Bounding the number of self-blocking occurrences of SIRAP*

Moris Behnam, Thomas Nolte
Mälardalen Real-Time Research Centre
P.O. Box 883, SE-721 23 Västerås, Sweden
moris.behnam@mdh.se

Reinder J. Bril
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5612 AZ Eindhoven
The Netherlands

Abstract

This paper presents a new schedulability analysis for hierarchically scheduled real-time systems executing on a single processor using SIRAP: a synchronization protocol for inter subsystem task synchronization. We show that it is possible to bound the number of self-blocking occurrences that should be taken into consideration in the schedulability analysis of subsystems. Correspondingly, we present two novel schedulability analysis approaches with proof of correctness for SIRAP. An evaluation suggests that this new schedulability analysis can decrease the analytical subsystem utilization significantly.

1 Introduction

The amount of functionality realized by software in modern embedded systems has steadily increased over the years. More and more software functions have to be developed, implemented and integrated on a common shared hardware architecture. This often results in very complex software systems, where the functions both are dependent on each other for proper operation, and are interfering with each other in terms of, e.g., resource usage and temporal performance.

To remedy this problem inherent in hosting a large number of software functions on the same hardware, research on platform virtualization has received an increased interest. Looking at real-time systems, research has focused on partitioned scheduling techniques for single processor architectures, which includes hierarchical scheduling where the CPU is hierarchically shared and scheduled among software partitions that can be allocated to the system functions. Hierarchical scheduling can be represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., tasks), and CPU resources are allocated from a parent node to its children nodes. Hence, using hierarchical scheduling techniques, a system can be decomposed into well-defined parts called subsystems, each of which receives a dedicated CPU-budget for execution. These subsystems may contain tasks and/or other subsystems that are scheduled by a so-called subsystem internal scheduler. Tasks within a subsystem can be allowed to synchronize on logical resources (for example a data structure, a memory map of a peripheral device, etc.) requiring mutually exclusive access by the usage of traditional synchronization protocols such as, e.g., the stack resource policy (SRP) [1]. More recent research has been conducted towards allowing tasks to synchronize on logical resources requiring mutual exclusion across subsystem boundaries, i.e., a task resident in one subsystem shall be allowed to get exclusive access to a logical resource shared with tasks from other subsystems (global shared resource). To prevent excessive blocking of subsystems due to budget depletion during global shared resource access, advanced protocols are needed.

One such synchronization protocol for hierarchically scheduled real-time systems executing on a single processor is the subsystem integration and resource allocation policy (SIRAP) [3], which prevents budget depletion during global resource access. SIRAP has been developed with a particular focus of simplifying parallel development of subsystems that require mutually exclusive access to global shared resources. However, a challenge with hierarchical scheduling is the complexity of performing (or formulating) a tight (preferably exact) analysis of the system behavior. Schedulability analysis typically relies on some simplified assumptions and when the system under analysis is complex, the negative effect of these simplifying assumptions can be significant.

In this paper we look carefully at SIRAP’s exact behavior and we identify sources of pessimism in its original local schedulability analysis, i.e. the analysis of the schedulability of tasks of a subsystem. By bounding the number of self-blocking occurrences† that are taken into consideration

---

*The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

† A simpler version of bounding self-blocking was presented in [9]. That paper assumes the same maximum self-blocking at every budget supply, which in our case may make the results more pessimistic than the original analysis of SIRAP. In this paper, we consider the maximum possible self-blocking that may occur at each budget supply.
in the analysis, we develop two new and tighter schedulability analysis approaches for SIRAP assuming fixed-priority pre-emptive scheduling (FPPS). We present proofs of correctness for the two approaches, and an evaluation shows that they can decrease the analytical subsystem utilization. In addition, the evaluation shows that neither approach is always better than the other. The efficiency of these new approaches is shown to be correlated with the nature of the system and in particular the number of accesses made to logical shared resources.

The outline of this paper is as follows: Section 2 outlines related work. In Section 3 we present our system model and background. Section 4 outlines the SIRAP protocol followed by an example motivating the development of a new schedulability analysis in Section 5. Section 6 presents our new analysis, which is evaluated in Section 7. Finally, Section 8 concludes the paper.

2 Related work

Over the years, there has been a growing attention to hierarchical scheduling of real-time systems. Deng and Liu [6] proposed a two-level Hierarchical Scheduling Framework (HSF) for open systems, where subsystems may be developed and validated independently. Kuo and Li [10] presented schedulability analysis techniques for such an HSF assuming a FPPS system level scheduler. Mok et al. [7, 13] proposed the bounded-delay virtual processor model to achieve a clean separation between applications in a multi-level HSF. In addition, Shin and Lee [14] introduced the periodic resource model (to characterize the periodic CPU allocation behavior), and many studies have been proposed on schedulability analysis with this model under FPPS [11, 4] and under Earliest Deadline First (EDF) scheduling [14, 16]. However, a common assumption shared by all above studies is that tasks are independent.

Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [5], BROE [8], and SIRAP [3]. Unlike SIRAP, HSRP does not support subsystem level (local) schedulability analysis of subsystems, and the system level schedulability analysis presented for BROE is limited to EDF and can not be generalized to include other scheduling policies.

3 System model and background

We consider a two-level HSF using FPPS at both the system as well as the subsystem level\(^2\), and the system is executed on a single processor.

**System model** A system contains a set \( R \) of \( M \) global logical resources \( R_1, R_2, \ldots, R_M \), a set \( S \) of \( N \) subsystems \( S_1, S_2, \ldots, S_N \), and a set \( B \) of \( N \) budgets for which we assume a periodic resource model [14]. Each subsystem \( S_s \) has a dedicated budget associated to it. In the remainder of the paper, we leave budgets implicit, i.e. the timing characteristics of budgets are taken care of in the description of subsystems. Subsystems are scheduled by means of FPPS and have fixed, unique priorities. For notational convenience, we assume that subsystems are indexed in priority order, i.e. \( S_1 \) has highest and \( S_N \) has lowest priority.

**Subsystem model** A subsystem \( S_s \) contains a set \( \mathcal{T}_s \) of \( n_s \) tasks \( \tau_1, \tau_2, \ldots, \tau_{n_s} \), with fixed, unique priorities that are scheduled by means of FPPS. For notational convenience, we assume that tasks are indexed in priority order, i.e. \( \tau_1 \) has highest and \( \tau_{n_s} \) has lowest priority. The set \( \mathcal{R}_s \) denotes the subset of global logical resources accessed by \( S_s \). The maximum time that a task of \( S_s \) may lock a resource \( R \in \mathcal{R}_s \) is denoted by \( X_{sk} \). This maximum resource locking time \( X_{sk} \) includes the critical section execution time of the task that is accessing the global shared resource \( R \) and the maximum interference from higher priority tasks, within the same subsystem, that will not be blocked by the global shared resource \( R_k \). The timing characteristics of \( S_s \) are specified by means of a subsystem timing interface \( S_s(P_s, Q_{sk}, \mathcal{X}_s) \), where \( P_s \) denotes the (budget) period, \( Q_{sk} \) the budget that \( S_s \) will receive every subsystem period \( P_s \), and \( \mathcal{X}_s \) the set of maximum resource locking times \( \mathcal{X}_s = \{ X_{sk} \} | \forall R \in \mathcal{R}_s \).

