# Algorithms for Temperature-Aware Task Scheduling in Microprocessor Systems

Marek Chrobak[1], Christoph Dürr[2], Mathilde Hurand[2], and Julien Robert[3]

[1] Department of Computer Science, University of California, Riverside, CA 92521.
Supported by NSF grants OISE-0340752 and CCF-0729071.
[2] CNRS, LIX UMR 7161, Ecole Polytechnique, France.
Supported by ANR Alpage.
[3] Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France.

**Abstract.** We study scheduling problems motivated by recently developed techniques for microprocessor thermal management at the operating systems level. The general scenario can be described as follows. The microprocessor temperature is controlled by the hardware thermal management system that continuously senses the chip temperature and automatically reduces the processor's speed as soon as the thermal threshold is exceeded. Some tasks are more CPU-intensive than other and thus generate more heat during execution. The cooling system operates nonstop, reducing (at an exponential rate) the deviation of the processor's temperature from the ambient temperature. As a result, the processor's temperature, and thus the performance as well, depends on the order of the task execution. Given a variety of possible underlying architectures, models for cooling and for hardware thermal management, as well as types of tasks, this gives rise to a plethora of interesting and never studied scheduling problems.

We focus on scheduling real-time jobs in a simplified model for cooling and thermal management. A collection of unit-length jobs is given, each job specified by its release time, deadline and heat contribution. If, at some time step, the temperature of the system is $t$ and the processor executes a job with heat contribution $h$, then the temperature at the next step is $(t+h)/2$. The temperature cannot exceed the given thermal threshold $\tau$. The objective is to maximize the throughput, that is, the number of tasks that meet their deadlines. We prove that in the offline case computing the optimum schedule is NP-hard, even if all jobs are released at the same time. In the online case, we show a 2-competitive deterministic algorithm and a matching lower bound.

## 1 Introduction

*Background.* The problem of managing the temperature of processor systems is not new; in fact, the system builders had to deal with this challenge since the inception of computers. Since early 1990s, the introduction of

battery-operated laptop computers and sensor systems highlighted the related issue of controlling the energy consumption.

Most of the initial work on these problems was hardware and systems oriented, and only during the last decade substantial progress has been achieved on developing models and algorithmic techniques for microprocessor temperature and energy management. This work proceeded in several directions. One direction is based on the fact that the energy consumption is a fast growing function of the processor speed (or frequency). Thus we can save energy by simply slowing down the processor. Here, algorithmic research focussed on *speed scaling* – namely dynamically adjusting the processor speed over time to optimize the energy consumption while ensuring that the system meets the desired performance requirements. Another technique (applicable to the whole system, not just the microprocessor) involves *power-down strategies*, where the system is powered-down or even completely turned off when some of its components are idle. Since changing the power level of a system introduces some overhead, scheduling the work to minimize the overall energy usage in this model becomes a challenging optimization problem.

Models have also been developed for the processor's thermal behavior. Here, the main objective is to ensure that the system's temperature does not exceed the so-called *thermal threshold*, above which the processor cannot operate correctly, or may even be damaged. In this direction, techniques and algorithms have been proposed for using speed-scaling to optimize the system's performance while maintaining the temperature below the threshold.

We refer the reader to the survey by Irani and Pruhs [5], and references therein, for more in-depth information on the models and algorithms for thermal and energy management.

*Temperature-aware scheduling.* The above models address energy and thermal management at the micro-architecture level. In contrast, the problem we study in this paper addresses the issue of thermal management at the operating systems level. Most of the previous work in this direction focussed on multi-core systems, where one can move tasks between the processors to minimize the maximum temperature [9, 1, 2, 6–8, 4, 10]. However, as it has been recently discovered, even in single-core systems one can exploit variations in heat contributions among different tasks to reduce the processor's temperature through appropriate task scheduling [1, 4, 6, 7, 11]. In this scenario, the microprocessor temperature is controlled by the hardware dynamic thermal management (DTM) sys-

tems that continuously senses the chip temperature and automatically reduces the processor's speed as soon as the thermal threshold (maximum safe operating temperature) is exceeded. Typically, the frequency is reduced by half, although it can be further reduced to one fourth or even one eighth, if needed. Once the chip cools down to below the threshold, the frequency is increased again. In addition, the cooling system operates non-stop, reducing the deviation of the processor's temperature from the ambient temperature at an exponential rate. The general accepted model stipulates that there is a constant time $T$, such that after $T$ time units, the cooling system divided by 2 the difference between the processor's temperature and the ambient temperature. In this work we assume that this constant is exactly the length of a unit length job. The general case will be considered in the full version of this paper.

