Maximizing Rewards for Real-Time Applications with Energy Constraints

COSMIN RUSU, RAMI MELHEM, and DANIEL MOSSÉ
University of Pittsburgh

New technologies have brought about a proliferation of embedded systems, which vary from control systems to sensor networks to personal digital assistants. Many of the portable embedded devices run several applications, which typically have three constraints that need to be addressed: *energy*, *deadline*, and *reward*. However, many of these portable devices do not have powerful enough CPUs and batteries to run all applications within their deadlines. An optimal scheme would allow the device to run the most applications, each using the most amount of CPU cycles possible, without depleting the energy source while still meeting all deadlines. In this paper we propose a solution to this problem; to our knowledge, this is the first solution that combines the three constraints mentioned above. We devise two algorithms, an optimal algorithm for homogeneous applications (with respect to power consumption) and a heuristic iterative algorithm that can also accommodate heterogeneous applications (i.e., those with different power consumption functions). We show by simulation that our iterative algorithm is fast and within 1% of the optimal.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms: Algorithms

Additional Key Words and Phrases: Power management, reward-based, scheduling, real-time, operating systems

1. INTRODUCTION

The current developments in embedded technology have been largely responsible for the promotion of mobile, wireless, systems-on-a-chip, and other "computing-in-the-small" devices. Most of these devices have energy constraints, embodied by a battery that has a finite lifetime. Therefore, an essential element of these embedded systems is the way in which power/energy is managed.

This work has been supported by the Defense Advanced Research Projects Agency through the PARTS (Power-Aware Real-Time Systems) project under contract F33615-00-C-1736 and by NSF under grant ANI-0087609.
Authors' address: Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260; email: {rusu,melhem,mosse}@cs.pitt.edu.

In addition to the power management needs, some of these devices execute real-time applications, in which producing timely results is typically as important as producing logically correct outputs. In hard real-time environments, such as nuclear power plants or airborne navigation systems, missing deadlines can be catastrophic. However, there are other application areas in which an approximate but timely result may be acceptable. Examples of such applications areas include multimedia [Rajkumar et al. 1997] and image and speech processing [Chang and Zakhor 1993; Feng and Liu 1993; Turner and Peterson 1992]. These applications typically have a reward function that assigns *value* to the application, as a function of the amount of time the application is allotted. In other words, the more the CPU is used by a certain application, the more value it accrues.

The three constraints mentioned above, namely *energy*, *deadline*, and *reward* play important roles in the current generation of embedded devices. An optimal scheme would allow the device to run the most applications, each using the most amount of CPU cycles possible, without depleting the energy source while still meeting all deadlines. In this paper we propose a solution to this problem, and devise two algorithms, an optimal algorithm for homogeneous applications (with respect to power consumption) and a heuristic algorithm that can also accommodate heterogeneous applications (i.e., those with different power consumption functions).

Note that this problem differs from minimizing power consumption due to the extra constraints considered, namely deadlines and CPU usage. Clearly, minimizing the energy consumption of applications is useful, but does not consider the value/reward characteristics of different applications (e.g., it may be better to run an important application that consumes more energy than two much less important applications that consume almost no energy).

It is important to consider these three constraints simultaneously (reward, energy, and deadlines) since it allows system designers to determine the most important components of their system, or allows them to emphasize a subset of the system over another in a dynamic fashion. An example of such flexibility is when a system designer decides to maximize mission lifetime (e.g., a sensor network that should last as long as possible even if the application results are not as accurate) versus having a fixed mission time within which performance should be maximized (e.g., in the case of a system with a renewable power source that is able to replenish the system batteries). The optimization considered in this paper is mainly meant to be applied at design time. However, the proposed solution is efficient and will cause little overhead even when applied dynamically at run time to cope with changes in the execution environment, the task mix or the available energy profile.

The rest of this paper is organized as follows: We first describe related work. Section 2 explains in detail the task model and defines the problem. In Section 3 we present the properties of the optimal solution. Sections 4 and 5 describe the optimal algorithm for specific power functions and the iterative algorithm for the general case. Section 6 presents experimental results obtained through simulation. In Section 7 we conclude the paper.

1.1 Related Work

The issue of assignment of CPU cycles to different tasks has been studied through scheduling and operations research for decades. In the mid-80s, researchers started considering the trade-off between time and other metrics, such as value/reward [Clark et al. 1992]. In the mid-90s, researchers started considering a similar trade-off, between energy and time [Yao et al. 1995]. Below we describe representative works in these two fields. To the best of our knowledge, this is the first work that addresses a general framework for assignment of time/cycles to tasks when taking into account three types of constraints, namely *energy*, *deadline*, and *reward/value*.

The IC (imprecise computation) [Liu et al. 1991; Shih et al. 1991] and IRIS (increased reward with increased service) [Dey et al. 1996; Krishna and Shin 1997] models were proposed to enhance the resource utilization and provide graceful degradation in real-time systems. In the IC model every real-time task is composed of a mandatory part and an optional part. The mandatory part must be completed before the task's deadline to yield an output of minimal quality. The optional part is to be executed after the mandatory part while still before the deadline. The longer the optional part executes, the better the quality of the result. The algorithms proposed for IC applications concentrate on a model that has an upper bound on the execution time that can be assigned to the optional part, and the aim is usually to minimize the (weighted) sum of errors. Several efficient algorithms have been proposed to solve optimally the scheduling problem of aperiodic tasks [Liu et al. 1991; Shih et al. 1991]. A common assumption in these studies is that the quality of the results produced is a linear function of the precision error; more general error functions are not usually addressed.

An alternative model is the IRIS model with no upper bounds on the execution times of the tasks and no separation between the mandatory and optional parts (i.e., tasks may be allotted no CPU time). Typically, a nondecreasing concave reward function is associated with each task's execution time. In Dey et al. [1993, 1996], the problem of maximizing the total reward in a system of aperiodic tasks was addressed and an optimal solution for static task sets was presented, as well as two extensions that include mandatory parts and policies for dynamic task arrivals. Note that a zero-time mandatory part in the IC model makes it into IRIS model.

Both IC and IRIS focus on linear and concave (logarithmic, for example) functions because they represent most of the real-world applications, such as image and speech processing [Chang and Zakhor 1993; Feng and Liu 1993; Turner and Peterson 1992], multimedia applications [Rajkumar et al. 1997], information gathering [Grass and Zilberstein 2000], and database query processing [Vrbsky and Liu 1993]. We note that some real applications might suggest awarding no reward for partial executions or using step functions, but these cases have been shown in Liu et al. [1991] to be NP-complete. Furthermore, the reward-based scheduling problem for convex reward functions is NP-hard [Aydin et al. 2001c]. Periodic reward-based scheduling for "error-cumulative" (errors have an effect on future instances of the same task) and "error noncumulative" applications

was explored in Chung et al. [1993]. An optimal algorithm assuming concave reward functions and "error noncumulative" applications was presented in Aydin et al. [2001c].

In Rajkumar et al. [1997] a QoS-based resource allocation model (QRAM) was proposed for periodic applications. The reward functions are in terms of utilization of resources, and an iterative algorithm was presented for the case of one resource and multiple QoS dimensions; the QoS dimensions may be either dependent or independent. In Rajkumar et al. [1998], the QRAM work is continued by the authors with the solution for a particular audio-conferencing application with two resources (CPU utilization and network bandwidth) and one QoS dimension (sampling rate). Several resource trade-offs (e.g., compression schemes to reduce network bandwidth while increasing the number of CPU cycles) are also investigated, assuming linear utility and resource consumption functions. Adapting to the QRAM model, the problem analyzed in this work has two resources (CPU utilization and energy) and one QoS dimension (CPU cycles). However, we are interested in the general case of concave utility functions, whereas the work in Rajkumar et al. [1998] is limited to linear utility functions.

The variable voltage-scheduling (VVS) framework, which involves dynamically adjusting the voltage and frequency of the CPU, has recently become a major research area. Cubic power savings [Hong et al. 1998c; Yao et al. 1995] can be achieved at the expense of just linear performance loss. For real-time systems, VVS schemes focus on minimizing energy consumption in the system while still meeting the deadlines. Yao et al. [1995] provided a static off-line scheduling algorithm, assuming aperiodic tasks and worst-case execution times (WCET). Our model extends Yao et al.'s model with different power functions and task rewards at the expense of a more restrictive timing model (only frame-based and periodic task sets are considered in this paper). We are interested in maximizing the system utility rather than minimizing energy.

Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of periodic requests are proposed in Hong et al. [1998b]. Nonpreemptive power aware scheduling is investigated in Hong et al. [1998a]. For periodic tasks with identical periods, the effects of having an upper bound on the voltage change rate are examined in Hong et al. [1998c]. Slowing down the CPU whenever there is a single task eligible for execution was explored in Shin and Choi [1999]. VVS in the context of soft deadlines was investigated in Lorch and Smith [2001]. Cyclic and EDF scheduling of periodic hard real-time tasks on systems with two (discrete) voltage levels have been investigated in Krishna and Lee [2000]. The static solution for the general periodic model where tasks have potentially different power characteristics is provided in Aydin et al. [2001a]. Because real-time applications exhibit a large variation in actual execution times [Ernst and Ye 1997] and WCET is too pessimistic, much research has been directed at dynamic slack-management techniques [Aydin et al. 2001b; Gruian 2001; Mossé et al. 2000; Shin et al. 2001]. Many other VVS papers appeared in recent conferences and workshops, such as COLP or PACS.

