Compositional multiprocessor scheduling: the GMPR interface

Artem Burmyakov · Enrico Bini · Eduardo Tovar

Abstract Composition is a practice of key importance in software engineering. When real-time applications are composed, it is necessary that their timing properties (such as meeting the deadlines) are guaranteed. The composition is performed by establishing an interface between the application and the physical platform. Such an interface typically contains information about the amount of computing capacity needed by the application. For multiprocessor platforms, the interface should also present information about the degree of parallelism. Several interface proposals have recently been put forward in various research works. However, those interfaces are either too complex to be handled or too pessimistic. In this paper we propose the generalized multiprocessor periodic resource model (GMPR) that is strictly superior to the MPR model without requiring a too detailed description. We then derive a method to compute the interface from the application specification. This method has been implemented in Matlab routines that are publicly available.

Keywords Real-time scheduling · Compositional scheduling · Multiprocessors · Real-time interfaces

1 Introduction

Reusing application code is driven by the need to shorten the overall design time, and typically software components are developed in isolation, possibly by different devel-

opers. During the integration phase, all components are bound to the same hardware platform. Clearly, the integration must be performed in such a way that the properties of components are preserved even after the composition is made.

In real-time systems, the key property that has to be preserved during the integration phase is time predictability: a real-time application (or component) that meets all its deadlines when designed in isolation should also meet all deadlines when it is integrated with other applications on the same hardware platform. This property is often guaranteed by introducing an *interface* between the application and the hardware platform. Then the application is guaranteed over the interface, and the hardware platform must provide a *virtual platform* that conforms with the interface—a compliant virtual platform. The scheduling problem over a virtual platform is often called a *hierarchical scheduling* problem. In fact, each application task itself may contain another entire application in a hierarchical fashion.

The benefit of using an interface-based approach is significant. During the design phase the interface of an application is computed such that all timing requirements of the application are met. Then, during the integration phase the interfaces of all applications are bound to the same hardware platform. As a result, the interface allows to hide an internal complexity of an individual application, and this property is essential in the development of large-scale real-time systems.

Typically, interfaces, allowing the composition of real-time applications, specify details about the amount of resource that has to be provided by a compliant virtual platform. This information can be described with a varying degree of detail. For example, a very simple interface for a virtual processor can be just a fraction of the allocated time.

With the broad diffusion of multiprocessors, hierarchical scheduling problems have recently started to be considered over hardware platforms that provide a concurrent resource supply. The formulation of interface models for multiprocessors, however, requires the introduction of a new dimension: the *degree of concurrency*. This additional characteristic of the interface makes the problem to be addressed more challenging.

The problem in selecting the appropriate interface model is to find the best trade-off between accuracy and simplicity of the interface. A simple interface is intuitive and easy to use, but it tends to cause a significant pessimism in the resource abstraction. On the other hand, an accurate interface minimizes the pessimism, but is more complex in use, and it can be very difficult to compute. In this paper we propose a simple interface that is a generalization of the one previously proposed by Shin et al. (2008). Our novel approach keeps the simplicity of that interface while reducing significantly the pessimism in terms of the needed resource.

1.1 Related works

The problem of composing real-time applications is certainly not new. There actually have been numerous contributions in this area. Being fully aware of the impossibility to provide a full coverage of the topic, we describe in this section the works that, to our best knowledge, are more related to ours.

One of the first contributions to address the isolation of applications using *resource reservations* was published in Parekh and Gallager (1993). In that paper the authors introduced the generalized processor sharing (GPS) algorithm to share a fluid resource according to a set of weights. Mercer et al. (1994) proposed a more realistic approach where a resource can be allocated based on a required budget and period. Later on, Stoica et al. (1996) introduced the earliest eligible virtual deadline first (EEVDF) for sharing the computing resource, and Deng and Liu (1997) achieved the same goal by introducing a two-level scheduler (using EDF as a global scheduler) in the context of multi-application systems. Kuo and Li (1999) extended the approach to a fixed priority global scheduler. Kuo et al. (2000) extended their own work (Kuo and Li 1999) to multiprocessors. However, in those approaches the authors made very stringent assumptions such as not considering task migration and restricting to period harmonicity. Those assumptions restrict the applicability of the proposed solution.