Different tasks use different microprocessor units in different ways; in particular, some tasks are more CPU-intensive than other. As a result, the processor's thermal behavior – and thus the performance as well – depends on the order of the task execution. In particular, Yang *et al.* [11] point out that the accepted model for the microprocessor thermal behavior implies that, given two jobs, scheduling the "hotter" job before the "cooler" one, results in a lower final temperature. They take advantage of this phenomenon to improve the performance of the OS scheduler.

With multitudes of possible underlying architectures (for example, single- vs. multi-core systems), models for cooling and hardware thermal management, as well as types of jobs (real-time, batch, etc.), this gives rise to a plethora of interesting and never yet studied scheduling problems.

*Our model.* We focus on scheduling real-time jobs in a somewhat simplified model for cooling and thermal management. The time is divided into unit time slots and each job has unit length. These jobs represent unit slices of the processes present in the OS scheduler's queue. We assume that the heat contributions of these jobs are known. This is counterintuitive, but reasonably realistic, for, as discussed in [11], these values can be well approximated using appropriate prediction methods.

In our thermal model we assume, without loss of generality, that the ambient temperature is 0 and that the heat contributions are expressed in the units of temperature (that is, by how much they would increase the chip temperature in the absence of cooling). Suppose that at a certain time the processor temperature is $t$ and we are about to execute a job with heat contribution $h$ in the next time slot. In reality [11], during the execution of this job, its heat contribution is spread over the whole slot

and so is the effect of cooling; thus, the final temperature can be expressed using an integral function. In this paper, we use a simplified model where we first take into account the job's heat contribution, and then apply the cooling, where the cooling simply reduces the temperature by half.

Finally, we assume that only one processor frequency is available. Consequently, if there is no job whose execution does not cause a thermal violation, the processor must stay idle through the next time slot.

*Our results.* Summarizing, our scheduling problem can be now formalized as follows. A collection of unit-length jobs is given, each job $j$ with a release time $r_j$, deadline $d_j$ and heat contribution $h_j$. If, at some time step, the temperature of the system is $t$ and the processor executes a job $j$, then the temperature at the next step is $(t + h_j)/2$. The temperature cannot exceed the given thermal threshold $\tau$. The objective is to compute a schedule which maximizes the number of tasks that meet their deadlines.

We prove that in the offline case computing the optimum schedule is NP-hard, even if all jobs are released at the same time. In the online case, we show a 2-competitive deterministic algorithm and a matching lower bound.

## 2 Terminology and Notation

The input consists of $n$ unit-length jobs that we number $1, 2, ..., n$. Each job $j$ is specified by a triple $(r_j, d_j, h_j)$, where $r_j$ is its release time, $d_j$ is the deadline and $h_j$ is its heat contribution. The time is divided into unit-length slots and each job can be executed in any time slot in the interval $[r_j, d_j]$. The system temperature is initially 0 and it changes according to the following rules: if the temperature of the system at a time $u$ is $t$ and the processor executes $j$ then the temperature at time $u+1$ is $(t+h_j)/2$. The temperature cannot exceed the given thermal threshold $\tau$ that we assume to be 1. Thus if $(t+h_j)/2 > 1$ then $j$ cannot be executed at time $u$. Idle slots are treated as executing a job with heat contribution 0, that is, after an idle slot the temperature decreases by half.

Given an instance, as above, the objective is to compute a schedule with maximum *throughput*, where throughput is defined as the number of completed jobs. Extending the standard notation for scheduling problems, we denote the offline version of this problem by $1|r_i, h_i| \sum U_i$.

In the online version, denoted $1|\text{online-}r_i, h_i| \sum U_i$, jobs are available to the algorithm at their release time. Scheduling decisions of the algorithm cannot depend on the jobs that have not yet been released.

## 3 The NP-Completeness Proof

The special case of the problem $1|r_i, h_i|\sum U_i$, when all jobs have the same release time and the same deadline is denoted as $1|h_i|C_{\max}$: The objective value $C_{\max}$ stands for minimizing the maximum completion time of the jobs when there are no deadlines to respect, and the decision version of this optimization problem is exactly deciding if there is a feasible schedule where all jobs complete before a given common deadline.

