Precise Worst-case Blocking Time of Tasks under Priority Inheritance Protocol

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

Precise Worst-case Blocking Time of Tasks under Priority Inheritance Protocol

Eugenio Faldella and Daniela Loreti

Abstract—The problem of precisely computing the worst-case blocking time that tasks may experience is one of the fundamental issues of schedulability analysis of real-time applications. While exact methods have been proposed for more sophisticated protocols, the problem is indeed complex in case of the Priority Inheritance Protocol, even restricting the attention to uniprocessor systems, non-nested resource accesses, and non-self-suspending tasks. Besides a very simple method leading in general to loose upper bounds, only one algorithm of exponential complexity has been so far reported in literature to tighten such bounds.
In this work, we describe a novel approach which, leveraging an operational research technique for modeling the problem, computes the same tight bounds in polynomial time. We then discuss the scenarios in which, assuming no conditional statements in the tasks' code, the computed bounds derive from an actually impossible blocking chain, and we refine the initial model to more precisely compute the worst-case blocking times for any task set in any possible operating condition.

Index Terms—Hard Real-Time Multitasking Applications, Exclusive Access Resources, Basic Priority Inheritance Protocol, Binary Linear Programming

1 Introduction

The problem of scheduling a set of hard real-time tasks sharing exclusive access resources without missing any deadline represents a topic of paramount importance in every application domain characterized by tight timing constraints. When the tasks are scheduled according to a preemptive priority-driven strategy, classical synchronization mechanisms, like semaphores and monitors, are not enough to avoid situations in which a higher-priority task is blocked for an unpredictable amount of time by lower-priority tasks; a suitable protocol must be applied instead.

The basic Priority Inheritance Protocol (PIP), proposed by Sha, Rajkumar and Lehoczky in 1990 [1], [2], was the first protocol specifically conceived for preventing this phenomenon, called *unbounded priority inversion*, in uniprocessor systems, when tasks are scheduled according to a fixed-priority strategy—like Rate-Monotonic Priority Ordering (RMPO) or Deadline-Monotonic Priority Ordering (DMPO) [3], [4]. Other protocols, like the Priority Ceiling Protocol (PCP), Immediate Priority Ceiling Protocol (IPCP), and Stack Resource Policy (SRP), have been proposed afterwards for further limiting the maximum blocking time that a higher-priority task can experience. To this end, they resort to precautionary blocks of lower-priority tasks [1], [5].

Nowadays, even though it does not prevent a task from incurring multiple blocks and despite some persistent disagreement over its advantages and shortcomings [6], [7], PIP is still a very popular [8], [9], [10] and easy-to-use resource access protocol. It is indeed natively and transparently supported by several commercial real-time operating systems and Linux-related real-time kernels [11], [12], [13].

• *E. Faldella and D. Loreti are with the Department of Computer Science and Engineering, University of Bologna, Italy.*
E-mail: {eugenio.faldella,daniela.loreti}@unibo.it

Manuscript received XXXX; revised XXXX.

When dealing with multitasking applications characterized by hard completion deadlines, schedulability analysis is crucial [14], [15]. This study requires the knowledge of the worst-case *computation* time of each task, the maximum *interference* due to higher-priority tasks, as well as the maximum *blocking* time caused by lower-priority tasks—more accurately these parameters are evaluated, more effective is the analysis and significant its outcome. The focus of this work is on the maximum blocking time.

While exact methods for the computation of the worst-case blocking time have been proposed for protocols preventing chained blocking, only a very simple algorithm is usually employed for PIP [1], [14]. This technique—which assumes non-self-suspending tasks served under *uniprocessor* fixed-priority scheduling, and non-nested resource accesses—often provides just loose upper bounds. Definitely better results are generally achievable with an alternative algorithm of exponential complexity, based on an exhaustive search of the solution space. It is worthwhile pointing out, however, that also this algorithm—presented by Rajkumar in [15]—may sometimes lead to the identification of upper bounds that are not tight.

In this work, we start form the same assumptions of the previous works [1], [14], [15], and we propose a novel approach, which leverages an operational research technique, namely Binary Linear Programming (BLP), to compute the same blocking bounds identifiable through the Rajkumar's algorithm with polynomial—rather than exponential—complexity.

Additionally, confining our attention to *linear* tasks (i.e. tasks which do not contain conditional statements that affect the execution order of any of their critical sections), we also give a formal definition of the scenarios in which the computed bounds derive from actually impossible blocking patterns. Relying on this formalization, the initial model is extended so as to permit a more precise evaluation of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

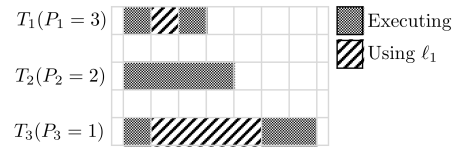


Fig. 1: Execution sequences of a sample application App1

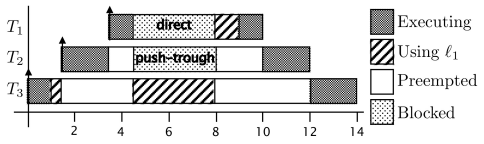


Fig. 2: Example of direct and push though blocking in App1

worst-case blocking time. In summary, the contributions of this paper can be listed as follows:

a) a BLP model which, thanks to its structure, leads to the same bounds of [15] in polynomial time;
b) a refinement of the BLP model and a demonstration of its ability to more precisely compute the worst-case blocking time in the case of linear tasks;
c) an extensive evaluation of the scalability, efficacy, and accuracy of the two proposed BLP methods with respect to existing techniques.

2 Background

Let us consider a uniprocessor system and a set of N tasks $\tau=\{T_1,\ldots,T_n,\ldots,T_N\}$, ordered by descending nominal priority $\{P_1,\ldots,P_n,\ldots,P_N\}$ (i.e. $P_l<P_h\ \forall l>h$), which share a set of M resources $\{\ell_1,\ldots,\ell_m,\ldots,\ell_M\}$ guarded by M distinct binary semaphores $\{S_1,\ldots,S_m,...,S_M\}$. Tasks can be periodic (with unknown periods and initial release offsets), or sporadic. For the sake of simplicity, we suppose that each task has a unique priority. Every task may access any resource more than once during its execution. The longest critical section of the task T_n guarded by the semaphore S_m is denoted with $Z_{n,m}$, and its duration with $L_{n,m}$. All resource accesses are not nested, and tasks do not self-suspend.

According to PIP, each task is initially assigned its nominal priority. When a task T_n tries to access a resource ℓ_m that is already held by a lower-priority task T_l, the priority of T_n is temporarily transferred to T_l. In this case, T_l is said to *inherit* the priority of T_n, and T_n is said to be *directly blocked* by T_l. When T_l exits its critical section and releases ℓ_m, the resource is acquired by the task with maximum priority blocked on that semaphore. The mechanism of priority inheritance is transitive, albeit it has been proven [14] that transitive inheritance can only occur in presence of nested critical sections. A lower-priority task T_l can block a higher-priority task T_n even if they do not share any resource. This case occurs when T_l inherits—as a consequence of a direct block—the priority of another task T_h with $P_h>P_n$. In this case, T_n is said to be subjected to a *push-through blocking*.

As an example, let us consider an application App1 consisting of three tasks sharing one resource. Fig. 1 summarizes the application details. In this example T_2 does not share any resource with T_3, but it can be nonetheless blocked if—as in Fig. 2—T_1 tries to access ℓ_1 when it is already held by T_3. As an effect of priority inheritance, T_2 is push-through blocked for the duration of the critical section in T_3.

In [1], the authors demonstrate that, when resource accesses are handled according to PIP's rules, the maximum number of blocks a task T_n can incur is given by $\min(l_n,\ s_n)$, where l_n and s_n are, respectively, the

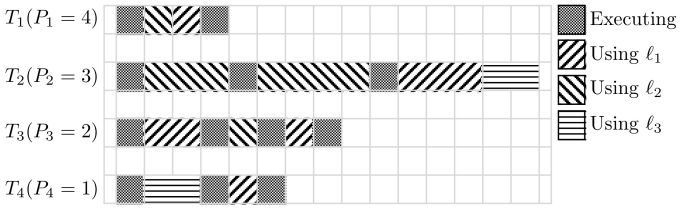


Fig. 3: Execution sequences of a sample application App2

number of lower-priority tasks and the number of distinct semaphores that can block T_n. The constraints deriving from PIP's rules can be more clearly and synthetically formulated [14] exploiting the notion of priority ceiling $\Pi(\ell_m)$ of a resource ℓ_m, defined as the highest priority of the tasks that use the corresponding semaphore S_m: $\Pi(\ell_m)=\max_{n=1,\ldots,N}(P_n|T_n \text{ uses } S_m)$, with $m=1,\ldots,M$. In particular, under PIP each task $T_n(n=1,\ldots,N-1)$ may be blocked *at most once*:

(I) by every lower-priority task T_l that uses any semaphore S_m with $\Pi(\ell_m)\geq P_n$,
(II) on every semaphore S_m with $\Pi(\ell_m)\geq P_n$ that is used by any lower-priority task T_l.

