Journal of Universal Computer Science, vol. 25, no. 4 (2019), 390-417
submitted: 15/10/18, accepted: 16/4/19, appeared: 28/4/19 © *J.UCS*

Scheduling Mandatory-Optional Real-Time Tasks in Homogeneous Multi-Core Systems with Energy Constraints Using Bio-Inspired Meta-Heuristics

Matías Micheletto
(Dep. Ing. Eléctrica y Computadoras - ICIC
Universidad Nacional del Sur, CONICET
Bahía Blanca, Buenos Aires, Argentina
matias.micheletto@uns.edu.ar)

Rodrigo Santos
(Dep. Ing. Eléctrica y Computadoras - ICIC
Universidad Nacional del Sur, CONICET
Bahía Blanca, Buenos Aires, Argentina
ierms@uns.edu.ar)

Javier Orozco
(Dep. Ing. Eléctrica y Computadoras - ICIC
Universidad Nacional del Sur, CONICET
Bahía Blanca, Buenos Aires, Argentina
jorozco@uns.edu.ar)

Abstract: In this paper we present meta-heuristics to solve the energy aware reward based scheduling of real-time tasks with mandatory and optional parts in homogeneous multi-core processors. The problem is NP-Hard. 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 power-mode is introduced together with a set of constraints that guarantee the real-time performance of the system. The meta-heuristics are the bio-inspired methods Particle Swarm Optimization and Genetic Algorithm. Experiments are made to evaluate the proposed algorithms using a set of synthetic systems of tasks. As these have been used previously with an Integer Lineal Programming approach, the results are compared and show that the solutions obtained with bio-inspired methods are within the Pareto frontier and obtained in less time. Finally, precedence related tasks systems are analyzed and the meta-heuristics proposed are extended to solve also this kind of systems. The evaluation is made by solving a traditional example of the real-time precedence related tasks systems on multiprocessors. The solutions obtained through the methods proposed in this paper are good and show that the methods are competitive. In all cases, the solutions are similar to the ones provided by other methods but obtained in less time and with fewer iterations.

Key Words: Reward base Scheduling; Multicore Systems; Energy Handling.
Category: D.1.5, D.4.0, D.4.1, D.4.7, D.4.8, I.1.2, I.6, J.6

1 Introduction

Real-time tasks are those that have to be executed correctly before a certain deadline [Stankovic 1988]. Several scheduling policies and optimal algorithms have been proposed for mono-processor systems [Liu and Layland 1973]. However, the actual trend in microprocessors architecture implements homogeneous multicore processors with independent cache memories and a common main memory. For this, real-time tasks scheduling is still an open issue. Some specific architectures propose heterogeneous multicore processors in which certain tasks are performed by a dedicated core, for example a digital signal processor (DSP) unit. In these cases, the scheduling can revert to a single core processor as the task execution in such unit is mandatory [Zhang et al. 2014].

There are three main approaches for scheduling real-time tasks in multicore systems. In the first one, called partitioned multiprocessor real-time scheduling, tasks are statically allocated to a particular core. No migrations are allowed between cores and each one should have a particular scheduler. In the second one, called global multiprocessor real-time scheduling, a single scheduler selects the higher priority task to run on any available core. Tasks may be preempted at any point and continue their execution on a different core. In this case, a full migration policy is adopted. Between both approaches, a restricted migration one is proposed in which tasks may execute different instances on different cores. However, once an instance has begun its execution in a core it finishes its execution in it. A new instance may execute in the same core or in a different one [Fisher 2007].

"Anytime algorithms" also known as imprecise computation, are those that improve the quality of the result when they execute for longer periods of time. It is the case of finding the roots of a function, image and speech processing in multimedia applications and navigation support among others [Aydin et al. 2001, Mo L. et al. 2018]. These algorithms have the particularity that after some iterations, a quality threshold is achieved that may be enough for the computation purpose; this is the mandatory part of the job. After achieving this threshold, the algorithm may optionally continue its execution improving the quality of the result. In [Liu et al. 1991], the authors introduced the mandatory/optional task model scheduling. The mandatory part guarantees the minimum quality in the result and should be completed before the deadline while the optional part improves the quality of the solution. Each optional part has associated a reward that is used by the scheduler to select the best one each time. In [Aydin et al. 2001] it is proved that under certain reward functions this is a NP-hard problem and they proposed an optimization mechanism to compute the best schedule.

Most processors can control the energy consumption by setting the voltage and frequency of operation (power-mode). If the scheduler is capable of modifying these parameters to reduce the power demand dynamically, it is said to be an energy-aware scheduler [Santos et al. 2012]. The power demand is the sum of the dynamic and static powers. The first one is proportional to the square of the frequency f, the capacitance C and the source voltage V while the second one is proportional to the amount of transistors in the processor [Pouwelse et al. 2001]:

$$P = P_d + P_s = Cf^2V + P_s. \tag{1}$$

When a processor reduces its power-mode, the energy consumption is reduced quadratically while the time necessary to execute the tasks is enlarged linearly. This imposes a trade-off that should be considered if real-time requirements are to be satisfied. In this paper we propose two methods to answer the following: How much improvement in the reward of a task is possible while satisfying all deadline restrictions with the minimum energy consumption? Some solutions have been provided for mono-processors systems [Santos et al. 2004] but, for multicore ones is still an open issue.

In a previous paper, [Mendez-Diaz et al. 2017] the authors proposed an Integer Linear Programming (ILP) optimal solution or exact solution for this problem. Based on the model they presented there, here we propose two methods based on bio-inspired meta-heuristics: genetic algorithms (GA) and particles swarm optimization (PSO) for solving the energy-aware mandatory/optional real-time multicore scheduling problem. We present a Pareto front analysis to evaluate the distance of the solution to a global optimum. The solution maximizes the reward of the system while keeping the energy consumption as low as possible. As far as we know, there is no previous work on mandatory/optional energy-aware scheduling analysis for homogeneous multicore real-time systems with these meta-heuristics. The experimental evaluation shows that the solutions obtained are within the Pareto Frontier and are found in a short time with just a few iterations.

2 Previous work

“Anytime algorithms” have been used for many years for different kind of iterative applications like the computation of polynomial roots, multimedia processing and autonomous robotic navigation among others [Shih and Liu 1995]. In [Chung et al. 1990] the authors proposed the scheduling of mandatory parts following traditional priority disciplines. Optional parts are scheduled in the background. There is no computation of slack time. In [Aydin et al. 2001] it is shown that the best results are obtained when the slack time is used to schedule the optional parts that provide more reward at that moment. As the authors use integer lineal programming techniques, the reward functions are restricted to be continuously differentiable. Although they proposed an extension for multiprocessor systems, they do not indicate the scheduling policy and the way tasks execute in the processors.

In [Santos et al. 2004, Santos et al. 2005] the use of slack time is proposed to provide a reward-based dynamic scheduler and a fault-tolerant energy-aware one respectively. Both approaches compute the possibility of advancing idle slots to execute optional parts or enlarged execution time by reducing the frequency of operation or repeating a task.

The extensive work presented in [Wanli and Chakraborty 2016] summarize different "approaches in automotive control systems that take implementation resources into consideration", showing that optimal resources management is a subject of interest in engineering areas. In [Santos et al. 2008], with a different approach, the authors analyze priority inversions and blocking in real-time tasks with precedence constraints.

In [Cheng and Wu 2018] authors show the case of implementing real time algorithm for distributed systems applied to the case of large and high-performance computing systems.

In [Likhachev et al 2008] and [Feller and Ebenbauer 2017], new techniques or methods improvements are presented with anytime algorithm approaches where it is taken into consideration time restrictions, and efficient use of time as in the case of real-time systems. In [Greco et al, 2011] authors address the case of implementing anytime control algorithms for linear systems in embedded platforms with real-time constraints. The authors formulated an ILP model to determine the rules for a stochastic scheduling approach.

In [Aydin et al. 2004] the authors introduce a static off-line method to compute the optimum frequency to execute each task assuming for each one the worst case execution time. It is proved that the problem of finding an optimal schedule with energy restrictions is equivalent to solving the scheduling of mandatory/optional tasks with concave reward functions.