We can show that this problem is NP-complete. Why is it interesting? In fact, one common approach in designing on-line algorithms for scheduling is to compute the optimal schedule for the pending jobs, and use this schedule to make the decision as to what execute next. The set of pending jobs forms an instance where all jobs have the same release time. This does not necessarily mean that the above method cannot work (assuming that we do not care about the running time of the online algorithm), but it makes this approach much less appealing, since reasoning about the properties of the pending jobs is likely to be very difficult.

**Theorem 1.** *The offline problem $1|h_i|C_{\max}$ is NP-hard.*

*Proof.* The reduction is from Numerical 3 Dimensional Matching. In the later problem we are given 3 sequences $A, B, C$ of $n$ non-negative integers each and an integer $m$. A 3-dimensional matching is a set of $n$ triplets $(a, b, c) \in A \times B \times C$ such that each number is matched exactly once. The problem consists in deciding if there is a 3-dimensional matching, such that all matches $(a, b, c)$ satisfy $a + b + c = m$.

Without loss of generality, we can assume (A1) that every $x \in A \cup B \cup C$ satisfies $x \leq m$ and (A2) that $\sum_{x \in A \cup B \cup C} x = mn$.

In this reduction we need two constants. Let be $\alpha = 1/13$ and $\epsilon = (1/8 - 3\alpha)/m > 0$. We also define the function $f : x \mapsto \alpha + \epsilon x$.

We construct an instance to the heating problem. In total there will be $4n + 1$ jobs, and all are released at time 0 and have deadline $4n + 1$. These jobs will be of two types:

1. First we have $3n$ jobs that correspond to the instance of Numerical 3 Dimensional Matching.
   - For every $a \in A$, there is a job of heat contribution $8f(a)$.
   - For every $b \in B$, there is a job of heat contribution $4f(b)$.
   - For every $c \in C$, there is a job of heat contribution $2f(c)$.

   We call these respectively $A$-, $B$- and $C$-jobs.
2. Next, we have $n + 1$ "gadget" jobs. The first of these jobs has heat contribution 2, and the remaining ones 1.75. We call these respectively 2- and 1.75-jobs.

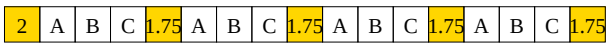


**Fig. 1.** The structure of the solution of the scheduling problem.

The idea of the construction is that the gadget jobs are so hot, that they need to be scheduled every 4-th time unit, separating the time into $n$ blocks of 3 consecutive time slots each. Now every remaining job has a heat contribution that consists in two parts. A constant part and a tiny part depending on the instance of the matching problem. The constant part is so large that in every block there is a single $A$-, $B$- and $C$-job and they are scheduled in that order. This defines a 3-dimensional matching. Now the gadget jobs are so hot, that they force every matching $(a, b, c)$ to satisfy $a + b + c \leq m$. Let's make this argument formal.

Suppose there is a solution to the matching problem. We show that there is a solution to the heating problem. Schedule the job 2 at time 0, and all other gadget jobs every 4th time slot. Now the remaining slots are grouped into blocks consisting of 3 consecutive time slots. Each $i$-th triplet $(a, b, c)$ from the matching is associated to the $i$-th block, and the corresponding $A$-,$B$- and $C$-jobs are executed in there in the order $A, B, C$, see figure 1. By construction every job meets its deadline, it remains to show that at no point the temperature exceeds 1. The non-gadget jobs have all heat contribution smaller than 1, by assumption (A1), so every execution of a non-gadget job would preserve that the temperature is not more than 1. Now we show by induction that right after the execution of a gadget job, the temperature is exactly 1. This is clearly the case after execution of the first job, since its heat contribution is 2. Now let $u$ be the time when a 1.75-job is scheduled, and suppose that at time $u-3$ the temperature was 1. Let $(a, b, c)$ be the matching associated to the slots between $u - 3$ and $u$. Then at time $u$ the temperature is

$$\frac{1}{8} + \frac{8f(a)}{8} + \frac{4f(b)}{4} + \frac{2f(c)}{2} = \frac{1}{8} + 3\alpha + (a + b + c)\epsilon = \frac{1}{8} + 3\alpha + m\epsilon = \frac{1}{4}.$$

This shows that at time $u + 1$ the temperature is again 1. We conclude that the schedule is feasible.

Now we show the remaining direction in the NP-hardness proof, namely that if there is a solution to the scheduling problem, then there is a solution to the matching problem. To that purpose, suppose that there is a solution to the scheduling problem. We first show that it has the form of figure 1. Since all $4n + 1$ jobs have deadline $4n + 1$, all jobs must be

scheduled without any idle time between time 0 and $4n+1$. This means that the gadget job of heat contribution 2 must be scheduled at time 0, because that is the only moment of temperature 0.