The problem of computing for *all task phasings* the worst-case blocking time B_n that each task T_n may experience is indeed complex, even in the case of non-nested critical sections [1]. A very simple algorithm, having $O(MN^2)$ complexity but often leading to the identification of loose upper bounds, is described by Buttazzo in [14]. This algorithm disregards the interleaving and length of independent executions—as their presence does not influence the worst-case blocking chain—and considers, for each lower-priority task T_l, the set $\gamma_{n,l}$ of the longest critical sections that may contribute to the blocking of T_n, i.e. $\gamma_{n,l}=\{Z_{l,m}|\Pi(\ell_m)\geq P_n\}$. B_n is then computed as $\min(B_n^l,\ B_n^s)$, where:

$$B_n^l=\sum_{l=n+1}^{N}\max_{m=1,\ldots,M}(L_{l,m}|Z_{l,m}\in\gamma_{n,l})$$

$$B_n^s=\sum_{m=1}^{M}\max_{l=n+1,\ldots,N}(L_{l,m}|Z_{l,m}\in\gamma_{n,l})$$

The upper bounds on the blocking time deriving from the application of this algorithm are often loose because conditions (I) and (II) are violated, i.e. two or more critical sections guarded by the same semaphore or belonging to the same task may contribute to the computation of B_n^l and B_n^s respectively. As an example, let us consider an application App2 (see Fig. 3) consisting of four tasks sharing three resources. Table 1 reports the duration of the longest critical sections of App2: each cell of coordinates T_n,ℓ_m reports $L_{n,m}$ (a 0-entry means that T_n does not access ℓ_m). The priority ceiling of the resources is indicated in parenthesis. The values of B_1, B_2 and B_3 (B_4 is clearly 0) deriving from

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

	$\ell_1(P_1)$	$\ell_2(P_1)$	$\ell_3(P_2)$
T_1	1	1	0
T_2	3	4	2
T_3	2	1	0
T_4	1	0	2

TABLE 1: Duration of the longest critical sections in App2.

the outlined algorithm are:

$$B_1{=}\min(B_1^l, B_1^s){=}\min(L_{2,2}+L_{3,1}+L_{4,1}, L_{2,1}+L_{2,2}){=}7$$
$$B_2{=}\min(B_2^l, B_2^s){=}\min(L_{3,1}+L_{4,3}, L_{3,1}+L_{3,2}+L_{4,3}){=}4$$
$$B_3{=}\min(B_3^l, B_3^s){=}\min(L_{4,3}, L_{4,1}+L_{4,3}){=}2$$

While B_2 and B_3 are correct, B_1 is clearly overestimated, because it derives from considering in B_1^l—against PIP condition (I)—the access to the same resource (ℓ_1) by two different tasks (T_3, T_4), and in B_1^s—against PIP condition (II)—the access to two different resources (ℓ_1, ℓ_2) by the same task (T_2).

Aiming to identify more accurate bounds, excluding combinations of critical sections contrasting with the above mentioned conditions, Rajkumar proposed a method [15] which exhaustively considers, for every task T_n, all the subsets of the possibly involved longest critical sections in the $N-n$ lower-priority tasks. The algorithm consists of three steps.

1. Build a one-level tree, called a *simpleton*, for each lower-priority task T_l that may contribute to the blocking of T_n. The simpleton associated to T_l has as many edges as the number of critical sections belonging to the set $\gamma_{n,l}$. The weight of the edge associated to $Z_{l,m}$ is the duration $L_{l,m}$ of the longest access to ℓ_m by T_l.
2. Combine together such one-level trees, starting from the simpleton associated with the highest priority task. Attach all other simpletons, one at a time in descending priority order, to the leafs of the tree under construction.
3. Explore all the paths from the root to the leaves of the final tree, associating to each leaf a weight given by the sum of the weights of the edges along the path. If a path contains more edges referring to the same resource, consider only the edge with the greatest weight. The worst-case blocking time of the task corresponds to the maximum weight of the leaves.

Fig. 4a exemplifies the three simpletons for the computation of B_1 in App2, corresponding to the pairs (T_1, T_2), (T_1, T_3) and (T_1, T_4). The simpletons are then combined together to create the final search tree (Fig. 4b). Correctly, the worst-case blocking time of T_1 results $B_1{=}L_{2,2}+L_{3,1}{=}6$ time unit (t.u.). Fig. 5 illustrates how the critical sections identified through the search tree can block T_1. This allows verifying that the computed maximum blocking time B_1 is actually *possible*, in the sense that (considering all release patterns) there exists a schedule in which T_1 exhibits a duration of priority inversion equal to the claimed bound B_1.

Although Rajkumar proposes a pruning technique [15] to reduce the search space, in the worst case the computation of the tree's path weights requires to check all the subsets of the longest critical sections in the lower-priority tasks that can contribute to the block, making the complexity of the algorithm exponential: $O(N^2M^N)$.

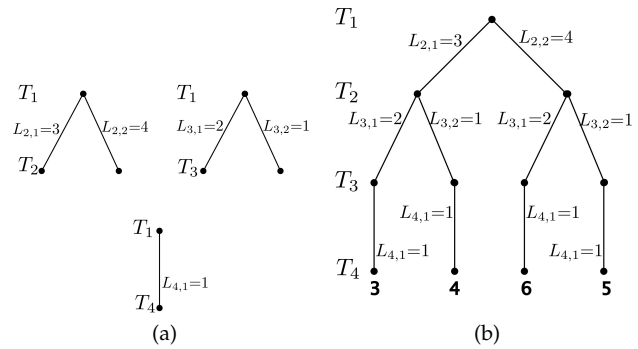


Fig. 4: Evaluation of B_1 in App2 according to Rajkumar's algorithm: simpletons (a) and final search tree (b).

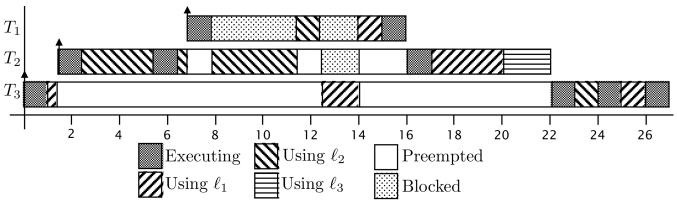


Fig. 5: Possible task scheduling of application App2 leading to the worst-case blocking time B_1 of T_1 (T_4 does not contribute to B_1).

3 Binary Linear Programming Formulation

Linear Programming (LP) is a class of optimization problems that involves only linear functions as regards both the objective and the constraints. Integer Linear Programming (ILP) problems are particular LP problems that envisage the set of decision variables in the solution to be composed of integer values only. A sub-class of ILP problems is represented by BLP problems, where the values of the decision variables are further bounded to be in $\{0, 1\}$.

Leveraging BLP, we first address the goal of computing the same maximum blocking times identifiable with the Rajkumar's algorithm [15]. Analogously to that algorithm, we assume that, during its execution, each task T_n may require to access any resource more than once, all resource accesses are not nested, and tasks do not self-suspend. Likewise [14] and [15], we disregard the presence and length of independent executions, and we define $\gamma_n{=}\bigcup_{l>n}\gamma_{n,l}$ as the set of all the longest critical sections that can block T_n (either directly or via push-through blocking). We associate a binary decision variable $x_{l,m}$ to every longest critical section $Z_{l,m}{\in}\gamma_n$ (with $1{\leq}n{<}l{\leq}N$). The role of each $x_{l,m}$ is to discriminate, on the basis of the value that the BLP solver will assign to it, whether the duration $L_{l,m}$ of the corresponding longest critical section $Z_{l,m}$ does contribute ($x_{l,m}{=}1$) or not ($x_{l,m}{=}0$) to the maximum blocking time B_n.

As implied by conditions (I) and (II), the set of critical sections in γ_n that most adversely contribute to the blocking of T_n must be selected in such a way to be compliant with the following constraints:

Constraint C1. *Select at most one of the critical sections belonging to the same lower-priority task T_l;*

Constraint C2. *Select at most one of the critical sections guarded by the same semaphore S_m.*

Therefore, the problem of computing the worst-case

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

blocking time B_n of task T_n for all task phasings can be straightforwardly modeled through BLP as follows:

$$\begin{aligned} &\textit{maximize } B_n = \sum_{\forall l,m | Z_{l,m} \in \gamma_n} x_{l,m} L_{l,m} \\ &\textit{subject to} \quad \forall\, T_l | \exists Z_{l,j} \in \gamma_n : \sum_{\forall m | Z_{l,m} \in \gamma_n} x_{l,m} \leq 1, \quad (i) \\ &\qquad\qquad \forall\, S_m | \exists Z_{i,m} \in \gamma_n : \sum_{\forall l | Z_{l,m} \in \gamma_n} x_{l,m} \leq 1, \quad (ii) \\ &\qquad\qquad \forall\, T_l, S_m | Z_{l,m} \in \gamma_n : \quad x_{l,m} \in \{0,1\}. \end{aligned} \tag{M1}$$

The first *(i)* and second *(ii)* sets of constraints in model (M1) correspond to the requirements C1 and C2, respectively.

Theorem T1. *The bounds computed through the BLP model* (M1) *are equal to those computed by the Rajkumar's method.*

Proof. By construction, the Rajkumar's method and (M1) start from the same search space: all the possible subsets of γ_n. Then, they both rule out the same solution candidates: those subsets of γ_n containing more than one critical section belonging to the same task or involving the same resource. The BLP model (M1) performs this exclusion by applying constraints *(i)* and *(ii)*, whereas Rajkumar by ensuring that any root-to-leaf path of the tree contains at most one branch for each task and for each resource.

Finally, among all the subsets of longest critical sections satisfying conditions (I) and (II), they both take the one (or those ones) with the larger sum of the durations: the Rajkumar's method considers the longest contribute for each resource in any root-to-leaf path, and then takes the maximum value on the leaves, whereas model (M1) uses the objective function, which maximizes the sum of the durations of the critical sections in the solution.

As Rajkumar's method and (M1) start from the same search space, rule out the same solution candidates, and compute the maximum in the same way, conclusion of Theorem T1 follows. □

3.1 Example

We apply the model (M1) to the computation of the worst-case blocking time B_1 of T_1 in App2. For each lower-priority task T_l (with $l>1$), we consider the set $\gamma_{1,l}$ of the longest critical sections that may contribute to B_1: $\gamma_{1,2}=\{Z_{2,1}, Z_{2,2}\}$, $\gamma_{1,3}=\{Z_{3,1}, Z_{3,2}\}$, and $\gamma_{1,4}=\{Z_{4,1}\}$. Given the complete set of the longest critical sections that may contribute to B_1, $\gamma_1=\{Z_{2,1}, Z_{2,2}, Z_{3,1}, Z_{3,2}, Z_{4,1}\}$, the BLP model to determine B_1 is defined as follows.

$$\begin{aligned} &\textit{maximize } B_1 = 3x_{2,1}+4x_{2,2}+2x_{3,1}+x_{3,2}+x_{4,1} \\ &\textit{subject to} \quad x_{2,1}+\ x_{2,2} \leq 1 \\ &\qquad\qquad x_{3,1}+x_{3,2} \leq 1 \quad (i) \\ &\qquad\qquad x_{4,1} \leq 1 \\ &\qquad\qquad x_{2,1} + x_{3,1} + x_{4,1} \leq 1 \quad (ii) \\ &\qquad\qquad x_{2,2} + x_{3,2} \leq 1 \\ &\qquad x_{l,m} \in \{0,1\}, \forall T_l, S_m | Z_{l,m} \in \gamma_1 \end{aligned}$$

Applying one of the ILP solving methods (e.g., cutting planes, branch-and-bound, branch-and-cut, etc. [16], [17]), the optimal solution of this problem is found to be:

$$x_{2,1}=0,\ x_{2,2}=1,\ x_{3,1}=1,\ x_{3,2}=0,\ x_{4,1}=0.$$

Analogously to the Rajkumar's method (Fig. 4b), (M1) states that $B_1=L_{2,2}+L_{3,1}=6$ t.u.

