electronics

MDPI

Article

Interference-Aware Schedulability Analysis and Task Allocation for Multicore Hard Real-Time Systems

José María Aceituno †, Ana Guasque †, Patricia Balbastre *,†, José Simó † and Alfons Crespo †

Instituto de Automatica e Informatica Industrial (ai2), Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain; aceituno@ai2.upv.es (J.M.A.); anguaor@ai2.upv.es (A.G.); jsimo@ai2.upv.es (J.S.); acrespo@ai2.upv.es (A.C.)

* Correspondence: patricia@ai2.upv.es

† These authors contributed equally to this work.

Abstract: There has been a trend towards using multicore platforms for real-time embedded systems due to their high computing performance. In the scheduling of a multicore hard real-time system, there are interference delays due to contention of shared hardware resources. The main sources of interference are memory, cache memory, and the shared memory bus. These interferences are a great source of unpredictability and they are not always taken into account. Recent papers have proposed task models and schedulability algorithms to account for this interference delay. The aim of this paper is to provide a schedulability analysis for a task model that incorporates interference delay, for both fixed and dynamic priorities. We assume an implicit deadline task model. We rely on a task model where this interference is integrated in a general way, without depending on a specific type of hardware resource. There are similar approaches, but they consider fixed priorities. An allocation algorithm to minimise this interference (Imin) is also proposed and compared with existing allocators. The results show how Imin has the best rates in terms of percentages of schedulability and increased utilisation. In addition, Imin presents good results in terms of solution times.

Keywords: multicore; hard real-time; scheduling; hardware resource contention

Citation: Aceituno, J.M.; Guasque, A.; Balbastre, P.; Simó, J.; Crespo, A. Interference-Aware Schedulability Analysis and Task Allocation for Multicore Hard Real-Time Systems. *Electronics* **2022**, *11*, 1313. https://doi.org/10.3390/electronics11091313

Academic Editor: Christos Volos

Received: 31 March 2022
Accepted: 19 April 2022
Published: 21 April 2022

1. Introduction

The use of embedded systems is nowadays spreading at an increasing speed, to all aspects of modern life as well as all phases of industrial production. The processing capability of multicore systems permits multiple embedded applications on a single shared hardware platform. Nevertheless, multicore systems add many sources of indeterminism, leading to a number of execution delays. These sources of indeterminism mainly involve shared hardware resources, such as buses, caches, and memories. It is necessary to analyse the temporal model in the context of a multicore system and not just when it is running without contenders [1]. Specifically, the interference appears when cores contend for these shared resources. In addition, in highly critical systems, the static plan must take the interference into account. If not, the system's feasibility may be jeopardized.

When this interference is taken into account, the model and the schedulability analysis is often limited to a particular type of hardware resource and even a particular type of memory.

Recently, in [2], a new task model is proposed that takes into account the interference of hardware shared resources in the context of multicore hard real-time systems. This model is general for any kind of hardware resource. However, there is no schedulability test for this new task model. This interference delay can be large and highly variable, posing a major challenge to the schedulability of real-time critical multicore systems.

Contribution: This paper proposes a schedulability test for multicore real-time systems for the model presented by Ref. [2], since in this work they did not present any schedulability analysis.

An allocation algorithm to minimise this interference is also proposed and compared with existing allocators. Our proposal is valid for both fixed and dynamic priorities.

The novelty of the contribution is the consideration of dynamic priority scheduling in a model that considers interference due to shared hardware resources. We use a general model that can be used with different types of shared hardware resources in contrast with other works that assume a very specific kind of resource. Other works assume only fixed priorities or that the interference is only valid for a specific type of shared resource [2].

The paper is organized as follows: Section 2 presents the relevant works in partitioned multicore systems scheduling. In Section 3, the system model used is presented. Section 4 presents the contention aware utilisation factor that is the basis for the schedulability analysis in Section 5. In Section 6 the allocation algorithm that minimises the interference is proposed. The evaluation of the proposal is presented in Section 7, while the conclusions and further work are given in Section 8.

2. Related Works

There is a lot of research about real-time multicore systems scheduling, and one of the most relevant surveys in this area is [3]. In multicore scheduling there are two main branches depending on the criticality required: partitioned and global scheduling. Since our scope of application is hard real-time, from now on we will assume partitioned scheduling. Partitioned multicore scheduling involves two phases: task to core allocation and task scheduling of each core.

The allocation of tasks to cores can be solved with bin packing techniques. We know that this problem is NP-hard in the strong sense [4]. Some of the most popular heuristics, cited in Refs. [5,6] are Worst fit (WF), First fit (FF), and Best fit (BF). Coffman et al. [7] describes many of these algorithms in detail.

It is important to note that the previous allocators do not take into account the unpredictability produced by shared hardware resources. In Ref. [8] a comprehensive analysis of all possible sources of indeterminism is presented. These sources are classified into primary and secondary. They consider primary sources such as caches, memory, FSB, and memory controller and secondary sources such as power saving strategies, hardware-prefetching, system management interrupts, and translation look-aside buffer.

Additionally, in [1] a state-of-the-art about contention delays is presented. This topic is also analysed in [9], where the sources of timing interference in single-core, multi-core, and distributed systems are presented. As stated in this paper, memory interference can render a system infeasible. It is shown in [10] that it is possible that memory interference can cause a worst-case response time increase of almost 300%, even for tasks that spend only 10% of their time fetching memory on an eight-core system.

There are some works that reduce interference delays by using modified task models. In [11], a new model called PREM (predictable execution model) is proposed. This proposal divides one task into two phases: communication and execution. A similar technique is used in [12], which schedules the system with the goal of minimising the makespan by letting the scheduler decide when it is appropriate to avoid interference. For DAG task models, Ref. [13] proposes a scheduling method that applies the LET (Logical Execution Time) paradigm and considers the communication timing between nodes to reduce interference delay due to shared hardware resources.

Other works, such as Ref. [14], proposes a feedback control scheme where critical and non-critical tasks are separated and assigned to different partitions for ensuring the execution of the critical tasks, so a hypervisor manages the multicore hardware system in order to limit memory bus access in non-critical cores measured with Performance Monitor Counters. In Ref. [15], the authors propose an analysis of memory contention as an optimisation problem, which tries to minimize memory interference. They split tasks intro three phases and consider multiple memory transactions issued during each phase. Other approaches, such as the one presented in Ref. [16], reduce contention using synchronisation-based interference models and specific memory allocation schemes.

The survey in Ref. [17] provides an overview on timing verification techniques for multicore real-time systems until 2018. This survey considers single shared resources (memory bus, shared cache, DRAM) and also multiple resources, which is what this work focuses on. The most relevant works come from the Multicore Response Time Analysis framework in Ref. [18], which provides a general approach to timing verification for multicore systems. They omit the notion of worst case execution time (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 cited in the survey, such as Ref. [19,20] consider the amount of time for shared resources accesses and the maximum number of access segments, which is out of the scope of this work.

Regarding mapping and scheduling, the survey presents several works grouped by the techniques that are used: bin-packing algorithms, genetic algorithms, ILP and MILP (Integer/Mixed Integer Linear Programming), etc. ILP and MILP techniques consider scratch-pad memories with different objectives: to minimise the initiation intervals of task graphs, to minimise the WCETs, to minimize the worst case response time (WCRT) of task graphs, etc. The presented techniques are not scalable to large numbers of cores in the system and the authors present heuristics that are scalable.

There are some works that introduce interference due to memory contention as a new parameter in the temporal model. In Ref. [21] WCRA (Worst Case number of shared Resource Accesses) is defined and added to the WCET. In the same way, Ref. [22] proposed the concept of interference-sensitive Worst-Case Execution Time (isWCET). A dynamic approach is presented in Ref. [23] so that, depending on the progress of each kernel, the dependencies of the isWCET schedules are reduced or completely eliminated. The concept of isWCET is similar to our work, but the proposals are centred around minimising the effect of interference with new scheduling methods while our work focuses on allocation algorithms and schedulability conditions. In Ref. [24] memory interference is analysed in a similar way to our work, as it is also represented as a parameter of the temporal model. However, their work is based on the interference produced only by the DRAM memory while our proposal is agnostic with respect to the shared hardware resource used. Their work also considers fixed priority scheduling while our proposal considers dynamic priorities.

The work in Ref. [25] provides 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 considers different types of interference (limited, direct, and indirect) and fixed priority scheduling policies. In contrast to this work, the task-to-cores allocation is known a priori.

In Ref. [2], 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.