**Task model** We consider the deadline-constrained sporadic hard real-time task model \( \tau_i(C_i, D_i, \{ c_{ik} \}) \), where \( T_i \) is a minimum inter-arrival time of successive jobs of \( \tau_i \). \( C_i \) is the worst-case execution-time of a job, and \( D_i \) is an arrival-relative deadline (\( 0 < C_i \leq D_i \leq T_i \)) before which the execution of a job must be completed. Each task is allowed to access an arbitrary number of global shared resources (also nested) and the same resource multiple times. The set of global shared resources accessed by \( \tau_i \) is denoted by \( \{ R \} \). The number of times that \( \tau_i \) accesses \( R_k \) is denoted by \( n_{ik} \). The worst-case execution-time of \( \tau_i \) during the \( a_k \)th access to \( R_k \) is denoted by \( c_{ik} \). For each subsystem \( S_s \), and without loss of generality, we assume that the subsystem period is selected such that \( 2P_s \leq T_{s\text{min}} \), where \( T_{s\text{min}} \) is the shortest period of all tasks in \( S_s \). The motivation for this assumption is that it simplifies the evaluation of resource locking time and in addition, allowing a higher \( P_s \) would require more CPU resources [15].

**Shared resources** To access a shared resource \( R_k \), a task must first lock the shared resource, and the task unlock the

\(^2\)Because the improvements only concern schedulability of subsystems, system level scheduling is not important for this paper. We also assume FPPS at the system level scheduler for ease of presentation of the model.

\(^3\)Because we only consider local schedulability analysis, we omit the subscript “s” from the task notation representing the subsystem to which tasks belong.
shared resource when the task no longer needs it. The time during which a task holds a lock is called a critical section. For each logical resource, at any time, only a single task may hold its lock.

SRP is a synchronization protocol proposed to bound the blocking time of higher priority tasks sharing logical resources with other lower priority tasks. SRP can limit the blocking time that a high priority task can face, to the maximum critical section execution time of a lower priority task that shares the same resource with \( \tau_i \). SRP associates a resource priority for each shared resource called resource ceiling which equals to the priority of the highest priority task (i.e. lowest task index) that accesses the shared resource. In addition, and during runtime, SRP uses system ceiling to track the highest resource ceiling (i.e. lowest task index) of all resources that are currently locked. Under SRP, a task \( \tau_j \) can preempt the currently executing task \( \tau_i \) only if \( i < j \) and the priority of \( \tau_j \) is greater than the current value of the system ceiling.

To synchronize access to global shared resources in the context of hierarchical scheduling, SRP is used in both system and subsystem level scheduling and to enable this, SRP’s associated terms resource, system ceiling should be extended as follows:

Resource ceiling: With each global shared resource \( R_k \), two types of resource ceilings are associated: an internal resource ceiling (\( rc_{sk} \)) for local scheduling and an external resource ceiling (\( RX_k \)) for system level scheduling. They are defined as \( rc_{sk} = \min\{i|\forall \tau_i \in T_s \land R_k \in \{R^i\}\} \) and \( RX_k = \min\{s|\forall \tau_i \in T_s \land R_k \in R_s\} \).

System/subsystem ceiling: The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/interal resource ceiling (i.e. highest priority) of a currently locked resource in the system/subsystem.

4 SIRAP

SIRAP prevents depletion of CPU capacity during global resource access through self-blocking of tasks. When a job wants to enter a critical section, it first checks the remaining budget \( Q' \) during the current period. If \( Q' \) is sufficient to complete the critical section, then the job is granted entrance, and otherwise entrance is delayed until the next subsystem budget replenishment, i.e., the job blocks itself. Conforming to SRP, the subsystem ceiling is immediately set to the internal resource ceiling \( rc \) of the resource \( R \) that the job wanted to access, to prevent the execution of tasks with a priority lower than or equal to \( rc \) until the job releases \( R \). The system ceiling is only set to the external resource ceiling \( RX \) of \( R \) when the job is granted entrance.

Figure 1 illustrates an example of a self-blocking occurrence during the execution of subsystem \( S_s \). A job of a task \( \tau_i \in T_s \) tries to lock a global shared resource \( R_k \) at time \( t_2 \). It first determines the remaining subsystem budget \( Q'^s \) (which is equal to \( Q' = Q_s - (Q^1 + Q^2) \), i.e., the subsystem budget left after consuming \( Q^1 \) + \( Q^2 \)). Next, it checks if the remaining budget \( Q'^s \) is greater than or equal to the maximum resource locking time \( (X_{ik})^s \) of the \( a^{th} \) access of the job to \( R_k \), i.e., if \( (Q'^s \geq X_{ik})^s \). In Figure 1, this condition is not satisfied, so \( \tau_i \) blocks itself and is not allowed to execute before the next replenishment period \( t_3 \) in Figure 1) and at the same time, the subsystem ceiling is set to \( rc_{sk} \).

Self-blocking of a task is exclusively taken into account in the local schedulability analysis. To consider the worst-case scenario during self-blocking, we assume that the \( a^{th} \) request of \( \tau_i \) to access a global shared resource \( R_k \) always happens when the remaining budget is less than \( X_{ik} \) by a very small value. Hence, \( X_{ik} \) is the maximum amount of budget that \( \tau_i \) can not use during self-blocking (also called the self-blocking of \( \tau_i \)). The effect of the interference from higher priority subsystems is exclusively taken into account in system level schedulability analysis; see [3] for more details.

![Figure 1. An example illustrating self-blocking.](image)

**Local schedulability analysis** The local schedulability analysis under FPPS is given by [14]:

\[
\forall \tau_i \exists t : 0 < t \leq D_{\tau_i}, \text{ sbf}_{FP}(i, t) \leq \text{sbf}_{s}(t),
\]

(1)

where \( \text{sbf}_{s}(t) \) is the supply bound function that computes the minimum possible CPU supply to \( S_s \) for every time interval length \( t \), and \( \text{rbf}_{FP}(i, t) \) denotes the request bound function of a task \( \tau_i \) which computes the maximum cumulative execution requests that could be generated from the time that \( \tau_i \) is released up to time \( t \). \( \text{sbf}_{s}(t) \) is based on the periodic resource model presented in [14] and is calculated as follows:

\[
\text{sbf}_{s}(t) = \begin{cases} 
  t - (g(t) + 1)(P_s - Q_s) & \text{if } t \leq V^g(t) \\
  (j - 1)Q_s & \text{otherwise,}
\end{cases}
\]

(2)

where \( g(t) = \max \{ [(t - (P_s - Q_s))/P_s], 1 \} \) and \( V^g(t) \) denotes an interval \( [(g(t) + 1)P_s - 2Q_s, (g(t) + 1)P_s - Q_s] \) in which the subsystem \( S_s \) receives budget. Figure 2 shows \( \text{sbf}_{s}(t) \). To guarantee a minimum CPU supply, the worst-case budget provision is considered in Eq. (2) assuming that tasks are released at the same time when the subsystem budget depletes (at time \( t = 0 \) in Figure 2) and the budget

\[4\text{How to determine } X_{ik} \text{ will be explained in the next subsection.} \]
was supplied as early as possible and all following budgets will be supplied as late as possible due to interference from other, higher priority subsystems.

\[ \text{calculateBudget of a subsystem } Q \text{ is given while the } \{X\} \in T \in \{0\}, \{1\} \text{ should satisfy} \]

\[ X \in \{0\}, \{1\} \text{ in Eq. (3).} \]