3.2 Characterization of the BLP Model

Like any optimization problem, (M1) can be written as:

$$\begin{aligned} &\textit{maximize} \quad \boldsymbol{L}^T \boldsymbol{x} \\ &\textit{subject to} \quad \boldsymbol{A}\,\boldsymbol{x} \leq \boldsymbol{b} \qquad \boldsymbol{x} \in \{0,1\} \end{aligned}$$

where $\boldsymbol{L}$ is the vector of the durations of the critical sections in γ_n, $\boldsymbol{x}$ is the vector of the binary decision variables, $\boldsymbol{A}$ is the matrix of the coefficients in the constraints, and $\boldsymbol{b}$ is the vector of constant terms. $\boldsymbol{A}$ has a column for each decision variable and a row for each constraint. In the worst-case scenario where the considered task T_n and all the lower-priority tasks access all resources, the number of decision variables would be $|\gamma_n|=(N-n)\times M$, whereas the number of constraints would be $N-n+M$.

In general ILP problems are $\mathcal{NP}$-hard. However, if the matrix of coefficients $\boldsymbol{A}$ is totally unimodular and the entries of vector $\boldsymbol{b}$ are all integer values, it can be demonstrated [18] that the optimal solution of the correspondent continuous relaxation (i.e with $\boldsymbol{x}\in\mathbb{R}^n$ instead of $\boldsymbol{x}\in\mathbb{Z}^n$) is a vector of integers and represents an optimal solution for the original ILP problem too. In that case, since LP$\in\mathcal{P}$, the optimal solution for the original ILP problem can be found in polynomial time in the worst case.

As regards (M1), it is easy to see that the vector $\boldsymbol{b}$ of constant terms is composed of integer values only: all right-hand sides are 1. Moreover, the following lemma holds.

Lemma L1. *The matrix $\boldsymbol{A}$ of the coefficients of* (M1) *is totally unimodular.*

Proof. An integer matrix is defined *totally unimodular* if all its non-zero minors are 1 or -1. Sufficient condition for a matrix $\boldsymbol{M}$ to be totally unimodular is to fulfill both the following requirements [17]: 1) all the entries are 0, 1 or -1; 2) every column contains at most two non-zero entries and the rows of $\boldsymbol{M}$ can be partitioned into two subsets $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ such that, if a column of $\boldsymbol{M}$ contains two non-zero entries of the same sign, one of them is in $\boldsymbol{M_1}$ and the other is in $\boldsymbol{M_2}$. Otherwise, if a column contains two non-zero entries of opposite sign, they are both in $\boldsymbol{M_1}$ or in $\boldsymbol{M_2}$.

Since in any constraint of model (M1) all decision variables either appear with coefficient 1 or do not appear, the matrix $\boldsymbol{A}$ is composed of just 1 and 0 entries, thus satisfying the first requirement of total unimodularity. Furthermore, $\boldsymbol{A}$ has a column for each decision variable and two sets of rows derived from the two sets of constraints *(i)* and *(ii)*. We name these two sets $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$, respectively. More precisely, $\boldsymbol{A}$ has the following form (0-entries are omitted):

$$\left[\frac{\boldsymbol{A_1}}{\boldsymbol{A_2}}\right] = \begin{array}{c} \\ n{+}1 \\ \vdots \\ l \\ \vdots \\ N \\ \hline 1 \\ \vdots \\ m \\ \vdots \\ M \end{array} \begin{array}{c} \begin{array}{cccc} x_{n+1,1} \cdots & x_{n+1,M} \cdots & x_{l,1} \cdots & x_{l,M} \cdots \quad x_{N,1} \cdots & x_{N,M} \end{array} \\ \left[\begin{array}{cccc} 1 \cdots\cdots 1 & & & \\ & & 1 \cdots\cdots 1 & \\ & & & 1 \cdots\cdots 1 \\ \hline 1 \ddots & & 1 \ddots & \quad 1 \ddots \\ & 1 & & \\ & & 1 & \quad 1 \end{array}\right] \end{array}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

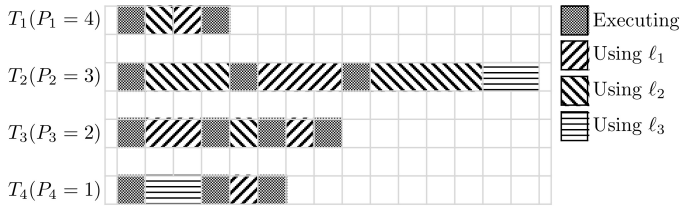


Fig. 6: Execution sequences of a sample application App3

Note that $\boldsymbol{A}$ may actually have fewer rows or columns as not all resources and tasks may contribute to B_n, but only those for which the corresponding $Z_{l,m}$ is in γ_n.

By construction, the sub-matrix $\boldsymbol{A_1}$ derived from *(i)* has a row for each lower-priority task T_l such that $\exists Z_{l,m} \in \gamma_n$. Since each constraint of set *(i)* regards a different T_l, a decision variable can appear with a coefficient different from 0 in at most one constraint of *(i)*, and as such, in at most one row of $\boldsymbol{A_1}$. Consequently, there cannot be more than one 1-entry in a column of $\boldsymbol{A_1}$.

By construction, the sub-matrix $\boldsymbol{A_2}$ derived from *(ii)* has a row for each semaphore S_m such that $\exists Z_{l,m} \in \gamma_n$. As we do not deal with nested resource accesses, each critical section—as well as each decision variable of (M1)—involves only one resource. As each constraint of *(ii)* regards a different S_m, a decision variable can appear with a coefficient different from 0 in at most one constraint of *(ii)*, and as such, in at most one row of $\boldsymbol{A_2}$. Consequently, there cannot be more than one 1-entry in a column of $\boldsymbol{A_2}$.

As constraints *(i)* and *(ii)* divide $\boldsymbol{A}$ in two sets of rows, each with at most one 1-entry per column, we can conclude that if a column of $\boldsymbol{A}$ contains two non-zero entries of the same sign, one of them is in $\boldsymbol{A_1}$ and the other is in $\boldsymbol{A_2}$. Therefore, also the second requirement for total unimodularity is fulfilled. Conclusion of Lemma L1 follows. □

As a consequence, it is possible to apply a resolution method (e.g., the simplex, ellipsoid, or Karmarkar's interior point method [17], [19]) to the continuous LP relaxation of (M1) in order to obtain its optimal solution in polynomial time in the worst case.

4 Model Refinement

Let us now consider application App3 in Fig. 6, where the only difference with respect to App2 is the order by which T_2 accesses the resources ℓ_1 and ℓ_2. The BLP computation (as well as the Rajkumar's method) leads to the same blocking time $B_1{=}L_{2,2} + L_{3,1}{=}6$ t.u.

However, according to PIP rules, this blocking chain is actually impossible. Indeed, in order to have T_1 blocked by $Z_{3,1}$ and $Z_{2,2}$, the corresponding resources must be acquired before T_1 is released. We can disregard T_1 for the moment and try to recreate the situation in which T_2 has acquired ℓ_2 to execute $Z_{2,2}$ and T_3 has acquired ℓ_1 to execute $Z_{3,1}$.

Looking at Fig. 7a, we note that, since T_3 has lower priority, it has to start first in order to be able to acquire ℓ_1 (at time t_1) before T_2 preempts it. Then, T_2 must have used ℓ_2 for three t.u. and—more importantly for our discussion—ℓ_1 for three t.u., before being able to actually acquire ℓ_2 to execute $Z_{2,2}$. The first of these conditions can be met while T_3 holds ℓ_1, whereas the second cannot. Indeed, if we focus

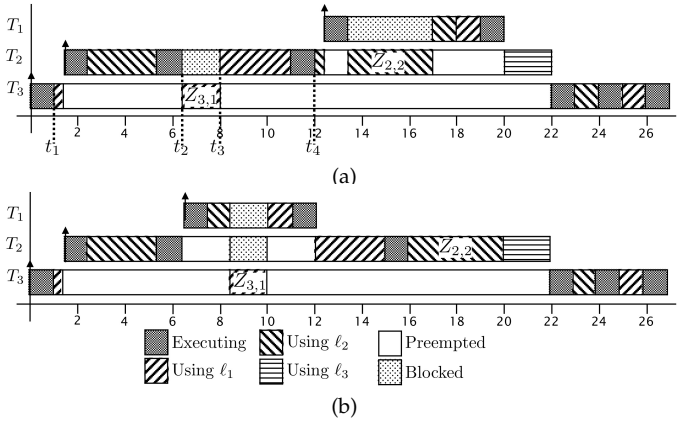


Fig. 7: In case of application App3, the worst-case blocking time of T_1 computed by (M1) (as well as by the Rajkumar's method) corresponds to an actually impossible situation. Fig. 7a and 7b report the two possibilities: T_1 blocked only on $Z_{2,2}$, and T_1 blocked only on $Z_{3,1}$.