1.2 Problem Definition

It was proved in Aydin et al. [2001b] that the problem of minimizing the energy consumption assuming WCET for tasks and convex power functions is equivalent with the problem of maximizing the rewards for concave reward functions assuming all the tasks run at the maximum speed. In this work we address the problem of maximizing the rewards assuming the VVS framework *and* a limited energy budget for frame-based and periodic task sets. Our goal is to execute a task set and maximize the rewards (under the IC model) without exceeding the deadline and the total energy available, which can be provided by an exhaustible source such as a battery. A formal definition of the problem will be presented in Section 2. The proposed algorithm determines the time and speed each task should be allocated.

2. TASK MODEL

We present the task model and problem definition for both periodic and aperiodic (frame-based) task sets and conclude that the problems to be solved are identical.

2.1 Frame-Based Task Sets

The system has to execute a task set $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$. Each task T_i is composed of a mandatory part M_i and an optional part O_i, with computational requirements expressed in terms of number of CPU cycles. The number of cycles is equal to the speed expressed in cycles per second (or the clock rate) multiplied by the execution time, $C_i = s_i \cdot t_i$. The worst-case number of cycles needed for the mandatory part is denoted by l_i, and the total worst-case number of cycles including the optional part is denoted by u_i. In other words, each task must receive a number of cycles between the lower bound l_i and the upper bound u_i.

There are no individual deadlines for the tasks, just a global deadline d, which is normally the length of a frame. The execution of a frame is to be repeated. Each task has to be executed exactly once during a frame, and the task can be executed without preemptions. All mandatory parts of all tasks must be finished before time d. We say that a task set $\mathbf{T}$ is *feasible* if every mandatory part M_i completes before the deadline.

The speed of task T_i, which is denoted by s_i, is physically constrained to be within certain lower and upper bounds $S_{\min}$ and $S_{\max}$. We assume that s_i can take any value in the given range (that is, voltages and frequencies can be changed continuously). The values of s_i and t_i are constrained by the following:

$$l_i \leq s_i \cdot t_i \leq u_i$$

$$S_{\min} \leq s_i \leq S_{\max}$$

$$\sum_{i=1}^{n} t_i \leq d\,.$$

We assume task T_i runs at single speed s_i and for each task T_i there is a corresponding power function P_i, which depends on its speed and on the switching

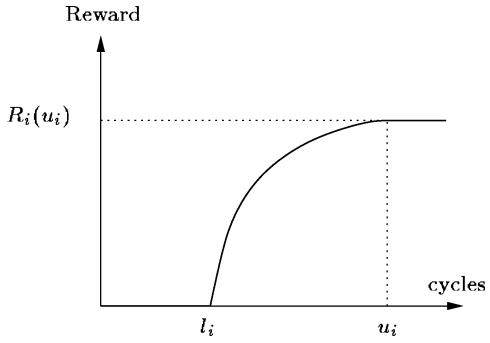


Fig. 1. Typical reward function.

activity of the task. We assume that task power functions can be derived through an offline analysis of the switching activity of tasks. For example, SPEC benchmarks consume between 15 and 35 W on an Alpha 21264 at 600 MHz and between 20 and 26 W on a Pentium Pro at 200 MHz [Joseph and Martonosi 2001] (similar variations exist for embedded processors). We consider the case of positive nondecreasing convex power functions (the second-order derivatives of the power functions P_i are positive). All theoretical results described in the next section are based on the fact that it is more energy efficient to run on reduced speeds, which only holds for convex power functions. The energy consumed by task T_i running for time t_i at speed s_i is $E_i = t_i \cdot P_i(s_i)$. There is a limited energy budget in the system, equally distributed between frames. The total energy consumed during a frame cannot exceed the frame energy budget, denoted by $E_{\max}$:

$$\sum_{i=1}^{n} t_i \cdot P_i(s_i) \leq E_{\max}.$$

The optional part of a task can execute only after its corresponding mandatory part completes. There is a reward function associated with each task; the reward function increases with the number of cycles that were allocated to the optional part. This reward is zero when the service time of the optional part is zero and is increasing with the amount of service time. There is no extra reward if the optional part receives more cycles than required. We consider the case of positive nondecreasing concave reward functions (the second-order derivatives of the reward functions R_i are negative). Note that for convex or partially convex reward functions, even assuming an infinite energy budget, determining the optimal solution that maximizes the total reward is NP-hard [Aydin et al. 2001c]. Our optimal solutions for homogeneous power functions (Section 4) and the iterative algorithm for general power functions (Section 5) are based on the concavity of the reward functions. Figure 1 shows a typical reward function, which is described by

$$R_i(C_i) = \begin{cases} 0 & \text{if } 0 \leq C_i \leq l_i \\ f_i(C_i) & \text{if } l_i \leq C_i \leq u_i \\ f_i(u_i) & \text{if } C_i \geq u_i. \end{cases}$$

The total reward is defined as the sum of all individual rewards. Our goal is to maximize the total reward. Mathematically, for unknowns t_i and s_i we solve the following:

$$\text{maximize} \quad \sum_{i=1}^{n} R_i(s_i \cdot t_i) \tag{1}$$

$$\text{subject to} \quad l_i \le s_i \cdot t_i \le u_i \tag{2}$$

$$S_{\min} \le s_i \le S_{\max} \tag{3}$$

$$\sum_{i=1}^{n} t_i \le d \tag{4}$$

$$\sum_{i=1}^{n} t_i \cdot P_i(s_i) \le E_{\max}. \tag{5}$$

2.2 Periodic Task Sets

Let $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ be a periodic task set. Similarly to aperiodic tasks in Section 2.1, each task T_i is specified by its lower and upper bounds l_i and u_i, the power function P_i and the reward function R_i. In addition, each task has a deadline (or period, or arrival rate) d_i, whereas in the frame-based model there was only a global period or deadline d. Let LCM be the least common multiple of all tasks' periods. If the energy budget $E_{\max}$ cannot be exceeded in time LCM, the problem is similar to Section 2.1: maximize the reward of the task set $\mathbf{T}$ without exceeding the available energy $E_{\max}$ in LCM time units.

Every LCM time units, each task T_i needs to execute $N_i = \frac{LCM}{d_i}$ instances. We denote the jth instance of task T_i by T_{ij}. Similarly, we denote the speed and execution time of T_{ij} by s_{ij} and t_{ij}, respectively. The total reward acquired by the task set is $\sum_{i=1}^{n} \sum_{j=1}^{N_i} R_i(t_{ij} \cdot s_{ij})$, and the total energy consumed is $\sum_{i=1}^{n} \sum_{j=1}^{N_i} t_{ij} \cdot P_i(s_{ij})$ (where P_i is defined as in the previous section).

We prove in the Appendix that in the optimal solution all instances of a task run at the same speed and for the same amount of time, that is, $s_{ij} = s_i$ and $t_{ij} = t_i$ for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, N_i$. Using this result, we present next the problem definition for periodic task sets.

Let $t_i' = t_i \cdot N_i$, that is, t_i' is the total execution time allocated to all the instances of task T_i each LCM time units. Define also $l_i' = l_i \cdot N_i$ and $u_i' = u_i \cdot N_i$. The total reward acquired by task T_i in LCM time units is $R_i'(s_i \cdot t_i') = N_i \cdot R_i(s_i \cdot t_i)$.

We define the utilization of a periodic task T_i as $U_i = \frac{t_i}{d_i}$, where t_i is the execution time of task T_i and d_i is the task period. Assuming EDF scheduling or any preemptive scheduling technique that can fully utilize the CPU, the schedulability condition for periodic task sets is $\sum_{i=1}^{n} U_i \le 1$, as proved in Liu and Layland [1973], if context switch overhead is assumed to be insignificant. Note that this holds true for frame-based task sets as well, the utilization of a task in this case being defined as $U_i = \frac{t_i}{d}$, where d is the frame size. Equations

(1)–(5) for the periodic case become:

$$\text{maximize} \quad \sum_{i=1}^{n} R_i'(s_i \cdot t_i') \tag{6}$$

$$\text{subject to} \quad l_i' \le s_i \cdot t_i' \le u_i' \tag{7}$$

$$S_{\min} \le s_i \le S_{\max} \tag{8}$$

$$\sum_{i=1}^{n} t_i' \le LCM \tag{9}$$

$$\sum_{i=1}^{n} t_i' \cdot P_i(s_i) \le E_{\max}. \tag{10}$$

Note that Equation (9) implies that $\sum_{i=1}^{n} U_i \le 1$ (as shown in the Appendix). Clearly, this is the same problem definition as for the frame-based task set. Observe that, due to the property that all instances run at the same speed and for the same amount of time, the number of task instances actually occurring in LCM time units does not increase the complexity of the problem, which is controlled only by the number of tasks.

