INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 24 (2017) 173–198
DOI: 10.1111/itor.12328

Energy-aware scheduling mandatory/optional tasks in multicore real-time systems

Isabel Méndez-Díaz[a], Javier Orozco[b], Rodrigo Santos[b] and Paula Zabala[a,b]

[a] *Dep. de Computación, Fac. Cs. Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina*
[b] *Dep. Ing. Eléctrica y Computadoras—IIIE, Universidad Nacional del Sur, CONICET, Buenos Aires, Argentina*
E-mail: imendez@dc.uba.ar [Méndez-Díaz]; jadorozco@gmail.com [Orozco]; ierms@criba.edu.ar [Santos]; pzabala@dc.uba.ar [Zabala]

Received 13 April 2015; received in revised form 14 June 2016; accepted 17 June 2016

Abstract

Reward-based scheduling of real-time systems of periodic, preemptable, and independent tasks with mandatory and optional parts in homogeneous multiprocessors with energy considerations is a problem that has not been analyzed before. The problem is NP-hard. In this paper, a restricted migration schedule is adopted in which different jobs of the same task may execute in different processors and at different power modes but no migration is allowed after the job has started its execution. An objective function to maximize the performance of the system considering the execution of optional parts, the benefits of slowing down the processor, and a penalty for changing the operation frequency is introduced together with a set of constraints that guarantee the real-time performance of the system. Different algorithms are proposed to find a feasible schedule maximizing the objective function and are compared using synthetic systems of tasks generated following guidelines proposed in previous papers.

Keywords: scheduling; integer programming; heuristics; combinatorial optimization

1. Introduction

In the past years, computer-based systems have been incorporated in everyday life. Previously, embedded systems understood as computer systems, often with real-time computing constraints, that form part of a larger system (Heath, 2002), were restricted to critical areas related to military applications, avionics, and sophisticated industrial controllers. Nowadays, they have become key components in different areas such as cars, home appliances, and entertainment, among others. The reduction in the costs of hardware, the availability of wireless links, the generalization of mobile smartphones, and Internet have pushed the limit creating new opportunities and applications. They exhibit dynamic behavior characterized by a set of dynamic parameters. Among them, the anytime algorithms represent a class of applications in which the tasks improve their response quality when they are executed for longer periods of time (Aydin et al., 2001). Some examples of this are: finding the roots of a function with the Newton–Raphson method, image and speech processing in multimedia applications, navigation systems, medical decision making, information gathering, or real-time heuristic search. The tasks implementing the algorithms have two parts. A mandatory part that has to be completed to achieve the minimum expected quality and an optional part that improves the quality of the result. Optional parts have associated a reward for their execution. If such applications have time constraints, they constitute a special scheduling problem as they have to be executed maximizing the reward associated to the execution of the optional parts while satisfying all the mandatory deadlines.

As mentioned before, there are several examples of mandatory/optional parts. For example, in control engineering the determination of the roots of the transfer function is of major importance as they are related to the stability of the system and the quality of the controlled output. Often the working conditions are time dependent so an adaptive control algorithm should be implemented (Astrom and Wittenmark, 1994). This implies the computation of the roots of the system online. The precision with which the roots are determined is an important issue, a minimum speed is necessary (mandatory computation) but if possible to increment it (optional part) the quality of the output will also improve. The reward associated to the optional execution is set by the designer of the application. In this way, if several tasks have mandatory and optional parts, the rewards are coherent from the designer point of view (Aydin et al., 2001).

In real-time computing, the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced (Stankovic, 1988). A task should provide a correct answer before a certain instant named deadline. There are many ways to order the execution of different tasks in a monoprocessor system to guarantee the time restrictions. In Liu and Layland (1973), it is proved that Rate Monotonic is optimal among fixed priorities scheduling policies and Earliest Deadline First is optimal for dynamic priorities. However, scheduling for multiprocessors or multicore processors platforms is still an open issue as there is no optimal polynomial solution (Fisher, 2007).

Computer systems and specially embedded systems face a major challenge with energy consumption as the processor consumes most of the energy producing serious pollution and waste of resources in the natural environment (Zhang et al., 2014).

The power demand of processors is the sum of the dynamic and static power: $P = P_d + P_s = CfV^2 + P_s$. The first one is produced by the switching of transistors and it is proportional to the frequency and the voltage. The second one is proportional to the leakage current and the amount of transistors in the processor.

Most modern processors can manage both frequency and voltage levels for controlling the energy consumed. However, reducing the operating frequency increases the time needed to execute a task imposing a trade-off between the schedulability and the energy consumption.

In embedded systems, the set of tasks to schedule and their parameters is known so two possible approaches are possible, online or offline. Offline scheduling is more predictable and secure for critical systems. In this paper, we propose an offline scheduling mechanism. As far as we know, there is no previous work on mandatory/optional energy-aware scheduling analysis for homogeneous multicore real-time systems. The main contribution of this work is to provide a solution to this problem. The paper presents an integer linear programming (ILP) model and several heuristic algorithms that are evaluated on instances generated following traditional guidelines in the literature.

The approach proposed in this paper allows a dynamic configuration of the system based on certain external parameters. For example, a system running on solar panels may privilege the execution of optional parts over the reduction in power mode during daylight and privilege a reduction of energy consumption reducing the optional execution rewards when no light is present.

The rest of the paper is organized in the following way. In Section 2 previous work is analyzed. In Section 3 the model of the system is presented. Sections 4 and 5 present the ILP model and the different heuristic algorithms, respectively. In Section 6 the experimental evaluation is described and the results are explained. Finally in Section 7 conclusions are drawn.

2. Previous work

In what follows, papers related to the mandatory/optional reward and homogeneous multicore multitask scheduling are discussed. The papers are grouped under different headings, in line with the (combined) aspects of interest here.

2.1. Mandatory/optional reward-based scheduling

"Anytime algorithms" have been in use for many years for traditional iterative-refinement applications. Initially, the idea of using this kind of algorithms was related to the minimization of the weighted sum of errors (Liu et al., 1991; Shih and Liu, 1995).

In Chung et al. (1990), a mandatory first schedule is proposed leaving different alternatives for scheduling the optional parts. The optional parts are scheduled in the background so no reshuffling of the slack time is performed. In Aydin et al. (2001), the different schemes are compared and it is shown that the best results are obtained when the background slack is allocated to the optional parts contributing the most to the total reward at that moment. The method is called Best Incremental Return and it is used as a yardstick. In the same paper, the authors present an offline reward-based method. They use linear programming maximizing techniques and because of this the reward functions need to be continuously differentiable. They also impose several constraints like executing the same amount of optional parts for each job of the tasks and this results in a rather rigid approach. In the paper, the authors mention a generalization of the objective function for the multiprocessor case but they do not mention anything about the task or job processor allocation method or the way in which tasks are scheduled in each processor. None of the previous papers consider the power demand in the analysis or the possibility of reducing the operational voltage/frequency to prolong the lifetime of the system.

In Aydin et al. (2004), the authors introduced a static offline solution to compute the optimal speed at the task level, assuming worst-case workload for each arrival. They also proposed a dynamic reclaiming algorithm for tasks that complete their execution without using their worst-case workload and an online, adaptive and speculative speed adjustment mechanism. The authors proved that the power-aware scheduling problem is equivalent to solving an instance of the reward-based scheduling problem when the reward functions are concave. The analysis is presented for the case of uniprocessor systems.

In contrast to Aydin et al. (2001, 2004) and Chung et al. (1990), in this paper the mandatory/optional scheduling is introduced for a homogeneous multiprocessor platform with energy-aware management and the only requirement for the reward functions is their computability at each instant.

2.2. Multiprocessor scheduling

Multiprocessor scheduling is an open issue as there is no optimal online algorithm and most feasibility tests are just sufficient conditions. In Hong and Leung (1988), it is proved that no optimal online multiprocessor scheduling algorithm for arbitrary jobs exists unless all jobs have the same relative deadline. Furthermore, in Dertouzos (1989) it is proved that even if the execution times are known precisely there is still no change. Finally, in Fisher (2007) it is proved that there is no optimal online algorithm for sporadic task sets with constraints or arbitrary deadlines.

In Andersson and Jonsson (2000), the authors compare the performance of systems with global and partitioned schedulers and show that the global one offers better results. In Andersson and Jonsson (2003), it is shown that for aperiodic task sets where future arrivals are unknown, no algorithm for tasks without migration capabilities can have a capacity bound greater than 0.50.

In Baruah and Carpenter (2003), the authors show that the feasibility analysis of priority-driven scheduling of periodic and sporadic task systems upon homogeneous multiprocessors is intractable (NP-hard in the strong sense), if each job may be assigned exactly one priority and it has to execute upon only a single processor.

In Zhang et al. (2014), a heuristic algorithm based on the particle swarm is proposed for the scheduling of real-time tasks in heterogeneous multiprocessors systems considering energy constraints. Although the solution the authors proposed is interesting, they do not consider a mandatory/optional reward model for the tasks.