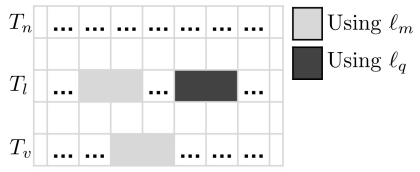


Fig. 8: Under PIP, T_l cannot hold ℓ_q while T_v holds ℓ_m

on instant t_2 of Fig. 7a, we see that, since T_2 requests ℓ_1, T_3 inherits its priority and continues to execute $Z_{3,1}$. Only after this critical section is completed, T_2 can acquire ℓ_1 (time t_3) for three t.u. and finally reach $Z_{2,2}$ (time t_4). But at this point $Z_{3,1}$ has completed already and cannot contribute to B_1. Therefore, in Fig. 7a we see that T_1 is only blocked on $Z_{2,2}$. This does not mean that T_1 cannot be blocked on $Z_{3,1}$ at all. Fig. 7b shows exactly this situation (T_1 is blocked on $Z_{3,1}$ but not on $Z_{2,2}$). Besides, this observation is totally independent from the order in which T_1 requests its resources. Actually, it is only related to the resources requested by the lower-priority tasks T_2 and T_3 even before T_1 is released. In particular, we see that $Z_{3,1}$ cannot contribute to the blocking of T_1 together with $Z_{2,2}$, because the latter is preceded by a critical section involving the same resource of $Z_{3,1}$.

More generally, if we consider three tasks T_n, T_l and T_v (with $P_n{>}P_l{>}P_v$) as in Fig. 8, such that T_l and T_v share the same resource ℓ_m, and T_l must access another resource ℓ_q after having released ℓ_m (with $\Pi(\ell_m), \Pi(\ell_q) \geq P_n$), we observe that only one of the following conditions can be true at the same time:

(a) the task T_v holds ℓ_m,
(b) the task T_l holds ℓ_q.

As *(a)* and *(b)* cannot happen at the same time, only one of the two critical sections (the one in T_v involving ℓ_m or the one in T_l involving ℓ_q) can contribute to the blocking time of the higher-priority task T_n. In light of this, it is evident that conditions (I) and (II) are not enough to precisely compute the worst-case blocking time B_n of task T_n. A third condition has to be taken into account:

(III) A task cannot be executing any critical section preceded by another critical section involving a resource that is held by a lower-priority task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

In order to formalize and demonstrate this condition, we need to modify our assumptions and introduce a novel notation able to describe *all* the critical sections—not just the longest ones—and the order in which they are executed.

4.1 Notation and properties

As in previous works, we assume again non-nested resource accesses and non-self-suspending, periodic or sporadic tasks executing on a uniprocessor system. Also, we now restrict our attention to *linear* tasks, i.e. tasks with no conditional statements that affect the execution order of the critical sections. We are aware that this is a strong assumption and the issues deriving from it are discussed in Section 7.

The k-th (k=1, 2, ...) critical section executed by task T_n (n=1, 2, ..., N) is denoted by $z_{n,k}$. The semaphore guarding the resource accessed during $z_{n,k}$ is denoted by $\sigma_{n,k}$ and the duration of $z_{n,k}$ by $d_{n,k}$. $\mathcal{Z}_n$=$\{z_{n,k}|k$=1, 2, ...$\}$ (n=1, 2, ..., N) denotes the set of the critical sections of the task T_n sorted by execution order—i.e. for any $k', k''|k'<k''$, the execution of $z_{n,k'}$ precedes that of $z_{n,k''}$ ($z_{n,k'} \prec z_{n,k''}$). Additionally, we need a set of symbols to clearly describe the scheduling of an application. Given an application $\mathcal{A}$ composed of a task set τ, we consider a generic PIP schedule $\mathcal{S}$ for $\mathcal{A}$, and we denote with a_n and f_n the instants when the task T_n is released and completed, respectively. Analogously, we denote with $a_{n,k}$ and $f_{n,k}$ the instants when the semaphore guarding the access to the critical section $z_{n,k}$ is respectively acquired and released by T_n. When PIP is used to enforce mutual exclusion in the access to shared resources, the following lemma holds.

Lemma L2. *If a PIP task schedule $\mathcal{S}$ states that a semaphore guarding the access to a critical section $z_{l,k}\in\mathcal{Z}_l$ is not acquired by task T_l before a higher-priority task T_n is released, then it can only be acquired after T_n is completed, i.e.*

$$\forall T_l, T_n, n<l : a_n<a_{l,k} \Rightarrow f_n<a_{l,k}$$

Proof. Under PIP rules, T_n is blocked by T_l only if T_l has been preempted while executing a critical section $z_{l,j}$ that can block T_n. If $a_n<a_{l,k}$, only a critical section preceding $z_{l,k}$ can block T_n, so we consider a generic $z_{l,j}$ with $j<k$. If such a critical section $z_{l,j}$ was indeed started and not finished before a_n, we know from (I) that it is the only critical section that will block T_n (recall that we assume non-self-suspending tasks). Any other critical section after $z_{l,j}$ (including $z_{l,k}$) is delayed until the end of T_n. $f_n<a_{l,k}$ follows. □

We can now give a formal definition of condition (III) employing the newly introduced notation.

Theorem T2. *For all triplet of critical sections $z_{n,k}, z_{n,k'}, z_{l,j}$ in a given application $\mathcal{A}$ such that $n<l, k<k'$ and $\sigma_{l,j}=\sigma_{n,k}$, any PIP task schedule of $\mathcal{A}$ is such that: $f_{l,j}<a_{n,k'} \veebar f_{n,k'}<a_{l,j}$*

Proof. By exhaustive case analysis. As $\sigma_{l,j}=\sigma_{n,k}$ and accesses to the same resource are mutually exclusive, if task T_n is within the critical section $z_{n,k}$, then T_l cannot acquire the semaphore guarding the access to $z_{l,j}$, and vice-versa. This entails that either $z_{l,j}$ is completely executed before $z_{n,k}$ starts or vice-versa, i.e. $f_{l,j}<a_{n,k} \veebar f_{n,k}<a_{l,j}$

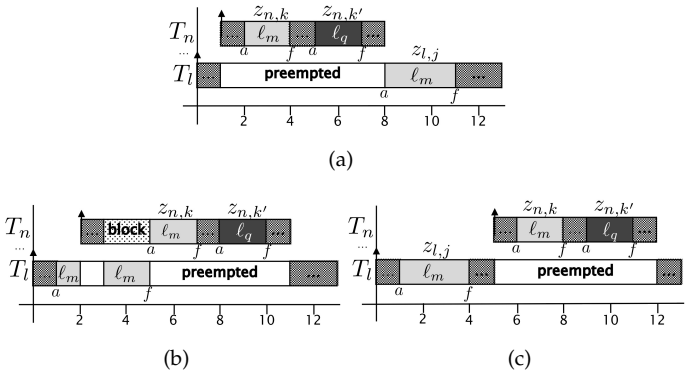


Fig. 9: Visual representation of theorem T2. As $z_{n,k}$ and $z_{l,j}$ both require ℓ_m, they cannot overlap in any scheduling for (II). As a result, under PIP rules, $z_{l,j}$ can be either executed after (Fig. 9a) or before (Fig. 9c and 9b) $z_{n,k}$. In particular, Fig.9c shows the case of $z_{l,j}$ completed before T_n starts, whereas Fig.9b shows T_n released while T_l is within the critical section $z_{l,j}$. As prescribed by T2, in none of these cases $z_{l,j}$ is executed while T_n is within $z_{n,k'}$.

Let us first consider the case $f_{l,j}<a_{n,k}$. As $k<k'$, necessarily $z_{n,k} \prec z_{n,k'}$ entailing that $f_{n,k}<a_{n,k'}$. We can therefore derive that in case $f_{l,j}<a_{n,k}$, the first conclusion of theorem T2 holds: $f_{l,j}<a_{n,k'}$.

Let us now consider the case $f_{n,k}<a_{l,j}$ (i.e. $z_{n,k}$ is completely executed before $z_{l,j}$). We know by construction that $a_n<f_{n,k}$ and we can therefore say $a_n<a_{l,j}$ in this case, i.e. the critical section $z_{l,j}$ is certainly started after T_n. As $n<l$, from Lemma L2 we know that, if the semaphore guarding the access to the critical section $z_{l,j}$ is not acquired before T_n is released (i.e. if $a_n<a_{l,j}$), then it can only be acquired after T_n is completed (i.e. $f_n<a_{l,j}$). It follows that the second conclusion of T2 holds: $f_{n,k'}<a_{l,j}$. □

Fig. 9 depicts the situations described by T2. Model (M1) can be now refined by taking into account theorem T2. To this end, we introduce a new set of symbols.

$\Psi_{n,l}$ ($1\leq n<l\leq N$) denotes the set of semaphores with priority ceiling greater than or equal to the priority of the task T_n, which can be acquired by a lower-priority task T_l, thus causing a direct or push-through block to the execution of T_n. We also define $\Psi_{n,*}$ as the set of all semaphores with a priority ceiling greater than or equal to the priority of the task T_n which can be acquired by any lower-priority task:

$$\Psi_{n,*}=\bigcup_{l>n} \Psi_{n,l}$$

$\Phi_{n,l,m}$ ($1\leq n<l\leq N$, $S_m\in\Psi_{n,l}$) denotes the set of critical sections guarded by semaphore S_m in the lower-priority task T_l, which can block (either directly or via push-through blocking) the higher-priority task T_n, i.e. $\Phi_{n,l,m}$=$\{z_{l,k}|\sigma_{l,k}=S_m, \Pi(\ell_m)\geq P_n\}$. In particular, we denote with $\varphi_{l,m}$ the critical section among those guarded by the semaphore S_m that is first executed by the task T_l, i.e. $\varphi_{l,m}=z_{l,k'}\in\Phi_{n,l,m}|z_{n,k'} \prec z_{n,k''}\ \forall z_{l,k''}\in\Phi_{n,l,m}, k'\neq k''$.