Solving the above problem described by Equations (6)–(10) gives the speed s_i and the total execution time t_i' each task T_i receives during LCM time units. The execution time of each instance of T_i is given by $t_i = \frac{t_i'}{N_i}$.

Note that our goal is to maximize the reward without exceeding an energy budget. The problem of minimizing the energy without taking the reward into account is solved in Aydin et al. [2001a]. Moreover, if minimizing the energy is also considered as a reward, then it is possible to formulate the objective function to be maximized in Equation (6) to depend on both $R_i'(s_i \cdot t_i')$ and $t_i' \cdot P_i(s_i)$. That is, the function R_i' in Equation (6) will be replaced by another function R_i'' that depends on s_i and t_i', while the constraints (7)–(10) remain unchanged.

In the rest of the paper we present the solution for frame-based task sets as described by Equations (1)–(5). Clearly, all results apply to periodic task sets as well.

3. PROPERTIES OF THE OPTIMAL SOLUTION

Before we start presenting the theoretical results, we introduce the nomenclature used in this paper. We define a *solution of (1)–(5)* to be a set of values $\mathbf{S}=\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$, that satisfy the constraints (2)–(5) and also maximize the reward as described by (1). We denote the value of this maximum reward by "optimal reward." Because there may be more than one solution that maximizes the reward, we define the *minimum-energy solution* to be a solution consuming the least amount of energy. Further, we denote by R_{ub} the maximum achievable reward (i.e., $R_{ub} = \sum_{i=1}^{n} R_i(u_i)$), which may be different from the optimal reward.

LEMMA 3.1. *If the optimal reward is less than R_{ub}, then in any solution of (1)–(5), either the entire available slack is used (i.e., the processor is fully utilized) or all the tasks run at the minimum speed.*

PROOF. By available slack we mean the CPU idle time. Using all the available slack in a schedule is equivalent to not having the CPU idle, that is, at any instance of time the processor is executing some task.

Assume that there exists a solution **S** in which a task T_a runs on some speed $s_a > S_{\min}$ and the entire available slack is not used. Since the optimal reward is less than R_{ub}, there also exists a task T_b that does not receive u_b cycles. We will show that a higher reward can be achieved for the same amount of energy, thus contradicting the fact that **S** is a solution of (1)–(5).

Using the available slack, the speed of a task T_k running at $s_k > S_{\min}$ can be reduced to some $s'_k < s_k$ while still providing the same number of cycles to the task. If $s_b > S_{\min}$, we pick task $k = b$ to reduce the speed. If $s_b = S_{\min}$, we pick task $k = a$. If the slack allows it, then we can reduce s_k to $s'_k = S_{\min}$, otherwise $S_{\min} < s'_k < s_k$ and the processor is fully utilized. Either way, the total reward is not changed as we keep the number of cycles given to task T_k constant. However, due to the convex nature of the power functions, the total energy is reduced.

Next, the saved energy can be used to increase the speed of T_b while keeping t_b constant. Note that it is always possible to increase s_b, as s_b is known to be less than $S_{\max}$ by the way we picked T_k in the previous step. By keeping t_b constant and increasing s_b, the number of cycles allocated to task T_b will increase. Thus, the total reward increases, contradicting the maximum-reward condition. □

LEMMA 3.2. *If the optimal reward is less than R_{ub}, then in any solution of (1)–(5), either the entire energy is used or all the tasks run at the maximum speed.*

PROOF. Assume that there exists a solution **S** in which a task T_a runs on some speed $s_a < S_{\max}$ and the entire available energy is not used. Also, since the optimal reward is less than R_{ub}, there exists a task T_b that does not receive u_b cycles. We will show how the extra energy can be used to increase the total reward, thus contradicting the fact that **S** is a solution of (1)–(5).

If $s_b < S_{\max}$, we can use the extra energy to increase the speed s_b while keeping t_b unchanged and thus the total reward is increased. If $s_b = S_{\max}$, we will improve the total reward as follows: first, using the available energy, the speed of T_a can be increased to some $s'_a > s_a$ while still providing the same number of cycles C_a to the task. If the available energy allows it we can have $s'_a = S_{\max}$, otherwise $S_{\max} > s'_a > s_a$ and the entire energy is used. This will create some slack that can be used to slow down T_b while keeping C_b constant. Next, the energy saved by slowing down T_b can be used to increase the speed s_b and thus the total reward. Either way, the total reward is improved, contradicting the maximum-reward condition. □

Theorem 3.3 combines the results of the above two lemmas, showing that if the reward is less than R_{ub}, which is to be expected for most task sets, a solution *must* use the most amount of slack possible (unless all tasks are already running at $S_{\min}$) and the most amount of energy possible (unless all tasks are running at $S_{\max}$), as shown in Figure 2.

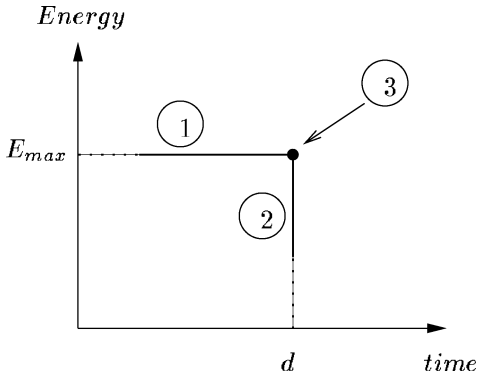


Fig. 2. Solutions of Theorem 3.3 are on the boundary of the design space. ① All tasks run at $S_{\min}$; all energy is used, some slack may exist. ② All tasks run at $S_{\max}$; there is no slack and some energy may be wasted. ③ $\exists T_i$, $S_{\min} < s_i < S_{\max}$; all slack and energy are used.

Theorem 3.4 extends Theorem 3.3, removing the restriction on the reward, and showing that there always exists at least one solution that uses the most amount of slack and energy possible. We will use this property in finding the optimal solution to (1)–(5) for specific power functions and also in the iterative algorithm for general power functions.

THEOREM 3.3. *If the optimal reward is less than R_{ub} then all solutions have one of the following properties: either (a) all the available slack and energy are used or (b) all the tasks are running at the same speed $S_{\min}$ or (c) all the tasks are running at the same speed $S_{\max}$.*

PROOF. If the optimal reward is less than R_{ub}, it follows from the above lemmas that there are only three possible types of solutions:

(1) All tasks run at the minimum speed, in which case the entire available energy is used and there may be some slack in the final schedule.
(2) All tasks run at the maximum speed, in which case the entire available slack is used but some energy may be wasted in the final schedule.
(3) There is at least one task T_i running at speed $S_{\min} < s_i < S_{\max}$. In this case, it follows from Lemmas 3.1 and 3.2 that the entire slack and all the available energy are used. □

THEOREM 3.4. *If the optimal reward is equal to R_{ub}, there exists a solution with one of the following properties: either (a) all the available slack and energy are used or (b) all the tasks are running at the same speed $S_{\min}$ or (c) all tasks are running at the same speed $S_{\max}$.*

PROOF. If the optimal reward is *exactly* R_{ub}, any solution can be transformed into a solution in which either all the available slack and energy are used or all the tasks run at the same speed $S_{\min}$ or $S_{\max}$, as described next.

If there is some available slack and a task T_i such that $s_i > S_{\min}$, we transform the solution by slowing down T_i to use as much as possible of the available slack while keeping C_i constant. The transformation is applied repeatedly

until either all tasks run at the minimum speed or all the available slack is used.

Next, if there is some unused energy and a task T_j such that $s_j < S_{\max}$, we transform the solution by speeding up T_j to use as much as possible of the extra energy while keeping t_j constant. The transformation is applied repeatedly until either all tasks run at maximum speed or all the available energy is used. □

The next theorem and corollaries present properties of the minimum-energy solutions that are used in Section 4 to identify the minimum-energy solution for specific power functions. Note that an implication of Theorem 3.3 is that if the optimal reward is less than R_{ub} then all solutions of (1)–(5) are also minimum-energy solutions. Theorem 3.5 shows how to compute the speed of all tasks if one task's speed (different from $S_{\min}$ and $S_{\max}$) is known. Knowing s_i, where $S_{\min} < s_i < S_{\max}$, the speed of all tasks T_j, $j \neq i$ can be determined by solving the equation $P_i(s_i) - s_i \cdot P_i'(s_i) = P_j(s_j) - s_j \cdot P_j'(s_j)$, where P_i' and P_j' denote the first-order derivatives of the power functions P_i and P_j, respectively. If the computed speed is less than $S_{\min}$, the solution is $s_j = S_{\min}$. Similarly, if $s_j > S_{\max}$, the solution is $s_j = S_{\max}$.