Moir and Ramamurthy (1999) proposed a hierarchical approach, where a set of P-fair tasks can be scheduled within a time partition provided by another P-fair task (called "supertask") acting as a server. However, the solution often requires the weight of the supertask to be higher than the sum of the weights of the served tasks (Holman and Anderson 2006).

Many independent works proposed to model the service provided by a uni-processor through a supply function. Feng and Mok (2002) introduced the bounded-delay resource partition model. Almeida et al. (2002) provided timing guarantees for both synchronous and asynchronous traffic over the FTT-CAN protocol by using hierarchical scheduling. Lipari and Bini (2003) derived the set of virtual processors that can feasibly schedule a given application. Shin and Lee (2003) introduced the periodic resource model also deriving a utilization bound. Easwaran et al. (2007) extended this model allowing the server deadline to be different from its period. Fisher and Dewan (2009) proposed an approximation algorithm to test the schedulability of a task set over a periodic resource.

More recently, some authors have addressed the problem of specifying an interface for applications executed upon multiprocessor systems, providing appropriate tests to verify schedulability of applications over that interface.

One of such works is described in Leontyev and Anderson (2008), where the authors proposed to use only the overall bandwidth requirement ω as interface for soft real-time applications. The authors proposed to allocate a bandwidth requirement of w onto $\lfloor w \rfloor$ dedicated processors, plus an amount of $w - \lfloor w \rfloor$ provided by a periodic server globally scheduled onto the remaining processors. An upper bound of the tardiness of tasks scheduled on such an interface was provided.

Shin et al. (2008) proposed the multiprocessor periodic resource model (MPR) that specifies a period, a budget and maximum level of parallelism of the resource provisioning. Khalilzad et al. (2012) later extended the MPR model, relaxing the assumption of fully synchronized virtual processors. Since our work is a generalization of the MPR, in Sect. 2.2 we describe the MPR in greater details.

Chang et al. (2008) proposed to partition the resource available from a multiprocessor by a static periodic scheme. The amount of resource is then provided to the application through a contract specification.

Bini et al. (2009) proposed the parallel supply function (PSF) interface of a virtual multiprocessor. This interface is designed to tightly capture the amount of resource provided by a virtual platform for very general supply mechanisms, which are not necessarily periodic. In their approach the authors do not reason on how to compute the interface parameters that guarantee the schedulability of a real-time application.

Lipari and Bini (2010) described an entire framework for composing real-time applications running over a multiprocessor. However, their proposed interface was extremely trivial.

Burmyakov et al. (2012) extended the multiprocessor periodic resource model (MPR) by specifying the minimal budgets for each level of parallelism. However, the assumption of integer budget values made the problem to compute an interface hardly tractable, even for a task set with a low utilization.

1.2 Contributions of the paper

The MPR model is one of the simplest interface models for the multiprocessor systems. In this paper we propose its extension, the GMPR model, which generalizes the MPR, reducing its pessimism while keeping its simplicity. To analyze schedulability over GMPR, we reuse the schedulability test proposed by Bini et al. (2009). We first improve this test by minimizing its run-time, and then, based on it, we derive several methods to compute the minimal GMPR which can guarantee a given set of tasks. We implement the algorithms to compute the GMPR in the Matlab environment. Then, we evaluate the GMPR against the MPR model to confirm a reduced resource utilization of GMPR, and therefore a significant reduction in the level of pessimism.