In this paper, it is assumed to denote a function that calculates this minimum budget \( \text{calculateBudget}(i, t) \) for all \( t \). Eq. (7) can be used to determine \( \tau_i \) at the system level, each subsystem \( S \) will be supplied as late as possible due to interference from higher priority tasks that can preempt the execution of \( \tau_i \) while accessing \( R_j \). Since \( 2P_s \leq T_s^{\text{min}} \), tasks with a priority higher than \( r_{c,s,k} \) can interfere at most once (the proof of Eq. (7) is presented in [2]).

\[ X_{i,k} = c_{i,k} + \sum_{h=1}^{r_{c,s,k} - 1} C_h. \] (7)

The self-blocking of \( \tau_i \), the higher priority tasks and the maximum self-blocking of the lower priority tasks are given in Eqs. (4)-(6). We can re-arrange these equations by moving all self-blocking terms into one equation \( I_S(i, t) \), resulting in corresponding equations \( I_H(i, t) \) and \( I_L(i) \):

\[ I_S(i, t) = \sum_{h=1}^{i-1} \left[ \frac{t}{T_h} \right] \left( \sum_{R_k \in \{R^h\}} \sum_{a=1}^{r_{n,h}} X_{h,ka} \right), \]

\[ I_H(i, t) = \sum_{t < h < i} \left[ \frac{t}{T_h} \right] C_h, \]

\[ I_L(i) = \max \{0, \frac{\max_{t=i}^{\infty} r_{c,s,k} (X_{i,k})}{\max_{R_k \in \{R^h\}, a=1}^{\infty} c_{i,k} + X_{i,k}} \}. \] (6)

Note that we use the outermost \( \max \) in (6) to also define \( I_L(i) \) in those situations where \( \tau_i \) can not be blocked by lower priority tasks. Looking at Eqs. (4)-(6), it is clear that \( \text{calculateBudget}(i, t) \) is a discrete step function that changes its value at certain time points \( \{t = a \times T_h \mid a \text{ is an integer number} \} \). Then for Eq. (1), \( t \) can be selected from a finite set of scheduling points [12].

The term \( X_{j,k,a} \) in these equations represents the self-blocking (resource locking time) of task \( \tau_j \) due to the \( a^{th} \) access to resource \( R_k \). Eq. (7) can be used to determine \( X_{i,k} \), where the sum in the equation represents the interference from higher priority tasks that can preempt the execution of \( \tau_i \) while accessing \( R_j \). Since \( 2P_s \leq T_s^{\text{min}} \), tasks

For the request bound function \( \text{rbf}_{FP}(i, t) \) of a task \( \tau_i \), and to compute the maximum execution request up to time \( t \), it is assumed that (i) \( \tau_i \) and all its higher priority tasks are simultaneously released, (ii) each access to a global shared resource by these tasks will generate a self-blocking, (iii) a task with priority lower than \( \tau_i \) that can cause a maximum blocking has locked a global shared resource just before the release of \( \tau_i \), and (iv) will also cause a self-blocking. \( \text{rbf}_{FP}(i, t) \) is given by [3]:

\[ \text{rbf}_{FP}(i, t) = C_i + I_S(i) + I_H(i, t) + I_L(i), \] (3)

where \( I_S(i) \) is the self-blocking of task \( \tau_i \), \( I_H(i, t) \) is the interference from tasks with a priority higher than that of \( \tau_i \), and \( I_L(i) \) is the interference from tasks with priority lower than that of \( \tau_i \), that access shared resources, i.e.,

\[ I_S(i) = \sum_{R_k \in \{R^h\}} \sum_{a=1}^{r_{n,h}} X_{h,ka}, \]

\[ I_H(i, t) = \sum_{h=1}^{i-1} \left[ \frac{t}{T_h} \right] (C_h + \sum_{R_k \in \{R^h\}} \sum_{a=1}^{r_{n,h}} X_{h,ka}), \]

\[ I_L(i) = \max \{0, \frac{\max_{t=i}^{\infty} r_{c,s,k} (X_{i,k})}{\max_{R_k \in \{R^h\}, a=1}^{\infty} c_{i,k} + X_{i,k}} \}. \] (6)

Subsystem timing interface In this paper, it is assumed that the period \( P_s \) of a subsystem \( S \) is given while the minimum subsystem budget \( Q_s \) should be computed. We use calculateBudget(\( S, P_s \)) to denote a function that calculates this minimum budget \( Q_s \) satisfying Eq. (1). This function is similar to the one presented in [14]. We can determine \( X_{s,k} \) for all \( R_k \in \mathcal{R}_s \) by

\[ X_{s,k} = \max_{\tau_i \in T_s, R_k \in \{R^h\}, a=1}^{\infty} c_{i,k} + X_{i,k}. \] (11)

We define \( X_s \) as the maximum resource locking time among all resources accessed by \( S \), i.e.

\[ X_s = \max_{R_k \in \mathcal{R}_s} (X_{s,k}). \] (12)

Finally, when a task experiences self-blocking during a subsystem period it is guaranteed access to the resource during the next period. To provide this guarantee, the subsystem budget \( Q_s \) should satisfy

\[ Q_s \geq X_s. \] (13)

System level scheduling At the system level, each subsystem \( S \) can be modeled as a simple periodic task. The parameters of such a task are provided by the subsystem timing interface \( S_c(P_s, Q_s, X_s) \), i.e. the task period is \( P_s \), the execution time is \( Q_s \), and the set of critical section execution times when accessing logical shared resources is \( X_s \). To validate the composability of the system under FPPS and SRP, classical schedulability analysis for periodic tasks can be applied; please refer to [3] for more details.
5 Motivating example

In this section, we will show that the schedulability analysis associated with SIRAP is very pessimistic if multiple resources are accessed by tasks and/or the same resource is accessed multiple times by tasks. We will show this by means of the following example.

<table>
<thead>
<tr>
<th>$T$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$R_k$</th>
<th>$\epsilon_{R_k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>6</td>
<td>100</td>
<td>$R_1$, $R_1$, $R_2$</td>
<td>1, 2, 2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>150</td>
<td>$R_1$, $R_2$</td>
<td>2, 1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>3</td>
<td>500</td>
<td>$R_2$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1. Example task set parameters

Example: Consider a subsystem $S_s$ that has three tasks as shown in Table 1. Note that task $\tau_1$ accesses $R_1$ twice, i.e., $r_{n_1,1} = 2$. Let the subsystem period be equal to $P_s = 50$. Using the original SIRAP analysis, we find a subsystem budget $Q_s = 23.5$. Task $\tau_2$ requires this budget in order to guarantee its schedulability, i.e. the set of points of time $t$ used to determine schedulability of $\tau_2$ is $\{100, 150\}$ and at time $t = 150$, $rbfp(2, 150) = sbf_s(150) = 47$.