In Ref. [26], partitioned scheduling that considers interference while making partition and scheduling decisions is presented. They present a mixed integer linear programming model to obtain the optimal solution but with a high computation cost and they also propose approximation algorithms. They only consider implicit deadline models. This paper differentiates between isolated WCET and the WCET with interference and overhead. They define an Inter-Task interference matrix, in which each element of the matrix is the interference utilisation between two tasks, considering the inflated WCET when two tasks run together. This work is similar to Ref. [2], but in Ref. [2] a general model is considered, valid for any type of shared hardware resource while Ref. [26] only interference due to cache sharing is considered.

Our work continues the research of Ref. [2], so we use the same temporal model to provide a schedulability bound. Therefore, our model does not only provide a schedulability analysis for both fixed and dynamic priorities with a general model for interference, but also a new allocation algorithm that minimizes this bound.

3. Problem Definition and Task Model

The aim of this work is to provide a schedulability condition and an interference-conscious allocator for the task model presented below. This task model is the same as the one presented in Ref. [2].

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

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

where C_i is the WCET, T_i is the period and I_i is the interference. We assume implicit deadlines, so the deadlines are equal to periods. When we refer to M_{τ_i}, we mean the core in which τ_i is allocated.

The hyperperiod of the task set, H, is defined as the least common multiple (lcm) of the periods of all the periodic tasks:

$$H = lcm\{T_i \mid i = 0, \ldots, n-1\} \tag{2}$$

We define A_i as the number of activations that τ_i has throughout H:

$$A_i = H/T_i \tag{3}$$

Since the goal of this paper is to obtain a schedulability test, we will assume a synchronous task system.

To visualize the concept of I_i, we can observe Figure 1, where the computation time (C_0) of task τ_0 is represented. As part of this computation time, the time in which the task performs read and/or write operations in memory has been differentiated (depicted with diagonal lines), as an example of access to a shared hardware resource. It is possible that when the task wants to do one of these r/w operations, the requested resource is busy. Or the other way around, that when the task is accessing this hardware resource, other tasks are blocked from accessing this resource. In any case, this time is the time considered as interference caused to other tasks in other cores I_i. In general, the term I_i is the time the task takes to access shared hardware resources. Although I_i is part of C_i, during the time the task is accessing the shared resource, other tasks on other cores will be delayed. So this interference time is defined independently of C_i, as will be used to represent the delay caused to other tasks. Although the parameter I_i is defined in [2], it is worth describing in detail by means of an example.

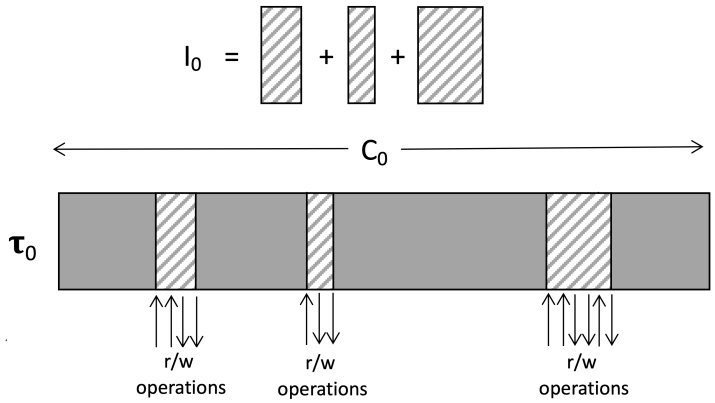

Figure 1. The interference time is included during the time execution.

Example of task set execution with interference: We have a task set composed of three tasks and three cores. Every task is allocated to a different core. τ_0 and τ_2 request

access to the same shared hardware resource, as $I_0, I_2 > 0$. However, τ_1 will not be affected by the interference of the other tasks, as $I_1 = 0$.

$$\tau_0 = (3, 6, 1) \quad \tau_1 = (5, 8, 0) \quad \tau_2 = (6, 12, 1)$$

The scheduling of this system is represented in Figure 2.

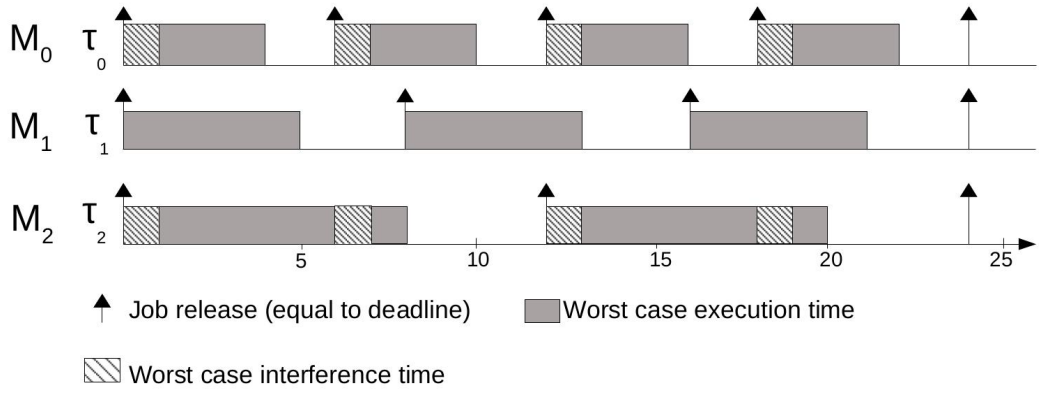

Figure 2. Example of interference.

As we can observe in Figure 2, on one hand, task τ_1 does not receive any interference from the other tasks, so its execution time is not affected by the contention. On the other hand, τ_0 and τ_2 suffer an increase in their execution time. At the beginning, both tasks (τ_0 and τ_2) suffer a delay of 1 unit because of the interference between them. In t = 6, τ_0 is released again and coincides with the first job of τ_2. In that moment, each task provokes a unit of interference in the execution of the other task. We represent the interference as unit times in unified blocks, positioned in the beginning of the execution in order to facilitate the calculation but always preserving the real magnitude of the contention.

Once the model has been defined, we will now present some parameters necessary for the development of the following sections.

From Figure 2, it is deduced that, if contention is considered, the total utilisation of a task does not only depend on its computation time and period, but also on the interference received from other tasks.

We consider that the interference parameter refers to a single type of hardware resource. If there are several types of hardware resources, the model would have to extend to as many interference parameters as there are hardware resources. Considering parameter I for all interference types would result in an even more pessimistic model.

It should be taken into account that a task τ_j will cause interference to other task τ_i when the following conditions are met:

- τ_i and τ_j are not allocated to the same core ($M_{\tau_i} \neq M_{\tau_j}$);
- τ_i and τ_j are active at the same time;
- τ_i and τ_j have at least 1 unit of interference ($I_i > 0$, $I_j > 0$)

Therefore, the real utilisation (we refer to real utilisation to the utilisation that includes the interference delay) of a task, U_i', is:

$$U_i' = U_i + U_i^{int} \tag{4}$$

where $U_i = C_i/T_i$ and U_i^{int} are the utilisation due to the interference caused by other tasks to τ_i:

$$U_i^{int} = \frac{I_i^T}{H} \tag{5}$$

where I_i^T the total interference that a task τ_i receives from other tasks in a period T_i.

Therefore:

$$U_i' = \frac{C_i}{T_i} + \frac{I_i^T}{H} \tag{6}$$

Hence, the utilisation of a core would be the sum of all the task utilisations:

$$U'_{M_k} = \sum_{\tau_i \in M_k} U_i' \tag{7}$$

And the utilisation of all the system would be the sum of all the core utilisations:

$$U'_\tau = \sum_{\forall k} U'_{M_k} \tag{8}$$

4. Contention Aware Utilisation Factor

In this section, we are going to provide an upper bound to U_i^{int}, so we will be able to provide a schedulability test. This bound will be called U_i^{iub}. First, we need two definitions, which were also introduced in Ref. [2]:

Definition 1 ([2]). *A task is defined as a receiving task when it accesses shared hardware resources and it suffers an increment of its computation time due to the interference produced by other tasks allocated to other cores.*

Definition 2 ([2]). *A task is defined as a broadcasting task when it accesses shared hardware resources and it provokes an increment of 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 $I_j > 0$.

To estimate U_i^{iub}, we will calculate the maximum total interference that will depend on the maximum number of activations of a broadcasting task that fall within a receiving task.