In [Hong and Leung 1988], [Dertouzos and Mok 1989] and [Fisher 2007], it is proved that there is no optimal on-line scheduling discipline for multiprocessors. In [Baruah and Carpenter 2003], a feasibility analysis for multiprocessors systems is presented for the case of homogeneous processors with sporadic tasks and it is proved to be NP-hard.

In [Zhang et al. 2014] an heuristic algorithm based on the particle swarm is proposed for the scheduling of real-time tasks in heterogeneous multiprocessors systems considering energy constraints. The authors do not consider a mandatory/optional reward model for the tasks. 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.

In [Mendez-Diaz et al. 2017] the authors introduced an ILP model for finding the exact solution to the problem here proposed. They also proposed in the paper several simple heuristics to cut down the time required to reach an optimal solution. A similar case is addressed in [Mo L. et al. 2018], where the authors use a MILP model and decompose the problem into two smaller sub-problems with fewer variables and constraints: an ILP-based Master Problem (MP) for task-to-processor allocation and frequency-to-task assignment decisions, and an LP-based Slave Problem (SP) for task scheduling and task adjustment decisions. In [Micheletto et al. 2015] we presented a short version of the present paper that did not include the Pareto frontier analysis, nor the experimental evaluation and implementation details.

3 System model

In this section we introduce the system model. Time is discretized in slots and it is noted with natural numbers, $h = 1, 2, \ldots$. All the events in the system are assumed to happen at the beginning of a slot. The length of the slot is a design issue and depends on different aspects [Mendez-Diaz et al. 2017].

At first we consider a set of independent, periodic and preemptable tasks defined as $\tau = 1, 2, \ldots, N$. In section 10 we consider the case in which the tasks are not independent. Each task is described by a tuple (m, o, P, D, r) where m is the worst case execution time of the mandatory part, o is the maximum desirable execution time for the optional part, P is the period of the task, which is assumed equal to the relative deadline $P = D$. The execution of the optional part produces a reward that is represented by $r = f(o)$. 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 time window, named hyperperiod and notated H, in which the system repeats itself. For this reason it is enough to provide a schedule for the first $H = lcm\{P_1, P_2, \ldots, P_N\}$. Each task, τ, is a stream of jobs or instances. In the hyperperiod, each task has H/P_τ instances or jobs.

A restricted-migration schedule is assumed. In the hyperperiod H there are $\sum_\tau^N H/P_\tau$ jobs enumerated in $\rho_\tau = \{1, 2, \ldots, H/P_\tau\}$ to be scheduled. A job ρ_τ becomes active at $h_{\rho_\tau} = (\rho - 1)P_\tau$ and finishes its execution at ϕ_{ρ_τ}. At this point, it is useful to introduce for every ρ_τ, the set of instants at which they are active, $H_{\rho_\tau} = \{h : a_{\rho_\tau} \leq h \leq a_{\rho_\tau} + D_\tau - 1\}$. A feasible system is one in which $\forall \rho_\tau,\ \phi_{\rho_\tau} < h_{\rho_\tau} + D_\tau$.

The tasks run on a set of M homogeneous cores, $\pi = \{1, 2, \ldots, M\}$. Each one may work at different power-modes noted, $f = \{1, 2, \ldots, E\}$, where E indicates the number of voltage/frequency pairs. Lower power-modes provide a benefit as the system saves energy increasing battery life. Reducing the power-mode increases the execution time of the tasks, so m_τ^f and o_τ^f indicate the worst case execution time of mandatory and optional parts for task τ at a particular power-mode f.

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 in a multi-core processor with restricted migration. The allocation is made at job level. In other words, a job that starts its execution on a certain core finishes its execution on the same core, whether it runs mandatory, optional, or mandatory and optional parts. It is also assumed that a power-mode is selected on a per job basis. Under this assumption, for example, a task may run its first job in core π_1 at power-mode f_2 and the second job in core π_2 at power-mode f_1 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 is the combination of two problems already proved to be NP-Hard. One is the restricted migration scheduling of jobs in multiprocessors as proved in [Burns 1991, Fisher 2007]. The other is the optimization of mandatory/optional reward-based scheduling [Aydin et al. 2001, Santos et al. 1997] with arbitrary reward functions.

4 Formulation

In this section we formalize the restricted-migration multiprocessor real-time scheduling optimization problem. The objective is to find a schedule that for each slot in $h = 1, 2, \ldots, H$, allocates a job ρ_τ of task τ, at a certain processor π with power-mode f in such a way that all mandatory parts finish their execution before their deadlines while maximizes the reward from optional parts with the minimum power-mode [Mendez-Diaz et al. 2017].

To do this, the following binary variables are defined. First, related to mandatory jobs, variable $y_{\tau\pi}^{\rho f}$ takes value 1 iff job ρ of τ runs in π at power-mode f and variable $x_{\tau\pi}^{fh} = 1$ iff at slot h, π runs any job of τ at power-mode f. Secondly, related to processors, variable g_π^{fh} takes value 1 iff at slot h, π runs at power-mode f and variable $z_\pi^h = 1$ iff at slot h, π changes power-mode. Finally, for optional jobs, variable $u_{\tau d}^{\rho f}$ takes value 1 iff job ρ runs d optional slots of a task τ at power-mode f. The execution of d optional slots at power-mode f produces a reward $r_{\tau d}^f$ for $d \in \{0, 1, \ldots, o_\tau^f\}$. When the processor executes at reduced power-modes it has also a reward that represents the low energy consumption; this reward is modeled by c_f. Even if reducing the power-mode has benefits, it is not good to keep changing it often as this increase the load of the processor. For this reason, a cost k is added to avoid successive power-mode changes. Based on the previous definition the objective function of the problem may be written as:

$$\max \alpha \sum_{\rho_\tau=1}^{H/P_\tau} \sum_{f=1}^{E} \sum_{d=1}^{o_\tau^f} r_{\tau d}^{f} u_{\tau d}^{\rho f} + \beta \left(\sum_{f=1}^{E} c_f \sum_{h=1}^{H} \sum_{\pi=1}^{M} g_\pi^{fh} - k \sum_{\pi=1}^{M} \sum_{h=1}^{H} z_\pi^h \right) \quad (2)$$

There are two terms in the objective function (2). The first one represents the reward obtained from the execution of optional parts. The second one represents the saving produced by reducing the power-mode considering the cost associated to the change. The schedule may have the maximum reward from the execution of optional parts but at the cost of a high power demand. On the other side, the processor may execute at the lowest possible power-mode reducing the energy consumption to the minimum but with no reward for the execution of optional parts. This is clearly a trade-off problem so two tuning parameters, α and β, are used as knobs to bias the solution either to the maximization of the reward or the minimization of energy consumption. For example, battery powered systems may privilege the low power-mode operation over the execution of optional parts. Suppose an embedded system that works on solar panels during daylight and batteries during dark hours. Two schedules can be computed for each type of power-source. In the first case the execution of optional parts for a better quality of application is preferred while in the second case the schedule optimizes the energy consumption.

The objective function is subject to the following constraints.

$$\sum_{\pi}^{M} \sum_{f}^{E} y_{\tau\pi}^{vf} = 1 \qquad \forall \rho_\tau \quad (3)$$

$$\sum_{h \in H_{\rho_\tau}} x_{\tau\pi}^{fh} \leq (m_\tau^f + o_\tau^f) y_{\tau\pi}^{\rho f} \qquad \forall \rho_\tau; \forall \pi; \forall f \quad (4)$$

$$\sum_{\pi}^{M} \sum_{h \in H_{\rho_\tau}} x_{\tau\pi}^{fh} \geq m_\tau^f \sum_{\pi}^{M} y_{\tau\pi}^{\rho f} \qquad \forall \rho_\tau; \forall f \quad (5)$$

$$\sum_{\tau}^{N} \sum_{f}^{E} x_{\tau\pi}^{fh} \leq 1 \qquad \forall \pi; h = 1, ..., H \quad (6)$$

$$\sum_{d=0}^{o_\tau^f} u_{\tau d}^{\rho f} = \sum_{\pi}^{M} y_{\tau\pi}^{\rho f} \qquad \forall \rho_\tau; \forall f \quad (7)$$