THEOREM 3.5. *All minimum-energy solutions of (1)–(5) have the following properties:*

(1) *If T_i and T_j are two tasks in the minimum-energy solution such that $S_{\min} < s_i < S_{\max}$ and $S_{\min} < s_j < S_{\max}$, then $P_i(s_i) - s_i \cdot P_i'(s_i) = P_j(s_j) - s_j \cdot P_j'(s_j)$.*
(2) *If $s_i = S_{\min}$ and $S_{\min} < s_j < S_{\max}$, then $P_i(s_i) - s_i \cdot P_i'(s_i) \leq P_j(s_j) - s_j \cdot P_j'(s_j)$.*
(3) *If $s_i = S_{\max}$ and $S_{\min} < s_j < S_{\max}$, then $P_i(s_i) - s_i \cdot P_i'(s_i) \geq P_j(s_j) - s_j \cdot P_j'(s_j)$.*

where P_i' and P_j' denote the first-order derivatives of the power functions P_i and P_j, respectively.

PROOF. Let **S** be a minimum-energy solution of (1)–(5) in which each task T_i receives C_i cycles. Since **S** is a minimum-energy solution, there is no other solution in which each task T_i receives exactly C_i cycles for less energy. Thus, **S** has to also be a solution of the following optimization problem:

$$\text{minimize} \quad \sum_{i=1}^{n} t_i \cdot P_i(s_i) \tag{11}$$

$$\text{subject to} \quad \sum_{i=n}^{n} t_i = d \tag{12}$$

$$t_i \cdot s_i = C_i \tag{13}$$

$$S_{\min} \leq s_i \leq S_{\max}. \tag{14}$$

Using Lagrangian multipliers method and Kuhn–Tucker conditions [Luenberger 1984], any solution of (11)–(14) must satisfy the following:

$$P_i(s_i) + \lambda_0 + \lambda_i \cdot s_i = 0 \tag{15}$$

$$t_i \cdot P_i'(s_i) + \lambda_i \cdot t_i - \mu_i^1 + \mu_i^2 = 0 \tag{16}$$

$$\mu_i^1 \cdot (S_{\min} - s_i) = 0 \tag{17}$$

$$\mu_i^2 \cdot (s_i - S_{\max}) = 0 \tag{18}$$

where λ_0, λ_i, $\mu_i^1 \geq 0$, $\mu_i^2 \geq 0$, $i = 1, 2, \ldots, n$ are constants.

The three properties in the theorem are proved next.

(1) Assume that T_i and T_j are two tasks in $\mathbf{S}$ such that $S_{\min} < s_i < S_{\max}$ and $S_{\min} < s_j < S_{\max}$. Then, from (17) and (18) it follows that $\mu_i^1 = \mu_i^2 = \mu_j^1 = \mu_j^2 = 0$. From (16) it results that $\lambda_i = -P_i'(s_i)$ and $\lambda_j = -P_j'(s_j)$. By substitution in (15), $P_i(s_i) - s_i \cdot P_i'(s_i) = P_j(s_j) - s_j \cdot P_j'(s_j)$.

(2) Assume now that $s_i = S_{\min}$ and $S_{\min} < s_j < S_{\max}$. Then, $\mu_i^2 = \mu_j^1 = \mu_j^2 = 0$. Since $\mu_i^1 \geq 0$, it results from (16) that $\lambda_i \geq -P_i'(s_i)$ and $\lambda_j = -P_j'(s_j)$. By substitution in (15), $P_i(s_i) - s_i \cdot P_i'(s_i) \leq P_j(s_j) - s_j \cdot P_j'(s_j)$.

(3) Similarly, if $s_i = S_{\max}$ and $S_{\min} < s_j < S_{\max}$, then $\mu_i^1 = \mu_j^1 = \mu_j^2 = 0$ and since $\mu_i^2 \geq 0$ it follows that $P_i(s_i) - s_i \cdot P_i'(s_i) \geq P_j(s_j) - s_j \cdot P_j'(s_j)$. □

COROLLARY 3.6. *If all tasks have identical power functions, then all tasks run at the same speed in the minimum-energy solution.*

PROOF. Let $P(s)$ be the convex power function of all tasks. We first show that the function $P(s) - s \cdot P'(s)$ is strictly decreasing. Using this fact, we will prove by contradiction that all tasks must run at the same speed.

By derivation, we obtain $(P(s) - s \cdot P'(s))' = -s \cdot P''(s)$, where $P''(s)$ is the second-order derivative of the power function $P(s)$. By the convexity of $P(s)$, $P''(s) > 0$ and it follows that $(P(s) - s \cdot P'(s))'$ is negative. Thus, $P(s) - s \cdot P'(s)$ is strictly decreasing.

Assume now that in the minimum-energy solution there are two tasks T_i and T_j such that $S_{\min} \leq s_i < s_j \leq S_{\max}$. Then, $P(s_i) - s_i \cdot P'(s_i) > P(s_j) - s_j \cdot P'(s_j)$. By Theorem 3.5, this can be true only if $s_i = S_{\max}$ or $s_j = S_{\min}$, contradicting the assumption that $s_i < s_j$. □

Corollary 3.7 shows that for power functions of the type $P_i = \alpha_i \cdot s^q$, all tasks run at the same power in the minimum-energy solution. The exceptions are the tasks running at $S_{\min}$, which have high-power coefficients α_i and the tasks running at $S_{\max}$, which have low-power coefficients α_i.

COROLLARY 3.7. *For power functions of the type $P_i = \alpha_i \cdot s^q$, where q is constant for all tasks, the minimum-energy solutions have the following properties:*

(1) *All tasks that do not run at minimum or maximum speed consume the same amount of power $\boldsymbol{P}$.*

(2) *All tasks running at $S_{\min}$ have a power consumption higher than $\boldsymbol{P}$.*

(3) *All tasks running at $S_{\max}$ have a power consumption less than $\boldsymbol{P}$.*

PROOF. The proof follows directly by substituting $P_i = \alpha_i \cdot s^q$ in Theorem 3.5. $P_i(s_i) - s_i \cdot P_i'(s_i) = P_j(s_j) - s_j \cdot P_j'(s_j)$ becomes $\alpha_i \cdot s_i^q - q \cdot \alpha_i \cdot s_i^q = \alpha_j \cdot s_j^q - q \cdot \alpha_j \cdot s_j^q$ and from here $P_i(s_i) = P_j(s_j)$. □

It can be proved in a similar way that if q can take n distinct values $q_1, q_2, \ldots, q_n$, then in the minimum-energy solutions all tasks not running at $S_{\min}$ or $S_{\max}$ can only run at one of the n power levels $P_1, P_2, \ldots, P_n$ (depending on the power exponent), all tasks running at $S_{\min}$ have a power consumption higher than $\max(P_1, P_2, \ldots, P_n)$, and all tasks running at $S_{\max}$ consume less than $\min(P_1, P_2, \ldots, P_n)$.

4. OPTIMAL SOLUTIONS FOR SPECIFIC POWER FUNCTIONS

In Aydin et al. [2001c] a polynomial time algorithm for optimally solving the following problem was given

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} f_i(t_i) \\ \text{subject to} \quad & l_i \le t_i \le u_i \\ & \sum_{i=1}^{n} t_i = d\,. \end{aligned}$$

Also, the algorithm that solves this problem, called OPT-LU, assumes that functions f_i are concave (or linear). The complexity of OPT-LU is $O(n^2 \cdot \log n)$. We will use the OPT-LU technique as part of our solution, as follows: we replace the variables s_i with constants and eliminate Equation (5) from our model. This will reduce the problem to a form that the OPT-LU algorithm can solve. We always look for solutions that consume all the available energy and slack, as seen in Theorem 3.4. The cases when all tasks run at the same speed $S_{\min}$ or $S_{\max}$ are treated as special cases.

4.1 Optimal Solution for Identical Power Functions

Assuming identical power functions $P_i = P$, by virtue of Corollary 3.6 the total energy is minimized when all tasks run at the same speed s. In order to find the speed s that consumes all the energy in the given deadline we solve $P(s) = \frac{E_{\max}}{d}$. Because of the speed bounds there are three possible cases:

(1) $S_{\min} \le s \le S_{\max}$. In this case s is the optimal speed.
(2) $s > S_{\max}$. In this case there is plenty of energy to run all the tasks at the maximum speed. s is set to $S_{\max}$ and some of the energy has to be wasted.
(3) $s < S_{\min}$. The speed s is set to $S_{\min}$. Observe that the entire slack cannot be used as the available energy would be exceeded even if all tasks run at $S_{\min}$. Thus, the deadline is artificially reduced: $d' = \frac{E_{\max}}{P(S_{\min})}$.

After determining the speeds $s_i = s$ (and possibly adjusting the deadline), Equation (5) can safely be eliminated from the model, as the energy budget cannot be exceeded. Next, OPT-LU gives a solution if there is one.

Note that if the optimal reward is exactly R_{ub} the solution returned by OPT-LU is not necessarily the minimum-energy solution if $\sum_{i=1}^{n} t_i < d$. To compute the minimum-energy solution in this case, proceed as follows: let s be the common speed of all tasks and d be the deadline (both computed as described

above before solving OPT-LU). Also, let t_i be the execution time of task T_i as computed by OPT-LU. Then, in the minimum-energy solution each task runs at speed $s' = \max(s \cdot \sum_{i=1}^{n} t_i/d, S_{\min})$ for $t_i' = \frac{s \cdot t_i}{s'}$ units of time.