Definition 3. *Let $I_{j\to i}^{Tub}$ (in what follows, the expression $j \to i$ means that τ_j is a broadcasting task and τ_i is a receiving task) be the maximum total interference that τ_j can cause to τ_i in a period of τ_i, T_i.*

Definition 4. *Let $A_{j\to i}$ be the maximum number of activations of the broadcasting task τ_j that fall within an activation of the receiving task, τ_i.*

From the two previous definitions, it is clear that:

$$I_{j\to i}^{Tub} = A_i \cdot A_{j\to i} \cdot I_j \tag{9}$$

Note that $I_{j\to i}^{Tub}$ is an upper bound because the maximum number of interferences does not have to occur in all activations. Besides, $I_{j\to i}^{Tub}$ only takes into account the interference caused by τ_j. To calculate U_i^{iub}, we need to consider all tasks allocated to other cores. Therefore:

$$U_i^{iub} = \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}}{T_i} \tag{10}$$

From Equation (10), it is easy to see that we have reduced the study of U_i^{iub} to the study of $I_{j\to i}^{Tub}$.

Worst Case Estimation of $I_{j\to i}^{Tub}$

To calculate $I_{j\to i}^{Tub}$, we need to know the exact value of $A_{j\to i}$. To do this, we will calculate $A_{j\to i}$ assuming the following conditions:

- τ_i and τ_j are allocated to different cores;
- $I_i, I_j > 0$;
- $T_j \geq T_i$ (the period of the broadcasting task is greater than the period of the receiving task).

We will have further considerations of the case in which $T_j < T_i$.

Let us study the total interference received by τ_i. First, the maximum number of activations of τ_j that fall within an activation of τ_i is calculated:

$$A_{j\to i} = \left\lceil \frac{T_i - 1}{T_j} \right\rceil + K \tag{11}$$

being

$$K = \begin{cases} 0 & \text{If periods } T_i \text{ and } T_j \text{ are harmonics} \\ 1 & \text{Elsewhere} \end{cases}$$

Equation (11) expresses the relationship between the periods of broadcasting and receiving task modified by factor K and minus 1 to reflect the number of activations of the broadcasting task that fall within a period of the receiving task. With the division of periods, we obtain the whole times that a task is included in the other and we apply the ceiling function in order to obtain the greatest integer number of times. Moreover, the worst scenario happens if one of the tasks is shifted one unit of time. That is why the minus one is added to the formula.

Then, two cases are possible:

- Harmonic periods, with zero or small residues in the division of periods. Then, $K = 0$;
- Non-harmonic periods. Then, $K = 1$.

The above definition of $I_{j\to i}^{Tub}$ is very pessimistic in the sense that it considers that the maximum interference occurs equally in all activations. This is not always true, as it depends on how the activations of the two tasks coincide.

Once the definition of $I_{j\to i}^{Tub}$ is provided, let us consider the case in which $T_i > T_j$ (the period of the broadcasting task is shorter than the period of the receiving task) with the following example: let us suppose two tasks τ_i and τ_j allocated to a dual-core platform, with $I_i = 2I_j$. Figures 3 and 4 represent the maximum interference that may be produced when tasks τ_i and τ_j are executed simultaneously. Note that, for the sake of simplicity, we have not depicted the tasks computation times, but only the interference.

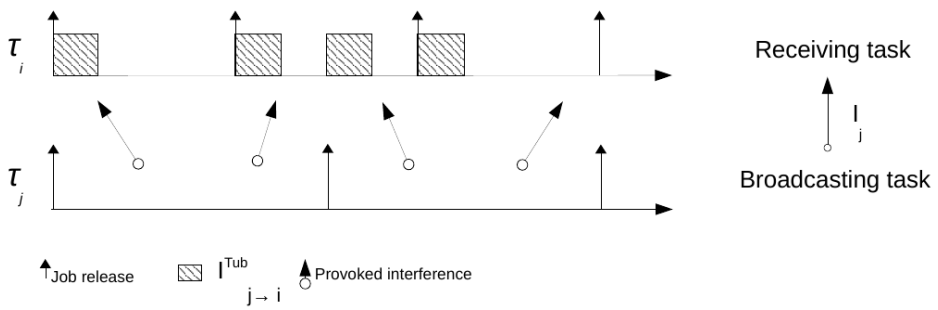

Figure 3. Example of interference from τ_j to τ_i.

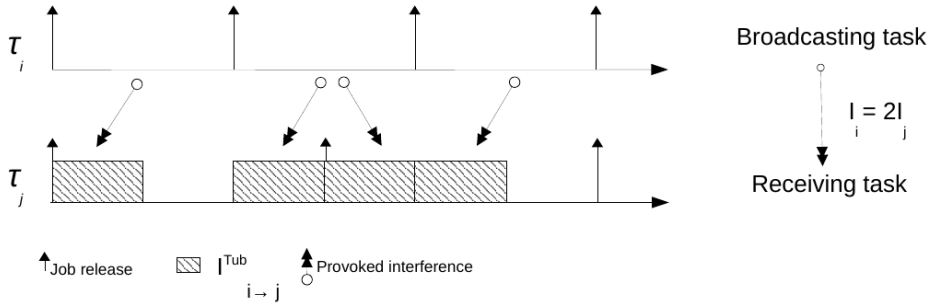

Figure 4. Example of interference from τ_i to τ_j.

Figure 3 considers that τ_j is the broadcasting task and τ_i the receiving task. In this case, the period of the broadcasting task τ_j is greater than the period of the receiving task τ_i, and the total received interference is equal to $I_{j\to i}^{Tub} = 4$. This scenario has been studied previously.

Let us consider now that τ_i is the broadcasting task and τ_j is the receiving task, as depicted in Figure 4. In this case, the period of the broadcasting task is shorter than the period of the receiving task. Could we apply Equation (11)? We are going to prove it numerically. Suppose that $T_i = 4$ and $T_j = 6$. Applying Equation (11), the maximum number of activations of the broadcasting task that fall within an activation of the receiving task is $A_{i\to j} = \left\lceil \frac{T_j - 1}{T_i} \right\rceil + 1 = 3$. As seen in Figure 4, τ_i never falls three times within an activation of τ_j. So Equation (11) can not be applied when the period of the broadcasting task τ_j is shorter than the period of the receiving task.

Then, we are proposing a methodology to relate the total interference between a broadcasting and a receiving task, regardless of the lengths of their periods.

Theorem 1. *The ratio of interference received and broadcast by τ_j to τ_i is:*

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub} \tag{12}$$

Proof. At time 0, as both tasks are released, interference is always introduced. Moreover, every time any task is released, I_j units of interference may be introduced. In particular, it happens as many times as activations the tasks have in a hyperperiod minus one, because the first activation has already been considered. Then, the total number of times is:

$$1 + \left(\frac{H}{T_i} - 1\right) + \left(\frac{H}{T_j} - 1\right) = \frac{H}{T_i} + \frac{H}{T_j} - 1$$

Then, the maximum number of interferences that a task τ_j provokes in τ_i is:

$$I_{j\to i}^{T} = \left(\frac{H}{T_i} + \frac{H}{T_j} - 1\right) I_j$$

Then, let us have a look at the total interference received by task j, $I_{i\to j}^{T}$. If we repeat the previous process, the maximum number of interferences that the task τ_i provokes in τ_j is:

$$I_{i\to j}^{T} = \left(\frac{H}{T_j} + \frac{H}{T_i} - 1\right) I_i$$

It is easy to deduce that the only difference between both situations is the coefficient of interference of the task that provokes the interference.

Therefore, we can conclude that the total interference that task j provokes to task i is related with the total interference that task i provokes to task j as follows:

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub}$$

□

Then, the total number of interferences between two tasks may be calculated interchangeably, from the point of view of both tasks. Thus, from now on and for calculus purposes, this work considers that, when two tasks interfere, the task with the biggest period provokes the interference on the task with the shortest period. If it is not the case, we will apply Equation (12).

To conclude with the example in Figures 3 and 4, as we deduced that $I_{j\to i}^{Tub} = 4$, from Equation (12):

$$I_{i\to j}^{Tub} = \frac{I_i}{I_j} I_{j\to i}^{Tub} = \frac{2I_j}{I_j} 4 = 8$$

and this result coincides with that depicted in Figure 4.

5. Schedulability Analysis

Once an upper value is given for $I_{j\to i}^{Tub}$, we can estimate an upper bound of the utilisation of a receiving task τ_i, U_i^{ub}, taking into account interferences due to contention.

$$U_i^{ub} = U_i + U_i^{iub} = U_i + \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}}{T_i}$$

Note that, if τ_i is not a receiving task, then $U_i^{iub} = 0$.