To evaluate $rbfp(i, t)$ for $\tau_1$, the SIRAP analysis assumes that the maximum number of self-blocking instances will occur for $\tau_1$ and all its lower and higher priority tasks. Considering our example, $I_s(2, 150)$ contains a total of 9 self-blocking instances; 6 self-blocking instances for task $\tau_1$ ($X_1, 1, 1, 1 = 1, X_1, 1, 1, 1 = 2, X_1, 1, 1, 2 = 2, X_1, 1, 2, 1 = 2, X_1, 2, 1, 1 = 2$), 2 for task $\tau_2$ ($X_2, 2, 1, 1 = 2, X_2, 2, 2, 1 = 1$), and 1 for task $\tau_3$ ($X_3, 2, 2, 1 = 1$) (see Eq. (8)), resulting in $I_s(2, 150) = 14$. Because $P_s = 50$ and $Q_s = 23.5$, we know that $\tau_2$ needs at least two and at most three activations of the subsystem for its completion. As no self-blocking instance can occur during a subsystem period in which a task completes its execution, the analysis should incorporate at most 2 self-blocking instances for $\tau_2$. This means that the SIRAP analysis adds 7 unnecessary self-blocking instances when calculating $rbfp(i, t)$, which makes the analysis pessimistic. If 2 self-blocking instances are considered and the two largest self-blocking values that may happen are selected (e.g. $X_1, 1, 2, 1 = 2, X_1, 2, 1, 1 = 2$), then $I_s(2, 150) = 4$ and a subsystem budget of $Q_s = 18.5$ suffices. For this subsystem budget, we once again find at most 2 self-blocking instances. In other words, the required subsystem utilization ($Q_s/P_s$) can be decreased by 27% compared with the original SIRAP analysis. This improvement can be achieved assuming that at most one self-blocking instance needs to be considered every budget period (the budget period is a time interval from the time when the budget replenished up to the next following budget replenishment time instant, for example in Figure 1, it starts at $t_1$ and ends at $t_1 + P_s = t_3$).

6 Improved SIRAP analysis

In the previous section, we have shown that the original analysis of SIRAP can be very pessimistic. If we assume that at most one self-blocking instance needs to be considered during every budget period then a significant improvement in the CPU resource usage can be achieved. Although multiple self-blocking instances can occur during one budget period, it is sufficient to consider at most one.

Lemma 1 At most one self-blocking occurrence, i.e. the largest possible, needs to be considered during each subsystem period $P_s$ of $S_s$ for the schedulability of $\tau_i \in T_s$.

Proof Upon self-blocking of an arbitrary task $\tau_j$ of $S_s$ due to an attempt to access $R_k$, the subsystem ceiling of $S_s$ becomes at most equal to the internal resource ceiling $rc_{sk}$. Once this situation has been established, the subsystem ceiling may decrease (due to activations of and subsequent attempts to access resources by tasks with a priority higher than $rc_{sk}$, i.e. the task index is lower than $rc_{sk}$), but will not increase during the current subsystem period. A task $\tau_i$ experiences blocking/interference due to self-blocking of an arbitrary task $\tau_j$ trying to access $R_k$ if and only if the internal resource ceiling $rc_{sk}$ of $R_k$ is at most equal to $i$ (i.e. $rc_{sk} \leq i$). Hence, as soon as $\tau_i$ experiences blocking/interference due to self-blocking, that situation will last for the remainder of the budget period, and additional occurrences of self-blocking can at most overlap with earlier occurrences. It is therefore sufficient to consider at most one self-blocking instance, i.e. the largest possible, per budget period.

6.1 Problem formulation

Lemma 1 proves that at each subsystem period, one maximum self-blocking can be considered in the schedulability analysis of SIRAP. That means the number of effective self-blocking occurrences at time instant $t$, that should be considered in the schedulability analysis, depends on the maximum number of subsystem periods that have been repeated up to time instant $t$. In other words, the number of self-blocking occurrences is bounded by the number of overlapping budget periods. However, the equations used for the local schedulability analysis Eqs. (2) and (3) can not express this bound on self-blocking because:

- The $sbf_s(t)$ of Eq. (2) is based on the subsystem budget and period, but is agnostic of the behavior of the subsystem internal tasks that cause self-blocking, and therefore also agnostic of self-blocking.
- The $rbfp(i, t)$ of Eq. (3) contains the self-blocking terms, but does not consider the subsystem period.

We propose two different analysis approaches in order to address the bound on self-blocking: the first approach is
based on using this knowledge (bound on the self-blocking) in the calculation of \( sbf_{FP}(i, t) \) and the second approach is based on using it in the calculation of \( sbf_s(t) \).

As long as we are still in the subsystem level development stage, we have all internal information including global shared resources, which task(s) access them and the critical section execution time of each resource access; information that is required to optimize the local schedulability analysis in order to decrease the CPU resources required to be reserved for the subsystem.

Before presenting the two analysis approaches that may decrease the required subsystem utilization compared to the original SIRAP approach, we will describe a self-blocking multi-set that will be used by these new approaches.

### 6.2 Self-blocking set

For each task \( \tau_i \), we define a multi-set \( G_i(t) \) containing the values of all self-blocking instances that a task \( \tau_i \) may experience in an interval of length \( t \) according to \( I^*_S(i, t) \); see Eq. (8). Similar to Eq. (8), the elements in \( G_i(t) \) are evaluated based on the assumption that task \( \tau_i \) and all its higher priority tasks are simultaneously released.

Note that \( G_i(t) \) includes all \( X_{jka} \) that may contribute to the self-blocking. Depending on the time \( t \), a number of elements will be taken from this list and, to consider the worst-case scenario, the value of these elements should be the highest in the multi-set. To provide this, we define a sequence \( G^\text{sort}_i(t) \) that contains all elements of \( G_i(t) \) sorted in a non-increasing order, i.e. \( G^\text{sort}_i(t) = \text{sort}(G_i(t)) \). Considering the example presented in Section 5, the sequence \( G^\text{sort}_2(150) \) for \( \tau_2 \) and \( t = 150 \) equals \( X_{1,1,2}, X_{1,1,2}, X_{1,2,1}, X_{1,2,1}, X_{1,1,1}, X_{1,1,1}, X_{2,2,1}, X_{3,2,1} \).

### 6.3 Analysis based on changing \( rbf \)

In this section we will present the first approach called IRBF that improves the local schedulability analysis of SIRAP based on changing \( rbf_{FP}(i, t) \). Note that as long as we are not changing the supply bound function \( sbf_s(t) \), Eq. (2) and the associated assumption concerning worst-case budget provision can still be used. As we explained before, the number of self-blocking occurrences is bounded by the number of overlapping subsystem budget periods. The following lemma presents an upper bound on the number of self-blocking occurrences in an interval of length \( t \).

**Lemma 2** Given a subsystem \( S_s \) and assuming the worst-case budget provision, an upper bound on the number of self-blocking occurrences \( z(t) \) in an interval of length \( t \) is given by

\[
z(t) = \left\lceil \frac{t}{P_s} \right\rceil.
\]

**Proof** Note that \( z(t) \) represents an upper bound on the number of subsystem periods that are entirely contained in an interval of length \( t \). In addition, the \( sbf_s(t) \) calculation in Eq. (2) is based on the worst-case budget provision, i.e. task \( \tau_i \) under consideration is released at a budget depletion when the budget was supplied as early as possible and all following budget supplies will be at least as possible. From the release time of \( \tau_i \), if two self-blocking occurrences happen, at least one \( Q_s \) must be fully supplied and another \( Q_s \) (at least partially). Hence, \( t > P_s - (Q_s - X^1) + P_s = 2P_s - (Q_s - X^1) \) for \( 0 < X^1 \leq Q_s \leq P_s \), where \( X^1 \) is a (first) self-blocking; see Figure 3(a). This assumption is satisfied for \( t > P_s \). Similarly, we can prove that for \( b \) self-blocking occurrences, \( t > b \times P_s \).

