IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 March 2023, accepted 20 March 2023, date of publication 23 March 2023, date of current version 29 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3261130

RESEARCH ARTICLE

Optimized Scheduling of Periodic Hard Real-Time Multicore Systems

JOSÉ MARÍA ACEITUNO[1], ANA GUASQUE[1], PATRICIA BALBASTRE[1], FRANCISCO BLANES[1], AND LUIGI POMANTE[2], (Member, IEEE)

[1]Instituto de Automática e Informática Industrial, Universitat Politècnica de València, 46022 València, Spain
[2]Center of Excellence DEWS, Università degli Studi dell'Aquila, 67100 L'Aquila, Italy

Corresponding author: Ana Guasque (anguaor@ai2.upv.es)

This work was supported in part by Ministerio de Ciencia e Innovación (MCIN)/Agencia Estatal de Investigación (AEI)/10.13039/501100011033 under Grant PLEC2021-007609, in part by the European Union NextGeneration EU/Plan de Recuperación, Transformación y Resiliencia (PRTR) [MOBILITY IN THE CITY OF THE FUTURE. PREPARING CITIES FOR THE NEW 2030 MOBILITY THROUGH THE 4 SPANISH POLYTECHNIC UNIVERSITIES (METROPOLIS)], and in part by MCIN/AEI/10.13039/501100011033/ [Modelos y plataformas para sistema informáticos industriales predecibles, seguros y confiables (PRESECREL)] under Grant PID2021124502OB-C41.

ABSTRACT Multicore systems were developed to provide a substantial performance increase over monocore systems. But shared hardware resources are a problem as they add unpredictable delays that affect the schedulability of multicore hard real-time systems. In recent years much effort has been put into modelling interference and proposing scheduling techniques to mitigate its negative effect. Using one of these models, we propose a scheduling technique, based on Integer Linear Programming (ILP) that, in combination with a task to core allocator, not only achieves a feasible schedule but also reduces the interference produced by shared hardware resources in the context of hard real-time multicore systems. The evaluation shows how interference is reduced ($\approx$ 83.47%) and schedulability is increased ($\approx$ 12.25%) compared to traditional heuristics.

INDEX TERMS Integer linear programming, optimization, partitioned systems, real-time systems, static scheduling.

I. INTRODUCTION

The use of embedded systems is widespread, not only in the industrial sector, but in all aspects of modern life. The processing power of multicore systems allows multiple embedded applications to be used on a single shared hardware platform.

In sectors where applications are highly critical, no failure is allowed as it can have catastrophic consequences. In such applications, due to certification requirements, the allocation of all resources must be static [1]. In this paper we will focus on the static allocation of temporal resources, i.e. on the scheduling of multicore systems for highly critical applications.

Existing theory in the field of multicore systems shows that scheduling on such systems is complex (NP-Hard). The generation of cyclic plans from the temporal requirements of several applications in a multicore system requires the use of techniques and heuristics that attempt to achieve feasible schedules in limited time. A large number of additional elements such as the different criticality of the applications, the assignment of tasks to cores, the management of energy consumption, the optimisation of the operating system performance, etc., adds to the complexity and difficulty of generating feasible and efficient schedules.

But there is not only difficulty in generating a feasible plan in multicore systems, it is also difficult to estimate the computation time of the tasks. In multicore systems, there are some sources of indeterminism due to the use of shared hardware resources such as memory, memory bus or cache [2]. This causes contention between tasks in different cores, which is reflected in delays in task execution. These delays, also called interferences, are non-deterministic and pose a challenge in multicore scheduling techniques. The position paper CAST-32A on multicore processors [1] identifies topics that could impact the safety, performance and integrity of airborne software systems and lists a set of objectives to help addressing multicore certification challenges. One of these challenges is

The associate editor coordinating the review of this manuscript and approving it for publication was Sergio Consoli.

interference due to contention, that is absorbing considerable research efforts in both real-time industry and academic community.

In recent years much effort has been put into modelling interference and proposing scheduling techniques to mitigate its negative effect. Two different approaches are commonly used to model interference: using a model specific to the type of shared hardware or proposing a general model that is valid for any type of hardware. The former gives a more accurate interference value but it is only valid for the hardware for which it has been calculated. Moreover, this interference value is added to the Worst Case Execution Time (WCET), making it a very pessimistic solution. The latter obtains a higher value of the interference but by adding this parameter to the temporal model independently of the WCET, it is possible to obtain a less pessimistic model. But it is necessary to propose a temporal model to incorporate this new interference parameter.

Regarding the scheduling of such systems, there are also numerous works that have addressed the scheduling and schedulability of partitioned real-time multicore systems (migration is not allowed, as the allocation must be static). This scheduling is done in two phases: first, the tasks are assigned to the cores, and then each core schedules its tasks. If interference is not taken into account, the scheduling of each core is independent. But if interference is taken into account, the execution of tasks in one core affects the scheduling of the other cores. This way, the timing correctness of the hard real-time system becomes more complex.

The main objective of this work is to obtain a static plan for critical multicore systems that is not only able to execute tasks within their deadlines but also to reduce interference as much as possible. To do this, we will use ILP techniques in the scheduling phase.

A. CONTRIBUTION AND OUTLINE

The main contributions of this work are:

- Proposal of a general ILP formulation for scheduling multicore systems with the goal of reducing interference due to shared hardware resources.
- Improvement of the above proposal to schedule independently each busy period to improve efficiency of the ILP technique (rolling horizon approach).
- Proposal of a heuristic (combined scheduler) that, based on known schedulers, chooses the best in terms of interference reduction for each busy period.
- Proposal of combination of the rolling horizon approach and the combined scheduler to achieve a reduction of the interference.
- Evaluation of our proposal in combination with two allocators in order to know which combination of allocator and scheduler achieves the best performance in terms of schedulability and interference reduction.

This paper is organised as follows: Section II briefly comments relevant papers in the area, Section III defines the temporal model used and the problem to solve while in Section IV a first approximation of the ILP model is presented. In order to improve the presented model, Section V presents the concept of busy period in partitioned multicore systems and Section VI proposes a scheduling technique to reduce interference. With the results of the previous sections, in Section VII, we present a ILP technique that obtains a scheduling plan that minimizes interference. The evaluation is presented in Section VIII and conclusions and further work is detailed in Section IX.

II. RELATED WORKS

There has been a trend towards using multicore platforms due to their high computing performance. From the key results in the field in 2006, there is a lot of research about real-time multicore systems. Some of the main surveys in the area are [3] and [4].

This work is focused in partitioned hard real-time systems that take into account interference due to contention. The most recent survey about interference and mitigation of its effect in the scheduling can be found in [5] (until 2021).

Regarding contention models (specific or general), many works consider a specific hardware shared resource: memory bus ([6], [7]), scratchpad memories and DRAM ([8], [9]), etc. For example, in [10], it is performed an analysis of interference due to accesses to DRAM in heterogeneous commercial-off-the-shelf (COTS) MPSoC platforms focused on mixed-criticality systems. Our approach, on the contrary, is to consider a general model, independent and valid for any kind of shared hardware.

The works that consider a general model are closer to our work. Altmeyer et al. [11] presented a Multicore Response Time Analysis (MRTA) framework, that provides a general approach to timing verification for multicore systems. They omit the notion of WCET and instead directly target the calculation of task response times through execution traces. They start from a given mapping of tasks to cores and assume fixed-priority preemptive scheduling. Other works as [12] or [13] come from the MRTA framework. In [14], a schedulability test and response time analysis for constrained-deadline systems is proposed. They analyse the amount of time for shared resource accesses and the maximum number of access segments, which is out of the scope of this work. They also assume that task priority levels are assigned *a priori*. Choi et al. in [15] propose a conservative modeling technique of shared resource contention supporting dependent tasks, in contrast to our work, that considers independent tasks. They also assume fixed-priority scheduling. The work presented in [16] considers constrained-deadline sporadic task sets and a fixed priority scheduling. Here, tasks are represented by a sequence of segments, each of which has execution requirements and co-runner slowdown factors with respect to sets of other segments that could execute in parallel with it. Our model is more general, in the sense that we do not split tasks into segments.