Now we claim that all 1.75-jobs have to be scheduled every 4-th time unit. For a proof by contradiction assume that at some times $u$ and $u+3$ two gadget jobs are scheduled. The lightest job that can be scheduled at times $u+1, u+2$ would be a $C$-job of heat contribution $2f(0) = 2\alpha$. The temperature at time $u+1$ is at least $1.75/2 = 7/8$. Therefore the temperature at time $u+3$ is at least

$$\frac{7}{32} + \frac{3}{2}\alpha = \frac{7}{32} + \frac{3}{26} > 1/4.$$

This contradicts that a 1.75-job is scheduled at time $u+3$, since the temperature would exceed 1 at $u+4$. So we can conclude that the 1.75-jobs are scheduled every 4-th time unit. This partitions the remaining time slots into blocks of 3 time slots each.

We show now that every block contains exactly one $A$-job, one $B$-job and one $C$-job, in that order. For a proof by contradiction assume that some $A$-job is scheduled at the 2nd position of some block. Its heat contribution is at least $8f(0)$. The coolest jobs scheduled at position 1 and 3 have heat contribution $2f(0) = 2\alpha$ at least. Then the temperature at the end of the block is at least

$$\frac{7}{64} + \frac{\alpha}{4} + 2\alpha + \alpha = \frac{7}{32} + \frac{13}{4}\alpha = \frac{7}{32} + \frac{1}{4} > 1/4.$$

This would be too hot for the following 1.75-job. The same argument shows that $A$-jobs cannot be scheduled at position 3 in a block, and therefore the 1st position of a block is always occupied by an $A$-job. Now we show that a $B$-job cannot be scheduled at the 3rd position of some block. This is because otherwise the temperature at the end of the block would be again at least

$$\frac{7}{64} + \alpha + \frac{\alpha}{2} + 2\alpha = \frac{7}{32} + \frac{7}{2}\alpha > 1/4.$$

We showed that every block contains jobs that correspond to some matching $(a, b, c) \in A \times B \times C$. It remains to show that each matching satisfies $a + b + c = m$. Let $(a_i, b_i, c_i)$ be the matching corresponding to the $i$-th block, for $1 \leq i \leq n$. Let be $t_i = (3\alpha - (a_i + b_i + c_i)\epsilon)/2 + 7/8$. Also let be $t_0 = 1$. Then for $1 \leq k \leq n$ the temperature at the end of time $4k+1$ is

$$\sum_{i=1}^{k} \left(\frac{1}{16}\right)^{k-i} t_i \leq 1, \tag{1}$$

For convenience we write $p_i = 15/16 - t_i$. Assumption (A2) said that the average of $a_i + b_i + c_i$ over $1 \leq i \leq n$ is $m$. Therefore the average of $t_i$ (excluding $t_0$) is 15/16, and therefore we have

$$\sum_{i=1}^{n} p_i = 0. \tag{2}$$

Furthermore from (1) we have

$$\forall\, 1 \leq k \leq n : \sum_{i=1}^{k} 16^i p_i \leq 0. \tag{3}$$

We will show now that for every $i$, $p_i = 0$. This would imply that $a_i + b_i + c_i = m$, and imply that the matching problem has a solution. For a proof by contradiction suppose that $(p_i)$ is not all zero. Let $\ell$ be the smallest index such that $p_\ell > 0$ and

$$p_1 + \ldots + p_\ell \geq 0. \tag{4}$$

By minimality of $\ell$ we have for every $2 \leq k \leq \ell - 1$

$$p_1 + \ldots + p_{k-1} \leq 0 \quad \text{and} \quad p_k + \ldots + p_\ell \geq 0$$

So we can increase the coefficients of the variables $p_k, \ldots, p_\ell$ by some same amount and preserve (4), obtain

$$16p_1 + 16^2 p_2 + \ldots 16^2 p_{\ell-1} + 16^2 p_\ell \geq 0$$

and finally

$$16p_1 + 16^2 p_2 + \ldots 16^{\ell-1} p_{\ell-1} + 16^{\ell-1} p_\ell \geq 0.$$

Now using $p_\ell > 0$, we obtain

$$16p_1 + 16^2 p_2 + \ldots 16^{\ell-1} p_{\ell-1} + 16^{\ell} p_\ell > 0,$$

which contradicts (4). This completes the proof. □

## 4 An Online Competitive Algorithm

In this section we show that there is a 2-competitive algorithm. We will show, in fact, that a large class of deterministic algorithms is 2-competitive.