$$\sum_{f}^{E} \sum_{h \in H_{\rho_\tau}} x_{\tau\pi}^{fh} - \sum_{f}^{E} m_\tau^f \sum_{\pi \in \Pi} y_{\tau\pi}^{\rho f} = \sum_{f}^{E} \sum_{d=1}^{o_\tau^f} d u_{\tau d}^{\rho f} \quad \forall \rho_\tau \quad (8)$$

$$\sum_{f}^{E} g_\pi^{fh} = 1 \qquad \forall \pi; \forall h = 1, 2, \ldots, H \quad (9)$$

$$g_\pi^{fh} \geq \sum_{\tau}^{N} x_{\tau\pi}^{fh} \qquad \forall \pi; \forall f; h = 1, 2, \ldots, H \quad (10)$$

$$g_{\pi}^{fh} \geq g_{\pi}^{fh-1} - \sum_{f'}^{E} \sum_{\tau}^{N} x_{\tau\pi}^{f'h} \qquad \forall \pi \in \Pi; \forall f; h = 1, 2, \ldots, H \tag{11}$$

$$E z_{\pi}^{h} \geq \sum_{f}^{E} f g_{\pi}^{fh} - \sum_{f}^{E} f g_{\pi}^{fh+1} \qquad \forall \pi; h = 1, 2, \ldots, H-1 \tag{12}$$

$$E z_{\pi}^{h} \geq \sum_{f}^{E} f g_{\pi}^{fh+1} - \sum_{f}^{E} f g_{\pi}^{fh} \qquad \forall \pi; h = 1, 2, \ldots, H-1 \tag{13}$$

The set of constraints define the way in which jobs are allocated to the processors and the power-mode selection. For this, constraint (3) guarantees that each job is executed in just one processor at a unique power-mode while (4) guarantees that no more than $m_{\tau}^{f} + o_{\tau}^{f}$ slots of a task τ are executed at power-mode f and constraint (5) guarantees the execution of at least m_{τ}^{f} slots for each job of each task. Constraint (6) imposes the restriction of only one task for each slot in each processor. Similar to the mandatory parts, constraint (7) guarantees that for $d \in \{0, 1, \ldots, o_{\tau}^{f}\}$ exactly one of the variables $u_{\tau\rho}^{df}$ takes value 1 when job ρ_{τ} executes at power-mode f and takes value 0 when job ρ_{τ} executes at some other power-mode. Constraint (8) makes one the variable $u_{\tau d}^{\rho f}$ for d equal to the amount of optional slots that were executed for job ρ_{τ}. If one job is executing in processor π in slot h at power-mode f then constraints (9) and (10) impose a value of 1 to g_{π}^{fh} and constraint (11) forces an empty slot to keep the same power-mode of the last busy one because power-mode changes have a cost. When there is a power-mode change in slot h in processor π, constraints (12) and (13) impose a value of 1 on variable z_{π}^{h}.

In the case the processor has one or more dedicated cores (for example a floating point unit), tasks that should execute in these particular cores should have a high penalty associated for not doing so. This can be implemented with an additional restriction.

In this paper, we introduce two bio-inspired heuristics to find a solution to the scheduling problem described. The ILP model and its solutions are used as reference to compare the results obtained with the metaheuristics.

We define the optimization variable as a pair of matrices X and Y of M rows and H columns each, which represents the schedule to be evaluated. The element $X(\pi, h)$ denotes the task running on the processor π at time h and the element $Y(\pi, h)$ indicates the operating power-mode of the processor π at slot h.

4.1 Scheduling algorithm

In the model defined above, the optimization variable generates a high dimensional search space, which increases the number of possible solutions and reduces the chances of finding the global optimum. To simplify the problem it is proposed to transform or encode the solution arrays so that the optimization method works in a transformed space smaller than the original one. With this encoding, a large number of solutions that do not meet the conditions of feasibility are discarded. In this way, the optimization methods work on a reduced search space, which increases the efficiency of the algorithms.

In order to discard unfeasible solutions we start by defining an intermediate optimization variable composed by matrices V and W, where element $V(\tau, \rho_\tau)$ denote the core to which the job ρ of task τ, that is, ρ_τ is assigned and the element $W(\tau, \rho_\tau)$ idicates the power-mode at which job ρ_τ is executed. Because the number of jobs may be different for each task, each row of matrices V and W may have different number of elements.

The scheduling algorithm generates the X and Y matrices from V and W matrices. Although different jobs assignment strategies can be used in this situation, a simple mechanism based on Rate Monotonic Scheduling (RMS) [Liu and Layland 1973] adapted for the multi-core case is presented. In this scheduling algorithm, lower period tasks have higher priority. Pseudo-code in Algorithm 1 shows the scheduling procedure. The inputs of this algorithm are the problem description and matrices V and W and produces the matrices X and Y as outputs. First all mandatory parts are allocated and the optional parts are scheduled on available slots.

Algorithm 1 Scheduling algorithm pseudocode.

for all τ **do** ▷ For each tasks
  for all ρ_τ **do** ▷ For each job
    $h \leftarrow \rho_\tau \cdot P_\tau$. ▷ Slot number
    $f \leftarrow W(\tau, \rho_\tau)$. ▷ Power-mode
    $\pi \leftarrow V(\tau, \rho_\tau)$. ▷ Core number
    for all m_τ^f **do** ▷ Slots to allocate
      if isFree($X(\pi, h)$) **then** ▷ If slot h is not allocated
        $X(\pi, h) \leftarrow \tau$. ▷ Allocate τ
      else ▷ If slot h of π was allocated before
        $h := h + 1$. ▷ Next h
        if $h == (\rho_\tau + 1) \cdot P_\tau$ **then** ▷ If it is the last slot
          return $-\infty$. ▷ Deadline missed
        end if
      end if
    end for
  end for
end for

After mandatory parts are allocated, optional parts must be assigned to the remaining free slots where higher reward tasks are given chosen first. The allocation procedure is the same, except no deadline is considered.

Once matrix X is obtained with algorithm 1, power-mode allocation matrix Y, can be calculated element by element where $Y(\pi,h) = V(\tau,\rho_\tau)$. Because we know $\tau = X(\pi,h)$ and $\rho_\tau = \left\lfloor \frac{h}{P_\tau} \right\rfloor$, hence

$$Y(\pi,h) = V\left(X(\pi,h), \left\lfloor \frac{h}{P_{X(\pi,h)}} \right\rfloor\right) \tag{14}$$

5 Random Search Algorithm

The Random Search method (RS) is a crude optimization technique in that, at each iteration it is generated and evaluated a random solution and the best solution found is constantly being updated as best candidates appear.

5.1 Random search applied to the scheduling problem

To apply the method of Random Search to the actual problem, the array of elements composing the intermediate optimization variable are assigned by determinations of two random variables, one for the allocation to cores and the other for power-modes, as shown in Equations 15.

$$V(\tau,\rho_\tau) = \varPsi \quad \text{and} \quad W(\tau,\rho_\tau) = \varPhi. \tag{15}$$

If $\varPsi$ has an uniform distribution, then an equally assignment of jobs to the cores is achieved, but if $\varPhi$ has an uniform distribution, high and low power-modes are equiprobably, thus in some cases the utilization factor may be greater than the number of cores and feasible solutions will never be found. To avoid this, a tuning parameter that increases the probability of jobs being allocated with higher power-modes is used. This parameter is adaptive to allow lower power-modes when the proportion of feasible solutions increases. Q_r is the parameter representing the probability that a job has a power-mode different from the nominal and $1 - Q_r$ is the probability of executing the job with maximum power-mode. Thus, the allocation of power-modes has the probability distribution shown in equations 16 and 17.

$$P(\varPhi = f_i) = \frac{Q_r}{E}, \text{(with i=1,2,...,E-1)} \tag{16}$$

$$P(\varPhi = f_E) = 1 - Q_r + \frac{Q_r}{E} = 1 - \frac{Q_r \cdot (E-1)}{E}. \tag{17}$$

Whether Q_r is small, the generated solution is allocated with higher power-mode and therefore with less utilization factor. If during the main iterative process of this method it is computed a high proportion of feasible solutions, then the Q_r parameter is increased to achieve lower power consumption solutions. This way, a better exploring of the search space is achieved.