In [17], the interference due to contention is added to the temporal model. Instead of adding it to the WCET, they propose a scheduling algorithm that computes the exact value of interference and an allocator that tries to reduce this total interference. This model considers implicit deadlines in the system and both fixed or dynamic priorities can be used. However, no action is taken in the scheduling phase to reduce interference, only in the phase of allocating tasks to cores. Our aim is to extend this work to obtain a schedule that minimizes interference. A similar work is presented in [18]. They define the Multicore Resource Stress and Sensitivity (MRSS) task model that characterises how much stress each task places on resources and its sensitivity to such resource stress. This work also considers a general model to cope with different hardware resources but only fixed priority scheduling policies are considered (preemptive and non-preemptive). They propose a schedulability analysis and a priority assignment to maximize schedulability but no actions are taken to reduce interference.

All of the above works are based on a specific scheduling algorithm (fixed priority most of the time). This conditions the amount of interference that can occur. Our approach is to use an ILP technique, so that we do not depend on a specific scheduling algorithm and decisions about which task to execute at which time are made with the aim of reducing interference.

Regarding the use of ILP techniques in real-time systems, Guasque et al. [19] proposes a technique based on rolling horizons for monocore systems. In multicore systems, most works that use ILP for scheduling do so with the objective of minimising power consumption [20], [21]. In [22] a new method for solving complex scheduling problems of real-time in multicore platforms is proposed using a directed acyclic graph (DAG) to represent the scheduling of the workload, where each vertex represents a processor of the system. Our model differs from this work in the sense that we assume a periodic temporal model.

III. TASK MODEL AND PROBLEM STATEMENT

A. PERIODIC TASK MODEL

Let us suppose a multicore system with m homogeneous cores $(M_0, M_1, M_2, \ldots, M_{m-1})$ where a task set τ of n independent preemptive periodic or sporadic tasks should be allocated to. Each task τ_i is represented by the tuple:

$$\tau_i = (C_i, D_i, T_i, I_i) \tag{1}$$

where C_i is the WCET, D_i is the relative deadline, T_i is the period, and I_i is the interference factor over other tasks. Constrained deadlines are considered, so $D_i \leq T_i \quad \forall i$.

The term I_i is the time spent by a task in accessing some shared hardware resource. A typical case is memory read and write operations. Although I_i is part of C_i, during the time the task accesses the shared resource, other tasks on other cores will be delayed if they try to access the shared resource. Therefore, this interference time is defined independently of C_i, as it will be used to represent the delay caused to other tasks.

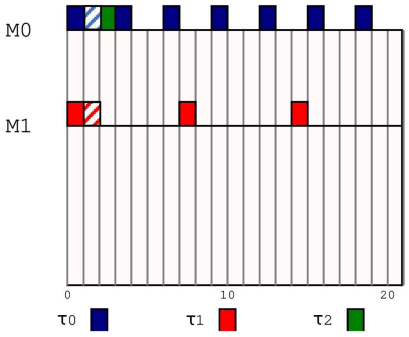

FIGURE 1. Example of chronogram under EDF.

A more detailed description of the interference factor parameter can be found in [17] but, because of its importance, we will illustrate how to schedule the task model with a simple example. First, we need the following definitions:

Definition 1 [17]: A task is defined as a receiving task when it accesses shared hardware resources and suffers an increase in its computation time due to the interference produced by other tasks allocated to other cores.

Definition 2 [17]: A task is defined as a broadcasting task when it accesses shared hardware resources and provokes an increase in computation time in other tasks allocated to other cores due to contention.

If $I_i = 0$, τ_i is neither broadcasting nor receiving task. If $I_i > 0$, τ_i will be a broadcasting and receiving task if there is at least one task τ_j in other core whose $I_j > 0$.

Then, let us proceed with the simple example. Let us assume a system with 2 cores: M_0 and M_1, and 3 tasks: τ_0, τ_1 and τ_2 with the following parameters: $\tau_0 = (1, 3, 3, 1)$, $\tau_1 = (1, 7, 7, 1)$, $\tau_2 = (1, 21, 21, 0)$. We say that τ_0 and τ_1 are broadcasting tasks since its $I_i \neq 0$. Tasks τ_0 and τ_2 are allocated to core M_0 and τ_1 is allocated to core M_1. Figure 1 shows the scheduling of the example under Earliest Deadline First (EDF) algorithm. The novelty is that when two tasks in different cores coincide in execution, they cause each other a delay that is the interference factor, if it is not 0. That is why τ_0 suffers 1 unit of interference (blue dash lines) that corresponds with I_1. And vice versa, τ_1 suffers 1 unit of interference (red dash lines) that corresponds with I_0.

The theoretical utilisation of a core M_k, U_{M_k}, is the fraction of processor time spent executing the tasks allocated to this core. It is calculated by summing up the theoretical utilisation of each task τ_i that belongs to that core ($U_i = C_i/T_i$). In a multicore system, the utilisation of a core does not only depend on the processor time spent executing the computation time of the tasks but also on the interference produced when tasks are executed simultaneously on different cores. For this reason, this paper defines the actual utilisation of a core M_k, U'_{M_k}, as the sum of the actual utilisations (U'_i) of the tasks that belong to that core. Therefore, the theoretical system utilisation U_τ is the sum of the utilisations of all cores

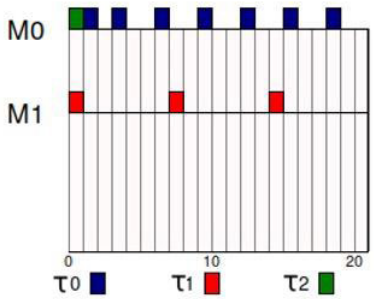

FIGURE 2. Example of chronogram.

and the actual system utilisation is denoted by U'_τ and takes into consideration the interference.

The hyperperiod of the task set is the smallest interval of time after which the periodic patterns of all the tasks are repeated. It is calculated as the least common multiple of the tasks' periods and it is denoted as lcm. During the hyperperiod, each task τ_i is activated $N_i = \text{lcm}/T_i$ times.

B. PROBLEM STATEMENT

As explained in the previous section, tasks can suffer delays due to interference of other co-runners tasks. This interference is unpredictable in the sense that depends on the tasks that are running in different cores at each instant of time. We model this delay in the worst case, so it can cause that a system becomes unfeasible if not taken into account. Different scheduling decisions can cause different amounts of interference. For example, Figure 2 shows the same example of Figure 1 but with a slightly different order of execution. Specifically, τ_2 is executed before τ_0. This means that now, τ_0 and τ_1 are not co-runners so they do not caused interference to each other. The result is that, by slightly changing the execution order, we have reduced the interference to 0. The resulting schedule does not correspond to any optimal priority assignment.

The aim of this work is to generate a static scheduling plan for multicore systems. Integer Linear Programming (ILP) techniques will be used to find a scheduling plan that reduces as much as possible the total system interference due to accesses to the shared hardware. We will assume that tasks are already allocated to cores and they cannot migrate. The solution obtained by ILP is the static scheduling plan that it is made off line so there is not overhead due to the time needed to find a solution.

IV. MULTICORE LP SCHEDULING

The first approximation to generate a static scheduling plan for multicore systems is to propose a complete LP model. This model addresses the multicore scheduling problem, assuming that the allocation phase has already been done.

Every optimization model has an objective function, which is the function on the decision variables that we wish (in this case) to minimize. The decision variables capture the results of the optimization. In a feasible solution, the computed values for the decision variables satisfy all of the model constraints. Finally, constraints capture a restriction on the values that a set of variables may take.

If we now apply it to our problem, the decision variables are: the tasks that are executed at each time on each core (binary, executed or not), the interference (binary, exists or not), and the response time of each task (continuous, time units). Then, the model we are dealing with is of the mixed integer type because it copes with integer and continuous variables. Our constraints are based on schedulability criteria: tasks must end before the arrival of their deadlines, multiple tasks can not be executed simultaneously in the same processor, etc. All constraints are linear and therefore, the problem is categorized as Mixed Integer Linear Problem (MILP).

In the following, we propose a complete MILP model that solves the multicore scheduling problem with interference. First, in Table 1 the notation is proposed.

The objective function to be solved is to minimize the interference and the response time of the tasks. The reason why the combination of two parameters has been chosen is because if we minimise interference alone, plans with many context switches could be produced:

$$\min \quad \text{Obj} = \frac{1}{\text{maxI}} \sum_{\substack{\forall i,k \in \tau \\ \forall a \in N_i \\ \forall b \in N_j}} m_{iakb} + \sum_{\forall i,a} \frac{w_{ia}}{D_i} \tag{2}$$

According to the problem statement, the objective function is defined in Equation 2, which is minimizing the total number of interferences (first term) and the total response time for all tasks in all system executions (second term). The problem is considered as multiobjective because it tries to reduce both interferences and response times.