Note that Eq. (14) accounts for a first self-blocking occurrence just after the release of \( \tau_i \), i.e. for \( t \) an infinitesimal larger than zero. For SIRAP, this release of \( \tau_i \) is assumed at a worst-case budget provision, e.g. at time \( t = 0 \) in Figure 2. At the end of the first budget supply (at time \( t = 2P_s - Q_s \) in Figure 2), where one complete self-blocking can occur, Eq. (14) has accounted for a second self-blocking, as shown in Figure 3(b). In general, at any time \( t \), the number of self-blocking occurrences evaluated using Eq. (14) will be one larger than the number of self-blocking occurrences that can happen in an interval with a worst-case budget provision. This guarantees that we can safely assume that the worst-case situation for the original analysis for SIRAP also applies for IRBF.

![Figure 3](image-url)

**Figure 3.** A subsystem execution with self-blocking.

After evaluating \( z(t) \), it is possible to calculate the self-blocking on task \( \tau_i \) from all tasks, i.e. lower priority tasks, higher priority tasks and \( \tau_i \) itself. Eq. (8), that computes the self-blocking on \( \tau_i \), can now be replaced by

\[
I^*_S(i, t) = \sum_{j=1}^{z(t)} G^\text{sort}_i(t)[j] = \left\lceil \frac{t}{P_s} \right\rceil.
\]
Note that if \( z(t) \) is larger than the number of elements in the set \( G_i^{\text{sort}}(t) \), then the values of the extra elements are equal to zero, e.g. if \( G_i^{\text{sort}}(t^*) \) has \( k_i \) elements (i.e. the number of all possible self-blocking occurrences that may block \( \tau_i \) in an interval of length \( t^* \)), then \( G_i^{\text{sort}}(t^*)[j] = 0 \) for all \( j > k_i \).

**Correctness of the analysis**  The following lemma proves the correctness of the IRBF approach.

**Lemma 3** Using the IRBF approach, \( \text{rbf}_{FP}(i, t) \) given by
\[
\text{rbf}_{FP}(i, t) = C_i + I^0_S(i, t) + I^1_H(i, t) + I^2_L(i)
\]
computes an upper bound on the maximum cumulative execution requests that could be generated from the time that \( \tau_i \) is released up to time \( t \).

**Proof** We have to prove that Eq. (15) computes an upper bound on the maximum request generated from self-blocking. As explained earlier, during a self-blocking, all tasks with priority less than or equal to the resource ceiling of the resource that caused the self-blocking, are not allowed to execute until the next budget activation. To consume the remaining budget, an idle task is executing if there are no tasks, with priority higher than the subsystem ceiling, released during the self-blocking. To add the effect of self-blocking on the schedulability analysis of \( \tau_i \), the execution time of the idle task during the self-blocking can be modeled as an interference from a higher priority task on \( \tau_i \). The maximum number of times that the idle task executes up to any time \( t \) is equal to the number of self-blocking occurrences during the same time interval and an upper bound is given by \( z(t) \). Furthermore, selecting the first \( z(t) \) elements from the \( G_i^{\text{sort}}(t) \) gives the maximum execution times of the idle task.

We also have to prove that a simultaneous release of \( \tau_i \) and all its higher priority tasks at a worst-case budget provision will actually result in an upper bound for \( I^0_S(i, t) \). To this end, we show that neither the actual number of self-blocking terms nor their values in an interval of length \( t^* \) starting at the release of \( \tau_i \) can become larger when a higher priority task \( \tau_h \) is either released before or after \( \tau_i \). We first observe that the number of self-blocking occurrences \( z(t^*) \) in an interval of length \( t^* \) is independent of the release of \( \tau_h \) relative to \( \tau_i \). Next, we consider the values for self-blocking.

A later release of \( \tau_h \) will either keep the same (worst-case) value for the self-blocking during \( t^* \) or reduce it (and may in addition cause a decrease of the interference in Eq. (5)). Releasing \( \tau_h \) earlier than \( \tau_i \) makes \( \tau_h \) receiving some budget and at the same time a self-blocking happens, before the release of \( \tau_i \) (remember, \( \tau_i \) is released at a worst-case budget provision). Furthermore, and at the end of time interval \( t^* \), new self-blocking caused by earlier releasing of \( \tau_h \), may be added to the self-blocking set \( (G_i(t^*)) \). However, since an earlier self-blocking happens (before the release of \( \tau_i \) this earlier self-blocking removes the effect of the additional self-blocking on \( G_i(t^*) \). For instance, an earlier release of \( \tau_h \) may \( i \) keep the self-blocking the same (if the additional self-blocking \( X^0 \) resulting from the earlier release of \( \tau_h \) during the last budget period is less than the one that was considered assuming all tasks are released simultaneously \( X^0 = X^j \); see Figure 4(b) or \( ii \) add or replace a self-blocking term in the last complete budget period contained in \( t^* \). For both cases of (ii), the new term for the additional activation of \( \tau_h \) will add the removal of a similar term for \( \tau_h \) at the earlier release of \( \tau_h \), effectively rotating the sequence of blocking terms as illustrated in Figure 4(c)-(d). Rotating the terms does not change the sum of the blocking terms, however, and the amount of self-blocking in \( t^* \) therefore remains the same.

**Figure 4.** Critical instant for two tasks.

**Example** Returning to our example, we find \( z(150) = 3 \) and that makes \( I^0_S(i, t) = 6 \) according to Eq. (15), we find a minimum subsystem budget \( Q_s = 19.5 \), which is better than the one obtained using the original SIRAP equations. The analysis is still pessimistic, however, because \( z(t) \) is an upper bound on the number of self-blocking occurrences rather than an exact number and in addition, \( t \) is selected from the schedulability test points set of \( \tau_2 \) rather than the Worst Case Response Time (WCRT) of the task. Note that the WCRT of \( \tau_2 \) is less than 150 which indicates remaining
Remark Based on the new analysis presented in this section, the following lemma proves that the results obtained from the analysis based on IRBF are always better than, or the same as, the original SIRAP approach.

Lemma 4 The minimum subsystem budget obtained using IRBF will be always less than or equal to the subsystem budget obtained using the original SIRAP approach.

Proof When evaluating \( \text{sbf}(i, t) \) for a task \( \tau_i \), the only difference between the original SIRAP approach and the analysis of IRBF is the calculation of self-blocking \( I^s_g(i, t) \) in Eq. (8) and \( I^s_g(i, t) \) in Eq. (15). To prove the correctness of this lemma we have to prove that \( I^s_g(i, t) \leq I^s_g(i, t) \). Because \( G_i^{\text{sort}}(t) \) is the sorted multi-set \( G_i(t) \) of values contained in \( I^s_g(i, t) \), the sum of all values contained in \( G_i^{\text{sort}}(t) \) is equal to \( I^s_g(i, t) \), i.e. when \( k_i \) is equal to the number of non-zero elements in \( G_i^{\text{sort}}(t) \), we have \( I^s_g(i, t) = \sum_{j=1}^{k_i} G_i^{\text{sort}}(t)[j] \). Since \( I^s_g(i, t) = \sum_{j=1}^{z(t)} G_i^{\text{sort}}(t)[j] \), we get \( I^s_g(i, t) < I^s_g(i, t) \) for \( z(t) < k_i \) and \( I^s_g(i, t) = I^s_g(i, t) \) for \( z(t) \geq k_i \). because \( G_i^{\text{sort}}(t)[j] = 0 \) for all \( j > k_i \). \( \Box \)

6.4 Analysis based on changing \( \text{sbf} \)

The effect of self-blocking in SIRAP has historically been considered in the request bound function (as shown in Sections 4 and 6.3). Self-blocking is modeled as additional execution time that is added to \( \text{rbf}_p(i, t) \) when applying the analysis for \( \tau_i \). In this section we use a different approach, called ISBF, based on considering the effect of self-blocking in the supply bound function. The main idea is to model self-blocking as unavailable budget, which means that the budget that can be delivered to the subsystem will be decreased by the amount of self-blocking. Moving the effect of self-blocking from \( \text{rbf} \) to \( \text{sbf} \) has the potential to improve the results, in terms of requiring less CPU resources, compared to the original SIRAP analysis.