$\Phi_{n,l,*}$ ($1\leq n<l\leq N$) is the set of critical sections of task T_l that can block the higher-priority task T_n:

$$\Phi_{n,l,*}=\{z_{l,k}|\sigma_{l,k}\in\Psi_{n,l}\}=\bigcup_{m|S_m\in\Psi_{n,l}} \Phi_{n,l,m}.$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

Analogously, $\Phi_{n,*,m}$ ($1 \leq n < N$, $S_m \in \Psi_{n,*}$) denotes the set of the critical sections guarded by the semaphore S_m in all the lower-priority tasks that can block T_n:

$$\Phi_{n,*,m} = \bigcup_{l>n} \Phi_{n,l,m}.$$

The overall set of critical sections in the lower-priority tasks which can block T_n is $\Phi_{n,*,*}$ ($1 \leq n < N$):

$$\Phi_{n,*,*} = \bigcup_{l>n} \Phi_{n,l,*} = \bigcup_{m | S_m \in \Psi_{n,*}} \Phi_{n,*,m}.$$

$\Theta_{n,l,m}$ ($1 \leq n < l \leq N$, $S_m \in \Psi_{n,l}$) denotes the subset of the critical sections in $\Phi_{n,l,*}$ which are guarded by a semaphore other than S_m and executed after $\varphi_{l,m}$, i.e. $\Theta_{n,l,m} = \{z_{l,k} \in \Phi_{n,l,*} | \sigma_{l,k} \neq S_m, \varphi_{l,m} \prec z_{l,k}\}$.

4.2 Worst-case blocking time computation

Aiming to refine the computation of the maximum blocking time B_n of a task T_n over *all tasks phasings*, we associate a binary decision variable $x_{l,k}$ to every critical section $z_{l,k} \in \Phi_{n,*,*}$. As in the previous BLP model, the role of each $x_{l,k}$ is to discriminate, on the basis of the value that the solver will assign to it, whether the duration $d_{l,k}$ does contribute ($x_{l,k}$ = 1) or not ($x_{l,k}$ = 0) to the maximum blocking time B_n.

The model (M1) must be reformulated to express C1 and C2 with the notation introduced in section 4.1, while condition (III) originates a third constraint.

Constraint C3. *For each pair (T_l, S_m) such that T_l can block T_n on a critical section guarded by the semaphore S_m, select either one of the critical sections following $\varphi_{l,m}$, or one of those involving S_m in tasks with priority lower than T_l.*

In light of this, C3 can be stated through a third set of constraints in the following BLP model:

$$\begin{aligned}
&\textit{maximize} \quad B_n = \sum_{z_{l,k} \in \Phi_{n,*,*}} x_{l,k} d_{l,k} \\
&\textit{subject to} \quad \forall T_l | l > n: \sum_{z_{l,k} \in \Phi_{n,l,*}} x_{l,k} \leq 1, \quad (i) \\
&\qquad \forall S_m \in \Psi_{n,*}: \sum_{z_{l,k} \in \Phi_{n,*,m}} x_{l,k} \leq 1, \quad (ii) \\
&\qquad \forall T_l, S_m | n < l < N \wedge S_m \in \Psi_{n,l}: \\
&\qquad\qquad \sum_{z_{l,k} \in \Theta_{n,l,m}} x_{l,k} + \sum_{z_{v,k} \in \bigcup_{v>l} \Phi_{n,v,m}} x_{v,k} \leq 1, \quad (iii) \\
&\qquad \forall T_l | l > n \qquad x_{l,k} \in \{0,1\}.
\end{aligned} \tag{M2}$$

Note that the set *(iii)* introduces a constraint for each pair (T_l, S_m) such that T_l can block T_n on a critical section guarded by the semaphore S_m. For each of these pairs, it considers the first critical section involving S_m in T_l, that is $\varphi_{l,m}$. In particular, the first sum of *(iii)* refers to the critical sections following $\varphi_{l,m}$ and not involving S_m, whereas the second sum addresses the set of critical sections involving S_m in any task T_v with priority lower than T_l. Constraints *(iii)* fulfil the statement C3 by imposing that only one critical section be selected from these two sets.

(M2) has a number of decision variables that depends on the total number of critical sections (not just the longest ones) relevant for the blocking of T_n. The dimension of (M2) is therefore $|\Phi_{n,*,*}|$. In the worst scenario where T_n and all lower-priority tasks access all resources, the sets of constraints *(i)* and *(ii)* have the same cardinality as model (M1) (i.e. $N-n$ and M, respectively), while the third set *(iii)* introduces an inequality for each pair (l, m) such that T_l can block T_n on a critical section involving S_m (i.e. $(N-n-1) \times M$). The sub-matrix derived from *(i)* and *(ii)* would be again totally unimodular. But, since the sub-matrix generated by *(iii)* has a variable number of 1-entries in each column, it deprives $\boldsymbol{A}$ of total unimodularity. For this reason, the third set of constraints allows computing a more accurate value for B_n, albeit it is no longer guaranteed that the optimal solution is found in polynomial time.

Algorithm 1 Constructive procedure for checking if a task T_n can be blocked by all the critical sections of a set Q_n.

Input: τ, a set of tasks ordered by descending nominal priority; T_n, a task of τ; $Q_n \subseteq \Phi_{n,*,*}$, a set of critical sections
Output: $\mathcal{S}$, a possible schedule where T_n is blocked by all the critical sections in Q_n, or *false* if it does not exist.

```
1:  t=0                                          ▷ schedule start time
2:  Y=∅                                          ▷ no semaphore acquired so far
3:  while τ≠∅ do
4:      T_l= task with lowest priority in τ
5:      if l>n and ∃ z_{l,k}∈Q_n then
6:          for each z_{l,k'} ⪯ z_{l,k} do
7:              if σ_{l,k'}∈Y then return false end if
8:          end for
9:          a_l=t                                ▷ T_l is released at time t
10:         t=a_{l,k}+ϵ                          ▷ execute T_l until the start of z_{l,k}
11:         Y=Y ∪ {σ_{l,k}}                      ▷ σ_{l,k} has been acquired
12:         Q_n=Q_n \ {z_{l,k}}
13:     else if l≤n then
14:         a_l=t
15:         t=t+ϵ
16:     end if
17:     τ=τ \ {T_l}
18: end while
19: if Q_n≠∅ then return false end if
20: return S
```

4.2.1 *Quality of the solution of* (M2)

The solution found by (M2) is precise in the worst possible case over all task phasings in the sense that, if all release patterns are possibile, there can exist a schedule in which the task T_n undergoes a blocking chain with duration equal to the solution of (M2). To prove this, we need to first provide a constructive procedure which, given a task set τ and a subset $Q_n \subseteq \Phi_{n,*,*}$, tries to build a worst-case schedule $\mathcal{S}$ where *all* the elements of Q_n contribute to the blocking of T_n. If such a schedule can be found, we say that Q_n corresponds to a *possible blocking chain* for T_n.

The procedure (presented in Algorithm 1) starts by considering all the tasks in τ by increasing order of priority. For each T_l with priority lower than T_n, if a critical section $z_{l,k}$ belonging to that task is found in Q_n (Line 5), then the task is "virtually" released and executed until the acquisition of the aforementioned critical section (Lines 9 and 10). Also, $\sigma_{l,k}$ is added to the set Y of semaphores acquired so far (Line 11). If such a critical section is not present in Q_n and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers

T_l has a priority lower than T_n, the task T_l is not released at all because its execution is not interesting for the goal of the procedure. T_n and all the higher-priority tasks are simply released one after the other by increasing priority (Line 14).

Line 19 ensures that all the critical sections in Q_n have been started at the end of the procedure. Otherwise, the procedure returns *false*. Conversely, Lines 6 to 8 ensure that all the critical sections in Q_n have not concluded before T_n is started (because in that case, the finished critical section could not contribute to the block of T_n).

Theorem T3. *Given a set $Q_n \subseteq \Phi_{n,*,*}$ of critical sections selected according to C1, C2 and C3, it is always possible to find a schedule in which T_n is blocked by all the elements in Q_n.*

Proof. By construction, Algorithm 1 returns a schedule $\mathcal{S}$ in which T_n is blocked by all the elements of Q_n only if the conditions of Lines 7 and 19 are never met.

Let us focus on Line 7 first. When iterating over a generic T_l with $l>n$ and considering one of its critical sections $z_{l,k} \in Q_n$, the condition $\sigma_{l,k'} \in Y$ is satisfied if either (*a*) there exists a $z_{l,k'}$ strictly preceding $z_{l,k}$ ($z_{l,k'} \prec z_{l,k}$) such that $\sigma_{l,k'}$ was acquired by a previously considered T_v (with $v>l$, recall that tasks are released by increasing priority); or *(b)* $z_{l,k}$ itself involves an already acquired semaphore $\sigma_{l,k}$ (note that the loop in Line 6 iterates also on $z_{l,k'}=z_{l,k}$). But from C3 we know that $\forall T_l, S_m$, the set Q_n contains either a critical section following $\varphi_{l,m}$ or a $z_{v,j}$ such that $v>l$ and $\sigma_{v,j}=S_m$. From which we deduce that $\forall z_{l,k}, z_{v,j} \in Q_n | v>l$ none of the critical sections preceding $z_{l,k}$ is guarded by the same semaphore of $z_{v,j}$ (i.e. $\sigma_{l,k'} \neq \sigma_{v,j} \ \forall \ z_{l,k'} | k'<k$). Therefore, case (*a*) is not possible because when considering T_l, no T_v ($v>l$) can have acquired in a previous iteration the same semaphore of any $z_{l,k'}$ preceding $z_{l,k}$. Furthermore, from C2 we know $\forall z_{l,k}, z_{v,j} \in Q_n, \sigma_{l,k} \neq \sigma_{v,j}$, so no other task can have acquired $\sigma_{l,k}$ in a previous iteration. Consequently, case (*b*) is not possible either, and we can conclude that the condition of Line 7 is never met when all the critical sections of Q_n are selected according to C2 and C3.