Therefore, it is important to note that minimization is not only about interference, but also about optimising interference and response times. Therefore, it would not be correct to talk about interference minimisation because we could obtain a plan with less interference but with many context switches, which is not desirable. Moreover, as we will see below, there will be cases where a solution cannot be found in a reasonable time, which is why alternatives are proposed in the following sections.

The range of values of interferences and response times are different. Thus, we need to normalize both variables to be of a similar scale. We scale response times by dividing by deadlines for each activation and task. The sum of interferences is scaled by the maximum possible interferences. In this way, both values are in the range [0,1] and there is a fair trade-off between competing objectives.

The maximum possible interference in the system is calculated from the array $\overrightarrow{v_{j \to i}}$, which is the activation pattern from a broadcasting task τ_j to a receiving task τ_i and was defined in [23]. Let us use this definition to obtain a pessimistic value called maxI, so that dividing the real number of interferences between maxI scales the interference objective in the range [0,1]. maxI (Equation 3) sums the interference that all the receiving tasks ($I_i \neq 0$) receive from broadcasting tasks

TABLE 1. Model notation of the MILP problem.

Sets and indices	
i	Tasks $\tau_i \in \{0, 1, 2, ..., n-1\}$
a	Activations of $\tau_i \in \{0, 1, 2, ..., N_i - 1\}$
j	Cores $M_j \in \{0, 1, 2, ..., m-1\}$
Parameters	
C_i	Worst case execution time of τ_i
D_i	Relative deadline of τ_i
d_{ia}	Absolute deadline of τ_i in activation a
T_i	Period of τ_i
I_i	Interference factor of τ_i over other tasks
N_i	Number of activations of τ_i
R_{ia}	$[a \cdot T_i, (a+1) \cdot T_i]$ Possible execution time interval for τ_i in activation a
o_{ij}	1 if τ_i is allocated to core M_j and 0 otherwise.
lcm	Hyperperiod of the task set
maxI	Maximum possible received interferences
Decision variables	
x_{iajt}	1 if τ_i in activation a is allocated to core M_j and executed at time t and 0 otherwise.
m_{iakb}	Interference matrix. 1 if execution of τ_i in activation a coincides with the execution of τ_k in activation b and 0 otherwise.
w_{ia}	Response time matrix. Response time of τ_i in activation a

$(I_j \neq 0)$ in other cores $(M_{\tau_i} \neq M_{\tau_j})$ in all its activations a, which are in the range $a \in [0, N_i - 1]$.

$$\text{maxI} = \sum_{\substack{\forall i,j \in \tau \\ M_{\tau_i} \neq M_{\tau_j} \\ I_i, I_j \neq 0}} \sum_{0 \leq a \leq N_i - 1} \overrightarrow{v_{j \to i}}[a] \cdot I_j \tag{3}$$

In the following sections we explain the constraints of the problem, dividing them according to their purpose.

A. ASSURANCE OF COMPUTATION TIME CONSTRAINTS

Constraint 4 ensures that all activations a of τ_i are executed in its activation intervals R_{ia}. Inside each of these intervals, the sum of the WCET (first term) and (if exists) the received interference (second term) must be executed.

$$\sum_{t \in R_{ia}} x_{iajt} = C_i \cdot o_{ij} + \sum_{\substack{k \in \tau \\ k \neq i \\ o_{ij} \neq o_{kj}}} m_{iakb} \cdot I_k \cdot o_{ij}$$
$$\forall i, a, j, k, b | k \in \tau, b \in N_k \text{ if } I_i, I_k \neq 0 \tag{4}$$

B. REAL-TIME CONSTRAINTS

This includes restrictions to ensure the temporary compliance of the system. In this sense, constraint 5 ensures that all tasks end before the arrival of their deadlines. Constraint 6 ensures that only one task is being executed at each point in time at each core. Constraint 7 ensures that tasks are not executed outside their activation intervals. Constraint 8 reduces some variables because of the known information. As the allocation is known *a priori*, x matrix can be reduced, assuming that if a task i is not allocated to a core M_j, $(o_{ij} = 0)$, then the execution of that task in that core is not possible $(x = 0)$.

$$t \sum_{\forall j} x_{iajt} \leq d_{ia} - 1 \quad \forall i, a, t | t \in R_{ia} \tag{5}$$

$$\sum_{\forall i,a} x_{iajt} \leq 1 \quad \forall j, t \tag{6}$$

$$x_{iajt} = 0 \quad \forall i, a, j, t | t \notin R_{ia} \tag{7}$$

$$x_{iajt} = o_{ij} \quad \forall i, a, j, t \text{ if } o_{ij} = 0 \tag{8}$$

C. INTERFERENCE CONSTRAINTS

This computes the total interference between each pair of activations and tasks, for all the tasks in the system. Constraints 9 and 10 calculate the produced interference. When activation a of τ_i and activation b of τ_k do coincide in execution at any time t, then m_{iakb} is equal to one. With constraint 11 we assure that m_{iakb} is equal to zero when tasks τ_i and τ_k do not coincide in execution. Because of the periods, there are activations that can not possibly overlap. In particular, this can not happen when the execution times for τ_i at activation a do not intersect with the execution times for τ_k at activation b.

$$m_{iakb} \geq x_{iajt} + x_{kblt} - 1 \quad \forall i, a, k, b, j, t | \forall k \in \tau, b \in N_k, t \in R_{ia} \cap R_{kb}, \text{ if } i \neq k \text{ and if } j \neq l \tag{9}$$

$$m_{iakb} = m_{kbia} \quad \forall i, a, k, b | k \in \tau, b \in N_k, \text{ if } k \neq i \tag{10}$$

$$m_{iakb} = 0 \quad \forall i, a, k, b | k \in \tau, b \in N_k \text{ if } k > i \text{ and if } R_{ia} \cap R_{kb} = \emptyset \tag{11}$$

D. RESPONSE TIME CONSTRAINT

Constraint 12 calculates the response time $(w_{ia} \in \mathbb{N}^+)$ of each activation of all tasks, as a function of the variable x.

$$w_{ia} \geq t \cdot x_{iajt} - aT_i + 1 \quad \forall i, a, j, t | t \in R_{ia} \tag{12}$$

E. DECISION VARIABLE DOMAIN

Constraints 13 and 14 represent the decision variable domains.

$$x_{iajt}, m_{iakb} \in \{0, 1\} \tag{13}$$

$$w_{ia} \geq 0 \tag{14}$$

Regarding the number of constraints, this is not always a good metric of the complexity of the model. For example, there are models with millions of variables and constraints that are solved in a few seconds and, on the contrary, seemingly simple models that take days to be solved. The complexity depends on the sparsity on the constraint matrix and the number of integer variables in linear or mixed-integer models, as is the case. A more complicated scenario happens with non-linear models or non-convex constraints. So we can not conclude that our model is complex because of the constraints. In fact, the proposed constraints only

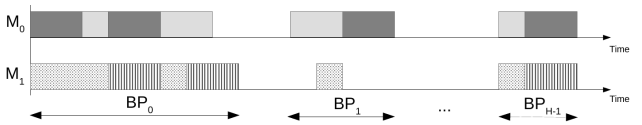

FIGURE 3. Busy period definition.

assure the proper functioning of a periodic system and evaluate the produced interference. The objective function is in charge of minimizing this interference.

The previous model solves the problem of scheduling a multicore hard real-time system. However, as stated in [19] the problem can become intractable, as its size is directly proportional to the number of tasks, task utilisation and hyperperiod. In this case, an additional dimension is added, namely the number of cores.

To overcome this drawback, we propose an alternative to efficiently solve the problem. The primary idea is to decompose the whole scheduling window (hyperperiod) into many short subproblems, called rolling horizons. The Rolling Horizon approach is used to reduce the computational time to solve big problems with many variables. In this work, each of these rolling horizons corresponds with a busy period that will be explained in the following section.

V. SYSTEM BUSY PERIODS

A busy period BP_i is an interval of time in which the processor has ready tasks to execute. The concept was first introduced in [24] for monocore. A way to calculate the busy periods in EDF is presented in [25].