The remainder of the paper is organized as follows. In Sect. 2 we briefly review the concepts and notations related to our research. In particular, we illustrate the drawbacks of the existing interface models by the examples of the PSF and the MPR models. In Sect. 3 we propose a new interface model called GMPR. Then, in Sect. 4 we adapt the schedulability test by Bini et al. (2009) over a virtual resource abstracted by a GMPR interface. In Sect. 5 we develop an algorithm to compute a feasible GMPR for a given task set. Later, in Sect. 6, we propose a technique to schedule GMPR interfaces. Finally, in Sect. 7 we evaluate the pessimism of GMPR against the MPR model.

2 Background on multiprocessor interfaces

In the past, there have been some proposals for multiprocessor interfaces. This section illustrates three of them (Leontyev and Anderson 2008; Shin et al. 2008; Bini et al. 2009). The interfaces are ordered by their increasing complexity and, consequently, by increasing accuracy of the guarantee test for applications running over the interface.

2.1 The multiprocessor bandwidth interface

Leontyev and Anderson (2008) proposed to use only the overall bandwidth requirement W (using their original notation) as an interface for soft real-time tasks. Being

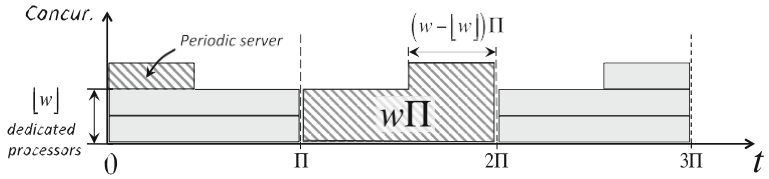


Fig. 1 The resource allocation over MBI with the bandwidth w

a multiprocessor interface, it is well acceptable to have $w > 1$. To schedule a task set, the authors proposed to allocate a bandwidth requirement of w onto $\lfloor w \rfloor$ fully dedicated processors, plus the bandwidth of $w - \lfloor w \rfloor$ provided by a periodic server globally scheduled onto the remaining processors (see Fig. 1).

We refer the interface model of Leontyev and Anderson (2008) as the multiprocessor bandwidth interface (MBI) and denote it as

$$\langle w, \Pi \rangle,$$

where w is the interface bandwidth and Π is the server period. Initially designed for soft real-time tasks, the MBI model can easily be extended for hard real-time systems. The advantage of the MBI is its simplicity and the reduced pessimism in the resource abstraction compared to many other existing models.

At the same time, there is a strong limitation of the MBI model as it requires $\lfloor w \rfloor$ fully dedicated processors. In a general case of the compositional scheduling, such a requirement cannot be always guaranteed by a virtual execution platform, for extended periods of time. To overcome this limitation, other different interface models have been introduced, as described in the next sections.

2.2 The multiprocessor periodic resource model (MPR)

The MPR model (Shin et al. 2008) is another simple resource abstraction. Its definition is given below.

Definition 1 A MPR model is modeled by a triplet

$$\langle \Pi, \Theta, m \rangle,$$

where Π is the time period and Θ is the minimal resource supply provided within each time interval $[k\Pi, (k+1)\Pi)$, with $k \in \mathbb{N}_0$, by at most m processors at a time. Often we also say that m is the *concurrency* (or the *degree of parallelism*) of the interface. The *utilization* of a MPR interface is the ratio $\frac{\Theta}{\Pi}$.

Since a MPR interface fixes only the aggregated parameters Π, Θ and m of the supply pattern, any feasible allocation of Θ resource units per time period Π with a parallelism m should preserve the schedulability of the underlying task set. It is then necessary to find the worst-case resource allocation for the MPR. Generalizing the result of Shin et al. (2008), derived for a case of integer Θ, the worst-case scenario for

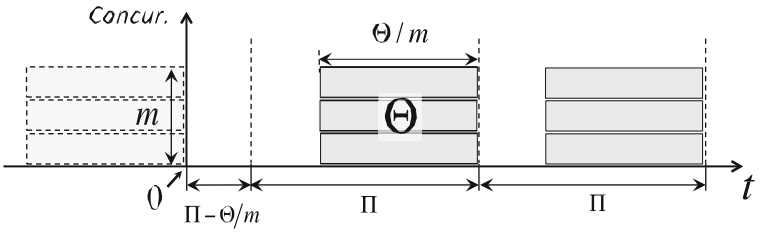


Fig. 2 The worst-case resource allocation over the MPR (Π, Θ, m). Instant 0 denotes the beginning of the worst-case interval