In Zhang and Guo (2014), the authors introduce a power-aware scheduling algorithm for real-time sporadic tasks. The algorithm is proposed for uniprocessor systems and schedules the tasks instances as they become active. If an instance finishes before its worst-case execution time, the algorithm dynamically reduces the power mode of the processor. This is different from our proposal in three important aspects: (i) it is for uniprocessor system while ours is for multiprocessor system, (ii) the model used is for sporadic task while our model is for periodic tasks, (iii) on demand schedule while our proposal is for an offline schedule computation.

In Kumar and Palani (2012), the authors proposed a genetic algorithm for scheduling tasks in multiprocessors systems with dynamic voltage scaling. The algorithm, however, does not contemplate real-time restrictions and focuses on optimizing the schedule length of the periodic task model subject to a minimum energy consumption.

In Rizvandi et al. (2011), the authors introduced an algorithm to determine the frequencies at which tasks should be executed to minimize the energy consumption. In order to do this, when a task finishes the difference between the actual and the worst-case execution times is used to slow down the processors. The multiprocessor scheduling is done with the Longest Processor Time First. This work, however, is different from our proposal because it is oriented to high-performance computing systems (HPC), tasks have only mandatory parts and the scheduler is dynamic.

In Albers et al. (2015), the authors propose two algorithms to schedule tasks in multiprocessors systems optimizing the energy consumption. The first algorithm uses a strongly combinatorial algorithm that relies on repeated maximum flow computations. The second algorithm works online and is based on two previous strategies: *Optimal Available* and *Average Rate*. The general idea is that each time a new job arrives, a new optimal schedule is computed using the offline algorithm based on the actual load. Although this work proposes a solution to the schedule of tasks with energy considerations in multiprocessors environments with time constraints, the task model is substantially different from the one proposed in our paper.

In Han et al. (2010), the authors introduce an online scheduling algorithm to schedule tasks with time constraints in variable speed processors. The algorithm improves previous solutions but is restricted to one processor and does not consider the task model with mandatory/optional parts. In Chan et al. (2011), the authors present an algorithm to solve the trade-off between energy consumption and throughput for the single-processor scheduling problem. To solve the problem, they introduce a *value* that represents a reward associated to the task for executing it. Tasks with low values are eventually dropped if the deadline cannot be met. This approach is different from the problem proposed in our paper as all tasks should finish by their deadlines in a multiprocessor environment and the reward comes from executing optional parts.

In Lam et al. (2008), the authors introduce an online algorithm to schedule tasks on multiprocessors systems without tasks migrations and considering energy consumption. The algorithm is quite simple and it uses a simple round-robin scheduling policy for distributing the jobs among the processors. The model differs since we admit job migrations and maximize the computation of optional parts.

3. System model

Given a set of real-time tasks with mandatory/optional parts to be run on a multicore hardware platform with power demand control, the best optional parts should be executed guaranteeing mandatory deadlines and minimizing energy consumption.

The time is assumed to be discrete and is noted $h = 1, 2, \dots$ Instant h and slot h are equivalent terms. Events are synchronized with the start of the slot. Although it is possible to analyze the problem considering time continuous, embedded systems usually operate with a discrete time base, slot, or timer tick. The duration of the slot is a designed parameter that depends on different aspects and it is not analyzed in this paper.

There is a set of tasks $\mathcal{S} = \{\tau_1, \tau_2, \dots, \tau_n\}$. Tasks are independent, periodic, and preemptive. Each task is described by a tuple $(m_i, o_i, P_i, D_i, r_i)$. m_i is the worst-case execution time of the mandatory part. o_i is the execution time of the optional part. P_i is the period of the task, which is assumed equal to the relative deadline $P_i = D_i$. The execution of the optional part produces a reward or benefit that is represented by $r_i : \{1, \dots, o_i\} \to \mathbb{R}$. This reward is computed for each slot and has no restrictions on the class of function. The least common multiple (lcm) of the periods defines a window in which the system repeats itself. For this reason, it is enough to provide a schedule for the first $H = lcm\{\forall P_i\}$. As stated in Section 1, each task, $\tau_i \in \mathcal{S}$, is a stream of jobs or instances $\rho_i^1, \rho_i^2, \dots, \rho_i^v, \dots, \rho_i^{H/P_i}$. H/P_i states for each task the amount of instances to be executed in the hyperperiod. Since a restricted-migration schedule is proposed, it is necessary to describe the set of jobs in the system. In the hyperperiod H there are $V = \sum_i^n H/P_i$ jobs to be scheduled and they are described in $\mathcal{V} = \{\forall i \in S, \rho_i^v \; v = 1, \dots, H/P_i\}$. A job ρ_i^v becomes active at $a_i^v = (v-1)P_i$ and should

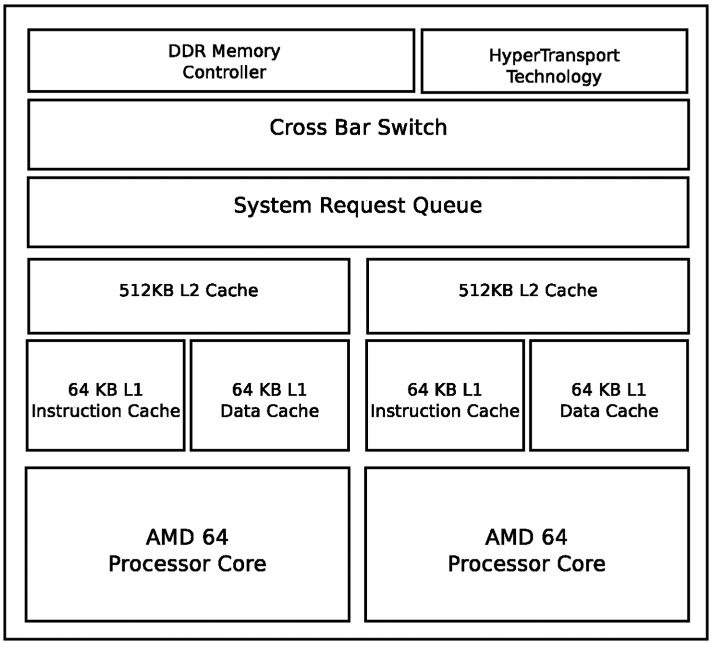

Fig. 1. Block diagram for AMD turion$^{\text{TM}}$X2 dual-core mobile processor.

finish its execution at $\phi_i^v < d_i^v = a_i^v + D_i$. At this point, it is useful to introduce for every $\rho_i^v \in \mathcal{V}$, the set of instants at which they are active, $H_i^v = \{h : a_i^v \leq h \leq a_i^v + D_i - 1\}$. A feasible system is one in which $\forall \rho_i^v \in \mathcal{V} \;\; \phi_i^v < d_i^v$.

There is also a set of homogeneous processors or cores, $\Pi = \{\pi_1, \pi_2, \dots, \pi_m\}$. Each core counts with at least one independent cache memory level, L1. For example, the AMD Turion$^{\text{TM}}$x2 Dual-Core Mobile Multiprocessor has two independent cores and each one has L1 and L2 cache memories. Figure 1 presents the block diagram. Each core may work at different power modes. In most cases there are two to eight different options from sleep mode, where the power demand is minimum, to maximum frequency operation. The power modes for each processor are notated $\mathcal{F} = \{f_0, f_1, \dots, f_F\}$, where F indicates the amount of frequency levels. Lower power-demand modes provide a benefit as the system saves energy and the batteries' life is enlarged.

The objective is to find an optimal schedule for maximizing the reward from the execution of optional parts with as little power demand as possible, while satisfying all mandatory deadlines. The allocation is made at job level. In other words, a job that starts its execution in a certain core finishes its execution in the same core. It is also assumed that a power mode is selected on a per job basis. Under this assumption, a task may run its first job in core A at power mode X and the second job in core B at power mode Y and in each case the amount of optional work may be different.

The problem stated in this way is NP-hard in the strong sense. This can be shown as it generalizes two NP-hard problems. If the instance does not have optional parts and has equal task periods, it is the well-known *bin-packing problem* (Garey and Johnson, 1979).

Furthermore, let us consider a *0-1 knapsack* instance of n items, maximum weight W, each item with weight w_i, and value v_i. The instance can be transformed to an instance of our problem with one

Table 1
System description

τ	m	o	P	R
1	1	2	4	3
2	1	2	4	3
3	1	1	6	5
4	1	1	6	5
5	2	4	12	1
6	2	4	12	1

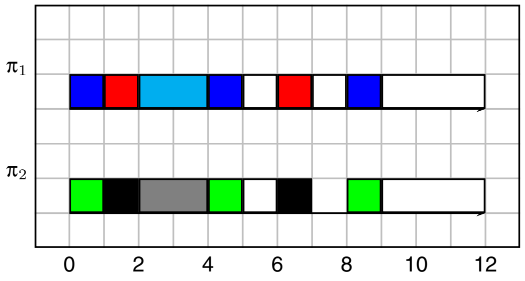

Fig. 2. Execution at full power without optional. τ_1 blue, τ_2 green, τ_3 red, τ_4 black, τ_5 cyan, τ_6 gray, empty slots in white.

processor and n tasks described by a tuple $(0, w_i, W, W, r_i)$ with $r_i(x) = 0$ if $x < w_i$ and $r_i(w_i) = v_i$ for $i = 1, \ldots, n$. Since the *0-1 knapsack problem* is known to be NP-hard (Karp, 1972), we conclude that our problem is also NP-hard.