6 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic optimization method. The method consists of a set of particles called swarm, and each particle represents different positions within the search space, and also has velocity so positions can be updated following simple rules. It is not required the objective function to be continuous nor differentiable, which makes this method an appropriated option to solve the problem presented here.

The variant used algorithm follows the next equations,

$$\overrightarrow{v_t} = \omega \overrightarrow{v_{t-1}} + \phi_g r_g \left(\overrightarrow{g} - \overrightarrow{x_{t-1}} \right), \tag{18}$$
$$\overrightarrow{x_t} = \overrightarrow{x_{t-1}} + \overrightarrow{v_t}. \tag{19}$$

where x_t and v_t are the position and velocity of a particle respectively and should not be confused with X and V which are the optimization variables. ω and ϕ_g are tuning parameters which controls the swarm behaviour, r_g is a uniformly distributed random number between 0 and 1 and $\overrightarrow{g}$ is the best position found by the swarm. This is the called PSO-VG approach as it only considers velocity and attraction to the swarm's best known position [Pedersen 2010].

The procedure is detailed in Algorithm 2. The algorithm is composed of two parts, the initialization of the swarm and the iterative procedure. The swarm initialization requires an initial population of feasible solutions. Once the particles have defined positions, the algorithm iterates through equations 18 and 19.

6.1 PSO applied to the scheduling problem

Each particle has an associated position which represent a solution in the search space. The dimensions of the position vector of particles are the same as the W arrays containing the allocation of jobs to cores and operation power-modes. The PSO initial population must consist of feasible solutions only, to make it possible to compare solutions through their quality values. To achieve this, random solutions are generated for the initial population of the swarm using a random search method. This operation represents an important part of the method's runtime and is generally greater if the utilization factor of the problem is high, because it is hard to find a feasible schedule. The initialization process of this method uses an adaptive Q_r parameter that works the same way as described in Section 5.1. Starting from a high value for Q_r the initializer tries to generate feasible solutions, and if a certain number of failed attempts occur, then Q_r is decreased. Each time a feasible solution is found, it is assigned to a particle's position and the next particle follows until all particles from the swarm population have an assigned position. Velocities are initialized as null vectors.

Once the swarm population has been generated, the algorithm begins the iterations following Equations 18 and 19 until the termination criterion is reached. Because the optimization variable defined in Section 4, has two independent components V and W, the equations of this technique can be decoupled. The Equations 20 to 23 show the element by element assignment for the task τ and job ρ_τ which is simply notated as ρ,

$$v_{1,\tau,\rho} = \omega_c v_{1,\tau,\rho} + \phi_c r_g \left(V(\tau,\rho) - V^{opt}(\tau,\rho)\right), \tag{20}$$

$$v_{2,\tau,\rho} = \omega_f v_{2,\tau,\rho} + \phi_f r_g \left(W(\tau,\rho) - W^{opt}(\tau,\rho)\right), \tag{21}$$

$$V(\tau,\rho) = V(\tau,\rho) + v_{1,\tau,\rho}, \tag{22}$$

$$W(\tau,\rho) = W(\tau,\rho) + v_{2,\tau,\rho}. \tag{23}$$

The swarm moves in an integer search space thus, each component of the velocity array of particles is rounded to the nearest integer and values of V and W are clamped so they do not exceed the search space limits. Generally the swarm mathematical dispersion is calculated as an indicator of the convergence of the method, but due to the size of the particles array and the number of dimensions of the search space, the computation cost increases considerably.

6.2 Parameter selection

To achieve a fast initialization of this method, the Q_r probability parameter value is selected to be $Q_r = 0.1$ and decreasing its value 0.01 every 1000 failed attempts of generating a feasible solution. If no particle has been initialized and Q_r parameter reaches the value $Q_r = 0$, the entire swarm is initialized with solutions of maximum power-mode only. However, lower power consumption solutions can be found because random components of velocity vectors allow particles to explore the search space.

Empirical tests show that high ω values enhance the swarm exploratory capability but reduce the swarm convergence and precision of results. On the other hand, low ϕ_g values increase the self-sufficiency of particles and make the algorithm behave in a similar way as random search method. Selecting $\omega = 0.5$ and $\phi_g = 1.5$ are good tuning parameters in most cases as it is shown in [Shi and Eberhart 1998].

Algorithm 2 PSO Method.

$q \leftarrow -\infty$ ▷ Global optima of the swarm.
for all i-th particle **do** ▷ For each particle
    $W_{1,\tau,\rho} \sim U(0, M)$ ▷ Generate an initial random position
    if $U(0, 1) < Q_r$ **then** ▷ With probability Q_r, do:
        $W_{2,\tau,\rho} \sim U(0, F)$ ▷ Assign random operation power-mode
    else
        $W_{2,\tau,\rho} = 0$ ▷ Assign maximum operation power-mode with probability $1 - Q_r$
    end if
    if $F_{obj}(x_i) > -\infty$ **then** ▷ If the actual particle is feasible
        $x_i \leftarrow W$ ▷ Add particle to initial population
        if $F_{obj}(x_i) > g$ **then** ▷ If the actual particle's quality is greater than the optimum
            $g \leftarrow x_i$ ▷ Remember best position of the swarm
            $q \leftarrow F_{obj}(x_i)$ ▷ Remember the optimum value of the swarm
        end if
    end if
end for
while Termination criterion is not verified **do**
    for i-th particle **do**
        $r_g \sim U(0, 1)$ ▷ Random number between 0 and 1
        $v_{i,\tau,\rho,1} \leftarrow \omega_c v_{i,\tau,\rho,1} + \phi_c r_g (g_{i,\tau,\rho,1} - x_{i,\tau,\rho,1})$ ▷ Update velocity
        $v_{i,\tau,\rho,2} \leftarrow \omega_f v_{i,\tau,\rho,2} + \phi_f r_g (g_{i,\tau,\rho,2} - x_{i,\tau,\rho,2})$ ▷ Update velocity
        $x_i \leftarrow x_i + v_i$ ▷ Update position of particles
        if $F_{obj}(x_i) > f_g$ **then** ▷ Update optima
            $g \leftarrow x_i$
            $q \leftarrow F_{obj}(x_i)$
        end if
    end for
end while

The objective function defined in Equation 2 in Section 4 is an integer function, so high exploratory swarm and integer tuning parameters are needed. In Table 1 selected values are shown.

7 Genetic Algorithm

A genetic algorithm (GA) performs numerical optimization mechanism imitating the process of natural selection of species. It consists of a population of so called chromosomes, and each chromosome contains a set of genes that encode the information of the optimization variable. This is an iterative method where chromosomes are selected to reproduce if the quality is over the average of the population, otherwise they are discarded. Then crossover occurs between two chromosomes and the offsprings are compared to the rest of the population. Mutation is simulated to introduce random variations of the genes that allow the origination of new genes to avoid the stagnation of the algorithm. This method is constantly improving the solutions quality because the individuals have to evolve for surviving and compete with others of each generation.

7.1 GA applied to the scheduling problem

Optimization variable V and W, as described in Section 4.1, are arrays of vectors, each one containing information about a task core allocation and power-mode job by job. Therefore, in the context of GA, each one of these vectors corresponds to the genes of a chromosome. Hence, the number of genes depends on the number of tasks to be scheduled. Crossover consists in swapping or combining genes of different chromosomes and results in a new pair of chromosomes or solutions, which are called offspring. Figure 1 illustrates a pair of parents chromosomes and the resulting offspring when the parents are combined.

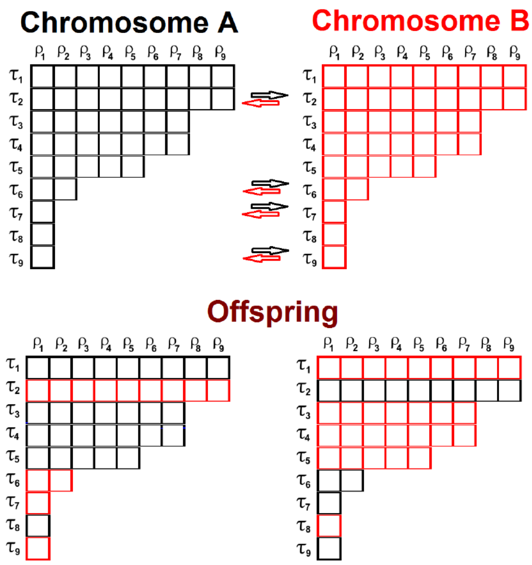


Figure 1: Crossover of chromosomes.