This upper bound is always greater or equal to the real utilisation of the receiving task i, U_i'. This results from the fact that:

$$I_i^T \leq \sum_{\tau_j \notin M_{\tau_i}} I_{j\to i}^{Tub}$$

Therefore, the upper bound of the utilisation of each core is defined as the sum of the upper bounds of the utilisations of the tasks that belong to that core, $U_{M_k}^{ub} = \sum_{\tau_i \in M_k} U_i^{ub}$. Similarly, the upper bound of the system utilisation is defined as the sum of the upper bound of the utilisations of all cores (or all tasks), $U_\tau^{ub} = \sum_{M_k} U_{M_k}^{ub}$. Then, the system will be schedulable if:

1. The upper bound of the utilisation of each task is less or equal to $n(2^{\frac{1}{n}} - 1)$ for fixed priorities and to one for dynamic priorities, Ref. [27]:

$$U_i^{ub} \leq \left\{ \begin{array}{ll} n(2^{\frac{1}{n}} - 1), & \text{for fixed priorities} \\ 1 & \text{for dynamic priorities} \end{array} \right\} \forall i = 0, \ldots, n-1$$

2. The upper bound of the utilisation of each core M_k is less or equal to one:

$$U_{M_k}^{ub} \leq 1 \quad \forall k = 0, \ldots, m-1$$

As a consequence of (i) and (ii), the upper bound of the system utilisation is less or equal to the number of cores [28], m:

$$U_\tau^{ub} \leq m$$

As utilisations U_i^{ub} are overestimated, we can conclude that if previous conditions are accomplished, the system will be schedulable.

Note that this is a necessary but not a sufficient condition.

Example

Let us show the procedure to be followed with an example: let us consider the following task set, $\tau = [\tau_0, \tau_1, \tau_2]$ with $\tau_0 = (2, 3, 0)$, $\tau_1 = (4, 8, 2)$, and $\tau_2 = (5, 12, 1)$, allocated to a system with three cores. τ_0 is allocated to core M_0, τ_1, in M_1, and τ_2, in M_2. Once the tasks are allocated to cores, the Earliest Deadline First (EDF) algorithm [27] schedules tasks in each core. The actual execution of the task set is shown in Figure 5.

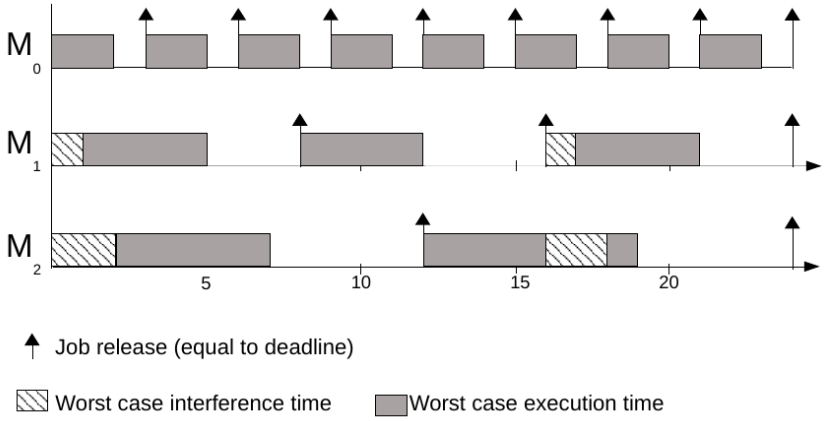

Figure 5. $\tau = [\tau_0, \tau_1, \tau_2]$ with $\tau_0 = (2, 3, 0)$, $\tau_1 = (4, 8, 2)$, and $\tau_2 = (5, 12, 1)$ allocated to a system with three cores.

We are applying the method described in Section 4 to calculate the upper bound of the system utilisation. Then we will compare it with the actual system utilisation, shown in Figure 5.

- Step 1: Define τ_j and τ_i. The task with the greatest period is the broadcasting task τ_j, and the receiving task is the task with the shortest period, τ_i. In this example, $\tau_j = \tau_2$ and $\tau_i = \tau_1$. As $I_0 = 0$, τ_0 is neither receiving nor broadcasting.
- Step 2: Calculate the maximum number of times that task τ_2 falls within an activation of τ_1, $A_{2\to1}$. As depicted in Figure 6, τ_2 falls twice within the second activation of τ_1. It may be calculated applying Equation (11):

$$A_{2\to1} = \left\lceil \frac{T_1 - 1}{T_2} \right\rceil + 1 = \left\lceil \frac{8-1}{12} \right\rceil + 1 = 1 + 1 = 2$$

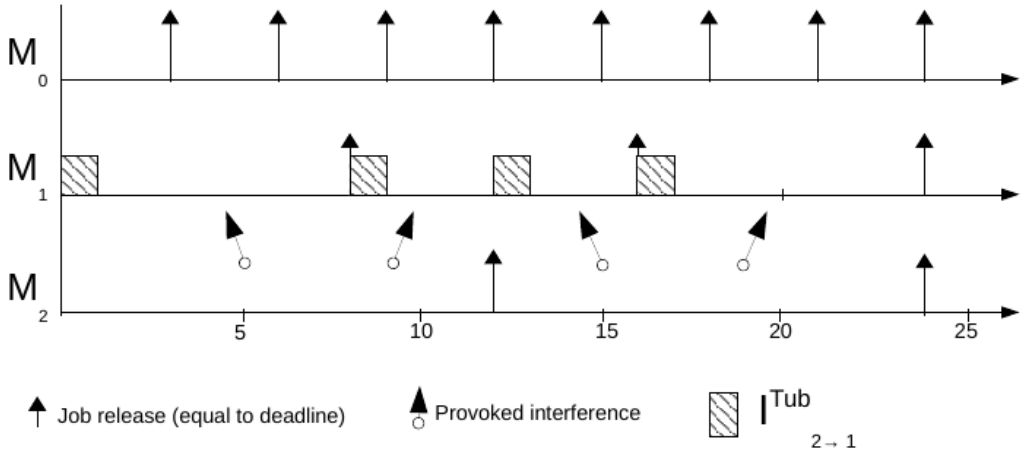

Figure 6. Upper bound of the interference received by τ_1, $I^{Tub}_{2\to1}$.

- Step 3: To calculate the maximum total interference that τ_2 can cause to τ_1. First, we calculate the number of activations of task τ_1 in a hyperperiod, $A_1 = H/T_1 = 3$. Following Equation (4):

$$I_{2\to1}^{Tub} = A_1 \cdot A_{2\to1} \cdot I_2 = 3 \cdot 2 \cdot 1 = 6$$

  The maximum total interference is only received in the second activation of τ_1, as shown in Figure 6, but we consider the worst case, in which all activations receive the maximum of interference from τ_2.
- Step 4: Calculate the total interference received by the other receiving tasks, τ_2, as shown in Figure 7. As the period of τ_2 is greater than the period of τ_1, we apply the Equation (12):

$$I_{1\to2}^{Tub} = \frac{I_1}{I_2} I_{2\to1}^{Tub} = \frac{2}{1} \cdot 6 = 12 \tag{13}$$

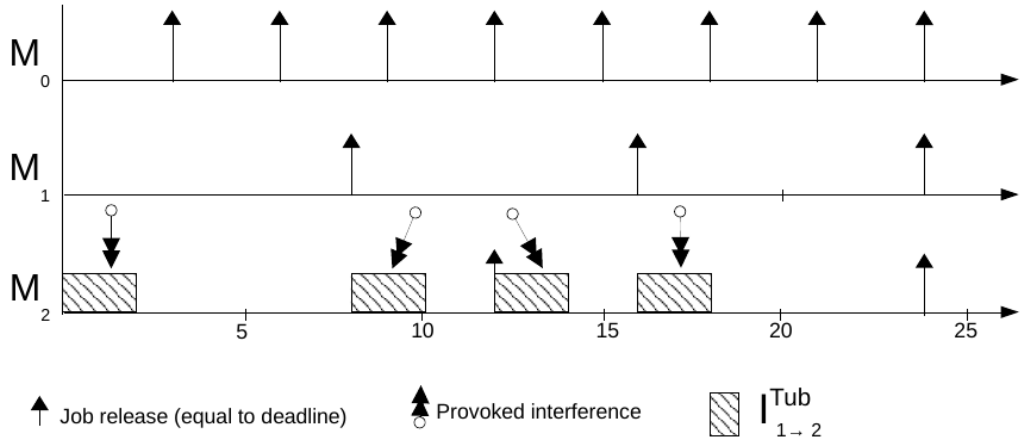

Figure 7. Upper bound of the interference received by τ_2, $I_{1\to2}^{Tub}$.