3.1. Example

In this section a simple example is introduced to illustrate different feasible schedules and clarify ideas. In Table 1, a mandatory/optional system composed of six periodic, independent, and preemptive tasks with constant rewards for each optional slot executed (linear reward function) is presented. The system runs on a two core processor with independent cache memories. Each core has two power modes: half power (execution times are doubled) and full power. We assume that executing at half-power mode has a gain of two while executing at full power is just zero but changing the power mode will cost one unit overall. With this setting, if task one executes its mandatory part and the two optional system will receive a reward of 6, but if it executes only its mandatory part at half-power mode it will need two slots contributing with a reward of 3 to the system.

Figure 2 shows a possible temporal evolution of the mandatory parts of each task. This is a feasible schedule but it is not the best one as there are empty slots, no optional parts are executed and no energy is saved.

The example shows the trade-off between the energy saving and the reward obtained from the optional parts. In Fig. 3, energy saving is privileged while in Fig. 4 the improved quality of some jobs is chosen. Between both extremes, several combinations may be possible. For tuning the problem the designer may include weights so he/she can bias the search of a solution.

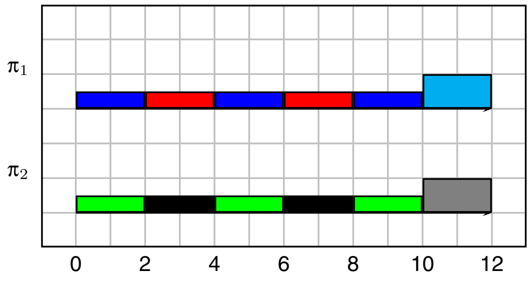

Fig. 3. Execution at half-power mode without optional parts. τ_1 blue, τ_2 green, τ_3 red, τ_4 black, τ_5 cyan, τ_6 gray.

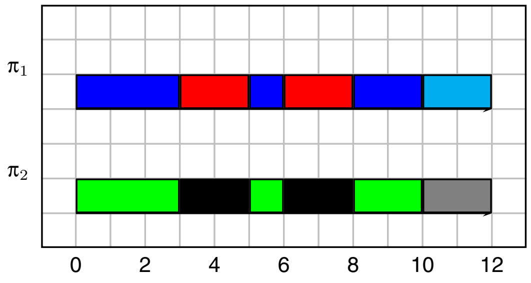

Fig. 4. Execution at full-power mode with optional parts. τ_1 blue, τ_2 green, τ_3 red, τ_4 black, τ_5 cyan, τ_6 gray.

Some real embedded systems may work on different modes depending on the actual situation of the system. For example, if a system operates based on solar panels during daylight and batteries during night hours, the two schedules are convenient for the two different situations. In the case of daylight, full advantage of the optional parts execution is obtained and during night the system operates with a reduced power mode but still meeting all mandatory deadlines. So the designer may obtain different schedules just by varying the weights accordingly to what is expected in each situation.

4. ILP formulation

Like many combinatorial problems, the general model of the system can be tackled with an ILP approach. The optimization problem involves the definition of an objective function and the associated constraints. The ILP solution identifies for each slot in the hyperperiod of the system and for each processor which job of each task and at what frequency has to execute.

For all tasks $\tau_i \in \mathcal{S}$, jobs $\rho_i^v \in \mathcal{V}$, processors $\pi_j \in \Pi$, frequency levels $f \in \mathcal{F}$, and slots $h = 1, \ldots, H$, the following binary variables are considered:

$$y_{iv}^{jf} = \begin{cases} 1 & \text{if job } \rho_i^v \text{ of } \tau_i \text{ runs in } \pi_j \text{ at frequency } f \\ 0 & \text{otherwise} \end{cases}$$

$$x_i^{kjf} = \begin{cases} 1 & \text{if at slot } k\ \pi_j \text{ runs } \tau_i \text{ at frequency } f \\ 0 & \text{otherwise} \end{cases}$$

$$g^{kjf} = \begin{cases} 1 & \text{if at slot } k\ \pi_j \text{ runs at frequency } f \\ 0 & \text{otherwise} \end{cases}$$

$$z^{kj} = \begin{cases} 1 & \text{if at slot } k\ \pi_j \text{ changes frequency} \\ 0 & \text{otherwise} \end{cases}$$

$$u_{iv}^{fd} = \begin{cases} 1 & \text{if job } \rho_i^v \text{ of } \tau_i \text{ runs } d \text{ optional slots at frequency } f, d = 0, \ldots, o_i^f \\ 0 & \text{otherwise.} \end{cases}$$

For each task τ_i, the execution of d optional slots at frequency f produces a reward r_i^{fd}, for $d \in \{0, \ldots, o_i^f\}$. The operation at different frequencies produces different rewards and these are modeled by c_f. The cost of changing the operational frequency is modeled by c. Using these definitions, the objective function of the problem may be written as:

$$\max \alpha \sum_{\rho_i^v \in \mathcal{V}} \sum_{f \in \mathcal{F}} \sum_{d=1}^{o_i^f} r_i^{fd} u_{iv}^{fd} +$$

$$\beta \left(\sum_{f \in \mathcal{F}} c_f \sum_{k=1}^{H} \sum_{j \in \Pi} g^{kjf} - c \sum_{j \in \Pi} \sum_{k=1}^{H} z^{kj} \right). \tag{1}$$

The objective function (1) maximizes the reward of the system from the execution of optional parts and the reduction of the frequency of operation in each processor. There are two main terms in the function. The first one computes the reward provided by the execution of the optional parts of the different jobs in the system. This term has a general tuning parameter α, which is used to weight the importance of the term.

The second one accounts the reward obtained by the frequencies at which the processors run. If the processor works with lower frequencies, the reward is higher because there is less energy demand. But changing the voltage/frequency combination is not something trivial and some time is required before the system can run properly with the new pair. For this reason, the second term of the objective function considers not only the reward of running at each frequency but also that changing the frequency may have a negative impact on the general behavior of the system. For this term, the tuning parameter is β. When energy consumption is more important than the reward obtained from executing optional parts, α is smaller than β.

The maximization is subject to the following restrictions:

$$\sum_{j \in \Pi} \sum_{f \in \mathcal{F}} y_{iv}^{jf} = 1 \qquad \forall \rho_i^v \in \mathcal{V} \tag{2}$$

$$\sum_{k \in H_i^v} x_i^{kjf} \leq (m_i^f + o_i^f) y_{iv}^{jf} \qquad \forall \rho_i^v \in \mathcal{V}; \forall j \in \Pi; \forall f \in \mathcal{F} \tag{3}$$

$$\sum_{j \in \Pi} \sum_{k \in H_i^v} x_i^{kjf} \geq m_i^f \sum_{j \in \Pi} y_{iv}^{jf} \qquad \forall \rho_i^v \in \mathcal{V}; \forall f \in \mathcal{F} \tag{4}$$

$$\sum_{i\in S}\sum_{f\in\mathcal{F}} x_i^{kjf} \le 1 \qquad \forall j\in\Pi; k=1,\dots,H \tag{5}$$

$$\sum_{d=0}^{o_i^f} u_{iv}^{df} = \sum_{j\in\Pi} y_{iv}^{jf} \qquad \forall \rho_i^v \in \mathcal{V}; \forall f\in\mathcal{F} \tag{6}$$

$$\sum_{f\in\mathcal{F}}\sum_{k\in H_i^v} x_i^{kjf} - \sum_{f\in\mathcal{F}} m_i^f \sum_{j\in\Pi} y_{iv}^{jf} = \sum_{f\in\mathcal{F}}\sum_{d=1}^{o_i^f} d u_{iv}^{df} \quad \forall \rho_i^v\in\mathcal{V} \tag{7}$$

$$\sum_{f\in\mathcal{F}} g^{kjf} = 1 \qquad \forall j\in\Pi; \forall k=1,\dots,H \tag{8}$$

$$g^{kjf} \ge \sum_{i\in\mathcal{S}} x_i^{kjf} \qquad \forall j\in\Pi; \forall f\in\mathcal{F}; k=1,\dots,H \tag{9}$$

$$g^{kjf} \ge g^{k-1jf} - \sum_{f'\in\mathcal{F}}\sum_{i\in S} x_i^{kjf'} \qquad \forall j\in\Pi; \forall f\in\mathcal{F}; k=1,\dots,H \tag{10}$$

$$Fz^{kj} \ge \sum_{f\in\mathcal{F}} f g^{kjf} - \sum_{f\in\mathcal{F}} f g^{k+1jf} \qquad \forall j\in\Pi; k=1,\dots,H-1 \tag{11}$$

$$Fz^{kj} \ge \sum_{f\in\mathcal{F}} f g^{k+1jf} - \sum_{f\in\mathcal{F}} f g^{kjf} \qquad \forall j\in\Pi; k=1,\dots,H-1 \tag{12}$$

Constraint (2) guarantees that each job is executed in just one processor at a unique frequency. Constraint (3) guarantees that no more than $m_i^f + o_i^f$ slots are executed at frequency f and constraint (4) guarantees the execution of at least m_i^f slots for each job of each task. Constraint (5) imposes the restriction of only one task for each slot in each processor.