Algorithms 3-7 shows a pseudo-code that illustrates the main procedures of the method. GA uses an initialization stage and an iterative process. The initialization used to generate the initial population of this method, which is shown in Algorithm 3, is the same as the one used in PSO (see Section 6.1).

The first step in the iteration process, shown in Algorithm 4, consists of sorting the chromosomes from highest to lowest quality. Then, the offspring number of each chromosome, is determined evaluating Equation 24,

$$osg_i = ||\frac{q_i}{\bar{q_i}}|| \tag{24}$$

Where $q_i = F_{obj}(x_i)$ is the quality of the i-th chromosome and $\bar{q_i}$ is the mathematical mean between the quality of all chromosomes. The number of the offspring population is equal to half of the initial population of individuals, and before the crossover, the worst half of the initial population is replaced by the half containing the calculated offspring. To perform crossover, the list of individuals is randomly sorted and parents are selected in consecutive pairs. The offspring is obtained by swapping genes from the parents. Genes of the core allocation array are selected with probability $Q_{co}core$ and the probability of swapping genes of the power-mode allocation array is $Q_{co}freq$. $Q_{co}core$ and $Q_{co}freq$ are tuning parameters. The procedure is detailed in Algorithm 5. Mutation, shown in Algorithm 6, is accomplished by choosing random components from the gene vector with probability $Q_m core$ if the vector corresponds to the core allocation array or $Q_m freq$ if the mutation is applied to a power-mode allocation vector. $Q_m core$ and $Q_m freq$ are also tuning parameters. The list of parents and offspring chromosomes is evaluated again by the objective function and sorted by quality and the process repeats until the termination criterion is reached, as shown in Algorithm 7.

7.2 Parameter selection

Crossover and mutation events take place in randomly selected genes where the probability of these occurrences are determined by the corresponding parameters listed in Table 1. Crossover probability values Q_{co} have the same effect than value $1 - Q_{co}$, so this parameter needs not to be greater than 0.5. As the combination between two core selection vectors has a greater quality change in the offspring than the obtained by the combination of two power-mode vectors, $Q_{co}core$ must be smaller than $Q_{co}freq$. On the other hand, high mutation probability overshadows the crossover effects which is the most important feature of the method, so the mutation probabilities must be selected as small as possible.

8 Pareto efficiency

A simplified approach to the problem is to consider the main variables that characterize the solutions: the processor's energy saving and the allocation of

Algorithm 3 GA Method: Initialization.

procedure INITIALIZATION
    for $i \leftarrow 1..N_{cr}$ **do** ▷ For each chromosome
        $V(\tau, \rho) \sim U(0, M)$ ▷ Assign initial random genes
        if $U(0,1) < Q_r$ **then** ▷ With probability Q_r, do:
            $W(\tau, \rho) \sim U(0, F)$ ▷ Random operation power-mode
        else
            $W(\tau, \rho) = 0$ ▷ Assign maximum operation power-mode with probability $1 - Q_r$
        end if
        if $F_{obj}(x_i) > -\infty$ **then** ▷ If the actual chromosome is feasible,
            $x_i \leftarrow (V, W)$ ▷ Add chromosome to initial population
        end if
    end for
end procedure

Algorithm 4 GA Method: Selection.

procedure SELECTION
    for $i \leftarrow 1..N_{cr}$ **do**
        $q_i \leftarrow F_{obj}(x_i)$ ▷ Determine each chromosome's quality
    end for
    Sort(x_i,fi_i,$i \leftarrow 1..N_{cr}$) ▷ Sort chromosomes by quality
    for $i \leftarrow 1..\frac{N_{cr}}{2}$ **do** ▷ Determine the offspring of the half best chromosomes
        $\bar{q}_i \leftarrow \frac{\sum_{i=0}^{\frac{N_{cr}}{2}-1} qi}{\frac{N_{cr}}{2}}$
        $osg_i \leftarrow ||\frac{qi_i}{\bar{q}_i}||$
    end for ▷ Double the list of chromosomes according to each chromosome offspring number
    $c \leftarrow 0, idx \leftarrow 0$
    for $i \leftarrow \frac{N_{cr}}{2}..N_{cr}$ **do**
        if $osg_{idx} > c$ **then**
            $x_i \leftarrow x_{idx}$
            $c \leftarrow c + 1$
        else
            $c \leftarrow 0$
            $idx \leftarrow idx + 1$
        end if
    end for
    Blend(x_i,$i \leftarrow \frac{N_{cr}}{2}..N_{cr}$) ▷ Blend the list of chromosome randomly
end procedure

Algorithm 5 GA Method: Crossover.

procedure CROSSOVER
    for $i \leftarrow \frac{N_{cr}}{2} : 2 : N_{cr}$ **do** ▷ Swap genes of chromosomes according to crossover probability
        for $\tau \leftarrow 0..N$ **do**
            if $U(0,1) < Q_{co}core$ **then**
                Swap($x_{i,\tau,1}$,$x_{i+1,\tau,1}$)
            end if
            if $U(0,1) < Q_{co}freq$ **then**
                Swap($x_{i,\tau,2}$,$x_{i+1,\tau,2}$)
            end if
        end for
    end for
end procedure

Algorithm 6 GA Method: Mutation.

procedure MUTATION
    for $i \leftarrow \frac{N_{cr}}{2}..N_{cr}$ **do**
        for $j \leftarrow 0..N$ **do**
            for $k \leftarrow 0 : \frac{H}{P_j}$ **do** ▷ Make random changes with probability $Q_m core$ and $Q_m freq$
                if $U(0,1) < Q_m core$ **then**
                    $x_{i,j,k,1} \sim U(0,M)$
                end if
                if $U(0,1) < Q_m freq$ **then**
                    $x_{i,j,k,2} \sim U(0,F)$
                end if
            end for
        end for
    end for
end procedure

Algorithm 7 GA Method: Optimization loop.

Initialization()
while Termination criterion is not verified **do**
    Selection()
    Crossover()
    Mutation()
end while

PSO		GA	
Population	30	Population	30
$\Phi_g core$	2	$Q_{co} core$	0.2
$\Phi_g frec$	2	$Q_{co} freq$	0.4
$\omega core$	2	$Q_m core$	0.02
$\omega frec$	2	$Q_m freq$	0.1

Table 1: Optimization parameters.

optional slots. Energy saving can be calculated as the difference between the nominal operating power-mode and the average operating one of all cores during the hyperperiod, that is

$$\bar{E} = 1 - \bar{F} = 1 - \sum_{h=1}^{H} f_h. \tag{25}$$

where f_h is the operation power-mode relative to the cores nominal one during the time slot h.

On the other hand, assuming that the allocation of optional slots is performed by an algorithm that maximizes the payoff by taking full advantage of the available slots, the second variable can be characterized as the optional utilization factor of the solution, that is

$$UF_o = \sum_{\tau=1}^{N} \frac{o_\tau}{P_\tau}. \tag{26}$$

Where o_τ is the optional parts of task τ at nominal power-mode. For convenience, instead of the optional utilization factor UF_o, we use the total utilization factor UF, that is, the sum of the mandatory and optional utilization factors. The nominal power-mode utilization factor indicates the number of mandatory and optional parts executed for all the tasks of the system. The number of slots assigned in this solution depends on the operating power-mode of the processor cores.