We extend the definition of busy periods applied to multicore systems:

Definition 3: In partitioned multicore systems, a busy period (BP_i) is the time interval in which there is no idle time in all the cores simultaneously.

Note that this definition is different for global multicore scheduling in [26] in which an *all busy period* is a consecutive time interval in which there are always ready jobs or waiting to be executed on every core.

According to the above definition, busy periods can be derived using the monocore busy periods.

For example, Figure 3 shows the characterization of the busy periods for a specific scheduling plan. The end of each busy period is the time in which all cores in the system are in idle state simultaneously.

Algorithm 1 shows how to calculate the busy periods in multicore systems from the busy periods[1] in monocore systems. It is obtained from all the execution intervals that compose the plan, including the executions in all cores. It is a matter of combining all intervals until a point in which there is no execution. It is important to note that monocore busy periods take into account the interference caused by other cores.

[1]Each interval is characterised by a start and an end and it is denoted as the following superscripts: BP_i=[BP_i^s,BP_i^e].

Algorithm 1 Obtaining Busy Periods (BP)

```
INPUT: Monocore busy periods (MBP) of all tasks in all cores from the scheduling plan → [MBP0,MBP1,...]
OUTPUT: Set of busy periods →[BP0,...,BPH-1]
procedure Obtaining busy periods
    Temporarily ordering of the execution intervals (by starting times)
    Initialise list of busy periods → BP=[MBP0]
    Initialise variables i←1, j←0
    for MBPi ∈ Execution Intervals [MBP1,...] do
        if BPj^e ≥ MBPi^s then
            if MBPi^e > BPj^e then
                BPj^e = MBPi^e
        else
            j = j + 1
            BPj←MBPi
        i = i + 1
```

Therefore, during a complete hyperperiod, cores alternate between idle and busy states that is, between busy and idle periods. Busy periods are independent of each other. This means that we can apply different scheduling strategies in each busy period and this does not affect the others.

Now that we have split the hyperperiod into busy periods and we know that we can schedule differently in each one without affecting the rest of the intervals, we are going to schedule each busy period with several conventional scheduling algorithms and we will choose the one that obtains the least interference. This is useful for two reasons:

- to have the calculation of each busy period that we need to develop the rolling horizon method (INPUT of Algorithm 2),
- to have a non-optimal solution in case the ILP method is not able to find one.

Note that in multicore scheduling with interference, the length of a busy period depends on the chosen scheduling algorithm. In the following section, we explain the technique of obtaining the scheduling of each busy period, which we have called the combined scheduler.

VI. COMBINED SCHEDULER

The Combined Scheduler (CS) is a scheduling algorithm that uses or combines different scheduling policies such as EDF or Deadline Monotonic (DM) [27] in a single schedule. The key of the CS is that it schedules interval by interval, and in each interval it can apply a different scheduling policy. The selected policy is the one that generates the best results for that interval at certain temporal parameters.

In the present case, the CS selects the policy that reduces the interference due to accesses to the shared hardware in each busy period. The CS chooses between the following scheduling policies: EDF, DM and their variants. The variants covered in this work are:

- Variant 1: consists of non-preempting a running task when a higher priority task is activated under certain conditions. The task is not preempted if the remaining time for the running task to finish is less than the computation time of the higher priority task. In this case, the running task would not be preempted, inheriting the priority of the task that wants to preempt it.
- Variant 2: consists of non-preempting a running task for N time units either in the case that the task is about to start execution or that the task has been previously preempted.

In sort, there are six schedulers to be chosen: classic EDF, classic DM, EDF variant 1, DM variant 1, EDF variant 2 and DM variant 2.

In [28], Salmani et al. propose MMUF algorithm, that combines fixed and dynamic scheduling to feasible schedule *transition overloaded* systems (load $\geq$ 100%). When the processor utilization is less or equal to one, EDF is a special case of the MMUF algorithm [29]. As our work is focused in critical systems, we will not find the case of an overloaded system. Therefore, EDF and its variants may be considered as variants of MMUF algorithm.

For a better understanding of the combined scheduler, an example is presented here. Let us assume a system with 2 cores: M_0 and M_1, and 4 tasks: τ_0, τ_1, τ_2 and τ_3 with the following parameters: $\tau_0 = (2, 6, 6, 0)$, $\tau_1 = (3, 10, 10, 1)$, $\tau_2 = (2, 7, 7, 1)$ and $\tau_3 = (3, 9, 9, 0)$. To simplify the example, we will assume that CS can only choose between classic EDF and EDF variant 1. As a first step, the CS schedules the task sets with classic EDF for the first busy period. As it can be seen in Figure 4, it obtains a busy period from instant 0 to instant 5 and 0 units of interference. Thus, the CS schedules the task sets for the same busy period again, but now with EDF variant 1. The result is also 0 units of interference as it can be seen in Figure 4, so in this busy period both policies obtain the same result. Since all options have the same result, the CS will choose the assigned policy by default, in this case, classic EDF. Once the first busy period is solved, CS continues the scheduling at instant 6 and the process is repeated again.

For the second busy period, we can see in Figure 5 that classic EDF obtains 2 units of interference while in Figure 6 EDF variant 1 obtains 0 units of interference. It is relevant to note that the size of the second busy period is different, from instant 6 to 16 (instead of 17 of classic EDF), and that is fully associated by the interference produced between the tasks. Therefore, the CS chooses EDF variant 1 as the appropriate policy for the second busy period. The CS would then continue with the same process to solve the next busy period in the schedule and so on until the hyperperiod is reached.

VII. ROLLING HORIZON MILP MODEL

The previous approach basically applies different known scheduling policies at each busy period and selects the one that provides better results in terms of interference. As each busy period is isolated from its neighbours by idle instants, each one can be scheduled independently as the CS does. However, the CS does not obtain an optimized solution in terms of minimum interference and this interference could be further reduced. In this section, we propose a MILP technique to obtain a static schedule with the optimization criteria of minimize interference.

We use the concept of *rolling horizon*. Figure 7 shows the main idea. The goal is to find a schedule for all the hyperperiod which is the scheduling horizon. As trying to solve the problem for the entire horizon is computationally

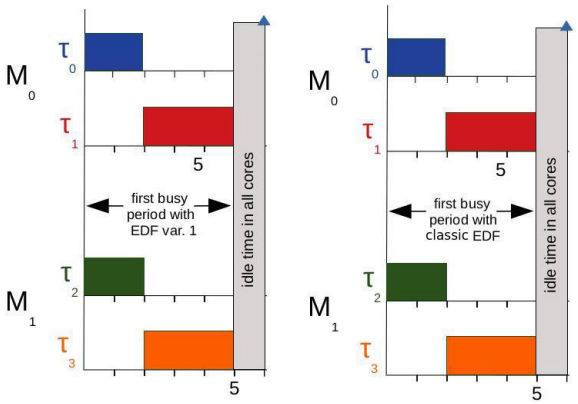

FIGURE 4. Resulting chronogram after scheduling the first busy period with EDF in variant 1 (left) and classic EDF (right).

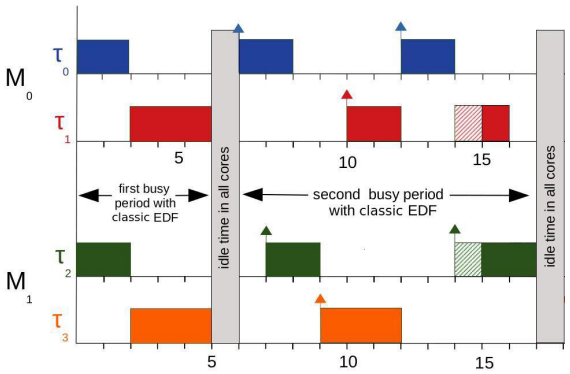

FIGURE 5. Resulting chronogram after scheduling the second busy period with classic EDF.

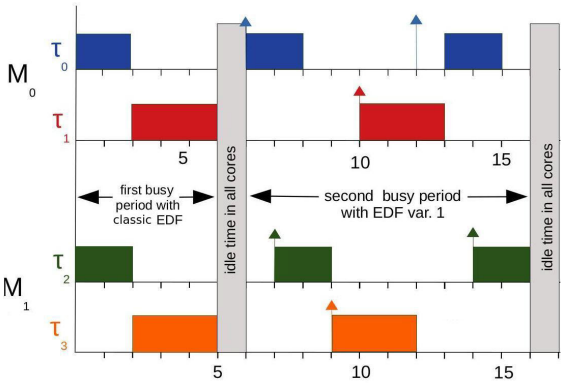

FIGURE 6. Resulting chronogram after scheduling the second busy period with EDF in variant1.