Constraint (6) guarantees that for $d = 0,\dots,o_i^f$ exactly one of the variables u_{iv}^{df} takes value 1 when job ρ_i^v executes at frequency f and takes value 0 when job ρ_i^v executes at some other frequency. Constraint (7) makes one the variable u_{iv}^{df} for d equal to the amount of optional slots that were executed for instance ρ_i^v.

If one job is executing in processor π_j in slot k at frequency f then constraints (8) and (9) impose a value of 1 to g^{kjf} and constraint (10) forces an empty slot to keep the same frequency of the last busy one.

When there is a frequency change in slot k in processor π_j, constraints (11) and (12) impose a value of 1 on variable z^{kj}.

Solution performance depends on several factors, with model size (number of constraints and integer variables) being a key point. In this case, variables and constraints are related to the number of tasks, processors, frequency levels, and the hyperperiod. In particular, the hyperperiod is critical as it is the largest and it can easily reach very high values. To avoid this, the designer should try to match the tasks' periods to obtain harmonic relations between them. This is possible as usually embedded systems tasks are related to the physical world and periods are set based on control or information-processing requirements. In these cases, it may be better to shorten a period to avoid a large hyperperiod even if the system becomes more stringent.

Successful techniques to solve ILP models are based on an intelligent enumeration of the feasible set of solutions like, for example, Branch and Cut algorithms. The basic idea is to guide the search by looking for upper and lower bounds of the optimum value discarding those sets of possible solutions in which the upper bound is lower than the lower bound. The value of the objective function in any feasible solution is a lower bound for the optimum. On the other hand, the upper bound used is found by solving a subproblem without the integrality constraints, which is a linear relaxation of the problem. If the upper bound of a set of solutions is equal or lower than the best lower bound, it is possible to discard the region. The success of the method depends on the quality of the bounds found in the process.

The model was solved using the standard Branch and Cut approach provided by the CPLEX (CPLEX Optimizer for z/OS) library (IBM, 2011). The time needed to find the optimum can be long even for middle size problems. In general, the upper bound is good but it is not easy to find feasible solutions with good lower bounds, especially when there are many frequency levels available. However, integer programming can be an effective and useful tool in generating high-quality heuristics, as discussed in the next section.

5. Heuristics

As we mentioned before, solving the proposed model by ILP tools is not computationally suitable due to the inherent problem complexity and its large number of variables and constraints. Hence, a heuristic solution approach is a practical method for the problem solution.

This paper proposes a two-stage heuristic approach. In the first phase, the main idea is to obtain feasible assignments of jobs to processors (mandatory parts meet their deadlines) by working at nominal frequency. Once a successful assignment has been found, a reduced version of the general problem is solved for each processor. Each problem considers only jobs assigned to the processor in the first phase, including mandatory and optional parts and the possibility to change the power mode.

Next we describe in detail the phases of the algorithm.

5.1. Two-phase approach

```
Data: $\mathcal{S}, \Pi, \mathcal{F}$
Result: Feasible schedule
  Phase one:
    Data: $\bar{\mathcal{S}} = \{(m_i, P_i) : \tau_i \in \mathcal{S}\}, \Pi$
    Result: Partition of $\mathcal{V}$ in $\mathcal{V}_1, \ldots, \mathcal{V}_m$, where $\mathcal{V}_i$ is the set of jobs allocated to processor $i$
  Phase two:
    For each processor $i = 1, \ldots, m$
      Data: $\mathcal{V}_i, \mathcal{S}, \mathcal{F}$
      Result: Feasible schedule to processor $i$.
```

5.1.1. Phase one

The first stage refers to the construction of an initial assignment of jobs to processors. They are allocated to processors with two different approaches: an exact method of integer programming; and greedy heuristics following certain priority orders.

- HILP
  The frequency levels are one of the factors that influence the performance of the proposed ILP model. However, if there is no solution at the nominal or maximum frequency, then there is no solution at any other frequency level. Based on this, we decided to restrict the number of frequency levels to the nominal one first. The performance of the algorithm improves substantially. The main idea is to achieve a feasible assignment of mandatory parts but, since in the second phase optional parts will be considered, it would seem profitable that such initial assignment allows processing as much as possible rewarding optional parts. This method is called HILP.
- Greedy heuristics
  The basic idea of a greedy allocation procedure is to assign available processor time sequentially to a job by following a priority order list. During each step, a job is chosen from the list and it is assigned to a processor. The algorithm finishes when all jobs are assigned or it is found that it is not possible to schedule the system according to the present scheme.

In case that there are mandatory and optional parts, the procedure should ensure that mandatory parts meet their deadlines. This goal is usually achieved giving lower priorities to optional parts. Then, if there are mandatory parts ready to be run, they are assigned to one processor. Taking into account that the final obtained allocation of mandatory parts does not depend on if we consider or not optional parts, we decide to discard the optional parts from the priority order list.

In order to introduce the general idea, let us consider the following definitions and notation. Let $L(j, h)$ be the list of jobs that have been allocated to processor π_j up to slot h and $\bar{L}(j, h)$ the ordered list of jobs still executing at slot h in processor π_j. In other words, $\bar{L}(j, h)$ contains the jobs that have not finished their execution at slot h yet.

Definition 5.1. *A job $\rho_i^v \in \bar{L}(j, h)$ is critical at slot h if it has to execute nonstop until the end to finish before its deadline, or what is the same if the amount of mandatory slots still to be executed is equal to $a_i^v + D_i - h$.*

Definition 5.2. *A processor π_j is critical at slot h if it has a critical job.*

Definition 5.3. *A processor π_j at slot h with more than one critical job is collapsed and makes the schedule not feasible.*

Definition 5.4. *At slot h, job ρ_i^v not allocated to any processor is named urgent if $h = a_i^v + D_i - m_i$.*

Definition 5.5. *The charge of processor π_j at slot h is:*

$$\mathcal{C}(j, h) = \sum_{\rho_i^v \in \bar{L}(j,h)} \frac{m_i - m_i^v(h)}{a_i^v + D_i - h},$$

where $m_i^v(h)$ is the amount of slots already executed by job ρ_i^v at instant h.

The basic scheme of the algorithm is as follows. Jobs are listed according to their priority and the release time and a *critical jobs* list is kept updated in order to ensure the schedule of *urgent* jobs. With this information for each slot in the hyperperiod ($h = 1, \dots, H$) jobs are distributed to processors and the order of execution is set. There are several possible situations. In all cases, the purpose is to keep the *processors' charge* as even as possible. First, it is necessary to check if any processor π_j has a *critical job* ρ_i^v. In that case, processor π_j continues executing ρ_i^v until completion. If at this stage any processor is *co-lapsed* the job allocation is not feasible and the assignment process is discarded. Second, if there are *urgent jobs* they should be allocated to processors that are not *critical* at that instant with the lowest *charge*. If there are more *urgent jobs* than available processors to handle them the schedule is unfeasible and the allocation is discarded. Third, if a job becomes active at instant h and has a higher priority than any of the current jobs executing in any of the processors, it preempts the job executing in the processor with the lowest charge. Finally, if a processor is idle, the highest priority job on the list is allocated to it. See Algorithm 1 for details.

In Fisher (2007) it is proved that for multiprocessors with restricted migrations of jobs there is no optimal (in the strong sense) priority assignment for dynamic scheduling tasks. For this reason, the way in which tasks and jobs are ordered for their allocation and later scheduling is an open issue. Usual priority orders are the earliest deadline first (Liu and Layland, 1973) and deadline (Audsley et al., 1991) or rate monotonic (Liu and Layland, 1973). In this paper, six different priority orders are considered, which are used for the allocation of jobs to processors in the cases that whenever no *critical* jobs are pending execution. If a *critical* job appears, the priority order is altered to schedule it instead. The priority orders considered are the following:

Decreasing mandatory utilization (H1): With this priority order the jobs with a higher mandatory utilization factor are allocated first. The rationale behind this consists in giving priority to those tasks that demand more processor bandwidth and leaving those that demand less for later.

Increasing mandatory execution time (H2): In this case, the order also includes the mandatory part too. Jobs with short execution times are scheduled first so jobs with longer execution times can be accommodated later.

Decreasing total utilization (H3): In contrast to the previous cases, the mandatory plus optional processor bandwidth demand is used to order the jobs. The rationale is, like in the first case, that allocating the *high demand* jobs to the processors first will facilitate the assignment of the low demand ones later.

Decreasing mandatory execution time (H4): Like in the third case, the mandatory execution times are used to order the allocation. However, instead of allocating the small jobs first, the big ones are allocated first. Like in the previous case, the idea is that allocating the big jobs first facilitates the allocation of small ones later.