- Step 5: Calculate the upper bound of the utilisation of each task.

$$U_0^{ub} = \frac{C_0}{T_0} = \frac{2}{3} = 0.667$$
$$U_1^{ub} = \frac{C_1}{T_1} + \frac{I_{2\to1}^{T}}{H} = \frac{4}{8} + \frac{6}{24} = 0.75$$
$$U_2^{ub} = \frac{C_2}{T_2} + \frac{I_{1\to2}^{T}}{H} = \frac{5}{12} + \frac{12}{24} = 0.91667$$

  From Figure 5, the actual utilisations of the tasks are:

$$U_0^{'} = \frac{16}{24} = 0.667 \leq U_0^{ub}$$
$$U_1^{'} = \frac{14}{24} = 0.5833 \leq U_1^{ub}$$
$$U_2^{'} = \frac{14}{24} = 0.5833 \leq U_2^{ub}$$

It is deduced that the actual utilisations of the tasks are equal or less than the upper bounds estimated in this paper. Therefore, the system is schedulable.

6. Task Allocation Algorithms

In the following, we first briefly discuss several existing allocation techniques. We then propose a new mapping strategy. Allocation is a problem that appears with multicore systems. It comes to answer the question: which processor will execute each task?

The main disadvantage of the partitioning approach to multicore scheduling is that the task allocation problem is analogous to the bin packing problem and is known to be NP-Hard [29].

6.1. Overview of Existing Heuristic Bin-Packing Algorithms

Refs. [5,6] detailed some of the most well-known bin-packing heuristics:

- First Fit (FF). Each item is always packed into the first bin where it fits. A new bin is open if any item does not fit in the open bin;
- Worst Fit (WF). Each item is placed into the open bin with the largest amount of room remaining. If it does not fit any bins, a new bin is opened.

One problem with the previous mentioned heuristics is that, if the items are not ordered properly, large items could not be packed efficiently. Therefore, items should be ordered effectively to avoid these problems. One way is to pack the items in order of decreasing weights or utilizations. This way, WF becomes WFDU (Worst Fit Decreasing utilisation), and FF becomes FFDU (First Fit Decreasing utilisation).

6.2. Overview of Aceituno's Method

In Ref. [2], a task model that takes into account interference delays due to contention of shared hardware resources is proposed. Moreover, three tasks-to-cores allocation methods are also presented.

In that paper, a discrepancy-based method is presented, which is defined as the difference between the maximum and minimum utilisation of a multicore system, UD_τ. One of the algorithms is focused on reducing the discrepancy, UDmin, and the other on maximizing it, UDmax. On the one hand, UDmin behaves as the heuristic WFDU, as both balance the load among cores. They present excellent schedulability ratios (98.1% and 100%, respectively) but reach high rates of increment of utilisation due to the system interference (2.3%). On the other hand, UDmax behaves as FFDU, as it unbalances the load among cores. In contrast to UDmin/WFDU, it reaches a low rate of increment of utilisation but their rates of schedulability are very low (around 43%). The Wmin allocator accounts for the possible interference produced for each task and it provides a low increment of utilisation (0.266%) and high schedulability ratio (up to 89%), with respect to previous algorithms.

6.3. Proposed Allocator Considering the Interference: Imin

Previous bin-packing and discrepancy-based algorithms do not take into account the interference produced when two or more tasks allocated to different cores coincide in execution. However, Wmin does take interference into account, although not exactly, but based on a binary matrix (W) of possible interference between tasks on other cores. This allocator tries to minimise the number of 1 s in the matrix, which represent a possible interference.

In this subsection, a new allocation algorithm to minimise the interference is proposed—Imin. As it has been analysed in previous sections, U_i^{ub} is the upper bound of the utilisation taking into account the maximum possible interference. The allocator Imin obtains the task allocation to cores that minimises this upper bound as much as possible. This allocator is based on integer linear programming to obtain an optimal solution. For that, we define a set of parameters and variables shown in Table 1.

Table 1. Model notation of the implicit deadline task model.

SETS AND INDICES	
i, j	Task index $i, j \in \{0, 1, 2, \ldots, n-1\}$
k	Core index $k \in \{0, 1, 2, \ldots, m-1\}$
PARAMETERS	
C_i	Worst case execution time of τ_i
T_i	Period of τ_i
U_i	Utilisation of τ_i
H	Hyperperiod of the task set τ
$I_{j \to i}^{Tub}$	Maximum interference that τ_j can cause to τ_i during $(0,T_i)$
DECISION VARIABLES	
U_i^{iub}	Upper bound of the utilisation due to interference of τ_i
U_i^{ub}	Upper bound of the total utilisation of τ_i
U_{M_k}	Utilisation of M_k
O_{ik}	Allocation matrix. 1 if τ_i is allocated to core k and 0 otherwise.

As has been previously stated, the objective is:

$$\text{Minimise} \sum_{\forall i} U_i^{ub} \tag{14}$$

s.t:

$$\sum_{\forall k} O_{ik} = 1 \qquad \forall i \tag{15}$$

$$U_i^{iub} = \frac{\sum_{\tau_j \notin M_{\tau_i}} I_{j \to i}^{Tub}}{T_i} \qquad \forall i \tag{16}$$

$$U_i^{ub} = U_i + U_i^{iub} \qquad \forall i \tag{17}$$

$$U_{M_k} = \sum_{\forall i} O_{ik} \cdot U_i \qquad \forall k \tag{18}$$

$$U_{M_k} \leq 1 \qquad \forall k \tag{19}$$

$$O_{ik} \in \{0, 1\} \tag{20}$$

$$U_i^{iub}, U_i^{ub}, U_{M_k} \geq 0 \tag{21}$$

Constraint (15) assures that a task is allocated to one and only one core. Equation (16) sets the value of the extra utilisation of a task provoked by the interference in its upper bound. Equation (17) sets the value of the total utilisation taking account the interference in its upper bound, which means in its maximum value, as it has been explained and established in this paper. The total utilisation per core is calculated as the sum of the utilisations of the tasks that belong to that core (Equation (18)) and its value should be less or equal to 1 (Equation (19)). Equations (20) and (21) represent the decision variable domains.

7. Evaluation

7.1. Experimental Conditions

To validate our proposed technique, we implemented a synthetic task generator to generate up to 5400 random feasible task sets with different configurations. The task generator works by calculating the following parameters:

- U_τ: utilisation of the task set calculated using the UUniFast discard algorithm [30];
- T_i: task periods, which are generated randomly in [20, 1000];
- C_i: computation times are deduced from the periods and the utilisations, as $C_i = U_i \cdot T_i$.

The experimental parameters of the evaluation are specified in Table 2. To ensure the reproducibility of the results, these parameters coincide with those used in [2]. The

only difference is that a wide range of theoretical utilisations and interference factors are also studied.

The theoretical utilisation of the maximum load of all cores varies between 50% and 75% of the maximum load of the system. For example, the maximum load of a system with 8 cores is 8, so for evaluation purposes, the initial utilisation is set to 4 (50%) and 6 (75%).

As is well shown in Table 2, the number of broadcasting tasks is 25% of the total number of tasks, except for scenarios 1–6 (2 cores), which is 50%. This is due to the fact that, if only one task is broadcasting, no interference will be produced. Each combination of the number of cores and utilisation is evaluated in three cases, with 10%, 20%, and 30% of interference over the WCET. It means that, for example, if the percentage of interference in a task set is established to 20% and a broadcasting task of this set has 10 units of WCET, then the interference of this broadcasting task should be 2 units.

Table 2. Experimental parameters selected for the evaluation process.

	Experimental Parameters																	
Number of cores	2						4						8					
Number of tasks	4						12						20					
Number of broadcasting tasks	2						3						5					
Theoretical utilisation	1.1			1.5			2.1			3			4			6		
Number of sets	900			900			900			900			900			900		
Interference (%) over WCET	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30
Scenario	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

With this setup, we apply the proposed Imin allocator and the already existing allocation methods (FFDU, WFDU, and Wmin) to the synthetic task sets. Once the allocation phase is complete, it must be checked if all tasks have been allocated to cores assuring that the maximum capacity per core is not exceeded, i.e., $U_{M_k} \leq 1 \; \forall k = 0, \ldots, m-1$. If any of the allocators cannot allocate the task set, this task set will be discarded and a new one is generated.

To perform the allocators Imin and Wmin, we use Gurobi optimizer 9.0 [31], from Gurobi Optimization, Inc. (Houston, TX, USA), which is the fastest and most powerful mathematical programming solver available for ILP, MILP and other problems. It provides a Python interface.