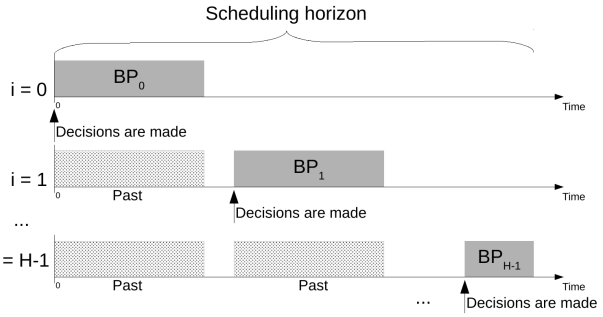

FIGURE 7. Rolling horizon description.

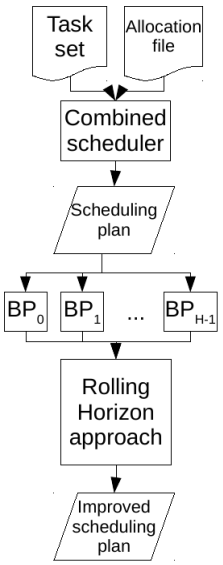

FIGURE 8. Rolling horizon schema.

very expensive, the problem is divided into smaller horizons in a "divide and conquer" strategy. In our case, each smaller horizon is a busy period.

The methodology is depicted in Figure 8. As seen in the Figure, once the CS computes the feasible scheduling plan for the task set, we deduce the system busy periods or rolling horizons.[2] In each rolling horizon we will solve a MILP problem, the Rolling Horizon MILP Algorithm (RHMA) listed in Algorithm 2.

The RHMA algorithm works as follows: it receives a task set, with a known task-to-core allocation and the set of busy periods from the plan generated by the CS. As the problem is to be posed as MILP, different parameters and variables must be introduced. They are defined in Table 2. For each rolling horizon (BP), all parameters from Table 2 are calculated from the data input (line 6). Then, the variable matrices are declared. In order to improve the efficiency of the algorithm, both variables and constraints are declared at the beginning and removed at the end of the interval. In this way, the size of the problem is significantly reduced. Moreover, some of the variables may be simplified because of known information from the input data, as happened in the complete model presented in Section IV. Then, the constraints and the objective are added and the solver starts the optimization (lines 10 and 11). If it reaches the optimal or a feasible solution in the specified time (a feasible scheduling plan in that rolling horizon), the scheduling plan will be saved (line 13), all the constraints and variables will be removed (line 17), the model will be updated (line 18) and then the system will move to the next rolling horizon. This is repeated in all rolling horizons. If, at some point, the solver can not feasibly schedule the tasks at any rolling horizon, this interval will be scheduled by the combined scheduler (line 15).

[2]The terms "busy periods" and "rolling horizons" are used interchangeably throughout this text.

Algorithm 2 Rolling Horizon MILP Algorithm (RHMA)

1: INPUT: Busy periods from CS, Task set and task-to-cores allocation
2: OUTPUT: Scheduling plan, σ
3: **procedure** Rolling horizon MILP algorithm
4: Temporarily ordering of busy periods $\rightarrow$ [BP_0,...,BP_{H-1}]
5: **for** $BP_h \in$ Busy Periods [BP] **do**
6: Calculate parameters
7: Define and reduce variable matrix x
8: Define and reduce variable matrix m
9: Define variable matrix w
10: add Constraints
11: set Objective and optimize
12: **if** Optimal or Feasible **then**
13: $\sigma_h = x_{iajt}$ (Save scheduling plan for BP_h)
14: **else**
15: Usage of the CS schedule in this BP
16: **if** h < H-1 **then**
17: Remove constraints and variables
18: Update the model

TABLE 2. Model notation of the RHMA.

Sets and indices	
i	Tasks $\tau_i \in \{0, 1, 2, ..., n-1\}$
a	Activations of $\tau_i \in \{0, 1, 2, ..., N_i - 1\}$
j	Cores $M_j \in \{0, 1, 2, ..., m-1\}$
h	Busy periods $BP_h \in \{0, 1, 2, ..., H-1\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Relative deadline of τ_i
d_{ia}	Absolute deadline of τ_i in activation a
T_i	Period of τ_i
I_i	Interference factor of τ_i over other tasks
N_i	Number of activations of τ_i
o_{ij}	1 if τ_i is allocated to core M_j and 0 otherwise.
S_{ih}	Activations of task τ_i executed in the busy period BP_h
R_{iah}	Possible execution time interval for task τ_i in its activation a in the busy period BP_h
T_h	Execution times for the busy period BP_h
lcm	Hyperperiod of the task set
maxI	Maximum possible received interferences
Decision variables	
x_{iajt}	1 if task i in activation a is allocated to core j and executed at time t and 0 otherwise.
m_{iakb}	Interference matrix. 1 if execution of τ_i in activation a coincides with the execution of τ_k in activation b and 0 otherwise.
w_{ia}	Response time matrix. Response time of τ_i in activation a

It is important to note that the difference between the methodology presented in Section IV and Algorithm 2 is that in Algorithm 2 the optimisation is performed in each interval and Section IV performs over the entire hyperperiod.

In the following, we are describing the objective function and the constraints for the MILP model. As happened in the complete model in Section IV, the objective function consists of minimizing the total number of interferences and the total response time for all tasks in all system executions.

$$\min \quad \text{Obj} = \frac{1}{\text{maxI}} \sum_{\substack{\forall i,k \in \tau \\ \forall a \in S_{ih} \\ \forall b \in S_{kh}}} m_{iakb} + \sum_{\forall i, a \in R_{iah}} \frac{w_{ia}}{D_i} \tag{15}$$

The constraints of the problem are defined as follows, similarly to the model presented in Section IV. The difference lies in the fact that the previous problem covered the entire hyperperiod and this model moves from interval to interval. Then, for each busy period BP_h:

$$x_{iajt} = o_{ij} \quad \forall i, a, j, t | i, a \in R_{iah}, t \in T_h \quad \text{, if } o_{ij} = 0 \tag{16}$$

$$m_{iakb} = 0 \quad \forall i, a \in R_{iah}, \forall k, b \in R_{kbh} \text{ if } k > i \text{ and if } R_{iah} \cap R_{kbh} = \emptyset \tag{17}$$

$$\sum_{\substack{a \in S_{ih} \\ t \in R_{iah}}} x_{iajt} = \text{len}(S_{ih}) \cdot C_i \cdot o_{ij} + \sum_{\substack{k \neq i \\ a \in S_{ih} \\ b \in S_{kh}}} m_{iakb} \cdot I_k \cdot o_{ij} \quad \forall i, j, \text{ if } o_{ij} \neq o_{kj} \text{ and if } I_i, I_k \neq 0 \text{ and if } \text{len}(S_{ih}), \text{len}(S_{jh}) > 0 \tag{18}$$

$$\sum_{t \in R_{iah}} x_{iajt} = C_i \cdot o_{ij} + \sum_{\substack{k \neq i \\ b \in S_{kh}}} m_{iakb} \cdot I_k \cdot o_{ij} \quad \forall i, j, a | a \in S_{ih}, \text{ if } o_{ij} \neq o_{kj} \text{ and if } I_i, I_k \neq 0 \text{ and if } \text{len}(S_{ih}), \text{len}(S_{jh}) > 0 \tag{19}$$

$$t \sum_{j \in J} x_{iajt} \leq d_{ia} - 1 \quad \forall i, a, t | a \in S_{ih}, t \in T_h \tag{20}$$

$$\sum_{\substack{\forall i \\ a \in S_{ih}}} x_{iajt} \leq 1 \quad \forall j, t | t \in T_h \tag{21}$$

$$m_{iakb} \geq x_{iajt} + x_{kblt} - 1 \quad \forall i, a, k, b, j, l, t | a \in S_{ih}, b \in S_{kh}, t \in T_h, \text{ if } i \neq k \text{ and if } j \neq l \tag{22}$$

$$m_{iakb} = m_{kbia} \quad \forall i, a, k, b | a \in S_{ih}, b \in S_{kh}, \text{ if } k \neq i \tag{23}$$

$$w_{ia} \geq t \cdot x_{iajt} - aT_i + 1 \quad \forall t, i, a, j | a \in S_{ih}, t \in T_h \tag{24}$$

$$x_{iajt}, m_{iakb} \in \{0, 1\} \tag{25}$$

$$w_{ia} \geq 0 \tag{26}$$

Constraints 16 and 17 reduce some variables because of the known information.

