed in the non-emergency time is set so that in the limit the temperature of the device is at most a constant fraction of the thermal threshold. When an emergency is detected, the online algorithm A switches to using the OA speed scaling algorithm, which is guaranteed to finish all jobs by their deadline. When no unfinished jobs are in danger of missing a deadline, the speed scaling algorithm A switches from OA back to the nonemergency constant speed policy. We show that A is $\frac{e}{e-1}(\ell + 3e\alpha^\alpha)$ -competitive for temperature, where $\ell = (2 - (\alpha - 1) \ln(\alpha/(\alpha - 1)))^\alpha \leq 2$. When the cube-root rule holds, this gives a competitive ratio of around 350. That is, the job instance can not be feasibly scheduled on a processor with thermal threshold $T_{\max}/350$. This compares to the previous competitive ratio of BKP when $\alpha = 3$ of around 6830. The insight that allowed for a better competitive ratio was that it is only necessary to run faster than this constant speed for brief periods of time, of length proportional to the inverse of the cooling parameter. By analyzing these emergency and non-emergency periods separately, we obtain a better bound on the competitive ratio than what was obtained in [4].

In section 4 we also show, using the same analysis as for A , a slightly improved bound on the temperature competitiveness of the energy optimal YDS schedule.

2 Batched Release

In this section, we consider the special case of the problem where all jobs are released at time 0. Instead of considering the input as consisting of individual jobs, each with a unique deadline and work, we consider the input as a series of deadlines, each with a cumulative work requirement equal to the sum of the work of all jobs due at or before that deadline. Formally, the input consists of n deadlines, and for each deadline d_i , there is a cumulative work requirement, $w_i = \sum_{j=1}^i p_j$, that must be completed by time d_i . With this definition, we then consider testing the feasibility of some schedule S with constraints of the form $W(S, d_i) \geq w_i$ where $W(S, d_i)$ is the total work of S by time d_i . We call these the *work constraints*. We also have the *temperature constraint* that the temperature in S must never exceed T_{\max} . Without loss of generality, we assume that the scheduling policy is to always run the unfinished job with the earliest deadline. Thus, to specify a schedule, it is sufficient to specify the processor speed at each point in time. Alternatively, one can specify a schedule by specifying the cumulative work processed at each point of time (since the speed is the rate of change of cumulative work processed), or one could specify a schedule by giving

the temperature at this point of time (since the speed can be determined from the temperature using Newton's law and the power function).

Before beginning with our analysis it is necessary to briefly summarize the equations describing the maximum work possible over an interval of time, subject to fixed starting and ending temperatures. First we define the function $UMaxW(0, t_1, T_0, T_1)(t)$ to be the maximum cumulative work, up to any time t , achievable for any schedule starting at time 0 with temperature exactly T_0 and ending at time t_1 with temperature exactly T_1 . In [4] it is shown that:

$$UMaxW(0, t_1, T_0, T_1)(t) = \left(\frac{1}{a}\right)^{\frac{1}{\alpha}} \left(\frac{T_1 - T_0 e^{-bt_1}}{e^{-bt_1} - e^{-\frac{bt_1\alpha}{\alpha-1}}}\right)^{\frac{1}{\alpha}} \left(\frac{b}{\alpha-1}\right)^{\frac{1}{\alpha}-1} \left(1 - e^{-\frac{bt}{\alpha-1}}\right) \quad (2)$$

The definition of the function $MaxW(0, t_1, T_0, T_1)(t)$ is identical to the definition of $UMaxW$, with the additional constraint that the temperature may never exceed T_{\max} . Adding this additional constraint implies that $MaxW(0, t_1, T_0, T_1)(t) \leq UMaxW(0, t_1, T_0, T_1)(t)$, with equality holding if and only if the temperature never exceeds T_{\max} in the schedule for $UMaxW(0, t_1, T_0, T_1)(t)$. A schedule or curve is said to be a *UMaxW curve* if it is equal to $UMaxW(0, t_1, T_0, T_1)(t)$ for some choice of parameters. A *MaxW curve/schedule* is similarly defined. We are only concerned with *MaxW* curves that are either *UMaxW* curves that don't exceed T_{\max} or *MaxW* curves that end at temperature T_{\max} . It is shown in [4] that these type of *MaxW* curves have the form:

$$MaxW(0, t_1, T_0, T_{\max})(t) = \begin{cases} UMaxW(0, \gamma, T_0, T_{\max})(t) & : t \in [0, \gamma] \\ UMaxW(0, \gamma, T_0, T_{\max})(\gamma) + (bT_{\max})^{\frac{1}{\alpha}} (t - \gamma) & : t \in (\gamma, t_1] \end{cases} \quad (3)$$

Here γ is the largest value of t_1 for which the curve $UMaxW(0, t_1, T_0, T_{\max})(t)$ does not exceed temperature T_{\max} . It is shown in [4] that γ is implicitly defined by the following equation:

$$\frac{1}{\alpha-1} T_0 e^{-\frac{b\gamma\alpha}{\alpha-1}} + T_{\max} - \frac{\alpha}{\alpha-1} T_{\max} e^{-\frac{b\gamma}{\alpha-1}} = 0 \quad (4)$$

2.1 Known Maximum Temperature

In this subsection we assume the thermal threshold of the device T_{\max} is known to the algorithm, and consider batched jobs. If there is a feasible schedule, our algorithm iteratively constructs schedules S_i satisfying the following invariant:

Definition 1. Max-Work Invariant: S_i completes the maximum work possible subject to:

- For all times $t \in [0, d_n]$, the temperature of S_i does not exceed T_{\max}
- $W(S_i, d_j) \geq w_j$ for all $1 \leq j \leq i$

By definition, the schedule S_0 is defined by $MaxW(0, d_n, 0, T_{\max})(t)$. The intermediate schedules S_i may be infeasible because they may miss deadlines after d_i , but S_n is a feasible schedule and for any feasible input an S_i exists for all i . The only reason why the schedule S_{i-1} cannot be used for S_i is that S_{i-1} may violate the i^{th} work constraint, that is $W(S_{i-1}, d_i) < w_i$. Consider the constraints such that for any $j < i$, $W(S_{i-1}, d_j) = w_j$. We call these *tight* constraints in S_{i-1} . Now consider the set of possible schedules $S_{i,j}$, such that j is a tight constraint in S_{i-1} , where intuitively during the time period $[d_j, d_i]$, $S_{i,j}$ speeds up to finish enough work so that the i^{th} work constraint is satisfied and the temperature at time d_i is minimized. Defining the temperature of any schedule S_{i-1} at deadline d_j as T_j^{i-1} , we formally define $S_{i,j}$:

Definition 2. For tight constraint $j < i$ in S_{i-1} ,

$$S_{i,j} = \begin{cases} S_{i-1} & : t \in [0, d_j) \\ UMaxW(0, d_i - d_j, T_j^{i-1}, T_i^{i,j})(t) & : t \in (d_j, d_i) \\ MaxW(0, (d_n - d_i), T_i^{i,j}, T_{\max})(t) & : t \in (d_j, d_n] \end{cases}$$

where $T_i^{i,j}$ is the solution of $UMaxW(0, d_i - d_j, T_j^{i-1}, T_i^{i,j})(d_i - d_j) = (w_i - w_j)$

We show that if S_i exists, then it is one of the $S_{i,j}$ schedules. In particular, S_i will be equal to the first schedule $S_{i,j}$ (ordered by increasing j) that satisfies the first i work constraints and the temperature constraint.

Algorithm Description: At a high level the algorithm is two nested loops, where the outer loop iterates over i , and preserves the max-work invariant. If the i^{th} work constraint is not violated in S_{i-1} , then S_i is set to S_{i-1} . Otherwise, for all tight constraints j in S_{i-1} , S_i is set to the first $S_{i,j}$ that satisfies the first i work constraints and the temperature constraint. If such a $S_{i,j}$ doesn't exist, then the instance is declared to be infeasible. The following lemma establishes the correctness of this algorithm.