To maximize the quality of a solution the number of free slots has to be minimum either because there are many optional parts being executed or the operation power-mode has been reduced for a reduced energy consumption. Therefore, the utilization factor of the solution must be as greater as possible. Thus there is a trade-off relationship between these two variables that characterize the solution and if the system schedule does not contain free slots, then

$$\frac{UF}{M} = \bar{F} = 1 - \bar{E}. \tag{27}$$

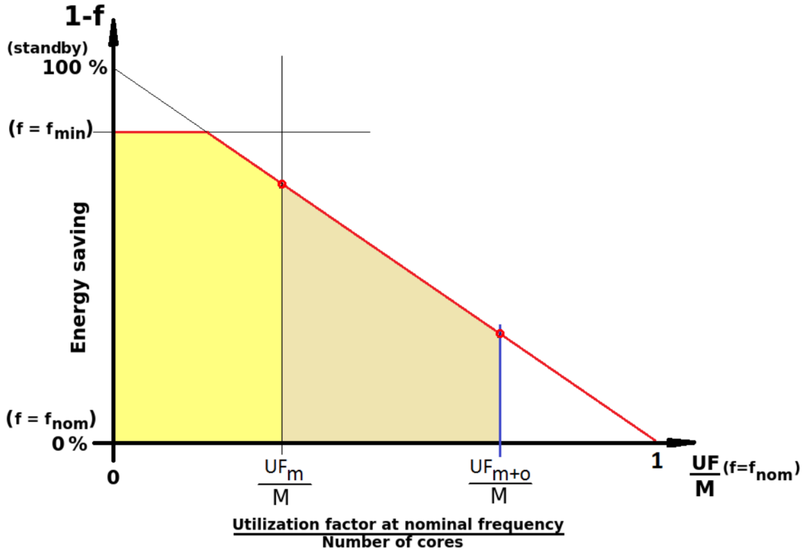


Figure 2: Pareto frontier representation

This linear relationship corresponds to the Pareto frontier and is illustrated in Figure 2, where it can be seen that the higher quality solutions will be closer to the Pareto frontier and their position along the curve depends on whether the saving energy has higher or lower priority than the allocation of optional parts.

The objective function can be used to compare solutions with identical assumptions on the tuning parameters α and β. However, Equation 27 provides a tool to compare solutions obtained with different values in the tuning parameters.

8.1 Example

Figure 3 shows the utilization factor and energy saving obtained for two different systems where the number of optional parts for each task was increased for each simulation up to the maximum possible utilization factor, that is the number of cores of the system. The tuning parameters were chosen such that $\alpha >> \beta$ so the allocation of optional parts has higher priority over the lowering of the operation power-mode. The problems used are described with detail in Section 9. The first set has the typical period and load distribution found in embedded systems while the second set is a synthetic one with uniform distribution in the utilization factor among the tasks. The schedule was solved using the GA with 10000 iterations and $\alpha = 10$ and $\beta = 1$. As it can be seen, for the first set of problems the data are over the Pareto frontier as described in Figure 2. In

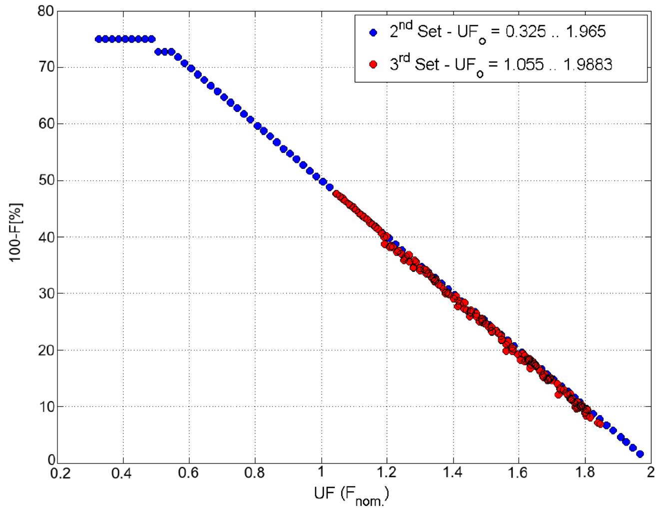


Figure 3: Pareto Frontier Example

the other case, there is a greater dispersion of the data because the problem is more complex, it has greater degrees of freedom and the optimal solutions do not reach the ideal position on the Pareto frontier.

Figure 4 shows in a three dimension plot how the distance to the Pareto Frontier changes when both α and β parameters are varied. As it is shown, when the $\alpha >> \beta$ the distance is increased. On the contrary, if $\beta > \alpha$ the method finds a solution that is on the Pareto Frontier.

9 Experimental evaluation

We consider, for the evaluation of the scheduling method, three kind of problems. The problems were taken from the literature and represent different situations in real-time applications. For the first two kinds, one hundred instances 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 available for each one. The reward for executing at different power-modes were 0, 1, 2 and 3 for 100%, 75%, 50% and 25% of the nominal one respectively. The cost for changing the power-mode was set equal to 8. The objective function was set with $\alpha = \beta = \gamma = 1$. The selection of the parameters is arbitrary. The purpose of this evaluation is to compare how fast and efficient are the different alternatives when compared with an ILP optimization procedure. We are not looking at the sensitivity of the different terms in the objective function. In what follows each set is described and the results presented.

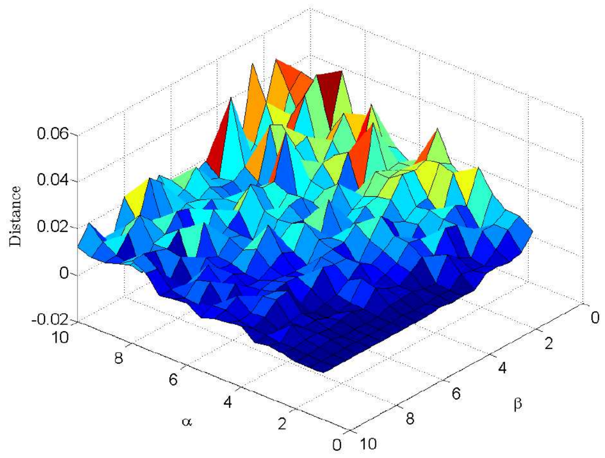


Figure 4: Pareto Frontier Distance as function of α and β

First set: In [Bini and Buttazzo 2005], the authors proposed the generation of synthetic systems to evaluate the performance of different scheduling tests. In this paper we follow the same algorithm to create one hundred instances of eight tasks each with utilization factors varying between 0.41 and 1.98. For the optional parts, reward values for the optional slots were randomly generated too.

Second set: In [Gai et al. 2003], the authors described the relations between periods and worst case execution times for embedded systems like the ones used to control the power-train in cars. We follow the same algorithm and create one hundred instances of twenty tasks each with utilization factors in the range of 0.44 to 2.354.

Third Set: In [Aydin et al. 2001], the authors introduced a set of problems to evaluate their method for scheduling optional parts. The proposed system has eleven tasks. Each task has a total execution time that includes both mandatory and optional parts. The experiments were done for 0%, 25%, 40%, 60% and 80% of the execution time as mandatory and the rest as optional. Each task has three reward functions, all of them are concave. We repeat the experiment but with multi-core allocation.

The results of the evaluation of each kind of problem were compared with those obtained by the ILP method with limited execution time as proposed in [Mendez-Diaz et al. 2017]. In Table 2 the comparative results of quality and required time to reach the solution are presented for three kind of problems. All values are in percentages relative to the ILP solution. The quality column (Q) shows the relation between the value of the objective function obtained by the proposed heuristic and that of the ILP method, in the same way the required time is the relation between the time used by the heuristic and the used by the ILP solver. The last row in the table presents mean values when considering all the evaluated instances for the different kind of problems. From results listed in Table 2, we conclude that the best solutions, in terms of distance to Pareto frontier and Quality, are obtained with the Genetic Algorithms (GA) heuristic. In second place comes PSO method and last, RS method, which in general gives better quality values, although distances to the Pareto frontier are greater than the ILP solutions. On the other hand, execution time (ET) of different algorithms exhibit inverse results, that is, RS is in first place, with almost 40% of the ILP execution time and in last place, GA with 72.68% of the total execution time of the ILP solver.

	RS			PSO			GA		
	Q	ET	DP	Q	ET	DP	Q	ET	DP
First set	87.96	7.3	77.74	94.54	8.97	36.51	104.29	18.26	30.82
Second set	90.22	76.7	139.79	90.78	106.91	91.83	100.11	135.51	35.92
Third set	84.45	10.63	287.78	87.53	12.23	143.95	102.11	16.62	40.82
Mean	88.82	39.81	121.25	92.38	54.74	90.76	102.20	72.68	35.85

Table 2: Comparative results. Q=Quality, ET=Execution time, DP = Distance to Pareto frontier

10 Precedence constraints

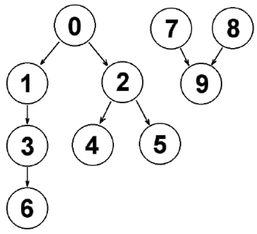
The precedence relationship between tasks is usually expressed by a directed acyclic graph or tree, where the nodes represent tasks and the edges, the order in which the tasks should run. The set of all tasks contained in a connected subgraph constitute a process and these tasks have all the same period.