4.2 Optimal Solution for Power Functions of the Type $P_i = \alpha_i \cdot s_i^q$ with No Speed Bounds

Since there are no speed bounds ($S_{\min} = 0$, $S_{\max} = \infty$), it follows from Corollary 3.7 that in the minimum-energy solution all tasks run at the same power P. Thus, the CPU power that fully utilizes the available energy in the given deadline is $P = \frac{E_{\max}}{d}$.

Next, the speeds s_i of all tasks T_i are determined from the equations $P_i(s_i) = P$. Having determined all speeds, Equation (5) is eliminated from the model, and the problem is transformed into an OPT-LU problem that returns a solution if there is one.

The solution returned is also minimum energy if the total reward is less than R_{ub}. If the reward is exactly R_{ub}, then, analogous to Section 4.1, the minimum-energy speed and execution time for task T_i are $s_i' = s_i \cdot \sum_{i=1}^{n} t_i/d$ and $t_i' = t_i \cdot d / \sum_{i=1}^{n} t_i$, respectively.

5. ITERATIVE ALGORITHM FOR GENERAL POWER FUNCTIONS

The algorithm in Section 4 cannot be easily extended for the case of general power functions because it is not possible to determine the speeds of tasks in polynomial time. The algorithm we propose in this section starts by solving the problem without energy constraints. Obviously, the optimal reward assuming limited energy $E_{\max}$ is bounded by the OPT-LU solution assuming all the tasks are running at speed $S_{\max}$. Having determined the execution times t_i for all tasks, we compute the total energy consumed by the formula $E = \sum_{i=1}^{n} t_i \cdot P_i(S_{\max})$. If $E \leq E_{\max}$ is true, the problem is solved. If not, we iteratively modify the unlimited-energy solution to satisfy the energy constraint.

Based on Theorems 3.3 and 3.4 we look for solutions that use all the available energy and slack. Thus, if the entire slack available is not used in the OPT-LU solution (i.e., $\sum_{i=1}^{n} t_i < d$), we artificially extend one task so that the entire slack available is used (for algorithm purposes).

Then, we iteratively transfer Δt units of time from one task T_i to another task $T_j, i \neq j$, by keeping the speed of T_i the same and reducing the speed of T_j so as to preserve the same number of cycles C_j. We refer to this type of transfer as *intertask* transfer. Specifically, by reducing t_i by Δt and keeping s_i unchanged, the number of cycles executed in T_i is reduced and thus the reward is reduced by $\Delta R_i = R_i(s_i \cdot t_i) - R_i(s_i \cdot (t_i - \Delta t))$. By increasing t_j by Δt and keeping C_j constant, the speed s_j becomes $s_j \cdot \frac{t_j}{t_j + \Delta t}$ and the total energy is reduced by $\Delta E_{ij} = \Delta_1 E_i + \Delta_2 E_j$, where $\Delta_1 E_i = \Delta t \cdot P_i(s_i)$ and $\Delta_2 E_j = t_j \cdot P_j(s_j) - (t_j + \Delta t) \cdot P_j(\frac{s_j \cdot t_j}{t_j + \Delta t})$.

We need to determine the tasks T_i and T_j to carry out the intertask transfer of Δt units of time. Since we want to maximize the energy saved while minimizing

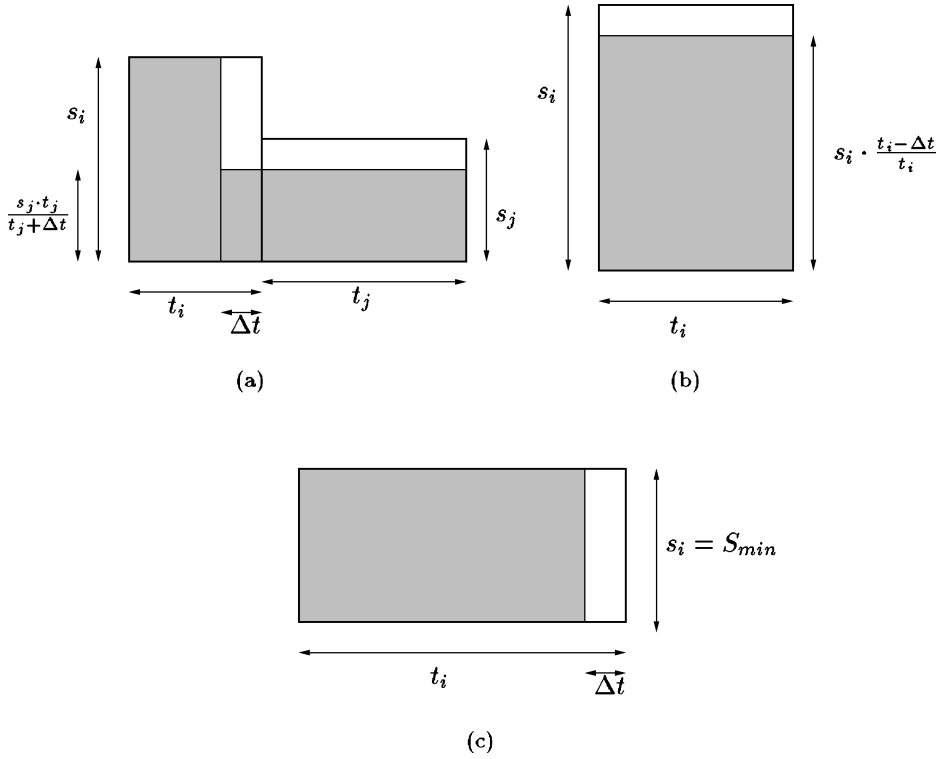


Fig. 3. (a) Intertask transfer, (b) intratask transfer and (c) reduction.

the loss in reward, we choose $\frac{\Delta_1 E_i + \Delta_2 E_j}{\Delta R_i}$ as our metric for deciding the best transfer. Observe that the best intertask transfer from T_i to T_j, $i \neq j$ can be computed in linear time by first determining the task T_j that maximizes $\Delta_2 E_j$. Task T_j is the task that exhibits the highest energy reduction when increasing its execution time by Δt and reducing its speed from s_j to $s_j \cdot \frac{t_j}{t_j + \Delta t}$. Thus, intertask transfers from task T_i to task T_k, where $k \neq j$ and $i \neq j$ need not be analyzed.

Intertask transfers are illustrated in Figure 3(a). The rectangles represent tasks expressed by speed (the height of the rectangle) and execution time (the width). The bold rectangles show the tasks before the transfer, and the shadowed rectangles show the tasks after the transfer.

Similarly, we introduce the *intratask* transfer, in which a task T_i can transfer Δt time units to itself by reducing its speed to $s_i \cdot \frac{t_i - \Delta t}{t_i}$, as shown in Figure 3(b). The reward is thus reduced by the same amount $\Delta R_i = R_i(s_i \cdot t_i) - R_i(s_i \cdot (t_i - \Delta t))$, and the energy is reduced by $\Delta E_{ii} = t_i \cdot [P_i(s_i) - P_i(s_i \cdot \frac{t_i - \Delta t}{t_i})]$. (Note that one cannot substitute $j = i$ in the formula for $\Delta E_{ij}, i \neq j$ to get the formula for ΔE_{ii}.) The intratask transfer does not change the execution time of task T_i or any other task. Specifically, the amount of computation (the number of cycles) in task T_i is reduced, which lowers the reward, in order to allow the task to execute at a lower speed without increasing the time allocated to task T_i. The

```
1 while E( S) > E_max do
  1.1 determine transferring task T_i and receiving task T_j
  1.2 if T_i and T_j exist, update S, E( S) and go to step 1
  1.3 while E( S) > E_max do
    1.3.1 determine task T_i from which to take Δt time units
    1.3.2 if such task exists, update S, E( S) and go to step 1.3
    1.3.3 return failure
2 return S={(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)}
```

Fig. 4. SOLVE(Δt, $\mathbf{S}$) algorithm.

result is a decrease in the energy consumption at the expense of a decrease in the reward.

Among all possible combinations of T_i and T_j, the transfer (either intertask or intratask) with the highest ratio $\Delta E_{ij}/\Delta R_i$ is chosen. The transfers stop when the total energy E of the current schedule satisfies $E \leq E_{\max}$.

If no transfer is possible and $E > E_{\max}$, it is the case that all tasks run on the minimum speed and the entire available slack is not used. For this situation a *reduction* is introduced: find the task T_i for which to reduce the execution time by Δt so that $\frac{\Delta E_i}{\Delta R_i}$ is maximized. Reductions are illustrated in Figure 3(c). If no reduction is possible (i.e., all tasks are already allotted their minimum number of cycles) and $E > E_{\max}$ the algorithm returns failure.