Increasing periods (H5): Used in monoprocessor systems, the rate monotonic priority order is one of the most used in real-time systems.

Earliest deadline first (H6): Like in the previous case, the use of this priority discipline is common in real-time systems and it is optimal in the case of monoprocessor systems.

Algorithm 1: Pseudocode for the first stage

```
Data: $\mathcal{S}$,$\Pi$
Result: Jobs allocation to processors
Tasks are ordered according to some priority discipline. $H \leftarrow$ lcm for all
the periods;
$h \leftarrow 1$ ;
while $h \leq H$ && no processors are collapsed do
    forall the $\pi_j$ critical processor with $\rho_i^v$ critical do
        push $\rho_i^v$ at the head of the pending jobs list for $\pi_j$:
            $\bar{L}(j,h) \leftarrow \rho_i^v \bar{L}(j,h-1) \setminus \rho_i^v$;
    end
    forall the $\rho_i^v$ urgent job do
        assign $\rho_i^v$ to the non critical processor with the lowest charge $\pi_j$:
            $L(j,h) \leftarrow \rho_i^v L(j,h-1)$;
            $\bar{L}(j,h) \leftarrow \rho_i^v \bar{L}(j,h-1)$;
            $m_i^v(h-1) \leftarrow 0$;
    end
    if there are more urgent jobs than non-critical processors then
        Stop. It is not possible to schedule the system.
    end
    forall the $\rho_i^v$ such that $h = a_i^v$ and is not still allocated do
        if there is a non-critical processor executing a job with lower
        priority than $\rho_i^v$, $\pi_j$ then
            allocate $\rho_i^v$ a $\pi_j$:
                $L(j,h) \leftarrow \rho_i^v L(j,h-1)$;
                $\bar{L}(j,h) \leftarrow \rho_i^v \bar{L}(j,h-1)$;
                $m_i^v(h-1) \leftarrow 0$;
        end
    end
    forall the processor $\pi_j$ such that $\bar{L}(j,h)$ is empty do
        if there is a job not allocated with $h \geq a_i^v$ then
            choose the highest priority $\rho_i^v$ and assign $\rho_i^v$ to $\pi_j$:
                $L(j,h) \leftarrow \rho_i^v L(j,h-1)$;
                $\bar{L}(j,h) \leftarrow \rho_i^v$;
                $m_i^v(h-1) \leftarrow 0$;
        end
    end
    forall the processor such that $\bar{L}(j,h)$ is not empty do
        execute a slot of the first pending job $\rho_i^v$:
            $m_i^v(h) \leftarrow m_i^v(h-1)+1$;
        if $m_i^v(h) = m_i$ then
            $\bar{L}(j,h) \leftarrow \bar{L}(j,h-1) \setminus \rho_i^v$;
        end
    end
    $h \leftarrow h+1$ ;
end
if $h \leq H$ then
    It is not possible to schedule the system.
end
else                                  16
    Return for each processor $\pi_j$ $L(j,H)$;
end
```

5.1.2. Phase two

When this initial stage is finished, there is a feasible allocation of jobs to processors, the slack or empty slots available within each processor is computed, and in the second stage the empty slots are used for maximizing the reward of the system by choosing a different power mode or executing optional parts. This second step is done with the ILP model for each processor but with the jobs already allocated. Although each of these subproblems are a reduced version of the general problem (i.e., just one processor and a subset of jobs), the ILP algorithm still takes time to find feasible solutions. Therefore, a sequential procedure is performed based on the fact that a solution with f frequency levels is also a solution for the case with $f + 1$ levels. Starting with one level of frequency, the ILP model is solved within a CPU time limit, considering the solution for f frequency levels as an initial lower bound for the model with $f + 1$ frequency levels.

6. Experimental evaluation

In this section, several experiments are presented to evaluate the model and the methods proposed before. The ILP and heuristics approaches are used to solve three different sets of synthetic problems generated from Aydin et al. (2001), Bini and Buttazzo (2005), and Gai et al. (2003). For the first two sets, 100 systems with different utilization factors and combinations of mandatory and optional parts were generated. For all the cases, two homogeneous processors were considered with four power modes or frequencies available for each one. The reward for executing at different frequencies were 0, 1, 2, and 3 for 100%, 75%, 50%, and 25% of the nominal speed, respectively. The cost for changing the power mode was set equal to 8. The experiments are divided in two parts. In the first one, we evaluate the ability of the algorithms to find a solution, the time required to do that, and the solution qualities. In the second part, we evaluate the influence of the α and β parameters in the kind of obtained schedule. To do that, the relation between them was changed and the characteristics of the solutions are analyzed.

All the algorithms were coded in C++ using CPLEX 12.4 optimizer with the default configuration for the Branch and Cut algorithm. The experiments were run on an Intel i7 of 3.4 GHz and 16 GB of RAM.

6.1. Algorithms performance

To evaluate the performance of different algorithms introduced before, we used the upper bound of the optimum value provided by the ILP solver. This value is used as a reference for the optimum to measure the quality of the solution found with the algorithms proposed. The gap is defined as the relative distance to it.

As was explained in Section 4, to reach the optimal solution, the ILP model with all the power modes available may take a long time even for middle size problems. In order to reduce this, we have programmed the ILP solver to incorporate different frequency levels one at a time. Each step was limited to 600 seconds so in the worst case the solver runs for 2400 seconds. However, in the first level the optimal solution was always found in just a few seconds and the total time used was closer

Table 2
Results for eight task first set

Method	Average gap	σ Gap	Average time	σ Time	% Solved
ILP	2.88	2.63	1138.20	504.26	100
H1	3.91	4.78	113.18	54.1	99
H2	3.86	5.03	102.72	47.8	100
H3	3.93	5.01	114.22	55.33	100
H4	3.52	4.5	110.93	56.10	90
H5	4.81	6.57	95.63	48.75	100
H6	3.88	4.76	113.70	55.16	99
HILP	1.98	1.89	116.10	68.4	100

Table 3
Results for 50 task first set

Method	Average gap	σ Gap	Average time	σ Time	% Solved
H1	1.83	1.75	246.20	26.87	100
H2	1.19	1.73	218.86	32.59	100
H3	1.80	1.73	245.00	27.46	100
H4	1.87	1.75	245.97	26.11	95
H5	1.66	1.84	236.71	31.44	100
H6	1.85	1.68	244.34	26.38	100
HILP	0.23	0.72	335.92	138.12	100

to 1800 seconds. In the case of the heuristics, the second stage that uses the ILP solver to obtain the optional/frequencies allocation were run for 40 seconds in each step. Since four power modes were considered, the total time dedicated to solve the ILP model in these cases was limited to 160 seconds for each processor.

First set: These problems were generated following the model presented in Bini and Buttazzo (2005). Each system has eight tasks and the total utilization factor was between 0.41 and 1.98. Reward values for the optional slots were also randomly generated. Table 2 presents the results obtained for this case. The first column refers to the method used, the second and third present the average gap and average standard deviation, the next two reports the average time required to achieve that gap and the average standard deviation, and the last column the percentage of instances solved.

The HILP approach provides the best solution with a gap below 2% in less than two minutes of computations. All the greedy heuristics provide gaps below 5% with average computation times of 107 seconds. The ILP solution is the second best but demands a long time to reach that value. It is also important to remark the fact that most of the algorithms find a solution for all the cases and only H4 has a poor performance in this aspect as it fails to find a solution in 10% of the problems.

We notice that the percentage of systems solved with the heuristics methods is quite significant. So it is worth considering the use of the simplified method for this kind of problems, which produces good results but almost 10 times faster.

In Table 3, we have also experienced with larger instances (50 tasks), looking for identifying if the behavior of the algorithms follows the same pattern. It is worth mentioning that ILP could not

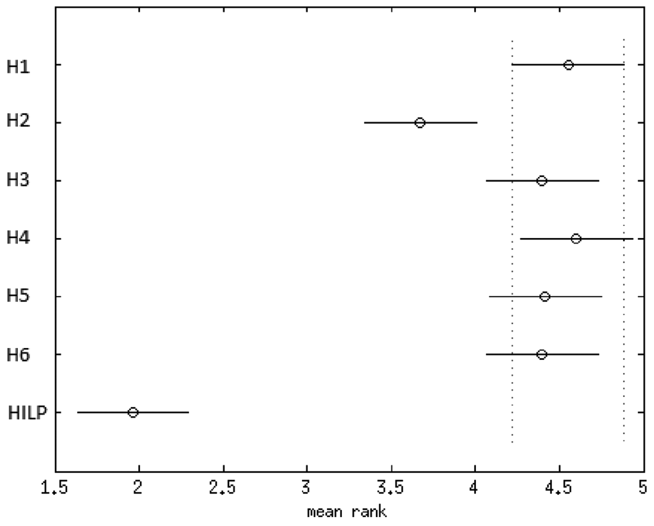

Fig. 5. Friedman test for 50 task first set.

obtain solutions within a reasonable time. Then, ILP was not considered for these instances and the gap is calculated as the relative distance to the best solution found by any of the applied methods for the respective instance.