A task must initiate its execution only when all their predecessors have completed their assigned slots including mandatory and optional parts. Optional parts of a predecessor task cannot be allocated after the corresponding slots of the successor task, and because the execution of optional parts of a predecessor may prevent or postpone the execution of the mandatory part of the successor, the system may result unfeasible. To avoid this, the amount of optional slots to be executed should be determined beforehand. This is achieved by adding a third variable to the set of intermediate optimization variables, V and W, that indicates the maximum number of optional parts that each task will run on each instance, and is called Z. The scheduling algorithm assigns the slots to be occupied by the successor task once all optional parts of the previous job have been completed. Then the optimization algorithm must determine the optimal number of optional parts to be executed by every task avoiding the unfeasibility condition.

Algorithm 8 Precedence constrained tasks scheduling.

```
for all $\psi$ do                                   ▷ Process number
    for all $\rho$ of $\psi$ do                     ▷ Instance number of $\psi$
        for all $\tau$ of $\psi$ do                 ▷ Task number of $\psi$
            getPriority($\tau$,$\rho$)
            firstSlot$_\tau \leftarrow \rho \cdot P_\psi$
        end for
        sortByPriority($\tau$)
        for all $\tau$ of $\psi$ do
            $h \leftarrow$firstSlot$_\tau$          ▷ Slot number
            $\pi \leftarrow V(\tau, \rho)$          ▷ Core number
            $f \leftarrow W(\tau, \rho)$            ▷ Power-mode number
            $o \leftarrow Z(\tau, \rho)$            ▷ Optional parts
            for all $m_\tau^f + o_\tau^f$ do
                while $X(\pi, h)$ taken do
                    $h \leftarrow h + 1$            ▷ Proceed with next slot
                    if k == $(\rho + 1) \cdot P_\psi$ then     ▷ End of period
                        return $-\infty$            ▷ Does not meet deadline
                    end if
                end while
                $X(\pi, h) \leftarrow \tau$         ▷ Asign $\tau$
            end for
            for all t son of $\tau$ do              ▷ For each son of $\tau$
                if firstSlot$_t$ < h+1 then
                    firstSlot$_t \leftarrow h + 1$
                    if h+1 == $(\rho + 1) \cdot P_\psi$ then
                        return $-\infty$            ▷ No time for run $t$
                    end if
                end if
            end for
        end for
    end for
end for
```

In the case that two or more tasks have the same immediate predecessor within a process tree, it is necessary to determine the order in which they are



Ψ	τ	m	o	P	R
	0	1	2		1
	1	1	1		2
	2	1	1		1
1	3	2	0	15	0
	4	1	1		3
	5	1	2		1
	6	1	1		2
	7	2	0		0
2	8	1	1	10	1
	9	3	2		1

Figure 5: System definition

executed. It may be better to execute them in parallel on different cores or to execute them one after the other on a single core. A priority associated to each task is useful to schedule them.

The calculation of the priority begins assigning to each node, the run time required to complete both mandatory and optional parts for the current instance. Once these values are assigned, the process tree is traversed in postorder direction where each node is added to the maximum priority among their children. Every time a task is allocated, the last slot occupied must be marked, so the allocation of subsequent tasks starts from that slot.

Algorithm 8 shows the scheduling strategy proposed for assigning precedence related tasks. The inputs and outputs are the same that for algorithm 1.

10.1 Examples

Figure 5 shows a system composed of two processes with seven and three tasks respectively. The table shows, for each task, optional and mandatory runtimes, periods and rewards. This problem was solved using GA. The obtained results are shown in Figure 6.

It can be seen that in the solution found, the core number 1 runs at nominal power-mode all the time while core number 2 executes two slots at 50% of the nominal power-mode and the rest of the time it does it at 25% of the nominal power-mode. Tasks 5 and 9 are the only ones to run optional parts.

A more complex example is presented in [Tindell et al. 1992] where the system consists of 11 processes and 43 tasks in total to be executed in eight homogeneous cores. As the system tasks have no optional parts, only the energy consumption is optimized. To solve this problem with GA, the crossover and mutation parameters where greatly reduced and a population of 30 chromosomes

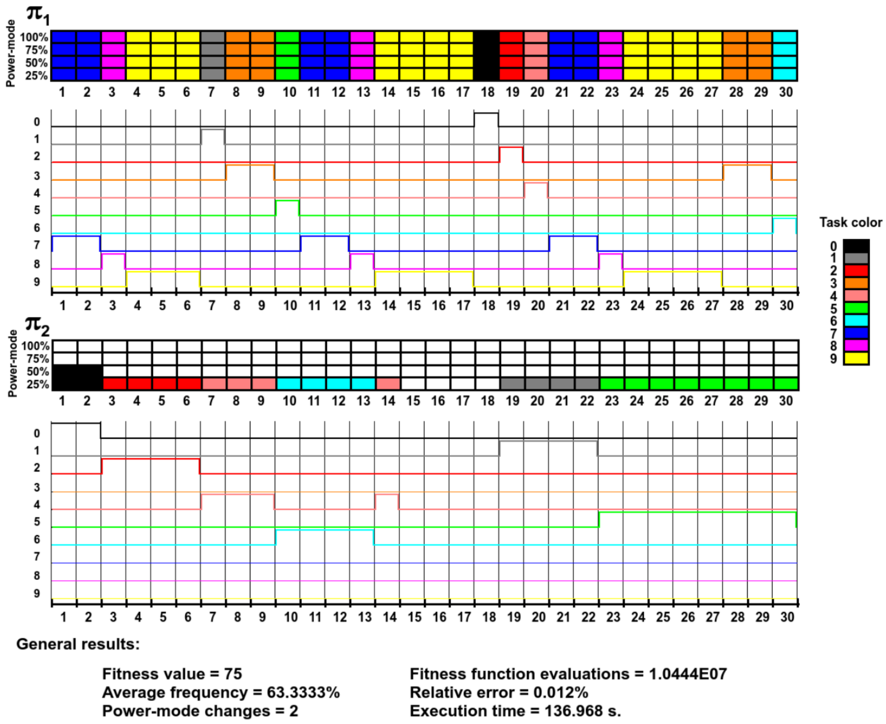


Figure 6: Results for the first proposed example.

was used. The following results were obtained:

```
Quality = 4065
Average power−mode = 67.314%
Power−mode changes = 41
Objective function evaluations = 2.6696e+07
Relative error = 0.00296077
Execution time = 16168.3 s.
```

The implemented system has a total utilization factor $U = 4.49$ so the minimum power with eight processors can be achieved at an average operation frequency of not less than 56% of the nominal value. The optimization process was limited to 1 million iterations, which were completed in a total time of 4 hours, 29 minutes and 28 seconds running under Ubuntu 14.04 on a desktop computer with 4Gb RAM and 3.2GHz Intel Core i5 CPU.

The fitness function was evaluated 29.696 million times and 3.304 million attempts were unfeasible solutions. The precedence restriction limits the number of feasible schedules and the possible ways to allocate tasks and power-modes is reduced significantly. It is unknown whether is possible or not to achieve the minimum average operation frequency with this restriction. Therefore, the optimal solution will be the one closest to this limit in terms of power consumption. In the solution found, the average operation frequency is 67.314%, 11% more than the minimum value. The more iterations are performed, the better the solution found will be, so the execution time or the maximum number of iterations must be determined according to a cost-benefit relation between the computational cost and the quality of the solution.

11 Conclusion

In this paper we used two different bio-inspired meta-heuristics to compute offline schedules for reward based mandatory/optional tasks with energy considerations for homogeneous multi-core systems. The problem is NP-hard as it has been proved in previous work. The algorithms introduced were evaluated using different problems from the real-time systems bibliography. The extensive simulations show that the solutions found are equivalent to those obtained with ILP procedures but demanding considerably less execution time. Besides, the Pareto Frontier analysis shows how the search can be guided towards the minimization of energy demand or the maximization of the reward using the tuning parameters. In any case, the solutions found are close to the frontier. Besides these contributions, an extension to deal with precedence related tasks was also considered. The proposed algorithms can solve several real scheduling problems that arise for embedded systems based on multicore architectures.