The evaluation process is executed on an Intel Core i7 CPU with 16 GB of RAM (Santa Clara, CA, USA).

Once all allocations are validated, the task sets are scheduled following the contention aware scheduling algorithm proposed in [2] which takes into account the interference. The selected priority-based algorithm for the mentioned scheduling algorithm is EDF [27] but any fixed-priority algorithm could also be used. As an output of this phase, the real utilisation of the system, U'_τ, is obtained.

After the scheduling phase, the scheduling plans must be validated in order to ensure that the temporal constraints are met throughout the hyperperiod. In addition, the following performance parameters are measured to compare different methods:

- Schedulability ratio. It is calculated as the relation between the number task sets with feasible scheduling plans and the number task sets with feasible allocations, expressed as a percentage;
- Increased utilisation. It is calculated as the relation between the increase in utilisation with respect to the theoretical utilisation. This is calculated as $1 - \sum_k \frac{U_{M_k}}{U'_{M_k}} = 1 - \frac{U_\tau}{U'_\tau}$;
- Relation between the upper bound of utilisation, U^{ub}_τ, and the actual utilisation, U'_τ.

7.2. Experimental Results

Depending on the experimental parameters specified in Table 2, different scenarios are defined (numbered from 1 to 18).

Figures 8 and 9 depict the comparison of schedulability ratio and the increased utilisation for each scenario and each allocator. In Figure 8 it is shown that, as was expected, that generally, as the number of cores and tasks increases, the schedulability ratio decreases. Regarding the percentage of interference, it is shown that, as a general rule, with the same number of cores and tasks, the higher percentage of interference, the lower is the schedulability for all the allocators, even though there are some exceptions to this rule, such as FFDU in scenarios 7, 8, and 9 or scenarios 10, 11, and 12. After an analysis of the data with the exception cases, we realized that these exceptions are just provoked by the randomness of the tasks generated as input data. In general terms, the allocators FFDU and WFDU are affected by the increase of interference percentage more than Wmin and Imin. This property can be clearly shown, for example, in scenarios 7, 8, and 9 of Figure 8, where the percentage of interference is 10, 20, and 30%, respectively. In these cases, FFDU and WFDU have very different results depending on the amount of interference but Wmin and Imin maintain almost the same values in spite of the variations in the amount of interference.

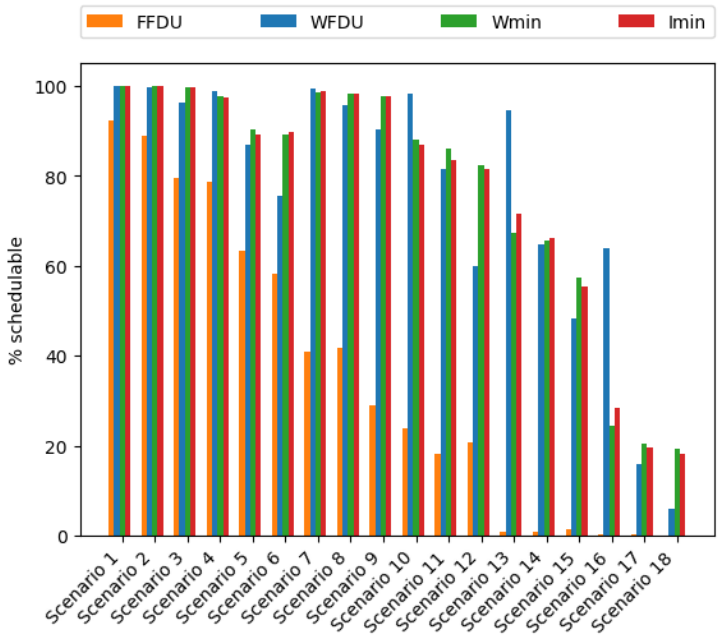

Figure 8. Percentage of schedulability task sets for each allocator depending on the scenario.

Figure 9 shows the increased utilisation for each scenario. The results vary a lot depending on the scenario and the allocator. In the case of FFDU, the increased utilisation with 2 cores is clearly higher than 4 and 8 cores. In the case of WFDU the increased utilisation is always the highest with respect to other allocators. In the case of Wmin and Imin the increased utilisation is significantly lower with respect to WFDU.

In Figures 8 and 9, average values of all scenarios of schedulability and increased utilisation are represented for each allocator. In Figure 10 it is shown that Imin has the highest rate of schedulability from all the allocators with a 76.83%. We should be aware that Wmin also has a high index of schedulability, almost the same as Imin, 76.8%. The FFDU allocator has the worst percentage of schedulability with 35.56% and WFDU has an acceptable rate of schedulability with 76.48%.

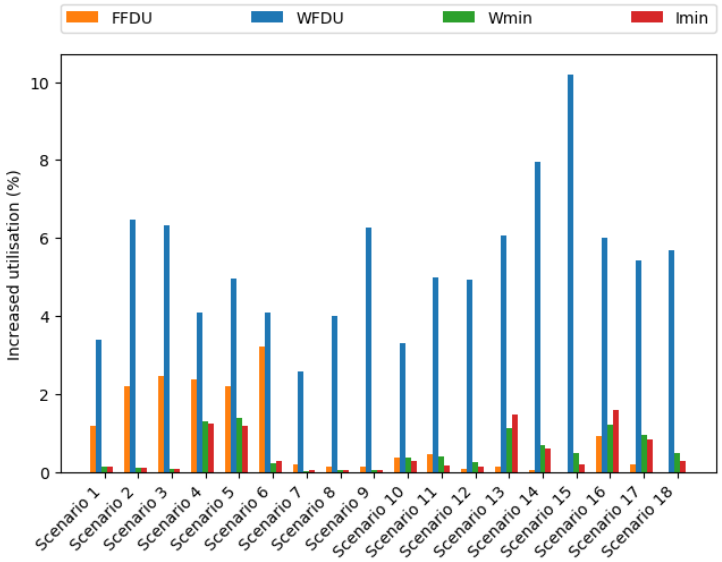

Figure 9. Increased utilisation resulted after the scheduling for each allocator depending on the scenario.

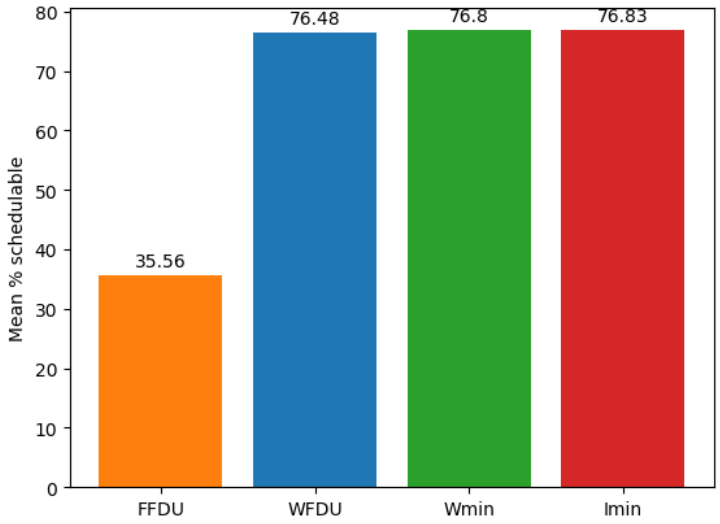

Figure 10. Average percentage of schedulable task sets depending on the allocators.

It is important to note that the results in [2] showed that WFDU could always ensure schedulability for all the task sets (100%). In our case, this behaviour has not been reproduced. This difference lies in the interference coefficient selected in the evaluation process. In Ref. [2], $I_i = 1 \ \forall \tau_i \in \tau$. In this work, $I_i \geq 0.1 \cdot C_i \ \forall \tau_i \in \tau$, which is always greater or equal than 1. Adding small interferences does not affect the schedulability of allocators that balance the load, as WFDU does. In fact, when more interference is considered, these results vary. With bigger interferences, the percentage of schedulability is reduced.

In Figure 11 it is shown that the WFDU algorithm has by far the highest increase in utilisation while Wmin and Imin have the best results. For WFDU and FFDU, the schedulability ratio and increased utilisation are directly proportional, which is not the desired behaviour. This is not the case of Wmin and Imin, in which a high schedulability ratio does not suppose an increment in the interference and then, in the utilisation of the system.