Notice that HILP remains being the best option providing in most of the cases the best solution, although the computational time increased slightly compared to the other procedures. Furthermore, procedure H4 is still the one that fails to find solutions in more instances.

A comparison among the algorithms based on the average gap or average time could be influenced by a few very good or very bad results. Therefore, a more detailed analysis to observe if the differences are statistically significant is made by applying a nonparametric Friedman test (Friedman, 1937) with p value 0.05 and the null hypothesis that says that all the algorithms are equal.

For each algorithm, the mean ranking is computed. Given an instance, the ranking $rank_k$ of the observed result for algorithm k is calculated, assigning to the best of them the ranking 1, and to the worst the ranking 7. Then, an average measure is obtained from the rankings of each method for all the instances. Clearly, the lower the ranking, the better the associated algorithm.

Figure 5 shows the mean ranking of the methods with the confidence intervals. According to the Friedman test, two algorithms are significantly different in statistical terms if their intervals in the corresponding axis do not overlap. As it is shown, HILP gets the best ranking and the test finds significant performance differences with regard to all other algorithms, followed by H2. Moreover, the plot suggests that there are not significant difference among the other algorithms.

In order to confirm the conclusions about the performance of HILP, we also apply Wilcoxon signed rank test (Wilcoxon, 1945) for detecting performance differences between the best ranked algorithm (HILP) and the remainder with 0.05 as the significance factor. We tested HILP with each other. The null hypothesis is $\mu_i = \mu_{HILP}$, where μ_i is the average gap obtained by each of the first six methods. The alternate hypothesis used is, $\mu_i > \mu_{HILP}$. Table 4 exhibits the values of Wilcoxon statistic. Whenever the p-value is greater than the critical value of Wilcoxon statistic for

Table 4
Wilcoxon test for 50 task first set

	HILP		
	Sum pos rank	Sum neg rank	p-Value
H1	2850	0	0
H2	1968	310	0
H3	2797.5	52.5	0
H4	2850	0	0
H5	2927.5	153.5	0
H6	2831.5	18.5	0

Table 5
Results for 20 task second set

Method	Average gap	σ Gap	Average time	σ Time	% Solved
ILP	3.45	2.85	1222.67	649.56	100
H1	2.37	1.82	148.51	79.15	100
H2	2.44	1.66	152.90	73.79	95
H3	2.46	1.82	147.61	80.39	100
H4	2.38	1.79	154.06	76.21	84
H5	2.63	1.99	147.33	76.11	100
H6	2.46	1.92	145.31	78.49	100
HILP	2.09	1.78	149.71	79.95	100

the chosen significance level, we should reject the null hypothesis and infer that μ_i is greater than μ_{HILP}, which means that heuristic HILP outperforms heuristic i. From Table 4, we can see that, HILP outperforms the remaining procedures, which confirms the conclusions drawn earlier.

Second set: In this case, two sets (one with 20 tasks and one with 50 tasks) were generated for each system with utilization factors in the range of 0.44–2.354. Tasks periods were generated following the suggestions presented in Gai et al. (2003), where the authors described the general relations among tasks in embedded systems like the ones used to control the power train in cars. In Table 5 the results are shown for 20 task instances, where the percentage average gap, average standard deviation gap, average time used to find the solution, average standard deviation time, and the percentage of instances solved are presented.

Like in the previous set of problems, HILP is the method that has the lowest gap, gets to that point in less than three minutes, and solves all the problems. The ILP solution has the highest gap and longest resolution time but solves all the instances. Heuristics are in-between but it is remarkable the poor performance of H4, which only solves 84% of the instances.

Table 6 reports the results for instances with 50 tasks. It can be seen that $HILP$ is the unique procedure capable to solve all the instances and obtains the best solution in most of the cases. Moreover, the low performance of H4 is accentuated since it could only solve 34% of the instances.

As we did in the previous case to confirm the conclusions, Friedman and Wilcoxon tests were applied. We discard H4 from the comparison due to its low performance in the number of solved instances.

Table 6
Results for 50 task second set

Method	Average gap	σ Gap	Average time	σ Time	% Solved
H1	0.92	1.13	348.56	280.50	81
H2	1.70	2.12	333.51	265.55	92
H3	0.88	1.07	367.76	278.77	79
H4	0.40	0.57	496.37	252.55	34
H5	1.25	1.40	314.90	269.04	98
H6	1.02	1.19	308.06	275.06	99
HILP	0.10	0.40	308.86	274.34	100

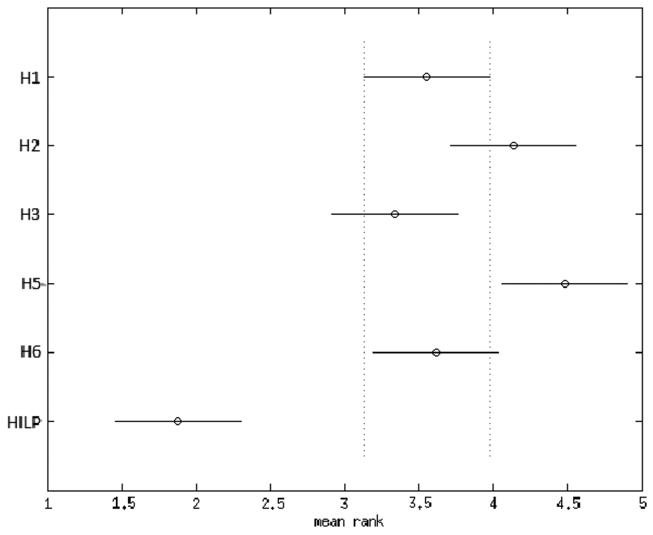

Fig. 6. Friedman test for 50 task second set.

Table 7
Wilcoxon test for 50 task second set

	HILP		
	Sum pos rank	Sum neg rank	p-Value
H1	2517	258	0
H2	3666	250	0
H3	2285	271	0
H5	3434	221	0
H6	3748	168	0

Figure 6 shows the Friedman mean rank plot, which confirms *HILP* has the best performance and it cannot be inferred significant difference among the other algorithms.

Table 7 shows the p values of Wilcoxon test when comparing *HILP* with each of the algorithms. All of them are below 0.05, which makes evident that *HILP* outperforms all the algorithms.

Table 8
Third set of problems

Task	P_i	$m_i + o_i$	$f_i^1(o)$	$f_i^2(o)$	$f_i^3(o)$
1	20	10	$15(1-e^{-t})$	$7\ln(20t+1)$	$5t$
2	30	18	$20(1-e^{-3t})$	$10\ln(50t+1)$	$7t$
3	40	5	$4(1-e^{-t})$	$2\ln(10t+1)$	$2t$
4	60	2	$10(1-e^{-0.5t})$	$5\ln(5t+1)$	$4t$
5	60	2	$10(1-e^{-0.2t})$	$5\ln(25t+1)$	$4t$
6	80	12	$5(1-e^{-t})$	$3\ln(30t+1)$	$2t$
7	90	18	$17(1-e^{-t})$	$8\ln(8t+1)$	$6t$
8	120	15	$8(1-e^{-t})$	$4\ln(6t+1)$	$3t$
9	240	28	$8(1-e^{-t})$	$4\ln(9t+1)$	$3t$
10	270	60	$12(1-e^{-0.5t})$	$6\ln(12t+1)$	$5t$
11	2160	300	$5(1-e^{-t})$	$3\ln(15t+1)$	$2t$

Table 9
Third set with exponential reward function

	$f_i^1(o)$									
	0		25		40		60		80	
Method	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
ILP	49.28	1385.72	32.04	1891.34	29.01	1902.53	51.93	2045.19	3.58	1927.14
H1	1.05	162.64	1.17	102.39	14.81	165.42	29.89	203.95	NS	
H2	1.05	162.57	1.96	129.23	16.30	170.04	NS		NS	
H3	1.02	163.09	15.43	123.22	16.63	168.50	29.89	203.30	NS	
H4	1.02	162.86	NS		NS		NS		NS	
H5	1.02	162.84	15.46	123.56	29.66	167.41	29.16	204.14	NS	
H6	1.02	162.81	2.31	124.03	14.41	137.79	31.30	206.71	NS	
HILP	0.74	145.38	1.16	203.08	22.58	261.93	32.52	473.73	3.58	399.93

Third set: In Aydin et al. (2001), the authors introduce a set of problems to evaluate their method for scheduling optional parts. The system proposed has 11 tasks. Each task has a total execution time that includes both the time required by the mandatory and optional parts and the evaluation is made considering different combinations of mandatory and optional parts. The experiments were done for 0%, 25%, 40%, 60%, and 80% of the demand as mandatory and the rest of the demand as optional. Each task has three reward functions, all of them are concave. In Table 8 the system is presented.

As the total utilization factor for this problem is 2.24 it is not possible to obtain a schedule for large mandatory proportions of jobs. Tables 9–11 present the results for the cases of 0%, 25%, 40%, 60%, and 80% of mandatory charge for the three reward functions. For each one the gap and time needed with each of the methods are shown.