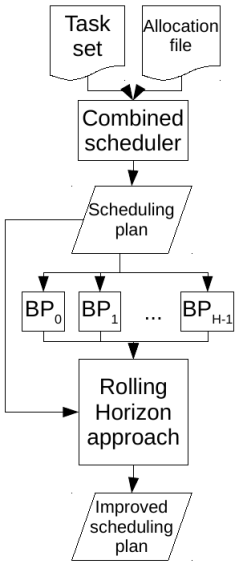

FIGURE 9. Rolling horizon with warm start schema.

Constraint 18 ensures that task τ_i completes all the computation time in the BP_h. At each busy period, a task can execute 0, 1 or more activations. Therefore, it has to complete its WCET as many times as activations occur plus the produced interference in that busy period. Similarly, constraint 19 ensures that each activation of each task is executed in its activation interval, including WCET and interference. Equation 20 ensures that all tasks end before the arrival of their deadlines. Constraint 21 ensures that only one task is being executed at each point in time at each core. Equations 22 and 23 calculate the produced interference.

Constraint 24 calculates the response time of each activation of all tasks. Equations 25 and 26 represent the decision variable domains.

A. RHMA WITH WARM START

A way of improving the RHMA performance is to make use of the CS scheduling plan, using it as an input of the RHMA (see Figure 9). The RHMA receives both the scheduling plan and the busy periods from the CS.

Warm starting information is an additional input data that allows the algorithm to speed up solving the problem, reducing fastly the distance from the optimality. In practice, it consists on providing manually starting solution vectors of the problem to the solver.

If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not feasible, the MILP solver tries to repair it. When it is difficult to find a good integer feasible solution for a problem, a warm start can significantly reduce the solution time. The effectiveness of the warm start in MILP solvers depends on many factors. Sometimes, warm starts do not help the solver to find solutions more quickly. For example, if a significant amount of time is expended proving optimality of a good solution (as is often the case), then there will be

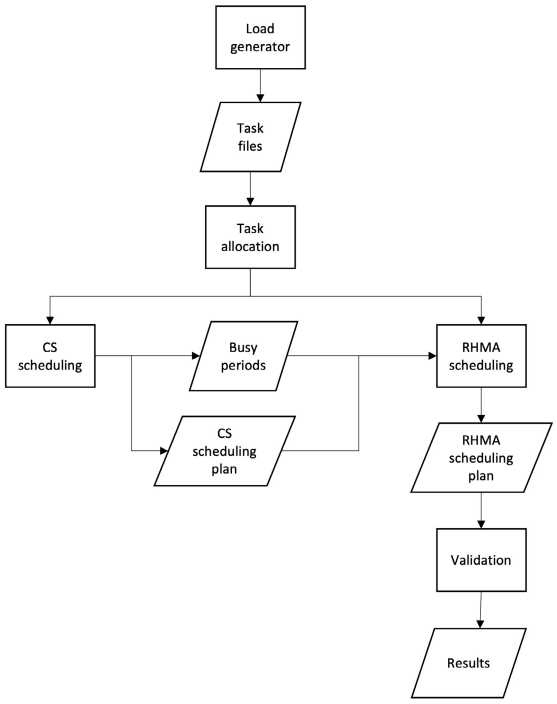

FIGURE 10. Experimental evaluation overview.

no noticeable change in run time by supplying a warm start. However, authors may consider providing feasible starting points to fix an upper bound on the objective value (in case of minimization) and thus can be used to prune nodes during the search [19].

The feasible starting point considered in this work is the scheduling plan provided by the CS.

VIII. EVALUATION

In order to assess the scheduling method proposed, we have conducted the evaluation with synthetic task loads. The methodology is shown in Figure 10, which consists of four main phases: load generation, allocation, scheduling and validation.

The load is obtained using a synthetic task generator. The inputs of this generator are: the number of cores M, the number of tasks n, the number of broadcasting tasks, the theoretical utilisation of the task set U_τ, and the interference of broadcasting tasks as a percentage of the WCET. The specific parameters used as inputs are shown in Table 3. Task parameters are obtained randomly to achieve the theoretical utilization.

The next step is the allocation of tasks to the different cores. Two different approaches are used for the allocation, Worst Fit Decreasing Utilisation (WFDU)[30] and Wmin[17]. We choose these two allocators based on the comparison made in [17] where Wmin showed the best results in terms of interference reduction and WFDU presents the best results results in terms of schedulability.

With the task set and the allocation, the CS schedules the task set and calculates the BPs, being both an input for the RHMA algorithm.

TABLE 3. Experimental parameters.

	2 cores	4 cores	8 cores
Theoretical utilisation	[1.2,1.8]	[2.5,3.5]	[5,6]
Number of tasks	[3,6]	[10,15]	[15,20]
Number of broadcasting tasks	[2,3]	[3,7]	[5,8]
Interference (% over WCET)	10%	10%	10%

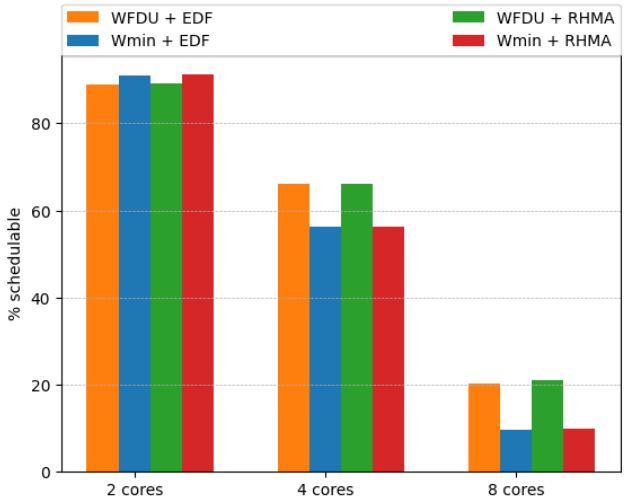

FIGURE 11. Percentage of schedulable task sets depending on the number of cores, allocators and scheduling algorithms.

The results obtained with the proposed RHMA will be compared with EDF scheduling. That is, we will compare two allocation algorithms with two schedulers resulting in four schedulers:

- WFDU+EDF
- Wmin+EDF
- WFDU+RHMA
- Wmin+RHMA

The comparison will be done in terms of schedulability and increased utilization. The schedulability measures the percentage of schedulable sets out of the total number of generated sets. The increased utilization is measured as:

$$1 - \sum_{k=0}^{k=m-1} \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau} \tag{27}$$

that is, it is the ratio between the actual utilization and the theoretical utilization.

In this work, we make use of Gurobi solver. It is available for students, faculty, and researchers to work with mathematical optimization at no cost. The experiments are performed on an Intel Core i7 3.2 GHz processor with 32 GB RAM, using Gurobi 9.5 as the MILP solver.

The results are depicted for schedulability in Figure 11.

In this figure, it can be seen that the percentage of schedulability decreases with the number of cores, as expected. Comparing allocators, WFDU obtains better schedulability rates than Wmin as concluded in [17]. With respect to the scheduling algorithms, RHMA improves EDF results in the sense that RHMA can schedule task sets that EDF is not

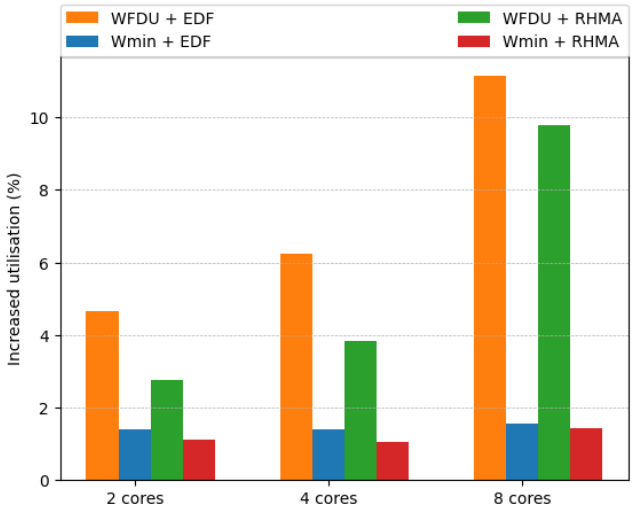

FIGURE 12. Increased utilisation of the task sets depending on the number of cores, allocators and scheduling algorithms.