Table 1 An example of a task set

i	C_i	T_i	D_i
1	1	30	30
2	4	40	40
3	11	50	50
4	15	60	60

an arbitrary Θ is the one depicted in Fig. 2, where time instant 0 denotes the beginning of the worst-case interval. Note that in the MPR case the contribution of each processor to the interface is Θ/m every period Π.

2.3 Comparison of the MBI and MPR models

The MBI model dominates MPR in terms of overall resource required to schedule an application: over the same time interval, MBI requires at most as much resource as MPR. However, unlike MBI, an MPR interface can be also provided over a platform in which the processors are not fully available (possibly due to the coexistence with other applications already consuming resource). In fact, by increasing the interface parallelism m, the requirement Θ/m on each processor decreases, making it possible to fit an interface on partially available platforms.

We illustrate this by an example. Consider a task set with the parameters reported in Table 1, to be scheduled by global EDF (GEDF) over a virtual platform. To compute interfaces, we apply the schedulability test of Lipari and Bini (2010), which is described in details later in Sect. 4. By setting the server period to $\Pi = 20$, we determine that the minimal MBI interface, guaranteeing the schedulability of the task set, requires 26 resource units every Π, while the MPR of the same concurrency $m = 2$ requires at least 30.8 units (see Fig. 3).

Let us now increase the MPR concurrency to $m = 3$. We immediately observe a reduction of resource to be provided by each virtual processor, from 15.4 to 11.4 units. For $m = 5$, the resource fraction decreases further to 10.4. Notice, however, that the overall resource Θ increases with m.

2.4 The parallel supply function (PSF)

The PSF was proposed by Bini et al. (2009) to characterize the resource allocation in hierarchical systems executed upon a multiprocessor platform. This interface is

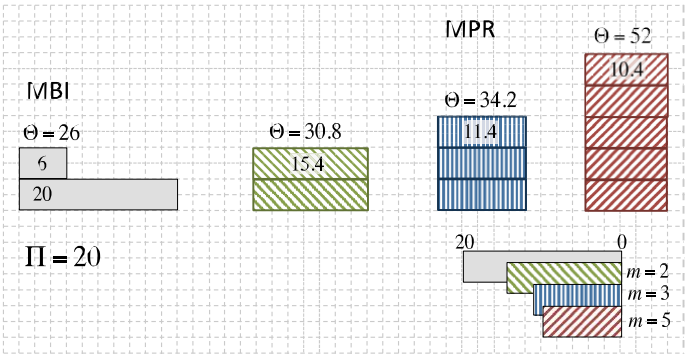


Fig. 3 Comparison of MBI and MPR

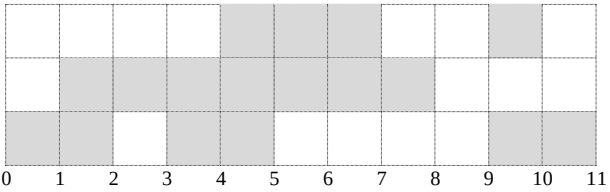


Fig. 4 From a resource schedule to the PSF interface

designed to tightly capture the amount of resource provided by a virtual platform for very general supply mechanisms, which are not necessarily periodic. As a drawback it is certainly quite complicated to be handled. Without entering into all the details of the definition (that can indeed be found in Bini et al. (2009)), we recall here the basic concepts.

Definition 2 The PSF interface of a multiprocessor resource is composed by the set of functions $\{Y_k\}_{k=1}^{m}$, where m is the number of virtual processors and $Y_k(t)$ is the *minimum* amount of resource provided in *any* interval of length t with a parallelism of *at most k*. The function $Y_k(t)$ is called the *level-k parallel supply function*.