Given a schedule, we will say that a job $j$ is *pending* at time $u$ if $j$ is released, not expired (that is, $r_j \leq u < d_j$) and $j$ has not been scheduled before $u$. If the temperature at time $u$ is $t$ and $u$ is pending, then we call $j$ *available* if $t + h_j \leq 2$, that is, $j$ is not too hot to be executed.

We say that job $j$ *dominates* job $k$ if $j$ is both cooler and has the same or smaller deadline than $k$, that is $h_j \leq h_k$ and $d_j \leq d_k$. Also we say that $j$ *dominates strictly* job $k$ if at least one of the inequalities is strict. An online algorithm is called *reasonable* if at each step (i) it schedules a job whenever if is available (the *non-waiting property*), and (ii) it schedules a pending job that is not strictly dominated by another pending job. The class of reasonable algorithms contains, for example, the algorithm which schedules the coolest available job, and the algorithm which schedules the earliest deadline available job.

**Theorem 2.** *Any reasonable alg. for* $1|\textit{online-}r_i, h_i|\sum U_i$ *is* 2*-competitive.*

*Proof.* We define a charging scheme that maps jobs executed by the adversary to jobs executed by the algorithm in such a way that no job in the algorithm's schedule gets more than two charges.

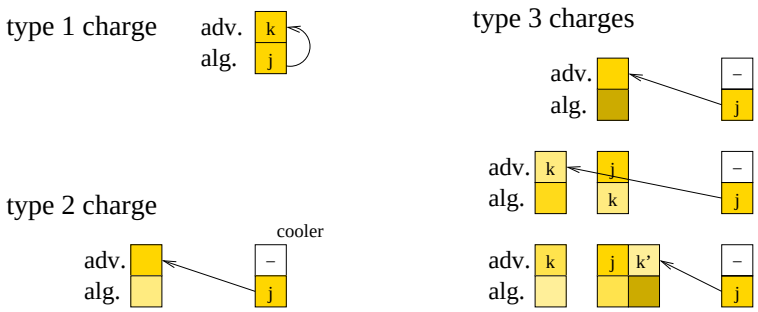


**Fig. 2.** the different types of charges

Fix an instance, and consider both the schedule produced by the algorithm and some arbitrary schedule, say produced by some adversary. Suppose that the adversary schedules $j$ at time $v$. There will be three types of charges.

*Type 1 Charges:* If the algorithm schedules a job $k$ at time $v$, charge $j$ to $k$. Otherwise, we are idle, and we have two cases.

*Type-2 Charges:* Suppose that the algorithm is hotter than the adversary at time $v$ and cooler at $v + 1$. In this case we charge $j$ to the previous *heating* step, by which we mean a time $u < v$, where either the

algorithm schedules a job which is strictly hotter than the one scheduled by the adversary, or where only the algorithm schedules something and the adversary is idle.

*Type-3 Charges:* Suppose now that the algorithm is hotter than the adversary at $v+1$ or cooler at $v$ (and therefore also at $v+1$). Now we argue that $j$ has been scheduled by the algorithm at some earlier time $u$: In the case where the algorithm was cooler at $v$, $j$ would have been available and we assumed the algorithm is non-waiting. Now consider the case where the algorithm was hotter than the adversary at time $v+1$. Its temperature is at most $1/2$ at that time, so the temperature of $j$ can be at most 1. So again it would have been executable at time $v$. This observation is also true for any job cooler than $j$, and this will be useful later.

So let $u < v$ be the execution time of $j$ by the algorithm. To find a job that we can charge $j$ to, we perform *pointer chasing* of sorts: If, at time $u$, the adversary is idle or schedules an equal or hotter job, then we charge $j$ to itself (its "copy" in the algorithm's schedule). Otherwise, let $k$ be the strictly cooler job scheduled by the adversary at $u$. Now we claim that the algorithm schedules $k$ at some time before $v$. Indeed, if it were still pending at $u$, since the algorithm never schedules a dominated job, its deadline is not before the one of $j$, in particular it is after $v$. By our earlier observation it would have been executable at $v$, so at $v$ job $k$ is not pending anymore. We iterate to the time the algorithm executes $k$. This iteration will end at some point, since we deal with cooler and cooler jobs.