Lemma 1. Assume a feasible schedule exists for the instance in question. If S_{i-1} is infeasible for constraint i , then S_i is equal to $S_{i,j}$, where j is minimized subject to the constraint that $S_{i,j}$ satisfies the first i work constraints and the temperature constraint.

2.2 Unknown Maximum Temperature

In this section we again consider batched jobs, and consider the objective of minimizing the maximum temperature ever reached in a feasible schedule. Let Opt be the optimal schedule, and T_{\max} be the optimum objective value. We know from the previous section that the optimum schedule can be described by the concatenation of $UMaxW$ curves C_1, \dots, C_{k-1} , possibly with a single $MaxW$ curve, C_k , concatenated after C_{k-1} . Each C_i begins at the time of the $(i-1)^{st}$ tight work constraint and end at the time of the i^{th} tight work constraint. Our algorithm will iteratively compute C_i . That is, on the i^{th} iteration,

C_i will be computed from the input instance and C_1, \dots, C_{i-1} . In fact, it is sufficient to describe how to compute C_1 , as the remaining C_i can be computed recursively. Alternatively, it is sufficient to show how to compute the first tight work constraint in Opt.

To compute C_1 , we need to classify work constraints. We say that the i^{th} work constraint is a *UMaxW constraint* if the single cumulative work curve that exactly satisfies the constraint with the smallest maximum temperature possible corresponds to equation (2). Alternatively, we say that the i^{th} work constraint is a *MaxW constraint* if the single cumulative work curve that exactly satisfies the constraint with the smallest maximum temperature possible corresponds to equation (3). We know from the results in the last section every work constraint must either be a *MaxW* constraint or a *UMaxW* constraint. In Lemma 2 we show that it can be determined in $O(1)$ time whether a particular work constraint is a *UMaxU* constraint or a *MaxW* constraint. In Lemma 3 we show how to narrow the candidates for *UMaxW* constraints that give rise to C_1 down to one. The remaining constraint is referred to as the *UMaxW-winner*. In Lemma 5 we show how to determine if the *UMaxW-winner* candidate is a better option for C_1 than any of the *MaxW* candidates. If this is not the case, we show in Lemma 6 how to compute the best *MaxW* candidate.

Lemma 2. *Given a work constraint $W(S, d_i) \geq w_i$, it can be determined in $O(1)$ time whether it is a *UMaxW* constraint or a *MaxW* constraint.*

Proof. For initial temperature T_0 , we solve $UMaxW(0, d_i, T_0, T_i)(d_i) = w_i$ for T_i as in the known T_{\max} case. Now we consider equation (4) for γ with $T_{\max} = T_i$:

$$\frac{1}{\alpha - 1} T_0 e^{\frac{-b\gamma\alpha}{\alpha-1}} + T_i - \frac{\alpha}{\alpha - 1} T_i e^{\frac{-b\gamma}{\alpha-1}} = 0$$

If we plug in d_i for γ and we get a value larger than 0 then $\gamma < d_i$ and thus the curve $UMaxW(0, d_i, T_0, T_i)(t)$ must exceed T_i during some time $t < d_i$, thus the constraint is a *MaxW* constraint. If the value is smaller than 0 then $\gamma > d_i$, the curve $UMaxW(0, d_i, T_0, T_i)(t)$ never exceeds T_i , and thus the constraint is a *UMaxW* constraint. \square

Lemma 3. *All of the *UMaxW* constraints, but one, can be disqualified as a candidate for C_1 in time $O(n)$.*

Proof. Consider any two *UMaxW* constraints, i and j with $i < j$. We want to show that the two work curves exactly satisfying constraints i and j must be non-intersecting, except at time 0, and that we can determine which work curve is larger in constant time. This together with Lemma 2 would imply we can get rid of all *UMaxW* constraints but one in time $O(n)$ for n constraints. For initial temperature T_0 , we can fully specify the two curves by solving $UMaxW(0, d_i, T_0, T_i)(d_i) = w_i$ and $UMaxW(0, d_j, T_0, T_j)(d_j) = w_j$ for T_i and T_j respectively. We can then compare them at all times prior to d_i using equation (2), i.e., $UMaxW(0, d_i, T_0, T_i)(t)$ and $UMaxW(0, d_j, T_0, T_j)(t)$.