The algorithm that carries out the transfers and reductions is presented in Figure 4. The input parameters are the value of the interval Δt and the unlimited energy solution $\mathbf{S} = \{(S_{\max}, t_1), (S_{\max}, t_2), \ldots, (S_{\max}, t_n)\}$ (S is obtained by substituting $s_i = S_{\max}$ in (1)–(4) and solving the resulting OPT-LU problem). The algorithm computes the speed and execution time (s_i, t_i) for each task and returns the solution in the same form $\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$. $E(\mathbf{S})$ denotes the total energy of an intermediate solution $\mathbf{S}$ and is computed by the formula $E(\mathbf{S}) = \sum_{i=1}^{n} t_i \cdot P_i(s_i)$. Intertask and intratask transfers are in lines 1.1–1.2, reductions are in lines 1.3.1–1.3.2. Note that most tasks will already be running at $S_{\min}$ when we get to 1.3.2, therefore we choose to go to 1.3 instead of 1.1.

Clearly, the value of Δt will directly influence the quality and the running time of the solution. When Δt tends to zero, the solution tends to the optimum and the running time increases. However, as the experiments will show, the error obtained depends on other factors such as the amount of energy available. The Δt size that gives an acceptable solution cannot be determined a priori. Therefore, we propose an algorithm that starts with an initial Δt and iteratively refines the solution by halving Δt until a stopping criterion suggests that the quality of the solution is acceptable.

The stopping criterion we use for the quality of the solution is $\frac{R(S_{\Delta t}) - R(S_{2\cdot\Delta t})}{R(S)} \leq \varepsilon$, where $S_{\Delta t}$ is the solution obtained for the current Δt value, $S_{2\cdot\Delta t}$ is the solution obtained in the previous iteration, $R(\mathbf{S})$ denotes the reward of solution $\mathbf{S}$, and ε is a given threshold. Another stopping criterion, $\Delta t_{\min}$, is also used as a threshold for Δt, to prevent searching indefinitely for a solution. The algorithm is presented in Figure 5. In the next section we evaluate the algorithm through simulations.

1 assuming $S_i = S_{\max}$ for all $i = 1, 2, \ldots, n$, transform the problem into an OPT-LU problem. OPT-LU returns the execution times t_i for all tasks. If OPT-LU gives no solution, return failure

2 let $\mathbf{S}$=$\{(S_{\max}, t_1), (S_{\max}, t_2), \ldots, (S_{\max}, t_n)\}$, where t_i is the execution time of task T_i as returned by OPT-LU. Compute the total energy of the solution $\mathbf{S}$ by $E(\mathbf{S}) = \sum_{i=1}^{n} t_i \cdot P_i(S_{\max})$. If $E(\mathbf{S}) \leq E_{\max}$ return solution $\mathbf{S}$

3 if $\sum_{i=1}^{n} t_i < d$, extend one task so that the entire slack is used: $t_1 = d - \sum_{i=2}^{n} t_i$, update $\mathbf{S}$ and $E(\mathbf{S})$

4 initialize $\Delta t = \Delta t_{initial}$ and $R(S_{\Delta t}) = 0$

5 $R(S_{2 \cdot \Delta t}) = R(S_{\Delta t})$

6 if $\Delta t < \Delta t_{\min}$ return failure

7 $S_{\Delta t}$ = SOLVE(Δt, $\mathbf{S}$) (Figure 4)

8 if SOLVE(Δt, $\mathbf{S}$) returned failure, $R(S_{\Delta t}) = 0$, else $R(S_{\Delta t}) = \sum_{i=1}^{n} R_i(t_i \cdot s_i)$

9 if $R(S_{\Delta t}) = 0$ or $\frac{R(S_{\Delta t}) - R(S_{2 \cdot \Delta t})}{R(S_{\Delta t})} > \varepsilon$ then $\Delta t = \frac{\Delta t}{2}$, go to step 5

10 return $S_{\Delta t}$=$\{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$

Fig. 5. Variable Δt algorithm.

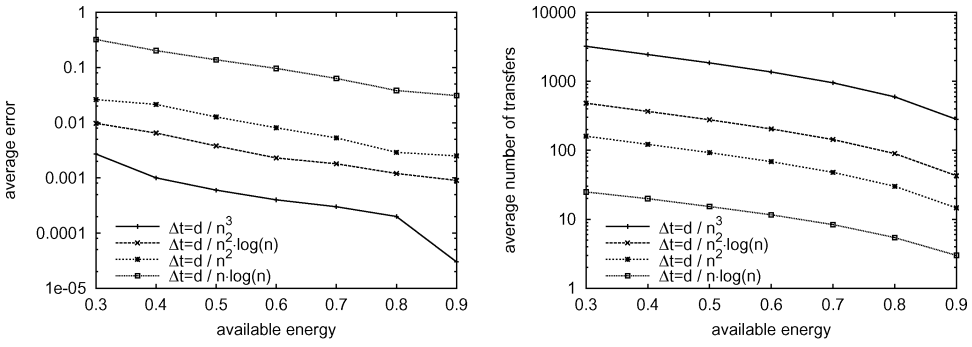
6. EXPERIMENTAL RESULTS AND VALIDATION

We simulated task sets with $n = 20$ tasks and identical power functions using the simpler version of our algorithm that uses a fixed Δt. The common power function we use for this experiment is $P = s_i^q$, where for each simulation run q was set to 2 or 3 (square and cubic power functions). We use linear reward functions of the type $R_i(t_i \cdot s_i) = \beta_i \cdot t_i \cdot s_i$, where the coefficients β_i were randomly chosen such that $\beta_i \in [0, n]$. The lower computation bounds l_i and u_i were randomly generated so that $l_i \in [1, n]$ and $u_i \in [l_i, l_i + n]$. The deadline d was generated in the range $[\sum_{i=1}^{n} \frac{l_i}{S_{\max}}, \sum_{i=1}^{n} \frac{u_i}{S_{\max}}]$. After solving the energy unrestricted OPT-LU, $E_{\max}$ was chosen so that it does not exceed the energy of the OPT-LU solution: $E_{\max} \in [\frac{1}{5} E_{\text{OPT-LU}}, E_{\text{OPT-LU}}]$. For the speed limits we used the normalized speeds $S_{\max} = 1$ and $S_{\min} = 0.5$. Similar results were obtained with narrower ranges for β_i, l_i, and u_i, as well as $S_{\min} < 0.5$. We simulated 1000 task sets for each point in the graphs.

The error of a solution $\mathbf{S}$ is defined as $e = \frac{R_{\text{opt}} - R(\mathbf{S})}{R_{\text{opt}}}$, where R_{opt} is the total reward of the optimal solution and $R(\mathbf{S})$ is the total reward of solution $\mathbf{S}$. Note that R_{opt} can always be computed for identical power functions, as shown in Section 4.1.

Figure 6 shows on a logarithmic scale the effect of the size of Δt and of the amount of available energy on the error of the solution and the number of steps (transfers) required to reach the solution, for a fixed Δt. We considered the following values for the size of Δt: $\Delta t = \frac{d}{n \cdot \log n}$, $\Delta t = \frac{d}{n^2}$, $\Delta t = \frac{d}{n^2 \cdot \log n}$, and $\Delta t = \frac{d}{n^3}$. The available energy is expressed as a percentage of $E_{\text{OPT-LU}}$, the energy consumed in the OPT-LU solution assuming $s_i = S_{\max}$.

Across all simulations, $\Delta t = \frac{d}{n^2 \cdot \log n}$ gives less than 1% absolute error and a relatively small number of transfers for identical power functions. However, if a better accuracy is desired or if the energy available is relatively small, $\Delta t = \frac{d}{n^2 \cdot \log n}$ might not be enough. On the other hand, for most task sets a higher Δt (fewer transfers) can achieve a satisfactory solution. The second algorithm (variable Δt size) was proposed to handle these problems.



(a) (b)

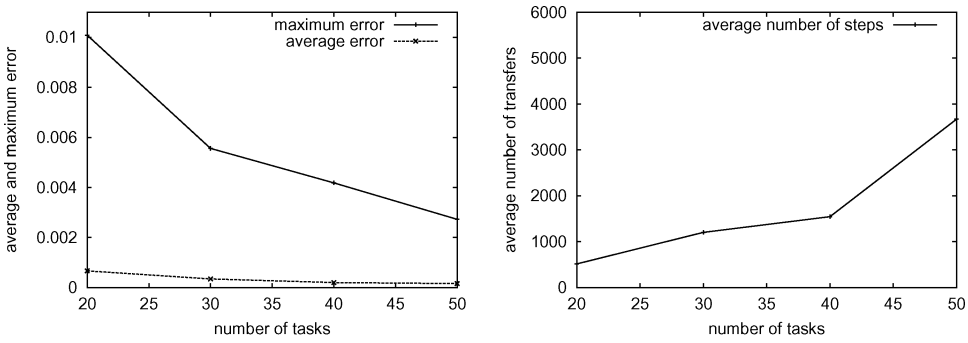
Fig. 6. Fixed Δt, identical power functions $P = \alpha \cdot s_i^q$: (a) average error and (b) average number of transfers, as a function of available energy.