To clarify this definition we propose an example. Consider that in the interval $[0, 11]$ the resource is provided by three processors according to the schedule drawn in gray in Fig. 4.

In this case $Y_1(11) = 10$ because there is always at least one processor available in [0,11] except in [8,9]. Then $Y_2(11) = 16$; that is found by summing up all the resources except one with parallelism 3 (provided only in [4,5]). Finally, $Y_3(11) = 17$; that is achieved by summing all the resources provided in [0,11]. In general, the parallel supply functions are also computed by sliding the time window of length t and by searching for the most pessimistic scenario of resource allocation. This minimization is somehow equivalent to the one performed on uni-processor hierarchical scheduling (Feng and Mok 2002; Lipari and Bini 2003; Shin and Lee 2003) for computing the supply function of a virtual resource.

Table 2 An example of a task set

i	C_i	T_i	D_i
1	6	40	40
2	13	50	50
3	29	60	60
4	27	70	70

Since the PSF can be computed for any possible resource allocation scheme, it is possible to compute it also for the MPR interface. The computation of the PSF interface $\{Y_k\}_{k=1}^{m}$ of a MPR enables the adaptation of schedulability tests developed over a PSF interface to a MPR interface. More details about the schedulability test will be provided in Sect. 4.

3 The generalized multiprocessor periodic resource (GMPR) model

The main drawback of the MPR interface is that it may require more computational capacity than needed, and therefore it has an undesirable level of pessimism in terms of resource allocation. Consider the task set with the parameters as depicted in Table 2, to be scheduled by global EDF (GEDF) over the MPR interface. In that table, for each task we provide its execution time, C_i, its period, T_i, and its deadline, D_i.

After setting the period of the interface $\Pi = 15$, we compute a MPR interface (Π, Θ, m) that can guarantee the task set. To check the schedulability, we reuse the PSF-based test proposed by Bini et al. (2009) (see Sect. 4 for details). Based on this test, we determine that the minimum feasible value of resource units to guarantee the schedulability is $\Theta = 39$. Notice that there is quite a significant gap between the utilization of the interface $\frac{\Theta}{\Pi} = 2.6$ and the utilization of the task set $\sum_i \frac{C_i}{T_i} = 1.28$.

As we will show in greater detail in the next sections, our proposed interface requires only 34 resource units per period, meaning that it has a utilization of $\frac{34}{15} = 2.267$ for the given example.

3.1 Model description

The main reason for the pessimism of the MPR is that the worst-case of the supply (Fig. 2) must be very conservative, if the only information in the interface is that an overall budget Θ is provided every Π. We propose to rectify this problem, as described below.

Definition 3 We define the GMPR interface model as

$$\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle,$$

where Π is the time period and Θ_k is the minimal resource supply provided within each time interval $[\Pi\Pi, (\Pi + 1)\Pi)$, $\Pi \in \mathbb{N}_0$, with a degree of parallelism of at most k. The values of Θ_k must satisfy the following constraints for any $k = 1, \ldots, m$ (for

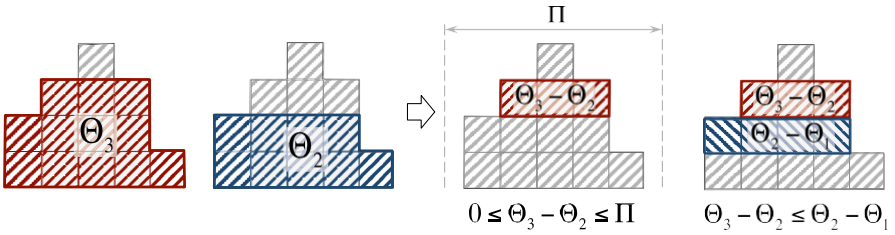


Fig. 5 Illustration of the constraints in the GMPR definition

convenience we denote $\Theta_k = 0,\ k \leq 0$):

$$0 \leq \Theta_k - \Theta_{k-1} \leq \Pi$$
$$\Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1}. \quad (1)$$

We assume that the interface parameters Π and $\Theta_1, \ldots, \Theta_m$ belong to R.