Note that for any two *UMaxW* curves defined by equation (2), a comparison results in the time dependent terms (t -dependent) canceling and thus one curve is greater than the other at all points in time up to d_i . Regardless of whether the larger work curve corresponds to constraint i or j , clearly the smaller work curve cannot correspond to the first tight constraint as the larger work curve implies a more efficient way to satisfy both constraints. To actually determine which curve is greater, we can simply plug in the values for the equations and check the values of the non-time dependent terms. The larger term must correspond to the dominating work curve. \square

In order to compare the *UMaxW*-winner's curve to the *MaxW* curves, we may need to extend the *UMaxW*-winner's curve into what we call a *UMaxW-extended* curve. A *UMaxW-extended* curve is a *MaxW* curve, describable by equation (3), that runs identical to the *UMaxW* constraint's curve on the *UMaxW* interval, and is defined on the interval $[0, d_n]$. We now show how to find this *MaxW* curve for any *UMaxW* constraint.

Lemma 4. *Any UMaxW constraint's UMaxW-Extended curve can be described by equation (3) and can be computed in $O(1)$ time.*

Proof. For any *UMaxW* curve satisfying a *UMaxW* constraint, the corresponding speed function is defined for all times $t \geq 0$ as follows:

$$S(t) = \frac{b}{(\alpha - 1)} \frac{1}{\alpha} \left(\frac{T_i - T_0 e^{-bd_i}}{e^{-bd_i} - e^{-\frac{bd_i \alpha}{\alpha - 1}}} \right)^{\frac{1}{\alpha}} e^{-\frac{bt}{\alpha - 1}}$$

Thus we can continue running according to this speed curve after d_i . As the speed is a constantly decreasing function of time, eventually the temperature will stop increasing at some specific point in time. This is essentially the definition of γ and for any fixed γ there exists a T_{\max} satisfying it which can be found by solving for T_{\max} in the γ equation. To actually find the time when the temperature stops increasing, we can binary search over the possible values of γ , namely the interval $(d_i, \frac{\alpha - 1}{b} \ln \frac{\alpha}{\alpha - 1}]$. For each time we can directly solve for the maximum temperature using the γ equation and thus the entire *UMaxW* curve is defined. We then check the total work accomplished at d_i . If the total work is less than w_i , then γ is too small, if larger, then γ is too large. Our binary search is over a constant-sized interval and each curve construction and work comparison takes constant time, thus the entire process takes $O(1)$ time. Once we have γ and the maximum temperature, call it T_γ , we can define the entire extended curve as *UMaxW*(0, γ , T_0 , T_γ)(t) for $0 \leq t < \gamma$ and $(bT_\gamma)^{1/\alpha} t$ for $t \geq \gamma$, in other words, *MaxW*(0, ∞ , T_0 , T_γ)(t) with $T_{\max} = T_\gamma$. \square

Lemma 5. *Any MaxW constraint satisfied by a UMaxW-Extended curve can't correspond to C_1 . If any MaxW constraint is not satisfied by a UMaxW-Extended curve then the UMaxW constraint can't correspond to C_1 .*

Proof. To satisfy the winning $UMaxW$ constraint exactly, we run according to the $UMaxW$ -extended curve corresponding to the $UMaxW$ constraint's exact work curve. Thus if a $MaxW$ constraint is satisfied by the entire extended curve, then to satisfy the $UMaxW$ constraint and satisfy the $MaxW$ constraint it is most temperature efficient to first exactly satisfy the $UMaxW$ constraint then the $MaxW$ constraint (if it is not already satisfied). On the other hand, if some $MaxW$ constraint is not satisfied then it is more efficient to exactly satisfy that constraint, necessarily satisfying the $UMaxW$ constraint as well. \square