We used the variable Δt version of our algorithm and compared the solution we obtained with the optimal solution. For this experiment we use different power functions of the type $P_i = \alpha_i \cdot s_i^q$, where q is constant for all tasks (for identical power functions we obtained quite similar results). As shown in Section 4.2, the optimal solution can always be computed for power functions of the type $P_i = \alpha_i \cdot s_i^q$ if there are no speed bounds. Thus, the power coefficients were generated so that in the optimal solution no task runs on the minimum or maximum speed. We ensured this by first generating the speeds s_i in the range $[S_{\min}, S_{\max}] = [0.5, 1]$ and the CPU power P at which all tasks run in the optimal solution. Only after that the coefficients α_i were generated such that $\alpha_i \cdot s_i^q = P$. We use concave reward functions of the type $R_i(C_i) = \ln(\beta_i \cdot C_i + 1)$ with random coefficients $\beta_i \in [0, n]$, where n is the number of tasks. As before, for each simulation $E_{\max}$ was randomly generated in the range $[\frac{1}{5}E_{\text{OPT-LU}}, E_{\text{OPT-LU}}]$. A large variation in power and reward coefficients was preferred in order to enforce a large variation in task speeds and execution times in the optimal solution. Even with such variation, simulations show that the algorithm converges to the optimal solution. Very similar results are obtained for narrower ranges for the coefficients (such as $\alpha_i \in [1, 2]$ and $\beta_i \in [1, 5]$).

Figure 7 shows the average and maximum error, as well as the average number of transfers for task sets of 20, 30, 40, and 50 tasks. For each result 1000 task sets were generated. For illustration purposes, the initial value for Δt was set to $\Delta t = \frac{d}{n^2}$ and for the stopping criterion we used $\varepsilon = 0.005$.

Clearly, the quality of the solution is satisfactory. For 95% of the tasks the number of transfers was less than the average. For some tasks however tens of thousands of transfers were required. We noticed that this only happened for extremely disproportionate task sets, that is, those with large differences among power coefficients and reward coefficients.

Observe that for the first two experiments it was possible to compare the algorithm with the optimal solution. Since we did not prove that the algorithm converges to the optimal, we consider these experiments necessary to evaluate



(a) (b)

Fig. 7. Variable Δt, $\varepsilon = 0.005$, different power functions $P_i = \alpha_i \cdot s_i^q$, available energy $E_{\max} \in [\frac{1}{5} \cdot E_{\text{OPT-LU}}, E_{\text{OPT-LU}}]$: (a) average and maximum error and (b) average number of transfers.

the performance of the algorithm. A third experiment is intended to be more realistic and assumes power functions extracted from real processor models. However, we have no way of computing the optimal solution in this case, and hence, we approximate the optimal with the solution obtained for a very small Δ_t (we used $\Delta t = \frac{d}{10^6}$) and report average and maximum errors relative to this solution. For comparison, we also used $\Delta t = \frac{d}{10^6}$ in the previous experiment for different power functions of the type $P_i = \alpha_i \cdot s^q$ and the maximum absolute error (exactly computed) across all simulations was less than 0.0001.

This third experiment shows the effect of the stopping criterion ε on the error of the solution, the number of transfers, and the number of Δt halvings required. We used a complex power function and limited the range of the power and reward coefficients in order to make the experiment more realistic. The different power functions are $P_i = \alpha_i \cdot [0.248s^3 + 0.225s^2 + \sqrt{(311s^2 + 282s) \cdot (0.014s^2 + 0.0064s)}]$ [Kumar and Srivastava 2001] with random coefficients $\alpha_i \in [0.5, 1.5]$. We used linear reward functions of the type $R_i(t_i \cdot s_i) = \beta_i \cdot t_i \cdot s_i$ with random coefficients $\beta_i \in [1, 5]$. For each result, 1000 task sets of 50 tasks were simulated using the variable Δt version of our algorithm. $\Delta t_{\text{initial}}$ was set to $\frac{d}{n \cdot \log n}$. We also experimented with $\Delta t = \frac{d}{n^2}$ and higher for the same values of ε. The first stopping test succeeded for most simulations, leading to a very small average and maximum error and a number of Δt halvings very close to 1; similar results were obtained for smaller values of ε.

Results are presented in Figure 8, where the x-axis is in log scale. The running time of the algorithm is roughly proportional to the number of transfers. Transfer time is linear in the number of tasks and depends on other factors like the complexity of the power and reward functions involved. For $\varepsilon = 0.01$ in Figure 8, the average running time of a transfer is 90 μs corresponding to an average total running time of 38 ms on a 400 MHz Pentium II. Decreasing the number of tasks to 20, the total running time is reduced to 1.5 ms. Further, if the power function in Figure 8 is replaced with the simpler $P_i = \alpha_i \cdot s_i^2$, the average running time of the algorithm is just half a millisecond.

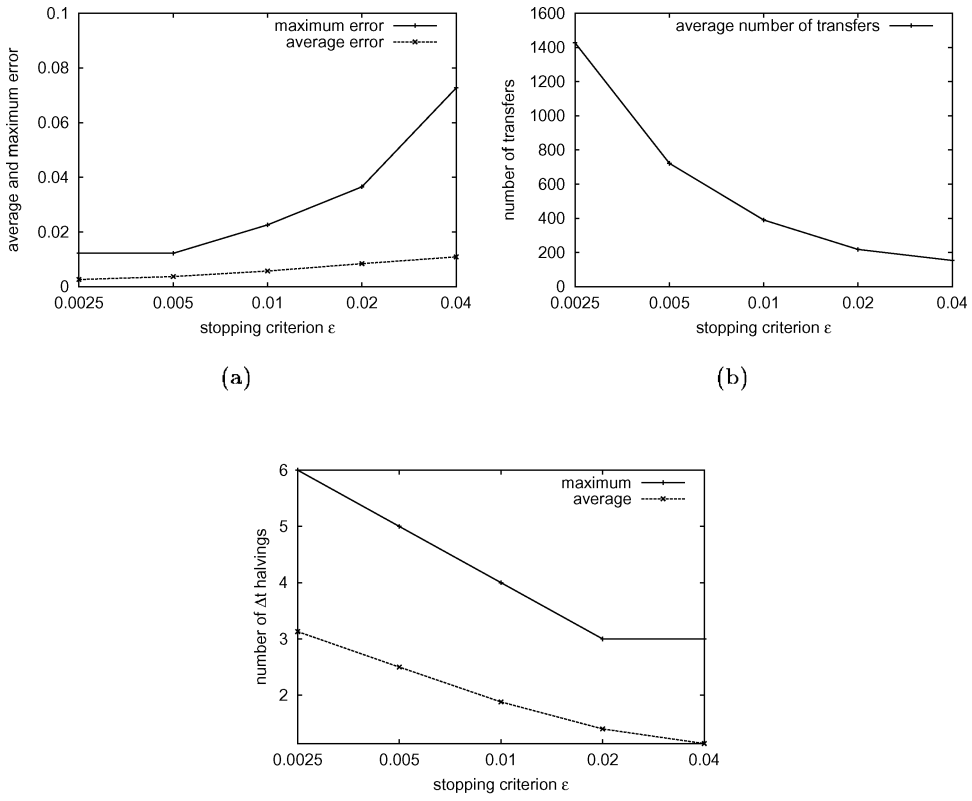


Fig. 8. Variable Δt, different power functions $P_i = \alpha_i \cdot [0.248 \cdot s^3 + 0.225 \cdot s^2 + \sqrt{(311.16 \cdot s^2 + 282.24 \cdot s) \cdot (0.014112 \cdot s^2 + 0.0064 \cdot s)}]$. Effect of stopping criterion ε on (a) average and maximum error, (b) average number of transfers, and (c) number of Δt halvings.

Observe that the absolute error is not necessarily less than ε, which is just the relative error between two consecutive solutions. Figure 8(c) shows the average and maximum number of Δt halvings required until the stopping criterion was met. As expected, as the value of ε increases, the average number of Δt halvings tends to 1. However, note that the number of halvings (maximum or average) is not large, even for small values of ε.

7. CONCLUSIONS AND FUTURE WORK

We presented an algorithm for maximizing the rewards under a limited energy budget, for both frame-based or periodic tasks. Our goal is to maximize the reward of the task set without exceeding the deadlines nor the available energy, and assuming different power functions and reward functions for each task. This work is the first that combines all three constraints, namely *energy*, *deadlines*, and *rewards*.

We devised an optimal algorithm for homogeneous power functions. Since all power functions are the same (or have the same form and different coefficients),

we are able to determine the power that each task consumes. From this, we are able to determine all task speeds, which are the same in case all functions are identical. However, the case of heterogeneous power functions (i.e., the general case) required an iterative heuristic solution. This solution starts off with an unlimited-energy solution, which is refined by changing time allocations and speed of tasks, until the energy and deadline constraints are met. Simulation results show that our iterative algorithm is fast and extremely accurate.