From the results it is clear that H4 is not capable of solving the problem with mandatory charge. Heuristics H1 and H3 can solve the schedule except for the case of 80% mandatory part and H2 is not able to solve it in the cases of 60% and 80%. Only ILP and HILP are capable of solving all the instances. Like in the previous set of problems, the HILP algorithm is the one that produces the best

Table 10
Third set with logarithmic reward function

	$f_i^2(o)$									
	0		25		40		60		80	
Method	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
ILP	45.87	2163.44	40.91	1951.99	34.42	1920.11	21.87	1934.83	0.00	217.76
H1	3.58	223.96	23.52	164.48	25.49	204.43	26.03	240.95	NS	
H2	3.58	224.01	20.46	201.90	25.76	226.10	NS		NS	
H3	3.58	224.11	23.59	213.43	25.88	240.86	26.03	240.98	NS	
H4	3.58	224.8	NS		NS		NS		NS	
H5	3.58	224.47	23.53	163.91	25.17	201.99	23.75	241.01	NS	
H6	3.58	224.97	19.86	163.98	25.59	204.95	25.59	240.83	NS	
HILP	9.59	473.71	24.12	284.21	34.42	363.34	21.87	407.94	0.03	616.116

Table 11
Third set with linear reward function

	$f_i^3(o)$									
	0		25		40		60		80	
Method	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
ILP	21.72	2409.23	17.65	1960.86	17.64	1979.44	8.63	1984.42	0.00	238.72
H1	20,55	241.47	22.03	240.74	28.28	236.23	18.39	224.42	NS	
H2	20.55	242.4	20.67	246.22	22.16	242.05	NS		NS	
H3	20.55	242.96	25.88	229.65	25.66	241.54	18.39	225.22	NS	
H4	20.55	242.78	NS		NS		NS		NS	
H5	20.55	242.55	23.50	241.72	25.34	240.03	14.01	243.82	NS	
H6	20.55	242.86	27.69	241.49	28.73	241.42	19.64	232.64	NS	
HILP	19.30	735.88	17.65	664.42	17.64	500.93	8.63	403.83	0.00	613.06

results considering the gap, the time needed to achieve that gap, and the number of problems solved. This behavior is even more evident when the reward function is linear. In the case of exponential and logarithmic functions, for some problems, the greedy heuristics show a slightly higher performance.

The overall evaluation shows that ILP is not convenient for finding a feasible schedule for the mandatory/optional reward based on energy considerations. Although a solution is always found in the end, it requires a long time. Greedy heuristics fail to find a feasible allocation when the mandatory utilization is high. The *HILP* method provides solutions to all the cases with better gaps. This approach, although not polynomial, resulted in a very good alternative as the time demanded was rather small. In general, the greedy heuristics proposed do not guarantee that a feasible schedule will be found in every case and they do not show remarkable different performances among them, but clearly H4 is the worst option.

6.2. *Sensibility of the schedule to α and β*

In this section, the sensitivity of the schedule to the set of α and β parameters is evaluated. The selection of these values influences the kind of schedule that is obtained as it is going to be shown.

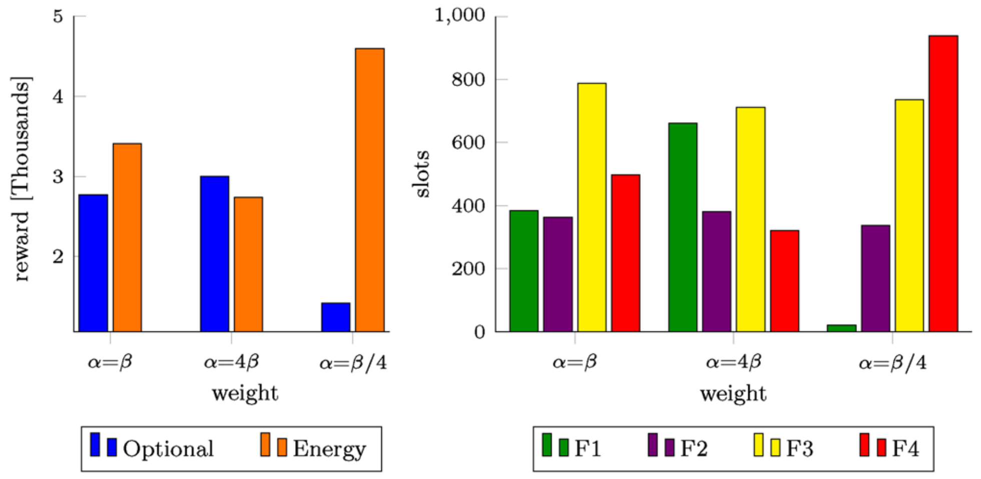

Fig. 7. First set of problems.

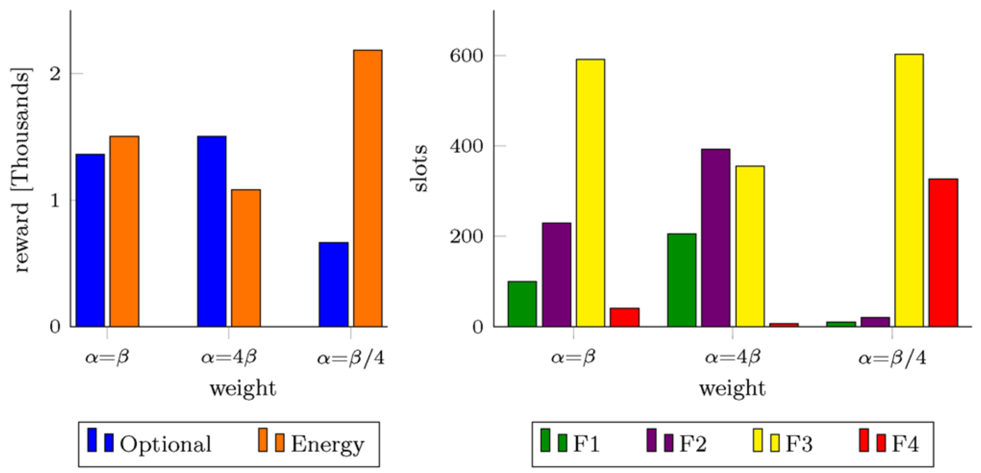

Fig. 8. Second set of problems.

In fact, these parameters determine if the schedule is oriented to save energy by reducing the power mode/frequency of the processor, to improve the quality of the tasks by executing more optional parts, or to a trade-off solution in which optional parts are executed while reducing the energy demand. The experiment was done with the HILP method as it is the one that produces the best results. Figures 7–9 show for the three different sets of problems, respectively, three different relations $\alpha = \beta$, $\alpha = 4\beta$, and $\alpha = \beta/4$. For each case, the average rewards obtained from the execution of optional parts and that obtained from the execution at lower frequencies are presented in the left while in the right the average number of slots executed at the different power modes/frequencies is presented. For the last set of problems, the overall average was made for the different reward functions and combinations of mandatory and optional parts.

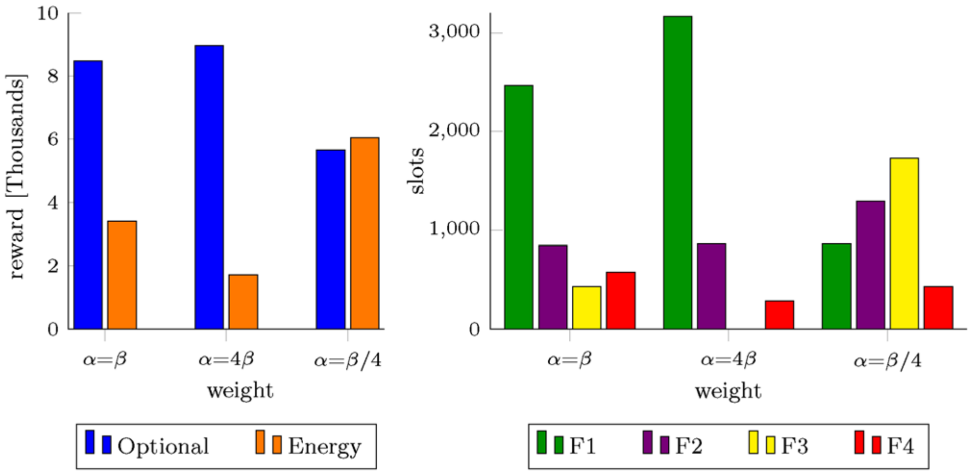

Fig. 9. Third set of problems.

The results show an accurate matching between the expected schedule orientation (optional or energy-aware) and the selection of values for α and β. This shows how different schedules of a system can be obtained for different scenarios. For example, a system working with solar panels during daylight and batteries in the night can have two different schedules that are adjusted to the needs of each situation by changing the relation between these parameters. When the energy consumption has major importance, fewer optional tasks are scheduled and greater number of slots are used at frequencies with lower power consumption in the solutions obtained by the algorithm. On the other hand, when the power consumption is not significant, the scheduler takes advantage of the optional tasks and the processors work at greater frequencies.