Now we show that any job scheduled by the algorithm will get at most two charges. Obviously, each job in the algorithm's schedule gets at most one type-1 charge. Every chain defining a type-3 charge is uniquely defined by the first considered job, and therefore type-3 charges are assigned to distinct jobs. This also holds for type-2 charges, since between any two time steps that satisfy the condition of the type-2 charge there must be a heating step, so the type-2 charges are assigned to distinct heating steps.

Now let $k$ be a job scheduled by the algorithm at some time $v$. By the previous paragraph, $k$ can get at most one charge of each type. If the adversary is idle at $v$, $k$ cannot get a type-1 charge. If the adversary schedules a job $j$ at time $v$ whose heat contribution is equal or smaller than $k$'s then $k$ cannot get the type-2 charge. If it schedules something hotter it cannot get the type-3 charge.

So in total every job scheduled by the adversary is charged to some job scheduled by the algorithm, and every job scheduled by the algorithm

receives no more than 2 charges, therefore the competitive ratio is not more than 2. □

## 5 A Lower Bound on the Competitive Ratio

**Theorem 3.** *Every deterministic online algorithm for* $1|\textit{online-}r_i, h_i|\sum U_i$ *has competitive ratio at least* 2.

*Proof.* We (the adversary) release a job $j \to (r_j, d_j, h_j) = (0, 3, 1)$. If the online algorithm schedules it at time 0, we release a tight job $k \to (1, 2, 1.7)$ and schedule it followed by $j$. If the algorithm does not schedule $j$ at time 0, then we do schedule it at 0 and release (and schedule) a tight job $k' \to (2, 3, 1.7)$ at time 2. In both cases we scheduled two jobs, while the algorithm scheduled only one, completing the proof. □

## 6 Final Comments

Many open problems remain. Perhaps the most intriguing one is to determine the randomized competitive ratio for the problem we studied. The proof of Theorem 3 can easily be adapted to prove the lower bound of 1.5, but we have not been able to improve the upper bound of 2; this is, in fact, the main focus of our current work on this scheduling problem.

Extensions of the cooling model can be considered, where the temperature after executing $j$ is $(t + h_j)/R$, for some $R > 1$. Even this formula, however, is only a discrete approximation for the true model (see, for example, [11]), and it would be interesting to see if the ideas behind our 2-competitive algorithm can be adapted to these more realistic cases.

In reality, thermal violations do not cause the system to idle, but only to reduce the frequency. With frequency reduced to half, a unit job will execute for two time slots. Several frequency levels may be available.

We assumed that the heat contributions are known. This is counterintuitive, but not unrealistic, since the "jobs" in our model are unit slices of longer jobs. Prediction methods are available that can quite accurately predict the heat contribution of each slice based on the heat contributions of the previous slices. Nevertheless, it may be interesting to study a model where exact heat contributions are not known.

Other types of jobs may be studied. For real-time jobs, one can consider the case when not all jobs are equally important, which can be modeled by assigning weights to jobs and maximizing the weighted throughput. For batch jobs, other objective functions can be optimized, for example the flow time.

One more realistic scenario would be to represent the whole processes as jobs, rather then their slices. This naturally leads to scheduling problems with preemption and with jobs of arbitrary processing times. When the thermal threshold is reached, the execution of a job is slowed down by a factor of 2. Here, a scheduling algorithm may decide to preempt a job when another one is released or, say, when the processor gets too hot.

Finally, in multi-core systems one can explore the migrations (say, moving jobs from hotter to cooler cores) to keep the temperature under control. This leads to even more scheduling problems that may be worth to study.

## References


1. F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-driven energy accounting for dynamic thermal management. In *Workshop on Compilers and Operating Systems for Low Power*, 2003.
2. J. Choi, C-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware task scheduling at the system software level. In *International Symposium on Low Power Electronics and Design,*, pages 213–218, 2007.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Co., 1979.
4. M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *SIGPLAN Not.*, 39(11):260–270, 2004.
5. S. Irani and K. R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
6. M. Martonosi J. Donald. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the International Symposium on Computer Architecture*, pages 78–88, 2006.
7. A. Kumar, L. Shang, L-S. Peh, and N. K. Jha. HybDTM: a coordinated hardware-software approach for dynamic thermal management. In *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation*, pages 548–553, 2006.
8. E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P. Bose. Investigating the effects of task scheduling on thermal behavior. In *the 3rd Workshop on Temperature-Aware Computer Systems*, 2006.
9. A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40(4):403–414, 2006.
10. J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, pages 5–5, 2005.
11. J. Yang, X. Zhou, M. Chrobak, and Y. Zhang. Dynamic thermal management through task scheduling. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2008. To appear.