Subsequent to this work we analyzed the case of discrete speed levels and step reward functions (no reward for partial execution) in Rusu et al. [2002]. The problem was proven in Aydin et al. [2001c] to be NP-complete for step reward functions even assuming continuous speed range. Heuristics were proposed in Rusu et al. [2002] that render close to optimal solutions for only a fraction of the running time of an optimal algorithm. In the future, we plan to consider the overhead of changing the speed, which can be an important factor for relatively short tasks.

APPENDIX

In Section 2.2 we claimed that for periodic task sets, in the optimal solution that maximizes the total reward without exceeding an energy budget, the speed and execution times of a periodic task T_i are constant at every instance. We present the proof here.

It has been shown in Aydin et al. [2001a] that, assuming different power functions for each task, in order to minimize the energy consumption of a periodic task set, all instances T_{ij} of a task T_i should run at the same speed $s_{ij} = s_i$. However, the task model presented in Aydin et al. [2001a] assumes that each instance T_{ij} should receive exactly C_i cycles and the only goal was to minimize energy. Since C_i and $s_{ij} = s_i$ are the same for all instances, it also followed that all task instances should run for the same amount of time $t_{ij} = t_i = \frac{C_i}{s_i}$.

Similarly, it has been shown in Aydin et al. [2001c] that, assuming concave reward functions for each task, in order to maximize the total reward of a periodic task set, all task instances of a periodic task should run for the same amount of time. The task model in Aydin et al. [2001c] assumes that all tasks run at the same speed and the only goal was maximizing the rewards.

We extend the proofs in Aydin et al. [2001a, 2001c] and show that the reward is maximized and the energy is minimized when the speed and execution time are constant at every task instance. Given any feasible schedule in which there exists i, j, k such that $t_{ij} \neq t_{ik}$ or $s_{ij} \neq s_{ik}$, we show that it is always possible to construct a better schedule (higher reward and less energy) in which all instances of a task run at the same speed and for the same amount of time. The common speed and execution time of all instances of a task T_i are $s_i = \sum_{j=1}^{N_i}(s_{ij} \cdot t_{ij}) / \sum_{j=1}^{N_i} t_{ij}$ and $t_i = \sum_{j=1}^{N_i} t_{ij}/N_i$, where N_i is the number of instances within the LCM, $N_i = \frac{LCM}{d_i}$.

Observe that in the new schedule the total number of cycles allocated to all instances of each task is the same as in the original schedule. Also, the total execution time of all instances of each task is unchanged. Due to the

convex nature of the power functions P_i and since the total execution time and number of cycles is preserved, it follows that in the new schedule each task T_i consumes less energy (or the same amount if $s_{ij} = s_i$ for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, N_i$). Similarly, due to the concave nature of the reward function R_i the total reward of each task T_i is higher in the new schedule (or the same reward if $s_{ij} \cdot t_{ij} = s_i \cdot t_i$ for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, N_i$).

However, we also need to show that the new schedule is feasible (i.e., no task misses its deadline). Since all instances of a task run for the same amount of time, the periodic task set is EDF schedulable if $\sum_{i=1}^{n} U_i \leq 1$. We know that $\sum_{i=1}^{n} \sum_{j=1}^{N_i} t_{ij} \leq$ LCM since we assumed the original schedule was feasible. Since $t_i = \sum_{j=1}^{N_i} t_{ij}/N_i$ it follows that $\sum_{i=1}^{n} N_i \cdot t_i \leq$ LCM or equivalently $\sum_{i=1}^{n} U_i \leq 1$ and thus the new schedule is EDF schedulable.

REFERENCES

AYDIN, H., MELHEM, R., MOSSÉ, D., AND ALVAREZ, P. 2001a. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS)*, Delft, Netherlands.

AYDIN, H., MELHEM, R., MOSSÉ, D., AND ALVAREZ, P. M. 2001b. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, London, UK.

AYDIN, H., MELHEM, R., MOSSÉ, D., AND ALVAREZ, P. M. 2001c. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers 50,* 2 (Feb.), 111–130.

CHANG, E. AND ZAKHOR, A. 1993. Scalable video coding using 3D subband velocity coding and multi-rate quantization. *IEEE Int. Conference On Acoustics, Speech and Signal Processing*.

CHUNG, J.-Y., LIU, J. W.-S., AND LIN, K.-J. 1993. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers 19*, 9 (Sept.), 1156–1173.

CLARK, R. K., JENSEN, E. D., AND REYNOLDS, F. D. 1992. Architectural overview of the alpha real-time distributed kernel. *USENIX Workshop on MicroKernels and Other Kernel Architectures*.

DEY, J. K., KUROSE, J., AND TOWSLEY, D. 1996. On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE Transactions on Computers 45*, 7 (July), 802–813.

DEY, J. K., KUROSE, J., TOWSLEY, D., KRISHNA, C. M., AND GIRKAR, M. 1993. Efficient online processor scheduling for a class of iris (increasing reward with increasing service) real-time tasks. In *Proceedings of ACM SIGMETRICS Conference on Measurement an Modeling of Computer Systems*, 217–228.

ERNST, R. AND YE, W. 1997. Embedded program timing analysis based on path clustering and architecture classification. In *Computer-Aided Design(ICCAD)*, 598–604.

FENG, W. AND LIU, J. W.-S. 1993. An extended imprecise computation model for time-constrained speech processing and generation. In *Proceedings of the IEEE Workshop on Real-Time Applications*.

GRASS, J. AND ZILBERSTEIN, S. 2000. A value-driven system for autonomous information gathering. *Journal of Intelligent Information Systems 14*, (Mar.), 5–27.

GRUIAN, F. 2001. Hard real-time scheduling using stochastic data and dvs processors. In *Proceedings of International Symposium on Low Power Electronics and Design*, 46–51.

HONG, I., KIROVSKI, D., QU, G., POTKONJAK, M., AND SRIVASTAVA, M. 1998a. Power optimization of variable voltage core-based systems. In *Proceedings of the 35th Design Automation Conference (DAC)*.

HONG, I., POTKONJAK, M., AND SRIVASTAVA, M. 1998b. On-line scheduling of hard real-time tasks on variable voltage processors. In *Computer-Aided Design (ICCAD)*, 653–656.

HONG, I., QU, G., POTKONJAK, M., AND SRIVASTAVA, M. 1998c. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid.

JOSEPH, R. AND MARTONOSI, M. 2001. Run-time power estimation in high-performance microprocessors. In *The International Symposium on Low Power Electronics and Design (ISLPED)*.

KRISHNA, C. M. AND LEE, Y. H. 2000. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS)*, Washington DC.

KRISHNA, C. M. AND SHIN, K. G. 1997. *Real-Time Systems*. McGraw-Hill, New-York.

KUMAR, P. AND SRIVASTAVA, M. 2001. Power-aware multimedia systems using run-time prediction. In *Proceedings of the 14th International Conference on VLSI Design (VLSID)*, Bangalore, India.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of ACM 20*, 1 (Mar.), 46–61.

LIU, J. W.-S., LIN, K.-J., SHIH, W.-K., YU, A. C.-S., CHUNG, C., YAO, J., AND ZHAO, W. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer 24*, 5 (May), 58–68.

LORCH, J. R. AND SMITH, A. J. 2001. Improving dynamic voltage scaling algorithms with pace. In *Proceedings of the ACM SIGMETRICS 2001 Conference*, Cambridge, MA.

LUENBERGER, D. 1984. *Linear and Nonlinear Programming*. Addison-Wesley, Reading MA.

MOSSÉ, D., AYDIN, H., CHILDERS, B., AND MELHEM, R. 2000. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, Philadelphia, PA.

RAJKUMAR, R., LEE, C., LEHOCZKY, J. P., AND SIEWIOREK, D. P. 1997. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*.

RAJKUMAR, R., LEE, C., LEHOCZKY, J. P., AND SIEWIOREK, D. P. 1998. Practical solutions for QoS-based resource allocation problems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*.

RUSU, C., MELHEM, R., AND MOSSÉ, D. 2002. Maximizing the system value while satisfying time and energy constraints. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, TX.

SHIH, W.-K., LIU, J. W.-S., AND CHUNG, J.-Y. 1991. Algorithms for scheduling imprecise computations with timing constraints. *SIAM Journal on Computing 20*, 3, 537–552.

SHIN, D., KIM, J., AND LEE, S. 2001. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers 18*, 3 (Mar.), 20–30.

SHIN, Y. AND CHOI, K. 1999. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th Design Automation Conference (DAC)*.

TURNER, C. J. AND PETERSON, L. L. 1992. Image transfer: An end-to-end design. In *SIGCOMM Symposium on Communications Architectures and Protocols*.

VRBSKY, S. V. AND LIU, J. W. S. 1993. Approximate—A query processor that produces monotonically improving approximate answers. *IEEE Transactions on Knowledge and Data Engineering 5*, 12 (Dec.), 1056–1068.

YAO, F., DEMERS, A., AND SHANKAR, S. 1995. A scheduling model for reduced cpu energy. *IEEE Annual Foundations of Computer Science*, 374–382.

Received February 2002; revised July 2002; accepted January 2003