Let us now consider Line 19. The while loop of Lines 3 to 18 iterates over the whole task set until $\tau=\varnothing$. At each iteration, a task is considered and at most one of its critical sections is removed from Q_n (Line 12). Since $Q_n \subseteq \Phi_{n,*,*}$ by construction, Q_n cannot contain critical sections of tasks with priority higher or equal to P_n. So, the only possibility for having $Q_n \neq \varnothing$ after the while loop is that the input Q_n contained more than one critical section for each task. But from C1 we know that there is at most one critical section per task in Q_n, i.e. $\forall z_{l,k}, z_{v,j} \in Q_n, l \neq v$ (recall that tasks do not self-suspend). As a consequence, the condition of Line 19 is never met when all the elements of Q_n are selected according to C1. Theorem T3 follows. □

4.2.2 Example

We apply (M2) to compute B_1 for App3. According to the notation in section 4.1, the ordered sets of critical sections in each task and their durations are:

$\mathcal{Z}_1=\{z_{1,1}, z_{1,2}\}$ $\quad \sigma_{1,1}=S_2, \sigma_{1,2}=S_1$
$\mathcal{Z}_2=\{z_{2,1}, z_{2,2}, z_{2,3}, z_{2,4}\}$ $\quad \sigma_{2,1}=S_2, \sigma_{2,2}=S_1, \sigma_{2,3}=S_2, \sigma_{2,4}=S_3$
$\mathcal{Z}_3=\{z_{3,1}, z_{3,2}, z_{3,3}\}$ $\quad \sigma_{3,1}=S_1, \sigma_{3,2}=S_2, \sigma_{3,3}=S_1$
$\mathcal{Z}_4=\{z_{4,1}, z_{4,2}\}$ $\quad \sigma_{4,1}=S_3, \sigma_{4,2}=S_1.$

$d_{1,1}=1,\ d_{1,2}=1.\ d_{2,1}=3, d_{2,2}=3,\ d_{2,3}=4,\ d_{2,4}=2.$
$d_{3,1}=2,\ d_{3,2}=1, d_{3,3}=1.\ d_{4,1}=2,\ d_{4,2}=1.$

The priority ceiling of the involved resources are $\Pi(\ell_1)=P_1,\ \Pi(\ell_2)=P_1,\ \Pi(\ell_3)=P_2$.

In order to compute B_1, all the critical sections in lower-priority tasks should be taken into account except those guarded by the semaphore S_3 because it has a priority ceiling $\Pi(\ell_3)=P_2$ lower than the priority of T_1. Hence:

$\Psi_{1,2}=\{S_1, S_2\}, \Psi_{1,3}=\{S_1, S_2\}, \Psi_{1,4}=\{S_1\}, \Psi_{1,*}=\{S_1, S_2\}.$
$\Phi_{1,2,*}=\{z_{2,1}, z_{2,2}, z_{2,3}\}$ $\quad \Phi_{1,*,1}=\{z_{2,2}, z_{3,1}, z_{3,3}, z_{4,2}\}$
$\Phi_{1,3,*}=\{z_{3,1}, z_{3,2}, z_{3,3}\}$ $\quad \Phi_{1,*,2}=\{z_{2,1}, z_{2,3}, z_{3,2}\}$
$\Phi_{1,4,*}=\{z_{4,2}\}.$
$\Phi_{1,*,*}=\{z_{2,1}, z_{2,2}, z_{2,3}, z_{3,1}, z_{3,2}, z_{3,3}, z_{4,2}\}.$

The BLP model is defined by first associating a binary decision variable to each critical section in $\Phi_{1,*,*}$: $(x_{2,1}, x_{2,2}, x_{2,3}, x_{3,1}, x_{3,2}, x_{3,3}, x_{4,2})$. Then, in order to write the constraints *(iii)* introduced in (M2), we also identify the relevant sets of critical sections for each lower-priority task T_l and each semaphore $S_m \in \Psi_{1,l}$:

(l=2, $\Psi_{1,2}=\{S_1, S_2\}$):
$\varphi_{2,1}=z_{2,2},\ \Theta_{1,2,1}=\{z_{2,3}\},\ \bigcup_{v>2} \Phi_{1,v,1}=\{z_{3,1}, z_{3,3}, z_{4,2}\}.$
$\varphi_{2,2}=z_{2,1},\ \Theta_{1,2,2}=\{z_{2,2}\},\ \bigcup_{v>2} \Phi_{1,v,2}=\{z_{3,2}\}.$
(l=3, $\Psi_{1,3}=\{S_1, S_2\}$):
$\varphi_{3,1}=z_{3,1},\ \Theta_{1,3,1}=\{z_{3,2}\},\ \bigcup_{v>3} \Phi_{1,v,1}=\{z_{4,2}\}.$
$\varphi_{3,2}=z_{3,2},\ \Theta_{1,3,2}=\{z_{3,3}\},\ \bigcup_{v>3} \Phi_{1,v,2}=\varnothing.$

This allow us to formulate the BLP model to determine B_1 for App3 as follows:

$$
\begin{aligned}
&max. B_1 = 3x_{2,1}+3x_{2,2}+4x_{2,3}+2x_{3,1}+x_{3,2}+x_{3,3}+x_{4,2} \\
&s.\ t. \quad x_{2,1}+x_{2,2}+x_{2,3} \le 1 \\
&\qquad x_{3,1}+x_{3,2}+x_{3,3} \le 1 \quad (i) \\
&\qquad x_{4,2} \le 1 \\
&\qquad x_{2,2}+x_{3,1}+x_{3,3}+x_{4,2} \le 1 \quad (ii) \\
&\qquad x_{2,1}+x_{2,3}+x_{3,2} \le 1 \\
&\qquad x_{2,3}+x_{3,1}+x_{3,3}+x_{4,2} \le 1 \\
&\qquad x_{2,2}+x_{3,2} \le 1 \quad (iii) \\
&\qquad x_{3,2}+x_{4,2} \le 1 \\
&\qquad x_{3,3} \le 1 \\
&x_{l,k} \in \{0,1\}, \forall 1<l \le 4.
\end{aligned}
$$

The BLP solver provides a novel solution for the maximum blocking time of the task T_1:

$x_{2,1}=1,\ x_{2,2}=0,\ x_{2,3}=0,\ x_{3,1}=1,\ x_{3,2}=0,\ x_{3,3}=0,\ x_{4,2}=0$
$B_1=d_{2,1}+d_{3,1}=5$ t.u.

Fig. 10 shows the possible blocking chain that causes B_1.

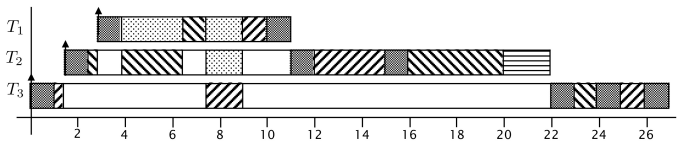


Fig. 10: Task schedule of App3 leading to the worst-case blocking time of T_1. The critical sections are denoted according to the legend of Fig. 5.

5 Empirical evaluation

In order to experimentally ascertain the validity of the proposed approach, we developed a simple software prototype[1], which allows easily specifying (through a graphical interface) arbitrary execution sequences, and calculating the worst-case blocking time of each task leveraging model (M2). In addition, we conducted an extensive evaluation of the proposed approach by comparing its performance with those of pre-existing techniques. The solution of the BLP model (M1) and (M2) are obtained through IBM ILOG CPLEX solver[2]. The study is conducted on an average hardware architecture: a 2,9 GHz quad-core Intel i7 with 16 GB RAM. The code to reproduce these experiments is available on GitHub[3].

5.1 Experimental setup

To evaluate the performance of the proposed approach, we randomly generate a set of applications with a variable number of tasks (N=$\{5, .., 100\}$) taking inspiration from the process described in [13]. For each task of any application, a number k of critical sections is randomly chosen (with uniform distribution) in the range $[k_{min}, k_{max}]$. In particular, applications with three different ranges are generated: $[5, 10]$ (*few* critical sections in each task), $[20, 30]$ (*many* critical sections in each task), and $[5, 20]$ (*mixed*, some tasks have few, some have many critical sections). The number M of shared resources in each application is varied across $\{5, 10, 15, 20\}$. For each critical section in each task, the involved resource is picked with uniform distribution in a set with cardinality M.

Since our approach only focuses on the computation of the maximum blocking time, the generation process disregards tasks' worst-case execution times, periods and deadlines, which would be instead necessary for schedulability analysis. The duration of each critical section does not influence the practicability of a blocking chain, but affects its duration. Therefore, it is randomly chosen (uniform distribution) in three different ranges $[1, 25]\mu$s, $[25, 50]\mu$s and $[50, 100]\mu$s, as to simulate applications with various critical section lengths. Although the choice of these parameters does not represent any specific real-life workload, it allows simulating a large variety of applications, characterized by low, medium and high resource contention. For each application generated in this way the probability p_{acc} of a task with k critical sections to access a certain resource at least once can be computed as $1 - (1 - 1/M)^k$. As k is uniformly distributed in the range $[k_{min}, k_{max}]$, we actually have:

$$p_{acc}=\frac{1}{k_{max} - k_{min} + 1} \sum_{k=k_{min}}^{k_{max}} 1 - \left(1 - \frac{1}{M}\right)^k$$

5.2 Evaluation approach

The evaluation is conducted by considering the applications generated according to the previously described procedure, and comparing the performance of our models (M1) and (M2) with those of Rajkumar [15] and Buttazzo [14]. In addition, we consider another relevant work in this field, Yang et al. [13], which assumes non-nested resource accesses and performs a unified comparison of several protocols, including PIP, employing the concept of "blocking fractions" in a multiprocessor scenario. Blocking fractions are the portion of longest critical section that can contribute to the blocking. This concept is used to define the decision variables of an LP model that maximizes the task's response time. Since the model in [13] is devoted to schedulability analysis, it is able to estimate other parameters besides the blocking bounds, such as the worst-case execution time and the maximum interference. Obviously, we restrict our attention to the LP model of the blocking bounds under PIP.

We perform three kinds of test.