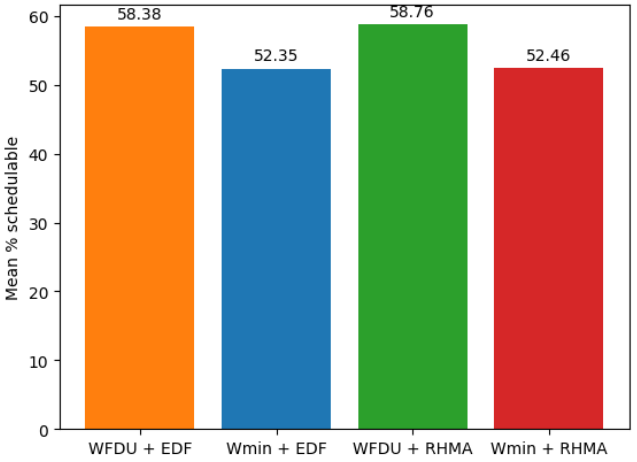

FIGURE 13. Percentage of average schedulable task sets depending on the allocators and schedulability algorithms.

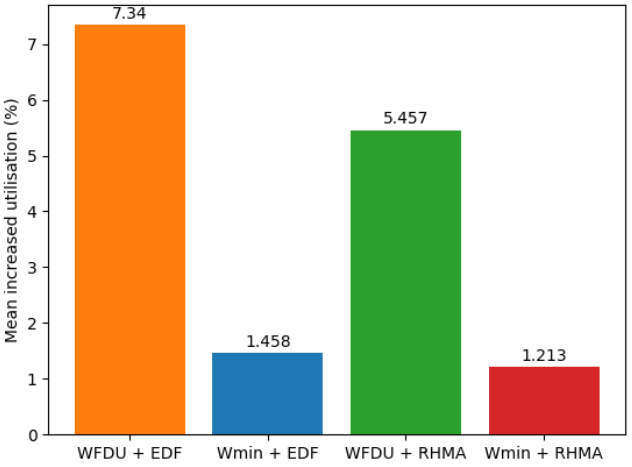

FIGURE 14. Percentage of average increased utilisation of the task sets depending on the allocators and schedulability algorithms.

capable of. Therefore, WFDU+RHMA is the best choice. As a conclusion, regarding schedulability, choosing the right allocator is key rather than the scheduler algorithm.

Figure 12 depicts the increased utilisation with respect to the theoretical utilisation.

As seen in this figure, to allocate tasks to cores using WFDU algorithm means a higher increased utilisation over using Wmin. This is due to the fact that Wmin tries to group tasks with interference together in the same core, in order to reduce the overall produced interference. WFDU balances the load considering only the theoretical utilisation of the tasks and not the interference. Regarding to the proposed scheduling algorithms, RHMA significantly reduces the increased utilisation, specially when tasks have been allocated using WFDU algorithm. That is because, as WFDU produces an allocator with high interference, RHMA has a wider range of action to reduce it. As Wmin already achieves an allocation with less interference, RHMA is less likely to reduce it.

In terms of figures, Figures 13 and 14 depict the average values of schedulability and increased utilisation. From Figure 13 it can be concluded that the combination of allocator and scheduler which offers better schedulability rates is WFDU and RHMA, 58.76%. These results are slightly better than WFDU with EDF, i.e., there are task sets that EDF can not schedule and RHMA can. The same happens with Wmin. In general, Wmin schedulability rates are lower than WFDU (52% vs 58%) but RHMA works better than EDF. Figure 14 shows that Wmin and RHMA is the combination that reduces the increased utilisation the most, only 1.213%. This is logical, as we are applying interference reduction techniques in both the allocator and the scheduler.

As it is said in previous sections, RHMA scheduler uses the CS algorithm to obtain a solution in some busy periods of large size, so it is relevant to make a comparison between RHMA and CS. In terms of increased utilisation, as it can be seen in Figure 15 the plans made by RHMA scheduler have a lower increased utilisation index compared to the plans strictly made by the CS. This happens regardless of which allocator is used for assigning tasks to cores, in our case with Wmin and WFDU allocators. It can be observed as well that, in the same way as it can be seen in previous graphics, with 8 cores the average of increased utilisation caused by interference is higher than the cases of 2 and 4 cores and the scheduling resulted after a Wmin allocation made less interference than the resulted from a WFDU allocation.

Since the role of the CS is to assist RHMA in some busy periods, it is interesting to know in how many BPs the RHMA is not able to find a solution and then, the CS schedule is used. Figure 16 represents the percentage of BPs with respect to the total number of BPs in a hyperperiod in which the scheduling solution is the one provided by the CS. The graphic shows that with 2 cores RHMA does not need to use the CS in any case. This is due to the fact that the size of the problem to be solved with 2 cores is much simpler than with more cores. However, with 8 cores the complexity of the scheduling problem increases a lot for each BP, so the RHMA tends to use the CS help up to more than 25% of the busy periods.

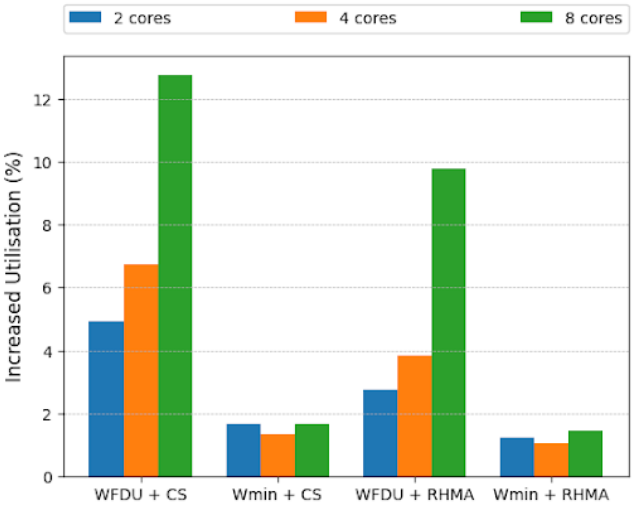

FIGURE 15. Comparison between RHMA and CS: Percentage of average increased utilisation depending on the allocators and schedulability algorithms.

FIGURE 16. Percentage of Busy Periods where RHMA needs the CS.

IX. CONCLUSION

In this paper, a new scheduling strategy is proposed with the aim of reducing interference due to the contention of shared hardware resources in the context of partitioned multicore scheduling. To reduce interference, actions can be taken in both phases of the plan generation, i.e. in the task-to-core allocation and in the subsequent scheduling. Our proposal reduces interference in the scheduling phase.

The first technique, the CS scheduler, is based on independently schedule each busy period in order to apply in each one the best scheduling heuristic in terms of interference reduction. The second one is based on a MILP model. In this case we have proposed the scheduling of all the hyperperiod at once but, due to its disadvantages, we propose another technique that, as the CS does, independently finds the scheduling plan that minimizes the interference. This technique based on rolling horizons (RHMA) is able to obtain solutions in a reasonable amount of time, as it divides the hyperperiod into smaller independent busy periods.

The evaluation done has showed that RHMA reduces the interference produced and it is the best solution when it is combined with an allocator that also takes into account the interference when allocating tasks to cores. If we compare RHMA with EDF, when no actions are taken in the allocation phase (WFDU), the reduction is noticeable (about 25%). But if we use in the allocation phase our previous proposal (Wmin) together with RHMA, then the reduction between EDF and RHMA is not so noticeable. However, Wmin has a lower schedulability compared to WFDU. The conclusion is that the most important factor in reducing interference is the choice of the allocator. If we allocate tasks so tasks with high interference factor do not coincide in the same processor, then the effect of interference is very low and subsequent actions in the scheduling phase do not have much impact. However, in highly critical systems, often the choice of which task is allocated in which core may not respond to interference reduction criteria but to other criteria such as isolation of highly critical tasks in specific cores (in the context of mixed criticality systems). In these cases, when we are not free to allocate tasks to cores as we wish, it is important to have an interference reduction strategy in the scheduling phase, hence the need for RHMA.