The "degree of parallelism" of a resource supply at time instant *t*, is the number of processors providing the resource at that instant. For example, an application which may have at most Π threads in parallel will not ever benefit from having a resource provided by $\Pi + 1$ processors simultaneously. Hence, for such an application, it does not make sense to have $\Theta_{\Pi+1}$ strictly larger than Θ_Π, since the extra amount of resource $\Theta_{\Pi+1} - \Theta_\Pi$ is provided at a too high parallelism that the application never exhibits.

The motivation for the constraints in Definition 3 is the following:

- $\Theta_k \geq \Theta_{k-1}$, because the overall supply at higher parallelism cannot decrease;
- $\Theta_k - \Theta_{k-1} \leq \Pi$, because the increment of supply at parallelism k (that is $\Theta_k - \Theta_{k-1}$) cannot exceed the length of the period;
- $\Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1}$, because the increment of supply at parallelism $k + 1$ (that is $\Theta_{k+1} - \Theta_k$) should not exceed the increment of supply at parallelism k (that is $\Theta_k - \Theta_{k-1}$). Otherwise some of the supply provided at parallelism $k + 1$ must instead be available at parallelism k.

Figure 5 illustrates an example of a resource supply over a GMPR interface with $\Pi = 6$, $\Theta_1 = 5$, $\Theta_2 = 9$, and $\Theta_3 = 12$.

A valid GMPR interface should guarantee the schedulability of a task set: any resource allocation compliant with the GMPR specification has to guarantee that all task deadlines are met.

The proposed GMPR interface model generalizes both MPR and MBI. In fact, a MPR interface (Π, Θ, m) is equivalent to a GMPR $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ with

$$\Theta_k = \frac{k}{m}\Theta, \quad k = 1, \ldots, m,$$

and a MBI interface (w, Π) is equivalent to a GMPR with

$$\Theta_k = k\,\Pi, \quad k = 1, \ldots, \lfloor w \rfloor$$
$$\Theta_{\lceil w \rceil} = w\,\Pi.$$

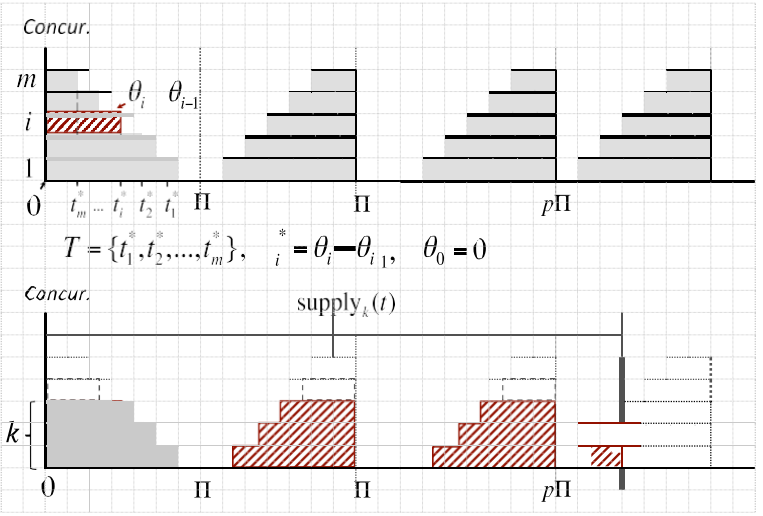


Fig. 6 The worst-case resource allocation over GMPR $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ (*top*) and the definition of the $\mathsf{supply}_k(t)$ function (*bottom*) proposed by Burmyakov et al. (2012)

3.2 Parallel supply functions of GMPR

To borrow the schedulability tests developed over the PSF interface (Bini et al. 2009), we compute the parallel supply functions $\{Y_k(t)\}_{k=1,\ldots,m}$ for the GMPR specification.