- *Scalability test*: in order to assess the scalability of models (M1) and (M2), we analyze the time required to compute the worst-case blocking time of all the tasks in any application. We consider how this time increases with the dimension of the problem, i.e. when the number of tasks in each application increases.
- *Efficacy test*: we compare the value of the solution obtained by the five methods to understand if there is a significant difference in the computed B_n.
- *Accuracy test*: to asses the quality of the solution found by each method, we check the practicability of the correspondent blocking times. To do so, we compute the ratio of solutions for which Algorithm 1 can find a possible worst-case schedule.

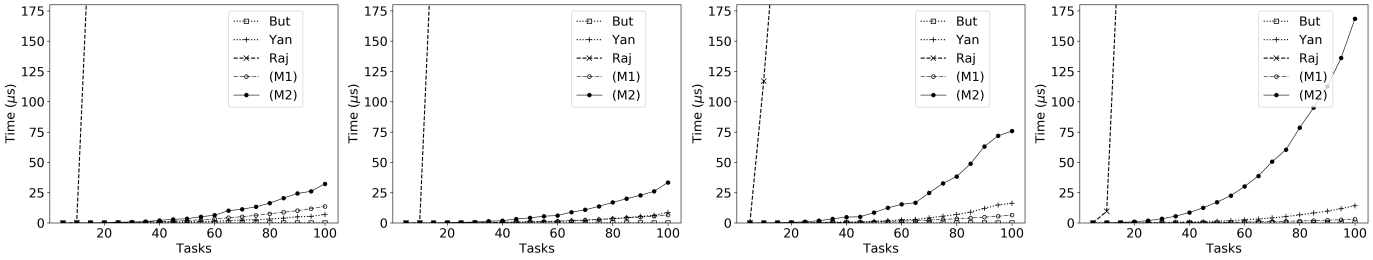
5.3 Results

Concerning the *Scalability test*, Fig. 11 reports the graphs comparing the time to generate the solution for the five methods. Among all possible combinations of the configuration parameters we chose the ones corresponding to four different resource contention scenarios: low (k=$[5, 10], M$=$20, d$=$[1, 25], p_{acc}$=0.32), medium (k=$[5, 10], M$=$10, d$=$[25, 50], p_{acc}$=0.54), high (k=$[5, 20], M$=$10, d$=$[25, 50], p_{acc}$=0.70), and very high (k=$[20, 30], M$=$5, d$=$[50, 100], p_{acc}$=0.99). Every point of the graph emphasizes the time needed to compute the blocking time for all the tasks of the considered application. For all four configurations, the method reported by Buttazzo ("But" series) is always the least time consuming—followed at short distance by Yang et al. ("Yan" series)—whereas Rajkumar's ("Raj" series) is the most because it explores the power set of all the longest critical sections in the lower-priority tasks that can contribute to the block. The scalability of our BLP models fall in between Buttazzo's and Rajkumar's methods. Nonetheless, it is important to remember that method (M1) computes the same upper bounds of Rajkumar with significant time saving.

The result of the *Efficacy test* is presented in Table 2, which reports the average value of the solution computed by the five methods for increasing application dimensions (i.e. number of tasks). Each value is averaged over all the tasks of 10 different applications with high resource contention. The bounds derived by Yang et al.'s method were cleared of the worst-case response times (including blocking and interference) derived for higher priority tasks,

1. https://github.com/dloreti/PIP_BTCalculator
2. https://www.ibm.com/products/ilog-cplex-optimization-studio
3. https://github.com/dloreti/pip_blocking/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers



(a) Low resource contention. (b) Medium resource contention. (c) High resource contention. (d) Very high resource contention.

Fig. 11: Comparison of the time to compute the worst-case blocking time in the four relevant resource contention scenarios.

Blocking time value (μs)				
# Tasks	Yang	Buttazzo	Rajkumar = (M1)	(M2)
10	464	322	317	282
20	420	328	327	317
40	320	310	310	306
60	379	329	329	326
80	424	331	331	329
100	542	330	330	327

TABLE 2: Comparison of the value for the worst-case blocking time computed by the five methods.

which are not considered by the other methods. Despite that, Yang et al.'s method provides rather loose bounds. This is indeed expected because the LP model in [13] is devoted to schedulability analysis in multiprocessor scenarios where (I) does not hold. Furthermore, differently from the present work, Yang et al. maximize the flexibility of their approach by allowing conditional statements in the code of the tasks. Hence, condition (III) is also inapplicable in their scenario. Therefore, the method [13] often rules out only the solutions not fulfilling condition (II). From Table 2 we also see that, while the values of Buttazzo, Rajkumar and model (M1) are all extremely similar, model (M2) is able to provide sensibly reduced solution values, especially for small and medium applications.

As regards the *Accuracy test*, Fig. 12 shows the ratio of computed worst-case blocking time which corresponds to an actually possible blocking chain. For Buttazzo, Yang, Rajkumar and model (M1) that percentage decreases significantly with the dimension of the application. This is indeed expected as the higher is the number of tasks, the more probable are long blocking chains in the solution, and as such, the more probable is that Buttazzo, Yang, Rajkumar and model (M1) make mistakes. The BLP model (M2) instead is always able to find a possible blocking chain.

6 Related work

As the majority of real-life systems need to deal with non-trivial synchronization problems at application- or kernel-level, locking protocols are a crucial component in any real-time operating system. Among the several proposed, we focus on PIP because of its significant practical relevance: priority inheritance is indeed supported by the POSIX real-time standard and hence implemented in several real-time kernels such as RTAI, VxWorks and QNX[4].

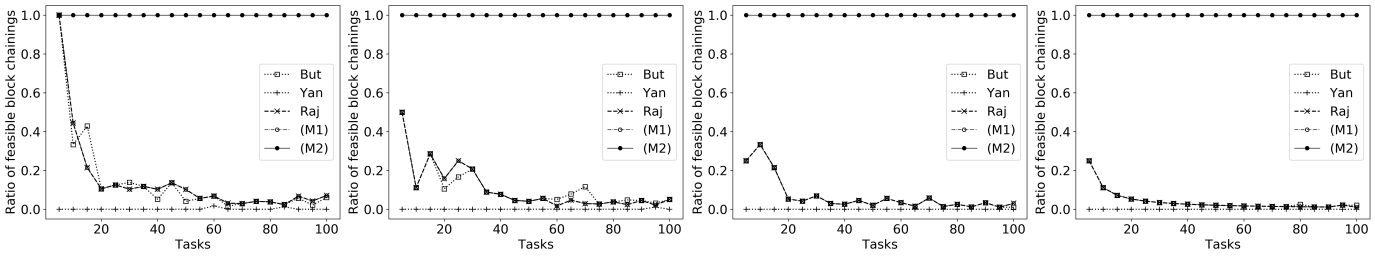
The estimation of the longest time each task can be blocked by another with lower-priority is one of the key points to assess the schedulability of priority-driven deadline-constrained real-time applications (together with the computation of the worst-case execution time and maximum interference). In case of uniprocessor systems, two main methods have been proposed for PIP: the technique exposed by Buttazzo in [14] (which is a natural consequence of the theorems originally exposed along with PIP in [1]), and the technique by Rajkumar in [15].

The importance of PIP is further confirmed by the relevance of works still focusing on it, now that multiprocessor systems have become a wide-spread reality for real-time applications [13], [20], [21], [22]. In particular, the works [21], [22] give a precise definition of *priority inversion blocking* for multiprocessor applications. As our work conversely addresses uniprocessor real-time systems, this term is straightforward to define in our context, and can be reduced to the notions of direct and push-through blocking originally given by the PIP designers (and reported in Section 2). In the paper [20], Easwaran and Andersson propose a response time analysis for PIP in the context of global fixed-priority scheduling (partially grounded on the previous work [23]). In that analysis, the authors assume non-nested resource accesses. We adopted the same simplifying (albeit rather impractical) limitation in the present paper. Other works specifically investigated the blocking caused by nested resource access on multiprocessor systems. In particular, Wieder and Brandenburg [24] analyze the complexity of accurately bounding that blocking time, finding it to be $\mathcal{NP}$-hard. Ward and Anderson [25], [26] propose a "pluggable" protocol to support unrestricted lock nesting while guaranteeing asymptotically optimal priority-inversion blocking bounds. Their work was later extended by Nemitz et al. [27] ensuring that both nested and non-nested requests are processed efficiently.

The employment of operational research concepts is not novel in the field of real-time schedulability analysis. In this regard, the seminal work [28] reports an LP-based comparison of four different protocols (MPCP, FMLP$^+$, DPCP, and DFLP) for partitioned fixed-priority scheduling. In the work [13], Yang et al. extend [28] taking into account other protocols. Interestingly the schedulability analysis of [13] highlights that PIP—together with FMLP—performs best over a wide range of scenarios. Analogously to our method, which extends model (M1) in (M2), the work [29] has recently enriched the LP model of [28] with an additional constraint to reduce the pessimism of the computed upper bounds. Indeed, one of the main advantages of LP-based analysis is its extendibility.

4. https://www.rtai.org - http://www.windriver.com/products/vxworks - http://www.qnx.com

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2020.3029328, IEEE Transactions on Computers



(a) Low resource contention. (b) Medium resource contention. (c) High resource contention. (d) Very high resource contention.

Fig. 12: Comparison of the ratio of possible blocking chains computed by the five algorithms.

Concerning spin locks, a popular alternative to semaphores, another operational research technique, namely Mixed Integer Linear Programming (MILP), has been employed in the work [30], and later extended in [31]. A combination of graph-based technique and ILP is employed in [32] to identify fine-grained blocking bounds for nested spin locks.

Since the aim of this work is just to provide a more efficient way to compute the worst-case blocking time we do not deepen the problems connected to schedulability analysis. We do not model the task's worst-case execution time, nor the interference due to higher priority tasks. Furthermore, differently from all operational research approaches described so far, our model (M2) starts from the strong assumption that tasks are linear. This is essential to formulate condition (III), and thus to improve the accuracy of the computed bounds. Nonetheless, it also represents a restriction that excludes the plain applicability of (M2) to many practical workloads (unless considering one at a time all the possible execution flows). Other relevant models in this field [13], [30], [31] do not suffer this loss of generality. In this regard, Melani et al. [33] recently enriched the model of Fonseca et al. [34] with a more efficient way to compute an upper-bound on the response time of conditional tasks.