References

[Aydin et al. 2004] Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P.: "Power-aware scheduling for periodic real-time tasks"; IEEE TC, IEEE Transactions on Computers, 53, 5 (2004), 584-600.

[Aydin et al. 2001] Aydin, H., Melhem, R., Mosseé, D., Mejía-Alvarez, P.: "Optimal reward-based scheduling for periodic real-time tasks"; IEEE TC, IEEE Transactions on Computers, 50, 2 (2001), 111–130.

[Baruah and Carpenter 2003] Baruah, S., Carpenter, J.: "Multiprocessor fixed-priority scheduling with restricted interprocessor migrations"; ECRTS, 15th Euromicro Conference on Real-Time Systems, IEEE, Porto (2003), 195-202.

[Bini and Buttazzo 2005] Bini, E., Buttazzo, G.: "Measuring the performance of schedulability tests"; RTS, Real-Time Systems, 30, 1-2 (2005), 129-154.

[Burns 1991] Burns, A.: "Scheduling hard real-time systems: a review"; SEJ, Software Engineering Journal, 6, 3 (1991), 116-128.

[Cheng and Wu 2018] , Cheng, A. M. K., Wu P.: 2Incorporating Deadline-Based Scheduling in Tasking Programming Model for Extreme-Scale Parallel Computing", RTSS, 2018 IEEE Real-Time Systems Symposium WIP Session, Nashville, USA 2018, 131-134.

[Chung et al. 1990] Chung, J. Y , Liu, J. W S, Lin, K. J.: "Scheduling periodic jobs that allow imprecise results"; IEEE TC, IEEE Transactions on Computers, 39, 9 (1990), 1156-1174.

[Dertouzos and Mok 1989] Dertouzos, M. L., Mok, A. K.:"Multiprocessor online scheduling of hard-real-time tasks"; IEEE TSE, IEEE Transactions on Software Engineering, 15, 12 (1989), 1497-1506.

[Feller and Ebenbauer 2017] Feller, C., Ebenbauer, C.: "A stabilizing iteration scheme for model predictive control based on relaxed barrier functions"; Automatica, 80, (2017), 328-339.

[Fisher 2007] Fisher, N. W.: "The Multiprocessor Real-Time Scheduling of General Task Systems"; PhD thesis, University of North Carolina at Chapel Hill, 2007.

[Gai et al. 2003] Gai, P., Di Natale, M., Lipari,G., Ferrari, A., Gabellini, C., Marceca, P.: "A comparison of mpcp and msrp when sharing resources in the janus multiple-processor on a chip platform"; 9th IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE, Toronto, (2003), 189-198.

[Greco et al, 2011] Greco, L., Fontanelli, D., Bicchi, A.: "Design and Stability Analysis for Anytime Control via Stochastic Scheduling"; IEEE TAC, IEEE Transactions on Automatic Control, 56, 4 (2011), 571-585.

[Hong and Leung 1988] Hong, K.S., Leung, J.Y.T.: "On-line scheduling of real-time tasks"; 19th IEEE Real-Time Systems Symposium, Madrid (1998), 244-250.

[Kumar and Palani 2012] Kumar, P., Palani, S.: "A dynamic voltage scaling with single power supply and varying speed factor for multiprocessor system using genetic algorithm"; International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME),IEEE, Salem (2012), 342-346.

[Likhachev et al 2008] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S.: "Anytime search in dynamic graphs"; Artificial Intelligence, 172,14 (2008), 1613-1643.

[Liu and Layland 1973] Liu, C. L., Layland, J.: "Scheduling algorithms for multiprogramming in a hard real-time environment"; J ACM, Journal of the Association for Computing Machinery, 20, 1 (1973), 46-61.

[Liu et al. 1991] Liu, J. W.-S., Lin, K.-J., Shih, W.-K., Yu, A.C.-S., Chun, C., Yao, J., Zhao, W.: "Algorithms for scheduling imprecise computations", IEEE Computer, 24, 5 (1991), 58-68.

[Mendez-Diaz et al. 2017] Mendez-Diaz, I., Orozco, J., Santos, R., Zabala, P.: "Energy-aware scheduling mandatory/optional tasks in multicore real-time systems"; ITOR, International Transactions in Operational Research, 24, 1-2 (2017), 173-198.

[Micheletto et al. 2015] Micheletto, M., Santos, R., Orozco, J.: "Using bioinspired meta-heuristics to solve reward-based energy-aware mandatory/optional real-time tasks scheduling"; 2015 Brazilian Symposium on Computing Systems Engineering (SBESC), IEEE, Foz do Iguazu (2015), 132-135.

[Mo L. et al. 2018] Mo, L., Kritikakou, A., Sentieys, O.: "Controllable QoS for Imprecise Computation Tasks on DVFS Multicores With Time and Energy Constraints"; IEEE JESTCS, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 8, 4 (2018), 708-721.

[Mo L. et al. 2018] Mo, L., Kritikakou, A., Sentieys, O.:"Energy-quality-time optimized task mapping on DVFS-enabled multicores"; IEEE TCADICS, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37, 11 (2018), 2428-2439.

[Pedersen 2010] Pedersen, M.: "Tuning & simplifying heuristical optimization"; PhD thesis, University of Southampton, 2010.

[Pouwelse et al. 2001] Pouwelse, J., Langendoen, K., Sips, H.:"Dynamic voltage scaling on a low-power microprocessor"; Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, ACM, New York (2001), 251-259.

[Santos et al. 1997] Santos, J., Ferro, E., Orozco, J., Cayssials, R.: "A heuristic approach to the multitask-multiprocessor assignmentproblem using the empty-slots method and rate monotonic scheduling"; RTS, Real-Time Systems, 13, 2 (1997), 167-199.

[Santos et al. 2012] Santos, R., Lipari, G., Bini, E., Cucinotta, T.: "On-line schedulability tests for adaptive reservations in fixed priority scheduling"; RTS, Real-Time Systems, 48, 5 (2012), 601-634.

[Santos et al. 2008] Santos, R., Lipari, G., Santos, J.: "Improving the schedulability of soft real-time open dynamic systems: The inheritor is actually a debtor"; JSS, Journal of Systems and Software, 81, 7 (2008), 1093-1104.

[Santos et al. 2005] Santos, R., Santos, J., Orozco, J.: "A least upper bound on the fault tolerance of real-time systems"; JSS, Journal of Systems and Software, 78, 1 (2005), 47-55.

[Santos et al. 2004] Santos, R., Urriza, J., Santos, J., Orozco, J.: "New methods for redistributing slack time in real-time systems: applications and comparative evaluations"; JSS, Journal of Systems and Software, 69, 1-2 (2004), 115-128.

[Shih and Liu 1995] Shih, W. K., Liu, J. W S.: "Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error"; IEEE TC, IEEE Transactions on Computers, 44, 3 (1995), 466-471.

[Stankovic 1988] Stankovic, J. A.: "Misconceptions about real-time computing"; IEEE Computer, 21, 17 (1988), 10-19.

[Tindell et al. 1992] Tindell, K. W., Burns, A., Wellings, A. J.: "Allocating hard real-time tasks: An np-hard problem made easy"; RTS, Real-Time Systems, 4, 2 (1992), 145-165.

[Shi and Eberhart 1998] Shi, Y., Eberhart, R. C.: "Parameter selection in particle swarm optimization"; 7th Annual Conference on Evolutionary Programming, Springer, San Diego (1998), 591-600.

[Wanli and Chakraborty 2016] , Wanli, C. and Chakraborty, S.: "Resource-aware Automotive Control Systems Design: A Cyber-Physical Systems Approach"; Foundations and Trends in Electronic Design Automation, 10, 1 (2016), 249-369.

[Zhang et al. 2014] Zhang, W., Xie, H., Cao, B. A, Cheng, A.: "Energy-aware real-time task scheduling for heterogeneous multiprocessors with particle swarm optimization algorithm"; MPE, Mathematical Problems in Engineering, (2014), 9 pages.