Figure 5 shows the supply bound function using the new approach, where \( Q_s \) is guaranteed every period \( P_s \), however, only a part (denoted \( Q^2 \)) from the \( j^{th} \) subsystem budget is provided to the subsystem after the release of \( \tau_i \), while the other part (denoted \( X^3 \)) of the \( j^{th} \) subsystem budget is considered as unavailable budget which represents the self-blocking time.

A new supply bound function should be considered taking into account the effect of self-blocking on the worst-case budget provision. In general, the worst-case budget provision happens when \( \tau_i \) is released at the same time when the subsystem budget becomes unavailable and the budget was supplied at the beginning of the budget period and all later budget will be supplied as late as possible. Note that self-blocking occurs at the end of a subsystem period, which means that unavailable budget is positioned at the end (last part) of the subsystem budget. The earliest time that the budget becomes unavailable relative to the start of a budget period is therefore \( Q_s - X^0 \). Conversely, the latest time that the budget will become available after a replenishment (starting time of the next budget period), is \( P_s - Q_s \). Hence, the longest time that a subsystem may not get any budget (called Blackout Duration \( BD \) is \( \sum (P_s - Q_s + X^0) \). Finally, each task has a specific set of self-blocking occurrences, which means that each task will have its own supply bound function. The new supply bound function \( \text{sbf}_s(i, t) \) for \( \tau_i \) is given by

\[
\text{sbf}_s(i, t) = \begin{cases} 
  t - (g(t) + 1)P_s + Q^0 + Q_s + \text{Sum}(g(t) - 1) & \text{if } t \in V^{\text{g(t)}}_s \ 
  \text{Sum}(g(t) - 1) & \text{if } t \in W^{\text{g(t)}} 
\end{cases}
\]

where

\[
\text{Sum}(t) = \sum_{j=1}^{t} Q_j
\]

and \( Q^0 = Q_s - X^3, V^{g(t)}_s \) denotes an interval \([g(t) + 1)P_s - Q^0 - Q_s, (g(t) + 1)P_s - Q^0 - X^{g(t)} \] when the subsystem gets budget, and \( W^{g(t)} \) denotes an interval \([g(t) + 1)P_s - Q^0 - X^{g(t)} \) during the \( g(t) \) self-blocking. The intuition for \( g(t) \) in Eq. (17) is the number of periods of the periodic model that can actually provide budget in an interval of length \( t \), as shown in Figure 5. To explain Eq. (17) let us consider the case for \( g(t) = 3 \). If \( t \in V^3 \), i.e. during the 3rd self-blocking time interval of length \( X^3 \), then the amount of budget supplied to the subsystem will by \( Q^1 + Q^2 + \text{Sum}(3) \). If \( t \in V^3 \), then the resource supply will equal to \( Q^1 + Q^2 + \text{Sum}(3) \) plus the value from the linearly increasing region (see Figure 5), otherwise, the budget supply is \( Q^1 + Q^2 + \text{Sum}(3) \).

Since we consider the effect of self-blocking in the supply bound function, we can now remove all self-blocking from \( \text{rbf}_p(i, t) \), i.e. \( I^s_g(i, t) = 0 \) in Eq. (8) and only Eqs. (9) and (10) are used to evaluate \( \text{rbf}_p(i, t) \). Hence, the local schedulability analysis is

\( \forall \tau_i \exists t : 0 < t \leq D_i, \text{rbf}_p(i, t) \leq \text{sbf}_s(i, t) \).

The final step on evaluating \( \text{sbf}_s(i, t^*) \) is to set the values of self-blocking \( X^j \) for \( 0 \leq j \leq g(t) \) such that the supply bound function gives the minimum possible CPU supply for interval length \( t^* \). To achieve this, \( X^j \) is evaluated as follows

\[
X^j = G_i^{\text{sort}}(t^*)[j]
\]

where \( 0 < j \leq g \) and \( X^0 = X^1 \) which is the largest self-blocking.
**Correctness of the analysis** The following lemma proves that setting the self-blocking according to Eq. (21) and \( X^0 = X^1 \) will make the supply bound function giving the the minimum possible CPU supply.

**Lemma 5** \( \text{sbf}_s(i, t) \) will give the minimum possible CPU supply for every interval length \( t \) if Eq. (21) and \( X^0 = X^1 \) are used to set the values of \( X^j \).

**Proof** To proof the lemma, we have to prove that the amount of budget supplied to a subsystem using Eq. (17) is the minimum and also the budget is supplied as late as possible. Using Eq. (21) will set the largest possible values of self-blocking at time \( t \) to \( X^1, X^2, \ldots, X^j \) and that will make the function \( s_{\text{turn}}(i) \) in Eq. (19) return the minimum possible value \( (Q_j^s = Q_s - X^j) \), which in turn will give the minimum \( \text{sbf}_s(i, t) \).

On the other hand, the blackout duration \( \text{BD} \) should be maximized to guarantee the minimum CPU supply. Since \( \text{BD} = 2P_s - 2Q_s + X^0 = 2P_s - Q^0 - Q_s \) (which equals to the starting time of the interval \( V^{(1)} \)), \( \text{BD} \) is maximized if \( X^0 = X^1 = G_i^{\text{sort}}(t^*)[1] \). This setting of \( X^0 \) will also maximize the starting time of the interval \( V^{(j)} \) \( j = 1, \ldots, g(t^*) \) (time interval when new budget is supplied) which delays the budget supply and decreases \( \text{sbf}_s(i, t) \) at any time instant \( t \). Considering the two mentioned factors will guarantee that Eq. (17) gives the minimum possible CPU resource supply.

Note that Eq. (21) uses the set \( G_i^{\text{sort}}(t) \), and the elements of the set are evaluated assuming that \( \tau_i \) and all tasks with priority higher than \( \tau_i \) are released simultaneously. In the previous section, we have shown that this assumption is correct considering the IRBF approach. For ISBF, setting \( X^0 = X^1 = G_i^{\text{sort}}(t^*)[1] \) makes the analysis more pessimistic than the actual execution since the first element in the set \( G_i^{\text{sort}}(t^*)[1] \) can only happen once before or after the release of \( \tau_i \). So the additional self-blocking \( X^0 \) is considered to maximize the time that tasks will not get any CPU budget, as proven in Lemma 5. If \( \tau_i \) or any of its higher priority tasks is released earlier than the beginning of the self-blocking \( X^0 \) then that task will directly get some budget and since we use \( X^1 \) self-blocking after the first budget consumption then \( X^0 \) should be removed (similar scenario is shown in Figure 4(c) but \( \tau_2 \) should be released at the time when self-blocking \( X_1 \) begins). As a result, and similar to the IRBF approach, same elements taken from \( G_i^{\text{sort}}(t^*) \) can at most be rotated if tasks are not released at the same time and that means the supply bound function at time \( t^* \) will not be decreased.