Burmyakov et al. (2012) proposed to compute the PSF using a classical approach in hierarchical scheduling. In that work the authors considered the worst-case scenario of the resource supply (depicted in Fig. 6) and defined $\mathsf{supply}_k(t)$ as the amount of resource available in $[0, t]$ by at most k concurrent processors (see Fig. 6). Then, the PSF $Y_k(t)$ was computed as

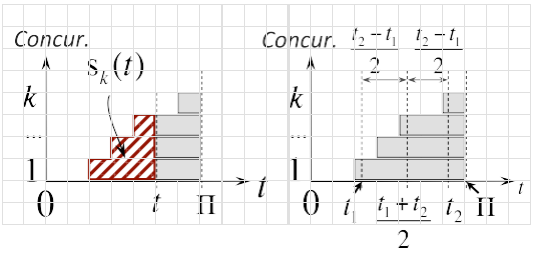
$$Y_k(t) = \min_{t_0 \in T} \left(\mathsf{supply}_k(t + t_0) - \mathsf{supply}_k(t_0)\right),$$

with $T = \{\Theta_i - \Theta_{i-1} \mid i = 1, \ldots, k\}$ being the set of time instants at which the supply by some processor ends.

Instead of the above mentioned approach, we now propose a significantly more efficient method to compute the Parallel Supply Functions $Y_k(t)$. We stress that this method is also applicable to the classical problems of hierarchical scheduling over a single processor (Lipari and Bini 2003; Shin and Lee 2003), as PSF is a generalization of the uni-processor supply function.

To compute the PSF $Y_k(t)$, let us first introduce an auxiliary function $s_k(t)$ over $t \in [0, \Pi]$. We define $s_k(t)$ as the overall amount of resource provided over the pattern of Fig. 7, in a time interval $[0, t]$. The function $s_k(t)$ has the property, formulated in the next lemma.

Fig. 7 Properties of the $s_k(t)$ function



Lemma 1 *Let* $s_k : [0, \Pi] \to \mathbb{R}$ *be defined as*

$$s_k(t) = \sum_{i=1}^{k} (t - (\Pi - (\Theta_i - \Theta_{i-1})))_0 . \tag{2}$$

Then, for any values $t_1, t_2 \in [0, \Pi]$, *we have*

$$s_k(t_1) + s_k(t_2) \geq 2\, s_k\left(\frac{t_1 + t_2}{2}\right). \tag{3}$$

Proof Consider the resource allocation over the time interval $[t_1, t_2]$ of Fig. 7. Time instant $t = \frac{t_1+t_2}{2}$ is the middle of this interval. Due to the alignment of the resource blocks to the right side, the resource in $[t_1, \frac{t_1+t_2}{2}]$ does not exceed the resource in $[\frac{t_1+t_2}{2}; t_2]$. It follows that

$$s_k\left(\tfrac{t_1+t_2}{2}\right) - s_k(t_1) \leq s_k(t_2) - s_k\left(\tfrac{t_1+t_2}{2}\right),$$

□

what leads us to (3).

The next theorem determines the worst-case scenarios of the resource supply which are then used to compute $Y_k(t)$.

Theorem 1 *The worst-case amount of resource provided over a GMPR interface* $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ *in an arbitrary time interval of length* t *is the minimum among the resources provided in* $[-\frac{t}{2}, \frac{t}{2}]$ *by any of the two patterns* S^{even} *and* S^{odd} *depicted in Fig. 8.*

Proof Let $\text{supply}_k^{\Gamma}(S, t)$ denote the resource provided by an arbitrary scenario S in the time interval $[-\frac{t}{2}, \frac{t}{2}]$ (of length t) at concurrency k. We next consider two cases depending on the interval length t: $t \leq \Pi$ and otherwise.