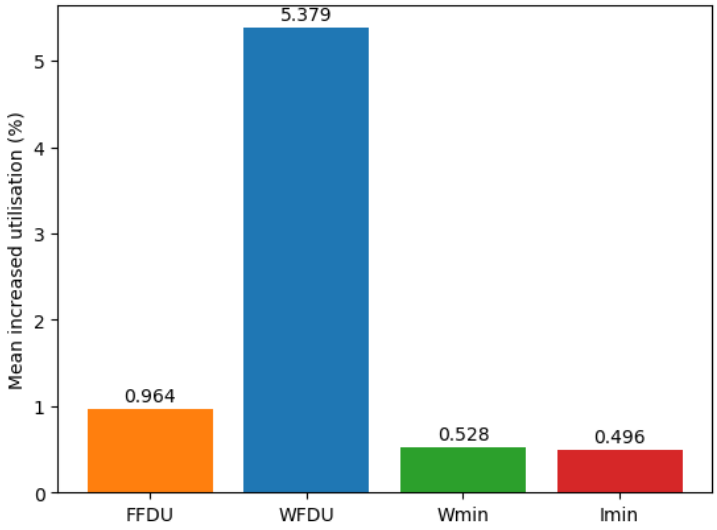

Figure 11. Average of increased utilisation depending on the allocators.

The comparison between the upper bound (U_τ^{ub}) and the real utilisation (U_τ') for the best allocators is depicted in Figure 12 (Imin), and Figure 13 (Wmin). They are expressed in terms of utilisation system percentage. As was expected, in both graphics, the real utilisation is always equal or less than its upper bound utilisation. Also, in both graphics, the more cores and tasks in the scenario, the more percentage of system utilisation is estimated.

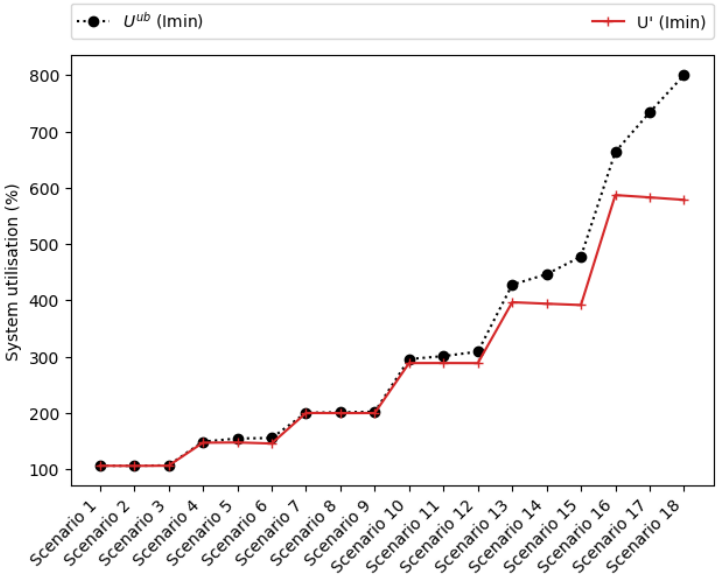

Figure 12. Imin algorithm: theoretical utilisation upper bound values compared to the real utilisation values measured after the scheduling.

In Figure 14, the comparison between the utilisation upper bound of Wmin and Imin is depicted. We can see that the results are very similar as in previous figures, but Imin achieves a slight lower value for U_i^{ub} because Imin minimises the upper bound while Wmin minimises W matrix. However, the results are very similar since both represent the possible interference between tasks in other cores.

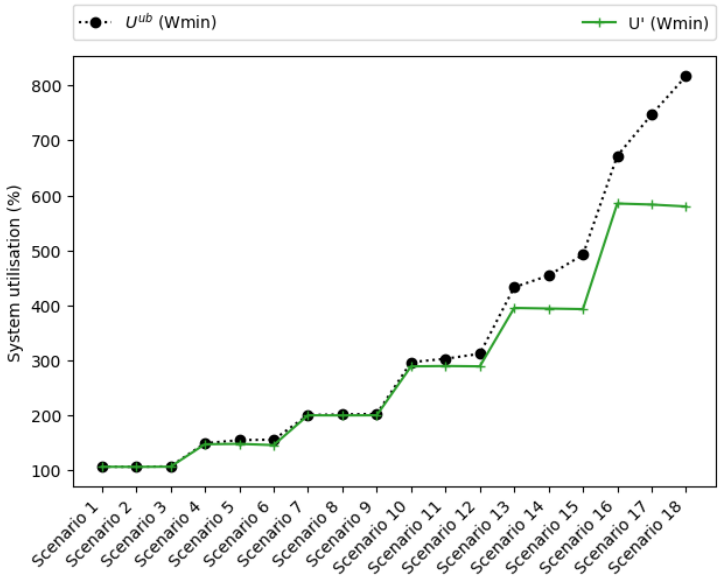

Figure 13. Wmin algorithm: theoretical utilisation upper bound values compared to the real utilisation values measured after the scheduling.

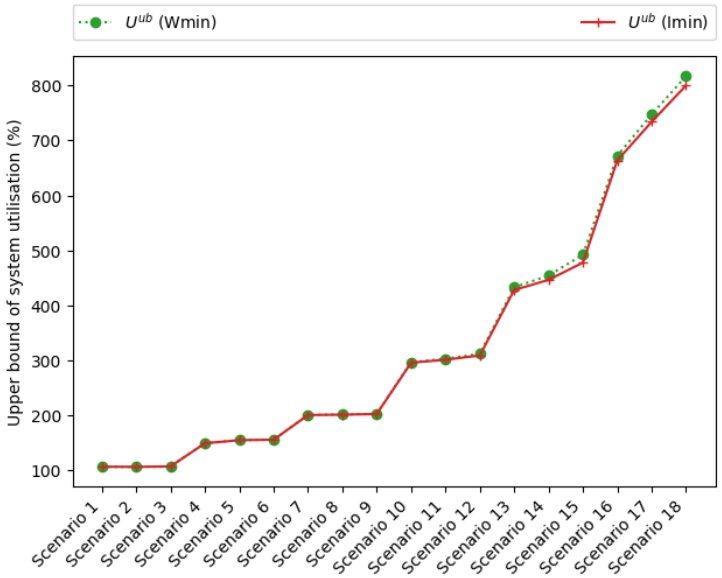

Figure 14. Comparison of Wmin and Imin utilisation upper bounds values.

Finally, the solution times for the MILP approaches are evaluated in Figure 15. It shows that, generally, as the number of cores increases, the solution time increases. This is more obvious in the cases of 8 cores scenarios (scenarios 13 to 18). This is because when the number of cores increases (and consequently, the number of tasks and broadcasting tasks), the complexity of the search of the optimal solution also increases. Moreover, it is observed that, specially in the 8 cores scenarios, the solution time for Imin is clearly lower than the solution time needed for Wmin. This is a clear advantage of Imin allocator.

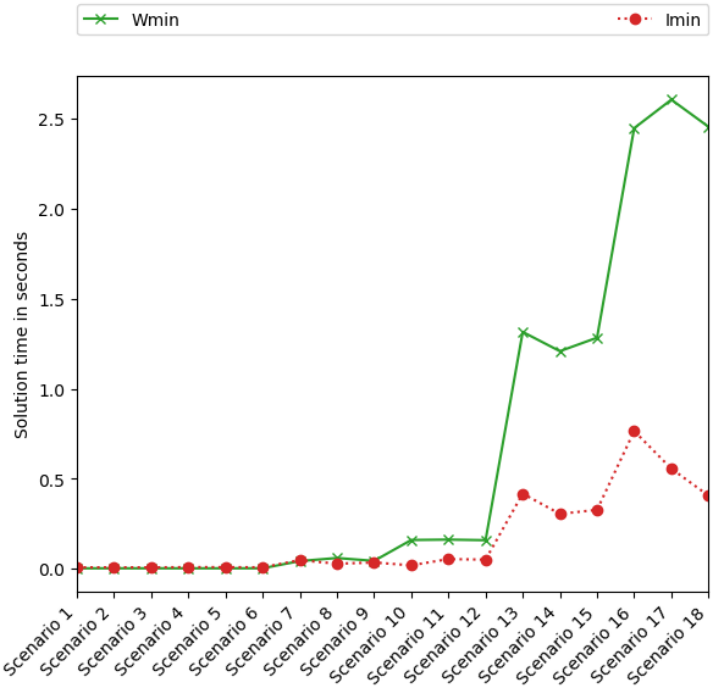

Figure 15. Solution times for MILP approaches depending on the experimental scenario.