As future work, the use of MILP techniques to reduce not only interference but also several parameters at the same time is proposed. We also aim to find a MILP technique that is able to plan the BPs where the CS solution is now needed.

REFERENCES

[1] C. A. S. T. (CAST). (Nov. 2016). *Multi-core Processors—Position Paper CAST-32A*. [Online]. Available: https://www.cast32a.com/files/cast-32a.pdf
[2] D. Dasari, B. Akesson, V. Nelis, M. A. Awan, and S. M. Petters, "Identifying the sources of unpredictability in COTS-based multicore systems," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2013, pp. 39–48.
[3] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surveys*, vol. 43, no. 4, pp. 1–44, Oct. 2011.
[4] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega, and F. J. Cazorla, "Contention in multicore hardware shared resources: Understanding of the state of the art," in *Proc. 14th Int. Workshop Worst-Case Execution Time Anal.*, vol. 39, 2014, pp. 31–42.
[5] T. Lugo, S. Lozano, J. Fernandez, and J. Carretero, "A survey of techniques for reducing interference in real-time applications on multicore platforms," *IEEE Access*, vol. 10, pp. 21853–21882, 2022.
[6] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee, "Response time analysis of COTS-based multicores considering the contention on the shared memory bus," in *Proc. IEEE 10th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Nov. 2011, pp. 1068–1075.
[7] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, and N. Stoimenov, "A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets," *Real-Time Syst.*, vol. 50, nos. 5–6, pp. 736–773, Nov. 2014.
[8] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 145–154.
[9] J. Xiao, S. Altmeyer, and A. Pimentel, "Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 199–208.
[10] M. Hassan and R. Pellizzoni, "Analysis of memory-contention in heterogeneous COTS mpsocs," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, vol. 165, 2020, p. 23.

[11] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *Proc. 23rd Int. Conf. Real Time Netw. Syst.*, Nov. 2015, pp. 129–138.
[12] R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "An extensible framework for multicore response time analysis," *Real-Time Syst.*, vol. 54, no. 3, pp. 607–661, Jul. 2018.
[13] H. Rihani, M. Moy, C. Maiza, R. I. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 67–76.
[14] W.-H. Huang, J.-J. Chen, and J. Reineke, "MIRROR: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources," in *Proc. 53rd Annu. Design Automat. Conf.*, 2016, pp. 1–6.
[15] J. Choi, D. Kang, and S. Ha, "Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 181–186.
[16] B. Andersson, H. Kim, D. D. Niz, M. Klein, R. Rajkumar, and J. Lehoczky, "Schedulability analysis of tasks with corunner-dependent execution times," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 3, pp. 1–29, May 2018.
[17] J. M. Aceituno, A. Guasque, P. Balbastre, J. Simó, and A. Crespo, "Hardware resources contention-aware scheduling of hard real-time multiprocessor systems," *J. Syst. Archit.*, vol. 118, 2021, Art. no. 102223.
[18] R. I. Davis, D. Griffin, and I. Bate, "Schedulability analysis for multicore systems accounting for resource stress and sensitivity," in *Proc. 33rd Euromicro Conf. Real-Time Syst. (ECRTS)*, vol. 196, Jul. 2021, p. 7.
[19] A. Guasque, H. Tohidi, P. Balbastre, J. M. Aceituno, J. Simo, and A. Crespo, "Integer programming techniques for static scheduling of hard real-time systems," *IEEE Access*, vol. 8, pp. 170389–170403, 2020.
[20] C. He, X. Zhu, H. Guo, D. Qiu, and J. Jiang, "Rolling-horizon scheduling for energy constrained distributed real-time embedded systems," *J. Syst. Softw.*, vol. 85, no. 4, pp. 780–794, Apr. 2012.
[21] J. F. Marquant, R. Evins, and J. Carmeliet, "Reducing computation time with a rolling horizon approach applied to a MILP formulation of multiple urban energy hub system," in *Proc. Int. Conf. Comput. Sci. (ICCS)*, vol. 51, 2015, pp. 2137–2146.
[22] S. Baruah, "An ILP representation of a DAG scheduling problem," *Real-Time Syst.*, vol. 58, no. 1, pp. 85–102, Mar. 2022.
[23] A. Guasque, J. M. Aceituno, P. Balbastre, J. Simó, and A. Crespo, "Schedulability analysis of dynamic priority real-time systems with contention," *J. Supercomput.*, vol. 78, no. 12, pp. 14703–14725, Aug. 2022, doi: 10.1007/s11227-022-04446-y.
[24] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proc. 11th Real-Time Syst. Symp.*, 1990, pp. 201–209.
[25] I. Ripoll, A. Crespo, and A. K. Mok, "Improvement in feasibility testing for real-time tasks," *Real-Time Syst.*, vol. 11, no. 1, pp. 19–39, Jul. 1996.
[26] F. Zhang and A. Burns, "A worst-case pattern of task load allocation and execution for multiprocessor global real-time scheduling," *Int. J. Simul.–Syst., Sci. Technol.*, vol. 17, p. 9, Jan. 2016.
[27] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[28] V. Salmani, S. T. Zargar, and M. Naghibzadeh, "A modified maximum urgency first scheduling algorithm for real-time tasks," *World Acad. Sci., Eng. Technol.*, vol. 1, no. 9, pp. 2814–2818, 2007.
[29] D. Stewart, D. Schmitz, and P. Khosla, "Implementing real-time robotic systems using CHIMERA II," in *Proc. IEEE Int. Conf. Syst. Eng.*, vol. 1, Aug. 1990, pp. 598–603.
[30] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Syst.*, vol. 9, no. 3, pp. 207–239, 1995.

JOSÉ MARÍA ACEITUNO was born in Valencia, Spain, in 1982. He received the B.S. degree in computer management from the University of Castellón, in 2012, and the M.S. degree in artificial intelligence from Universitat Politècnica de València (UPV), in 2016, where he is currently pursuing the Ph.D. degree in distributed systems. From 2016 to 2019, he was a Teacher of high-level web applications and multiplatforms with Ilerna Online, Spain.

ANA GUASQUE was born in Valencia, Spain, in 1987. She received the B.S. degree in industrial engineering, the M.S. degree in automation and industrial computing, and the Ph.D. degree in industrial engineering from Universitat Politècnica de València (UPV), in 2013, 2015, and 2019, respectively. She is currently a Researcher with UPV. Her main research interests include real-time operating systems, scheduling, and optimization algorithms and real-time control.

PATRICIA BALBASTRE received the degree in electronic engineering from Universitat Politècnica de València (UPV), in 1998, and the Ph.D. degree in computer science, in 2002. She is an Associate Professor of computer engineering with UPV. Her main research interests include real-time operating systems, dynamic scheduling algorithms, and real-time control, which have resulted in publications in prestigious journals (20) and conferences (57) in the field.

FRANCISCO BLANES received the Ph.D. degree in computing sciences from Universitat Politècnica de València (UPV). From 2012 to 2020, he was the Director of Instituto de Automática e Informática Industrial, UPV, where he is currently teaching distributed and embedded real-time systems. He has more than 90 scientific publications. During the last ten years, he has combined his research and teaching activities with innovation and collaboration with companies. He is responsible for many technology transfer projects in the area of smart manufacturing. His research interests include real-time embedded systems applied to robot control and distributed industrial control systems.

LUIGI POMANTE (Member, IEEE) received the Laurea (B.Sc. and M.Sc.) degree in computer science engineering from Politecnico di Milano, Italy, in 1998, the second-level university master degree in information technology from CEFRIEL (a Center of Excellence of "Politecnico di Milano"), in 1999, and the Ph.D. degree in computer science engineering from Politecnico di Milano, in 2002. Since 2006, he has been an Academic Researcher with the Center of Excellence DEWS, Università degli Studi dell'Aquila, Italy. Since 2010, he has been in charge of scientific and/or technical issues on behalf of DEWS in more than ten funded European and national research projects. His research interests include electronic design automation (in particular, electronic system-level HW/SW co-design) and networked embedded systems (in particular, wireless sensor networks). In such a context, he was the author (or coauthor) of more than 100 articles published in international and national conference proceedings, journals, and book chapters.