Lemma 6. *If all $UMaxW$ constraints have been ruled out for C_1 , then C_1 , and the entire schedule, can be determined in time $O(n)$.*

Proof. To find the first tight constraint, we can simply create the $MaxW$ curves exactly satisfying each constraint. For each constraint, we can essentially use the the same method as in Lemma 4 for extending the $UMaxW$ winner to create the $MaxW$ curve. The difference here is that we must also add the work of the constant speed portion to the work of the $UMaxW$ portion to check the total work at the constraint's deadline. However this does not increase the construction time, hence each curve still takes $O(1)$ time per constraint.

Once we have constructed the curves, we can then compare any two at the deadline of the earlier constraint. The last remaining work curve identifies the first tight constraint and because we have the $MaxW$ curve that exactly satisfies it, we have specified the entire optimal scheduling, including the minimum T_{\max} possible for any feasible schedule. As we can have at most n $MaxW$ constraints and construction and comparison take constant time, our total time is $O(n)$. \square

Theorem 1. *The optimal schedule can be constructed in time $O(n^2)$ when T_{\max} is not known.*

Proof. The theorem follows from using Lemma 3 which allows us to produce a valid $MaxW$ curve by Lemma 4. We then apply Lemma 5 by comparing the $UMaxW$ -winner's work at each $MaxW$ constraint. If all $MaxW$ constraints are disqualified, we've found the first tight constraint, else we apply Lemma 6 to specify the entire schedule. In either case, we've defined the schedule up to at least one constraint in $O(n)$ time. \square

3 Online Algorithm

Our goal in this section is to describe an online algorithm A , and analyze its competitiveness. Note that all proofs in this section have been omitted due to space limitations but can be found in the full paper.

Algorithm Description: A runs at a constant speed of $(\ell b T_{\max})^{1/\alpha}$ until it determines that some job will miss its deadline. Here, $\ell = (2 - (\alpha - 1) \ln(\frac{\alpha}{\alpha - 1}))^\alpha \leq 2$. At this point A runs according to the online algorithm OA. When enough work is finished such that running at constant speed $(\ell b T_{\max})^{1/\alpha}$ will not cause any

job to miss its deadline, A then switches back to running at the constant speed.

Before beginning, we briefly note some characteristics of the energy optimal algorithm, YDS, as well as some characteristics of the online algorithm OA. We require one main property from YDS, a slight variation on Claim 2.3 in [4]:

Claim 1. *For any speed s , consider any interval, $[t_1, t_2]$ of maximal time such that YDS runs at speed strictly greater than s . YDS schedules within $[t_1, t_2]$, exactly those jobs that are released no earlier than t_1 and due no later than t_2 .*

We also need that YDS is energy optimal within these maximal intervals. This is a direct consequence of the total energy optimality of YDS. Lastly note that YDS schedules jobs according to EDF. For more on YDS, see [2] and [4].

For the online algorithm OA, we need only that it always runs, at any time t , at the minimum feasible constant speed for the amount of unfinished work at time t and that it has a competitive ratio of α^α for total energy [4].

We will first bound the maximum amount of work that the optimal temperature algorithm can perform during intervals longer than the inverse of the cooling parameter b . This is the basis for showing that the constant speed of A is sufficient for all but intervals of smaller than $1/b$.

Lemma 7. *For any interval of length $t > 1/b$, the optimal temperature algorithm completes strictly less than $(lbT_{\max})^{1/\alpha} \cdot (t)$ work.*

We now know that if all jobs have a lifetime of at least $1/b$, A will always run at a constant speed and be feasible, thus we have essentially handled the competitiveness of A in non-emergency periods. Now we need to consider A 's competitiveness during the emergency periods, i.e., when running at speed $(lbT_{\max})^{1/\alpha}$ would cause A to miss a deadline. To do this, we will show that these emergency periods are contained within periods of time where YDS runs faster than A 's constant speed and that during these larger periods we can directly compare A to YDS via OA. We start by bounding the maximal length of time in which YDS can run faster than A 's constant speed.