The pessimistic assumption \( X^0 = X^1 = G_i^{\text{sort}}(t^*)[1] \) may affect the results of ISBF and the effect depends on the tasks and the subsystem parameters as shown in the following examples.

**Example** Returning to our example, based on the new supply bound function, we find a minimum budget \( Q_s = 18.5 \), since two instances of self-blocking can happen at \( t = 150 \). This is better than IRBF yielding \( Q_s = 19.5 \) and the original SIRAP where \( Q_s = 23.5 \). Note that assigning \( X^0 = 2 \) did not affect the results of ISBF.

However, it is not always the case that ISBF can give better results than the other approaches, as will be shown in the following example. Suppose a subsystem \( S_a \) with \( P_a = 100 \) and \( n \) tasks. The highest priority task \( \tau_1 \) is the task that requires the highest subsystem budget. \( \tau_1 \) has the following parameters, \( T_1 = 230, C_1 = 29.5 \), the maximum blocking

---

*Figure 5.* New supply bound function \( \text{sbf}_s(i, t) \).
from lower priority tasks that accesses a global shared resource $R_1$ is $B_1 = 6$ and $\tau_1$ accesses $R_1$ two times with critical section execution time $c_{1,1,1} = 1$ and $c_{1,1,2} = 1$. Using ISBF, the minimum subsystem budget is $Q_s = 39.2$ while using the other two approaches then $Q_s = 37.85$.

The reason that ISBF will require more subsystem budget than the other two approaches in the second example is that using ISBF, the maximum blocking $B_1 = 6$ is considered twice, i.e. $X^0 = X^1 = 6$, whereas the other approaches use the actual possible self-blocking $\{6, 1, 1\}$. Because the difference between the largest and the other self-blocking terms is high, ISBF requires a higher budget.

7 Evaluation

In this section, we evaluate the performance of the two presented approaches ISBF and IRBF, in terms of the required subsystem utilization, compared to the original SIRAP approach. Looking at the schedulability analysis of both IRBF and ISBF, the following parameters can directly affect the improvements that both new approaches can achieve:

- The number of global shared resource accesses made by a subsystem (including the number of shared resources and the number of times that each resource is accessed).
- The difference between the subsystem period and its corresponding task periods.
- The length of the critical section execution time, that affects the self-blocking time.

We will explain the effect of the mentioned parameters by means of simulation in the following section.

7.1 Simulation settings

The simulation is performed by applying the two new analysis approaches in addition to the original SIRAP approach on 1000 different randomly generated subsystems where each subsystem consists of 8 tasks. The internal resource ceilings of the globally shared resources are assumed to be equal to the highest task priority in each subsystem (i.e. $c_{sk} = 1$) and we assume $T_i = D_i$ for all tasks. The worst-case critical section execution time of a task $\tau_i$ is set to a value between $0.1C_i$ and $0.25C_i$, the subsystem period $P_s = 100$ and the task set utilization is 25%.

For each simulation study one of the mentioned parameters is changed and a new set of 1000 subsystems is generated (except when changing $P_s$; in that case the same subsystems are used). The task set utilization is divided randomly among the tasks that belong to a subsystem. Task periods are selected within the range of 200 to 1000. The execution time is derived from the desired task utilization. All randomized subsystem parameters are generated following uniform distributions.

7.2 Simulation results

Tables 2-4 show the results of 3 different simulation studies performed to measure the performance of the two new analysis approaches.

In these tables, $U_s^{IRBF} < U_s^{Orig}$ denotes the percentage of subsystems where their subsystem utilization $U_s = Q_s/P_s$ using IRBF is less than the subsystem utilization using the original SIRAP approach, out of 1000 randomly generated subsystems, and “Max I $(U_s^{IRBF}/U_s^{Orig})$” is the maximum improvement that the analysis based on IRBF can achieve compared with the original SIRAP approach, which is computed as $(U_s^{Orig} - U_s^{IRBF})/U_s^{IRBF}$. Finally, “Max D $(U_s^{ISBF}/U_s^{Orig})$” is the maximum degradation in the subsystem utilization as a result of using the analysis based on ISBF compared to the analysis using the original SIRAP approach. As we explained in the previous section, in some cases ISBF may require more CPU resources than the other two approaches.

- **Study 1** is specified having the number of shared resource accesses equal to 2, 4, 8, and 12, critical section execution time $c_{ij,k}$ is $(0.1 - 0.25) \times C_i$ and subsystem period $P_s = 100$. The intention of this study is to show the effect of changing the number of shared resources on the performance of the three approaches.

- **Study 2** changes the subsystem period (compared to Study 1) to 75 and 50 and keeps the number of shared resources to 12. As mentioned previously we use the same 1000 subsystems as in Study 1 and only change the subsystem period. The intention of this study is to show the effect of decreasing the subsystem period on the performance of the three approaches.

- **Study 3** decreases the critical section execution time to $(0.01 - 0.05) \times C_i$ (compared to Study 1) and keeps the number of shared resources to 12. The intention of this study is to show the effect of decreasing the critical section execution times on the performance of the three approaches.

Looking at the results in Table 2 (Study 1), it is clear that the improvements that both ISBF and IRBF can achieve become more significant when the number of shared resource accesses is increased. This is also clear in Figure 6 and Figure 7 that show the number of subsystems that have subsystem utilization within the ranges shown in the x-axis (the lines that connect points are only used for illustration) for 8 and 12 shared resource accesses, respectively. The reason is that the self-blocking $I^s_{\tau}(i, t)$ in Eq. (8), used by the original SIRAP approach, will increase significantly which will require more subsystem utilization. Comparing the values in the table, when the number of shared resources is 12 the analysis based on ISBF can decrease the subsystem utilization by 36% compared with the original SIRAP approach and the improvement in the median of subsystem utilization is about 12.5%. IRBF can achieve slightly less improvement than ISBF because the number of the considered
self-blocking \(z(t)\) is an upper bound. However, when the number of shared resources is low, e.g. 2, ISBF and IRBF can achieve some improvement compared with the original SIRAP, and in many cases ISBF requires higher subsystem utilization compared with the original SIRAP (about 48%). It is interesting to see that even if the number of shared resource access is low, ISBF and IRBF can achieve some improvements. Note that IRBF will never require more subsystem utilization than using the original SIRAP approach (see Lemma 4). Now, comparing the results of using ISBF and IRBF, we can see from the table that ISBF gives relatively better results, in terms of the number of subsystems that require less subsystem utilization, median and maximum improvement compared with IRBF if the number of shared resources accesses is high. The reason is that the possibility of having many large self-blocking will be higher which can decrease the effect of \(X^0\) on ISBF.