The decision about the weight for prioritizing one objective from the other is made by the user. An efficient frontier curve could help the decision maker to examine the trade-off between the two objectives. The idea is that the user analyzes if substantial improvement in energy consumption implies a degradation in reward of execution of optional tasks. In other words, if a large gain in processing optional tasks can be achieved with an acceptable energy consumption. To illustrate the point, we consider one instance with 20 tasks from the second set and plot the solutions achieved by executing HILP-independent optimizations, by varying each time weights $\alpha, \beta \in [0, 1], \alpha + \beta = 1$ in steps of 0.05. The results are presented in Table 12 and Fig. 10.

A first evaluation shows that all solutions are nondominated, that is none solution is better on both objectives than other solutions. The heuristic is capable of providing a set of possible scheduling with diverse function values.

For this instance, if we consider the greatest energy savings (4800) and the greatest benefit by running electives (7072), we can see that the variability of savings ranges up to 53% and 57%, respectively. Moreover, improvement in one of the two objectives is accompanied by a loss of the same order of percentage in the other. That is, significant energy savings cannot be achieved without sacrificing optional processing tasks and vice versa.

This type of analysis can help the scheduler to make his decision in choosing one of the solutions. For example, a good compromise between both functions could be the solutions $(6268, 2880)$ or $(6037, 2967)$.

Table 12
Solutions by varying weights α, β in steps of 0.05

	Reward by optional tasks	Reward by frequency reduction	No. of slots F1	No. of slots F2	No. of slots F3	No. of slots F4
1	7072	0	1916	4	0	0
2	7072	1708	108	1812	0	0
3	7066.5	1747	117	1803	0	0
4	7060	1804	76	1844	0	0
5	7040.5	1836	44	1876	0	0
6	7020.5	1854	42	1878	0	0
7	7007	1897	7	1913	0	0
8	6958.5	1920	0	1920	0	0
9	6727	2274	6	1514	400	0
10	6356	2780	60	900	960	0
11	6334.5	2856	8	952	960	0
12	6268	2880	0	960	960	0
13	6037	2967	4	809	1107	0
14	5526.5	3504	0	320	1600	0
15	5256	3840	0	0	1920	0
16	4894	3984	0	0	1760	160
17	4556	4240	0	0	1614	306
18	4390	4290	0	0	1543	377
19	3414.5	4329	0	0	1407	513
20	3322.5	4448	0	0	1280	640
21	0	4800	0	0	960	960

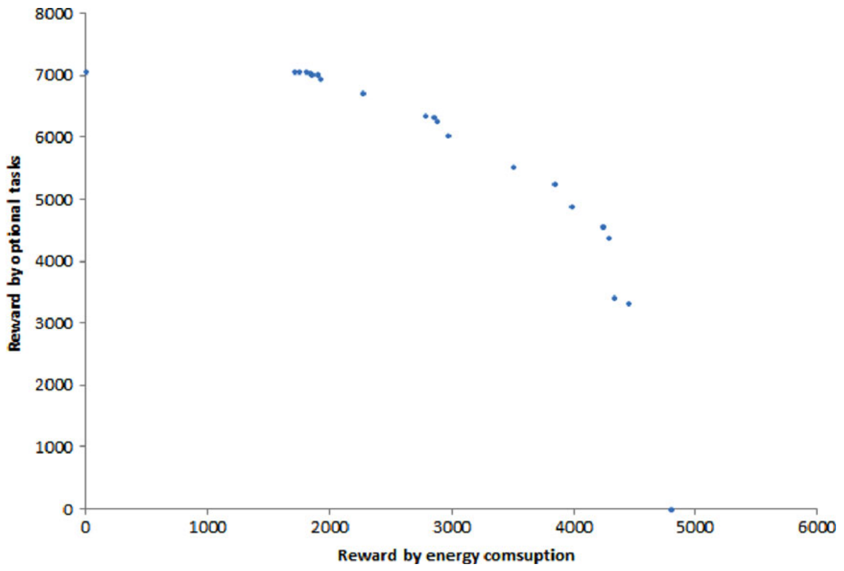

Fig. 10. Energy consumption and reward of optional tasks trade-off curve.

7. Conclusions

In this paper, the scheduling of energy-aware reward-based mandatory/optional tasks with time constraints in homogeneous multicore systems is shown to be an NP-hard problem in the strong sense as it is a combination of two problems already proved to be NP-hard (scheduling of restricted migration jobs in multiprocessors systems on one side and mandatory/optional reward-based scheduling with arbitrary reward functions on the other). Seven different algorithms have been introduced and evaluated using three sets of problems taken from the literature on real-time systems. The experiments show that a pure ILP approach requires a long time compared with an oriented ILP search like the one proposed in the HILP method. The heuristics introduced show good performance with a clear advantage for those using rate monotonic and earliest deadline first for ordering the jobs while distributing them among the processors.

The model proposed has two tuning parameters that can be used to orient the schedule according to certain conditions. As it was shown in the experiments, variations in the relation between these parameters produce different results improving the quality of the response (more optional parts executed), reducing the energy consumption (lower execution frequencies), or optimizing both in a trade-off.

References

Albers, S., Antoniadis, A., Greiner, G., 2015. On multi-processor speed scaling with migration. *Journal of Computer and System Sciences* 81, 7, 1194–1209.

Andersson, B., Jonsson, J., 2000. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In Proceedings of the Seventh International Conference on Real-Time Computing Systems and Applications, Cheju Island, Korea, pp. 337–346.

Andersson, B., Jonsson, J., 2003. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In EuroMicro Conference on Real-Time Systems, Porto, Portugal, pp. 33–40.

Astrom, K.J., Wittenmark, B., 1994. *Adaptive Control* (2nd edn). Addison-Wesley Longman, Boston, MA.

Audsley, N., Burns, A., Richardson, M., Wellings, A., 1991. Real-time scheduling: the deadline-monotonic approach. In Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA, pp. 133–137.

Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P., 2001. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers* 50, 2, 111–130.

Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P., 2004. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers* 53, 5, 584–600.

Baruah, S., Carpenter, J., 2003. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In Proceedings of the 15th Euromicro Conference on Real-Time Systems, Porto, Portugal, pp. 195–202.

Bini, E., Buttazzo, G., 2005. Measuring the performance of schedulability tests. *Real-Time Systems* 30, 1–2, 129–154.

Chan, H.-L., Lam, T.-W., Li, R., 2011. Tradeoff between energy and throughput for online deadline scheduling. *Sustainable Computing: Informatics and Systems* 1, 3, 189–195.

Chung, J., Liu, J., Lin, K., 1990. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers* 39, 9, 1156–1174.

Dertouzos, M., Mok, A., 1989. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering* 15, 12, 1497–1506.

Fisher, N.W., 2007. The multiprocessor real-time scheduling of general task systems. PhD thesis, University of North Carolina at Chapel Hill.

Friedman, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association* 32, 675–701.

Gai, P., Di Natale, M., Lipari, G., Ferrari, A., Gabellini, C., Marceca, P., 2003. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, pp. 189–198.

Garey, M., Johnson, D., 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Han, X., Lam, T., Lee, L., To, I., Wong, P., 2010. Deadline scheduling and power management for speed bounded processors. *Theoretical Computer Science* 411, 40–42, 3587–3600.

Heath, S., 2002. *Embedded Systems Design* (2nd edn). Butterworth-Heinemann, Newton, MA.

Hong, K., Leung, J., 1988. On-line scheduling of real-time tasks. In *R Proceedings of the Real-Time Systems Symposium*, Huntsville, AL, pp. 244–250.

IBM, 2011. Cplex optimizer for z/os. http://pic.dhe.ibm.com/infocenter/cplexzos/v12r4/index.jsp.

Karp, R., 1972. Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds) *Complexity of Computer Computations*, The IBM Research Symposia Series, Springer, New York, pp. 85–103.

Kumar P., Palani, S., 2012. A dynamic voltage scaling with single power supply and varying speed factor for multiprocessor system using genetic algorithm. In International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME), Salem, Tamilnadu, pp. 342–346.

Lam, T., Lee, L., To, I., Wong, P., 2008. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '08, ACM, New York, pp. 256–264.

Liu, C., Layland, J., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20, 1, 46–61.

Liu, J.W.-S., Lin, K.-J., Shih, W.-K., Yu, A.C.-S., Chun, C., Yao, J., Zhao, W., 1991. Algorithms for scheduling imprecise computations. *IEEE Computer* 24, 5, 58–68.

Rizvandi, N.B., Taheri, J., Zomaya, A.Y., 2011. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *Journal of Parallel and Distributed Computing* 71, 8, 1154–1164.

Shih, W., Liu, J., 1995. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers* 44, 3, 466–471.

Stankovic, J., 1988. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer* 21, 10, 10–19.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. *Biometrics* 1, 80–83.

Zhang, W., Xie, H., Cao, B., Cheng, A., 2014. Energy-aware real-time task scheduling for heterogeneous multiprocessors with particle swarm optimization algorithm. *Mathematical Problems in Engineering* 2014, Article ID 287475.

Zhang, Y., Guo, R., 2014. Power-aware fixed priority scheduling for sporadic tasks in hard real-time systems. *Journal of Systems and Softwares* 90, 128–137.