Lemma 8. *Any maximal time period where YDS runs at a speed strictly greater than $(lbT_{\max})^{1/\alpha}$ has length $< 1/b$.*

We call these maximal periods in YDS *fast periods* as they are characterized by the fact that YDS is running strictly faster than $(lbT_{\max})^{1/\alpha}$. Now we show that A will never be behind YDS on any individual job outside of fast periods. This then allows us to describe A during fast periods.

Lemma 9. *At the beginning and ending of every fast period, A has completed as much work as the YDS schedule on each individual job.*

Lemma 10. *A switches to OA only during fast periods.*

We are now ready to upper bound the energy usage of A , first in a fast period, and then in an interval of length $1/b$. We then use this energy bound to upper bound the temperature of A . We use a variation on Theorem 2.2 in [4] to relate energy to temperature. We denote the maximum energy used by an algorithm, ALG , in any interval of length $1/b$, on input I , as $C[ALG(I)]$ or simply $C[ALG]$ when I is implicit. Note that this is a different interval size than used in [4]. We similarly denote the maximum temperature of ALG as $T[ALG(I)]$ or $T[ALG]$.

Lemma 11. *For any schedule S , and for any cooling parameter $b \geq 0$,*

$$\frac{aC[S]}{e} \leq T[S] \leq \frac{e}{e-1} aC[S]$$

Lemma 12. *A is α^α -competitive for energy in any single maximal fast period.*

Lemma 13. *A uses at most $(\ell + 3e\alpha^\alpha)T_{\max}$ energy in an interval of size $1/b$.*

Theorem 2. *A is $(\frac{e}{e-1}(\ell + 3e\alpha^\alpha))$ -competitive for temperature.*

4 Additional Results

Theorem 3. *Using the technique from the previous section, it can be shown that the energy optimal offline algorithm, YDS , is $\frac{e}{e-1}(\ell + 3e)$ -competitive for temperature, where $15.5 < \frac{e}{e-1}(\ell + 3e) < 16.1$.*

References

1. Snowdon, D.C., Ruocco, S., Heiser, G.: Power management and dynamic voltage scaling: Myths and facts. In: Proceedings of the 2005 Workshop on Power Aware Real-time Computing, New Jersey, USA (2005)
2. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced cpu energy. In: FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society (1995) 374
3. Albers, S.: Algorithms for energy saving. In Albers, S., Alt, H., Nher, S., eds.: Efficient Algorithms. Volume 5760 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 173–186
4. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. *J. ACM* **54** (2007) 1–39
5. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro* **20** (2000) 26–44
6. Li, M., Yao, A.C., Yao, F.F.: Discrete and continuous min-energy schedules for variable voltage processors. Proceedings of the National Academy of Sciences of the United States of America **103** (2006) 3983–3987
7. Bansal, N., Bunde, D.P., Chan, H.L., Pruhs, K.: Average rate speed scaling. In: LATIN'08: Proceedings of the 8th Latin American conference on Theoretical informatics, Berlin, Heidelberg, Springer-Verlag (2008) 240–251

8. Bansal, N., Chan, H.L., Pruhs, K., Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming, Berlin, Heidelberg, Springer-Verlag (2009) 144–155
9. Chrobak, M., Dürr, C., Hurand, M., Robert, J.: Algorithms for temperature-aware task scheduling in microprocessor systems. In: AAIM '08: Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management, Berlin, Heidelberg, Springer-Verlag (2008) 120–130
10. Albers, S.: Energy-efficient algorithms. *Commun. ACM* **53** (2010) 86–96
11. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. *SIGACT News* **36** (2005) 63–76
12. Rao, R., Vrudhula, S.: Performance Optimal Processor Throttling Under Thermal Constraints. In: CASES '07: Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Salzburg, Austria, (2007) 257–266