<table>
<thead>
<tr>
<th>Number of shared resources</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{ISBF}^i))</td>
<td>0.2%</td>
<td>23.1%</td>
<td>98.7%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) &lt; (U_{ISBF}^i))</td>
<td>2.0%</td>
<td>33.3%</td>
<td>99.5%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) = (U_{ISBF}^i))</td>
<td>50.0%</td>
<td>29.0%</td>
<td>0.2%</td>
<td>0%</td>
</tr>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{IRBF}^i))</td>
<td>2.0%</td>
<td>31.0%</td>
<td>80.0%</td>
<td>90.0%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>35.6</td>
<td>37.0</td>
<td>40.8</td>
<td>43.6</td>
</tr>
<tr>
<td>Median ((U_{ISBF}^i))</td>
<td>35.6</td>
<td>36.9</td>
<td>38.8</td>
<td>39.3</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{ISBF}^i))</td>
<td>3.1%</td>
<td>5.7%</td>
<td>16.4%</td>
<td>30.6%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{ISBF}^i))</td>
<td>7.3%</td>
<td>14.4%</td>
<td>22.7%</td>
<td>36.7%</td>
</tr>
<tr>
<td>Max D ((U_{ISBF}^i) / (U_{ISBF}^i))</td>
<td>5.5%</td>
<td>3.9%</td>
<td>1.2%</td>
<td>0%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{IRBF}^i))</td>
<td>7.3%</td>
<td>8.8%</td>
<td>22.1%</td>
<td>17.2%</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{IRBF}^i))</td>
<td>5.5%</td>
<td>4.0%</td>
<td>2.0%</td>
<td>1.7%</td>
</tr>
</tbody>
</table>

**Table 2. Measured results of Study 1**

<table>
<thead>
<tr>
<th>(P_s)</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{ISBF}^i))</td>
<td>87.0%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) &lt; (U_{ISBF}^i))</td>
<td>83.0%</td>
<td>99.7%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) = (U_{ISBF}^i))</td>
<td>6.0%</td>
<td>0.1%</td>
<td>0%</td>
</tr>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{IRBF}^i))</td>
<td>55.0%</td>
<td>82.0%</td>
<td>90.0%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>36.0%</td>
<td>14.0%</td>
<td>8.0%</td>
</tr>
<tr>
<td>Median ((U_{ISBF}^i))</td>
<td>41.0%</td>
<td>42.3%</td>
<td>43.6%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>39.7%</td>
<td>39.3%</td>
<td>39.3%</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{ISBF}^i))</td>
<td>16.8%</td>
<td>30.3%</td>
<td>30.6%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{ISBF}^i))</td>
<td>17.3%</td>
<td>36.5%</td>
<td>36.7%</td>
</tr>
<tr>
<td>Max D ((U_{ISBF}^i) / (U_{ISBF}^i))</td>
<td>2.7%</td>
<td>0.7%</td>
<td>0%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{IRBF}^i))</td>
<td>4.4%</td>
<td>12.1%</td>
<td>17.2%</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{IRBF}^i))</td>
<td>2.7%</td>
<td>1.9%</td>
<td>1.7%</td>
</tr>
</tbody>
</table>

**Table 3. Measured results of Study 2**

<table>
<thead>
<tr>
<th>(C_{sub})</th>
<th>((1 - 5%) \times C_s)</th>
<th>((10 - 25%) \times C_s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{ISBF}^i))</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) &lt; (U_{ISBF}^i))</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>((U_{ISBF}^i) = (U_{ISBF}^i))</td>
<td>78.0%</td>
<td>90.0%</td>
</tr>
<tr>
<td>((U_{IRBF}^i) &lt; (U_{IRBF}^i))</td>
<td>8.0%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>35.6%</td>
<td>42.6%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>34.4%</td>
<td>39.3%</td>
</tr>
<tr>
<td>Median ((U_{IRBF}^i))</td>
<td>34.3%</td>
<td>38.7%</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{ISBF}^i))</td>
<td>5.0%</td>
<td>30.6%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{ISBF}^i))</td>
<td>7.0%</td>
<td>36.7%</td>
</tr>
<tr>
<td>Max I ((U_{ISBF}^i) / (U_{IRBF}^i))</td>
<td>2.1%</td>
<td>17.2%</td>
</tr>
<tr>
<td>Max I ((U_{IRBF}^i) / (U_{IRBF}^i))</td>
<td>0.4%</td>
<td>1.7%</td>
</tr>
</tbody>
</table>

**Table 4. Measured results of Study 3**

Looking at Table 3 (Study 2), it is clear that when the subsystem period is decreased, the improvement that ISBF and IRBF can achieve compared with original SIRAP is also decreased. Comparing the median of the subsystem utilization of the 1000 generated subsystems when changing the subsystem period, we can see that for the original SIRAP analysis the subsystem utilization is decreasing when decreasing the subsystem period. However, using the other two approaches, the subsystem utilization is increasing when decreasing the subsystem period. The reason for this behavior is that the number of self-blocking occurrences will increase when decreasing the subsystem period and in turn it will increase \(z(t)\) using IRBF, i.e. the number of \(X^j\) for ISBF. This will increase \(rbf_{fp}(i, t)\) using IRBF, and decrease \(rbf_s(i, t)\) using ISBF, at time \(t\) compared with

![Figure 6. Results of Study 1 for 8 global shared resources access.](image1)

![Figure 7. Results of Study 1 for 12 global shared resources access.](image2)
the case when the subsystem period is higher, and that will in turn require more subsystem utilization. Note that this case can happen when the number of shared resource accesses is high. So for a high number of global shared resource accesses, it is recommended to use larger subsystem periods that can decrease the subsystem utilization and at the same time decrease the number of subsystem context switches. Another interesting observation from this table is that the percentage of subsystems that require less subsystem utilization using ISBF compared with IRBF, is decreasing when decreasing the subsystem period. The reason is that more self-blocking occurrences will be considered in both ISBF and IRBF and that will increase the possibility of having a large difference between the considered self-blocking which will increase the effect of $X^0$ for ISBF.

In Study 3 we have decreased the range of the critical section execution times which will, in turn, decrease the self-blocking execution times. The results in Table 4 show that the improvements that ISBF and IRBF can achieve in terms of subsystem utilization compared with the original SIRAP approach, are decreased. The improvement in the subsystem utilization median using ISBF is decreased from 12.6% to 2% when decreasing the critical section execution time, and using IRBF it is decreased from 10.9% to 1.7%. The reason for this is that the total self-blocking $I^B_i (i,t)$ in Eq. (8) used by the original SIRAP approach, depends not only on the number of shared resource accesses but also on the size of the self-blocking $X_{i,ka}$.

8 Summary

In this paper, we have presented new schedulability analysis for SIRAP; a synchronization protocol for hierarchically scheduled real-time systems. We have shown that the original local schedulability analysis for SIRAP is pessimistic when the tasks of a subsystem make a high number of accesses to global shared resources. This pessimism is inherent in the fact that the original SIRAP schedulability analysis does not take the maximum number of self-blocking instances into account, when in fact this number is bounded by the maximum number of subsystem period intervals in which these resource accessing tasks execute. We have presented two new analysis approaches that take this bounded number of self-blocking instances into account; the first approach based on changing rbf and second approach based on changing srbf. We have identified the parameters that have effect on the improvement that these new approaches can achieve over the original SIRAP schedulability analysis and we have explored and explained the effect of these parameters by means of simulation analysis. The results of the simulation show that significant improvements can be achieved by the new approaches compared to the original SIRAP approach, if the number of accesses to global shared resources made by the tasks of a subsystem is high. Generalizing the analysis of this paper to include other scheduling algorithms, e.g. EDF, as a subsystem level scheduler, is a topic of future work.

Acknowledgment

The authors thank all reviewers for their constructive comments and suggestions.

References