7 Discussion

Like traditional methods [1], [15], (M1) assumes only non-nested resource accesses and non-self-suspending tasks, while (M2) refines the initial BLP by restricting attention to the special case of linear tasks, intending them as tasks with no conditional statements in the code, or tasks for which safe abstract execution sequences can be derived, so that the order of the critical sections in their code can be considered fixed. In fact, task's linearity is a strong assumption that is expected to make the applicability of (M2) to many real-life workloads rather difficult. Like other works on this topic, the one by Yang et al. that is considered in the experimental evaluation, does not suffer this restriction. As a result, the accuracy of the bounds of Yang et al. is limited with respect to (M2), but the generality of their analysis is greater. Therefore, we must remark that the advantages of model (M2) come at the price of reduced applicability.

The linear tasks assumption also affects the sustainability of any scheduling algorithm built on the theory of this work. The sustainability of an application is the expectation that, if the system is schedulable under its worst-case specifications, then it should remain schedulable when it "behaves better" than the worst-case [35]. If tasks have conditional statements, their execution presents alternative branches, each of which may include different sequences of critical sections. Seminal works on this topic are [36], [37], [38], [39]. Since (M1) does not consider the execution order of the critical sections, the computed values remain valid also in case of conditional statements. On the other hand, a scheduling strategy using (M2) on non-linear task sets would need to build (and solve) a different BLP model for each possible combination of branches in each task, making the computing effort impractical in most cases.

Another interesting discussion regards the possibility to further refine the blocking time values. For example in case of periodic tasks, both the models proposed in this work assume unknown periods and initial release offsets, and investigate the maximum blocking time over *all task phasings*. In particular, we claim the result of (M2) is such that, assuming all release patterns are possible, there exists a schedule producing the corresponding blocking chain. Nonetheless, if we knew the periods and initial release offsets of periodic tasks, an analysis of these parameters could reveal that the schedule found by Algorithm 1 is actually impossible because the release pattern generating that blocking chain can never occur. In that case, the assumption that all release patterns are possible would no longer hold and the upper bound found by (M2) would not be tight. In general, we cannot preclude the possibility that modeling also other task parameters besides those considered (e.g., release patterns, interference, worst-case execution time, presence/length of independent executions) could induce other constraints and further refine the maximum blocking time.

8 Concluding Remarks

In this paper, we adopt an operational research technique for the computation of the task maximum blocking time under PIP rules on uniprocessor systems. Focusing on non-nested resource accesses and non-self-suspending tasks, we employ BLP to determine the same upper bounds provided by Rajkumar with polynomial—rather then exponential—complexity. Then, restricting our attention to tasks with no conditional statements, we identify an additional constraint set that must be satisfied in order to obtain a possible blocking chain. Although the complexity of the resulting model is no longer polynomial in the worst case, the proposed solution is able to significantly improve the accuracy of the worst-case blocking time.

Research is underway to extend the BLP models in order to cope with nested resource accesses.

9 Acknowledgments

The authors wish to sincerely thank the anonymous Referees for their valuable comments, which have helped to improve the soundness and readability of the paper.

References

[1] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, 1990.
[2] D. Cornhill and L. Sha, "Priority inversion in Ada," *Ada Lett.*, vol. VII, no. 7, pp. 30–32, Nov. 1987.
[3] 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, 1973.
[4] J. Y. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Perform. Eval.*, vol. 2, no. 4, pp. 237–250, 1982.
[5] T. P. Baker, "Stack-based scheduling of realtime processes," *Real-time Syst.*, vol. 3, no. 1, pp. 67–99, 1991.
[6] V. Yodaiken, "Against priority inheritance," Finite State Machine Labs, Tech. Rep., Sep. 2004.
[7] M. Bertogna and S. Baruah, "Limited preemption edf scheduling of sporadic task systems," *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 579–591, Nov. 2010.
[8] D. Zöbel, D. Polock, and A. van Arkel, "Testing for the conformance of real-time protocols implemented by operating systems," *Electr. Notes Theor. Comput. Sci.*, vol. 133, pp. 315–332, 2005.
[9] A. Wellings, A. Burns, O. M. dos Santos, and B. Brosgol, "Integrating priority inheritance algorithms in the real-time specification for java," in *10th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC'07)*, Santorini, Greece, 2007, pp. 115–123.
[10] I. Bicchierai, G. Bucci, L. Carnevali, and E. Vicario, "Combining UML-MARTE and preemptive time Petri nets:an industrial case study," *IEEE Trans. Ind. Informat.*, pp. 1806–1818, Nov. 2013.
[11] G. Racciu and P. Mantegazza, *RTAI 3.4 Users Manual, rev 0.3*, Oct. 2006. [Online]. Available: \url{https://www.rtai.org}
[12] J. Lee and H. Kim, "Implementing priority inheritance semaphore on uC/OS real-time kernel," in *Proc. IEEE Workshop Softw. Technol. for Future Embedded Syst.*, Hokkaido, Japan, May 2003, pp. 83–86.
[13] M. Yang, A. Wieder, and B. Brandenburg, "Global real-time semaphore protocols: A survey, unified analysis,and comparison," in *IEEE Real-Time Syst. Symp.*, San Antonio,TX, 2015, pp. 1–12.
[14] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd Edition*, ser. Real-Time Syst. Series. Springer, 2011, vol. 24.
[15] R. Rajkumar, *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Norwell, MA, USA: Kluwer, 1991.
[16] G. B. Dantzig, *Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation*. New York, NY, USA: Wiley, 1951, ch. XXI.
[17] A. Schrijver, *Theory of linear and integer programming*. Wiley, 1999.
[18] I. Heller and C. B. Tompkins, *An extension of a theorem of Dantzig's*. NJ, USA: Princeton Univ. Press, 1956, ch. 14, pp. 247–254.
[19] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–396, 1984.
[20] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *30th IEEE Real-Time Syst. Symp.*, Washington, DC, USA, 2009, pp. 377–386.
[21] B. Brandenburg and J. Anderson, "Optimality results for multiprocessor real-time locking," in *31st IEEE Real-Time Syst. Symp.*, San Diego, CA, USA, 2010, pp. 49–60.
[22] B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, Univ. of North Carolina at Chapel Hill, NC, USA, 2011.
[23] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *28th IEEE Real-Time Syst. Symp.*, Tucson, AZ, USA, 2007, pp. 149–160.
[24] A. Wieder and B. Brandenburg, "On the complexity of worst-case blocking analysis of nested critical sections," in *IEEE Real-Time Syst. Symp.*, Rome, Italy, 2014, pp. 106–117.
[25] B. Ward and J. Anderson, "Supporting nested locking in multiprocessor real-time systems," in *24th Euromicro Conf. on Real-Time Syst.*, Pisa, Italy, 2012, pp. 223–232.
[26] ——, "Fine-grained multiprocessor real-time locking with improved blocking," in *Proc. of the 21st Int. Conf. Real-Time Netw. and Syst.*, New York, NY, USA, 2013, pp. 67–76.
[27] C. Nemitz, T. Amert, and J. Anderson, "Real-time multiprocessor locks with nesting: optimizing the common case," *Real-time Syst.*, vol. 55, no. 2, pp. 296–348, 2019.
[28] B. Brandenburg, "Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling," in *19th IEEE Real-Time Embedded Technol. Appl. Symp.*, Philadelphia,PA, 2013, pp. 141–152.
[29] Z. Ma, R. Kurachi, G. Zeng, and H. Takada, "Further analysis with linear programming on blocking time bounds for partitioned fixed priority multiprocessor scheduling," *J.Inf.Process.*, vol. 26, pp. 540–548, 2018.
[30] A. Wieder and B. Brandenburg, "On spin locks in AUTOSAR: blocking analysis of fifo, unordered, and priority-ordered spin locks," in *34th IEEE Real-Time Syst. Symp.*, Vancouver, Canada, 2013, pp. 45–56.
[31] A. Biondi and B. Brandenburg, "Lightweight real-time synchronization under P-EDF on symmetric and asymmetric multiprocessors," in *28th Euromicro Conf. Real-Time Syst.*, Toulouse, France, 2016, pp. 39–49.
[32] A. Biondi, B. Brandenburg, and A. Wieder, "A blocking bound for nested FIFO spin locks," in *IEEE Real-Time Syst. Symp.*, Porto, Portugal, 2016, pp. 291–302.
[33] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *27th Euromicro Conf. Real-Time Syst.*, Lund, Sweden, 2015, pp. 211–221.
[34] J. C. Fonseca, V. Nélis, G. Raravi, and L. M. Pinho, "A multi-DAG model for real-time parallel applications with conditional execution," in *Proc, 30th Annu. ACM Symp. Appl. Comput.*, Salamanca, Spain, 2015, pp. 1925–1932.
[35] A. Burns and S. K. Baruah, "Sustainability in real-time scheduling," *J. Comput. Sci. Eng.*, vol. 2, no. 1, pp. 74–97, 2008.
[36] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *31st IEEE Real-Time Syst. Symp.*, San Diego, CA, USA, 2010, pp. 259–268.
[37] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-time Syst.*, vol. 49, no. 4, pp. 404–435, 2013.
[38] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *33rd IEEE Real-Time Syst. Symp.*, San Juan, Puerto Rico, 2012, pp. 63–72.
[39] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of dags," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, 2014.

Eugenio Faldella is full professor at the Department of Computer Science and Engineering of the University of Bologna since 1990, where he teaches Digital Systems and Real-Time Systems. He is author of several publications in prestigious conferences and journals. At present, his research activity is mainly concerned with techniques for hardware-software co-design of real-time embedded systems for industrial automation applications.

Daniela Loreti is post-doc and assistant professor of Operating Systems at Department of Computer Science and Engineering, University of Bologna. She received her Ph.D. in Computer Science in 2016. Her research focuses on distributed systems for big data management and stream processing as well as parallel paradigms for high performance computing. She is also interested in the parallelization of artificial intelligence techniques in the fields of Machine Learning, Process Mining and expert systems.