8. Conclusions

This paper has proposed a schedulability analysis for task models that consider the delay produced by the contention of hardware shared resources in a hard real-time multicore system. Then, an allocation algorithm that minimises this contention, Imin, has been proposed and evaluated by comparison with other existing allocators.

After the results of the experimental evaluation, we can conclude that the allocation algorithm proposed in this paper, Imin, has the best rates in terms of percentages of schedulability and increased utilisation. In addition, Imin presents good results in terms of solution times. So, it is reasonable to affirm that Imin may be an eligible option as an allocator for the implementation of a multicore system, especially in those systems where the priority is to maximize the scheduling rate or to minimise the contention produced in hardware shared resources.

We plan to further investigate how to improve Imin in order to achieve a lower utilisation rate without decreasing the schedulability. We also plan to find a less pessimistic upper bound for the utilisation and extend the model to the constrained deadlines.

Author Contributions: Conceptualization, J.S.; investigation, J.M.A., A.G. and P.B.; methodology, J.M.A., A.G., P.B., J.S. and A.G.; project administration, A.C.; resources, A.G.; software, J.M.A. and A.G.; supervision, P.B., J.S. and A.C.; validation, J.M.A.; visualization, J.M.A. and A.G.; writing—original draft, J.M.A., A.G., P.B., J.S. and A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported under Grant PLEC2021-007609 funded by MCIN/ AEI/ 10.13039/ 501100011033 and by the "European Union NextGenerationEU/PRTR".

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fernandez, G.; Abella, J.; Quiñones, E.; Rochange, C.; Vardanega, T.; Cazorla, F. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis, Ulm, Germany, 8 July 2014.
2. Aceituno, J.M.; Guasque, A.; Balbastre, P.; Simó, J.; Crespo, A. Hardware resources contention-aware scheduling of hard real-time multiprocessor systems. *J. Syst. Arch.* **2021**, *118*, 102223. [CrossRef]

3. Davis, R.I.; Burns, A. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Comput. Surv.* **2011**, *43*, 35. [CrossRef]
4. Johnson, D. Near-Optimal Bin Packing Algorithms. Ph.D. Thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA, 1973.
5. Oh, Y.; Son, S.H. Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems. *Real-Time Syst.* **1995**, *9*, 207–239. [CrossRef]
6. Coffman, E.G.; Garey, M.R.; Johnson, D.S., Approximation Algorithms for Bin Packing: A Survey. In *Approximation Algorithms for NP-Hard Problems*; PWS Publishing Co.: Boston, MA, USA, 1996; pp. 46–93.
7. Coffman, E.G., Jr.; Csirik, J.; Galambos, G.; Martello, S.; Vigo, D., Bin Packing Approximation Algorithms: Survey and Classification. In *Handbook of Combinatorial Optimization*; Pardalos, P.M.; Du, D.Z.; Graham, R.L., Eds.; Springer: New York, NY, USA, 2013; pp. 455–531. [CrossRef]
8. Dasari, D.; Akesson, B.; Nélis, V.; Awan, M.A.; Petters, S.M. Identifying the sources of unpredictability in COTS-based multicore systems. In Proceedings of the 2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES), Porto, Portugal, 19–21 June 2013; pp. 39–48. [CrossRef]
9. Mitra, T.; Teich, J.; Thiele, L. Time-Critical Systems Design: A Survey. *IEEE Des. Test* **2018**, *35*, 8–26. [CrossRef]
10. Pellizzoni, R.; Schranzhofer, A.; Jian-Jia Chen.; Caccamo, M.; Thiele, L. Worst case delay analysis for memory interference in multicore systems. In Proceedings of the 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010), Dresden, Germany, 8–12 March 2010; pp. 741–746. [CrossRef]
11. Pellizzoni, R.; Betti, E.; Bak, S.; Yao, G.; Criswell, J.; Caccamo, M.; Kegley, R. A Predictable Execution Model for COTS-Based Embedded Systems. In Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium, Chicago, IL, USA, 11–14 April 2011; pp. 269–279. [CrossRef]
12. Rouxel, B.; Derrien, S.; Puaut, I. Tightening Contention Delays While Scheduling Parallel Applications on Multi-Core Architectures. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 164. [CrossRef]
13. Igarashi, S.; Ishigooka, T.; Horiguchi, T.; Koike, R.; Azumi, T. Heuristic Contention-Free Scheduling Algorithm for Multi-core Processor using LET Model. In Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Virtual Conference, 14–16 September 2020. pp. 1–10. [CrossRef]
14. Crespo, A.; Balbastre, P.; Simó, J.; Coronel, J.; Gracia Pérez, D.; Bonnot, P. Hypervisor-Based Multicore Feedback Control of Mixed-Criticality Systems. *IEEE Access* **2018**, *6*, 50627–50640. [CrossRef]
15. Casini, D.; Biondi, A.; Nelissen, G.; Buttazzo, G. A Holistic Memory Contention Analysis for Parallel Real-Time Tasks under Partitioned Scheduling. In Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Sydney, NSW, Australia, 21–24 April 2020; pp. 239–252. [CrossRef]
16. Reder, S.; Becker, J. Interference-Aware Memory Allocation for Real-Time Multi-Core Systems. In Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Sydney, Australia, 21–24 April 2020; pp. 148–159.
17. Maiza, C.; Rihani, H.; Rivas, J.M.; Goossens, J.; Altmeyer, S.; Davis, R.I. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Comput. Surv.* **2019**, *52*, 1–38. [CrossRef]
18. Altmeyer, S.; Davis, R.I.; Indrusiak, L.; Maiza, C.; Nelis, V.; Reineke, J. A Generic and Compositional Framework for Multicore Response Time Analysis. In Proceedings of the 23rd International Conference on Real Time and Networks Systems, Lille, France, 4–6 November 2015; Association for Computing Machinery: New York, NY, USA, pp. 129–138. [CrossRef]
19. Huang, W.H.; Chen, J.J.; Reineke, J. MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources. In Proceedings of the 53rd Annual Design Automation Conference; Association for Computing Machinery, Austin, TX, USA, 5–9 June 2016. [CrossRef]
20. Choi, J.; Kang, D.; Ha, S. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In Proceedings of the 2016 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 181–186.
21. Galizzi, J.; Vigeant, F.; Perraud, L.; Crespo, A.; Masmano, M.; Carrascosa, E.; Brocal, V.; Balbastre, P.; Quartier, F.; Milhorat, F. WCET and Multicores with TSP. In Proceedings of the DASIA 2014 DAta Systems In Aerospace, Warsaw, Poland, 3–5 June 2014.
22. Nowotsch, J. Interference-Sensitive Worst-Case Execution Time Analysis for Multi-Core Processors. Ph.D. Thesis, Fakultät für Angewandte Informatik, Universität Augsburg, Augsburg, Germany, 2014.
23. Skalistis, S.; Kritikakou, A. Dynamic Interference-Sensitive Run-time Adaptation of Time-Triggered Schedules. In Proceedings of the ECRTS 2020—32nd Euromicro Conference on Real-Time Systems, Virtual Conference, 7–10 July 2020; pp. 1–22. [CrossRef]
24. Kim, H.; De Niz, D.; Andersson, B.; Klein, M.; Mutlu, O.; Rajkumar, R. Bounding and Reducing Memory Interference in COTS-Based Multi-Core Systems. *Real-Time Syst.* **2016**, *52*, 356–395. [CrossRef]
25. Davis, R.I.; Griffin, D.; Bate, I. Schedulability Analysis for Multi-Core Systems Accounting for Resource Stress and Sensitivity. In *Leibniz International Proceedings in Informatics (LIPIcs), Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021), Online, 5–9 July 2021*; Brandenburg, B.B., Ed.; Schloss Dagstuhl—Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2021; Volume 196, pp. 7:1–7:26. [CrossRef]
26. Guo, Z.; Yang, K.; Yao, F.; Awad, A. *Inter-Task Cache Interference Aware Partitioned Real-Time Scheduling*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 218–226.
27. Liu, C.L.; Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]

28. Baruah, S.; Fisher, N. The partitioned multiprocessor scheduling of sporadic task systems. In Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05), Miami, FL, USA, 5–8 December 2005. [CrossRef]
29. Garey, M.R.; Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*; W. H. Freeman & Co.: New York, NY, USA, 1979.
30. Davis, R.I.; Burns, A. Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems. In Proceedings of the 2009 30th IEEE Real-Time Systems Symposium, Washington, DC, USA, 1–4 December 2009; pp. 398–409. [CrossRef]
31. *Gurobi Optimizer Reference Manual*; Gurobi Optimization, Inc.: Beaverton, OR, USA, 2019.