We recall that, from Definition 3 of GMPR, there always exists a time instant t^* such that the resource provided at concurrency k over each interval $[t^* + (p-1)\Pi, t^* + p\Pi]$, $p \in \mathbb{Z}$, equals to Θ_k. We refer t^* as the replenishment instant of a GMPR interface, and the time intervals $[t^* + (p-1)\Pi, t^* + p\Pi]$ are its replenishment cycles.

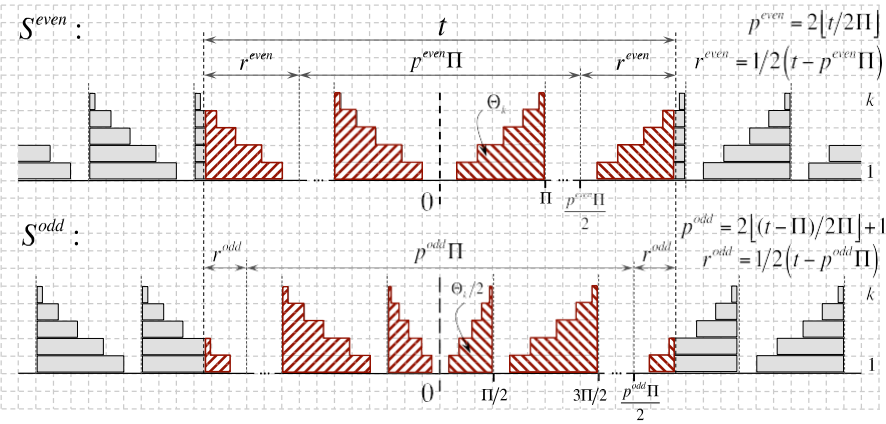


Fig. 8 The worst-case resource allocation patterns S^{even} and S^{odd} over GMPR $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$

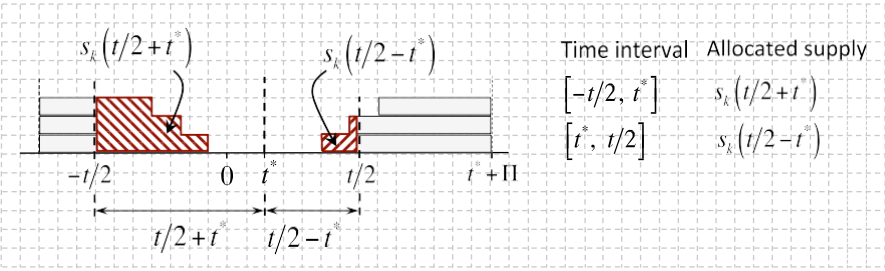


Fig. 9 Scenario S' for case 1a: $0 \le t^* \le \frac{t}{2} \le \frac{\Pi}{2}$

Case 1 $t \le \Pi$. There always exists a replenishment instant $t^* \in \left[-\frac{\Pi}{2}, \frac{\Pi}{2}\right]$ such that the resource provided in both intervals $[t^* - \Pi, t^*]$ and $[t^*, t^* + \Pi]$ is Θ_k each. Let us assume that $t^* \ge 0$; the proof for $t^* < 0$ is done by analogy.

As $t^* \in [0, \frac{\Pi}{2}]$ and $\frac{t}{2} \in [0, \frac{\Pi}{2}]$, the following two cases are possible:

$$0 \le t^* \le \tfrac{t}{2} \le \tfrac{\Pi}{2}$$
$$0 \le \tfrac{t}{2} \le t^* \le \tfrac{\Pi}{2}.$$

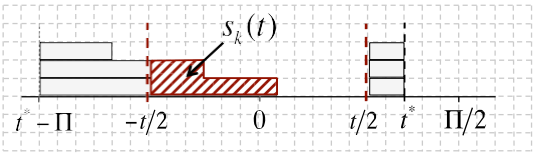
Each of these cases is considered below.

Case 1a $0 \le t^* \le \frac{t}{2} \le \frac{\Pi}{2}$. Let us transform the scenario S into S' by moving left any resource provided before t^* and by moving right any resource provided after t^*, as depicted in Fig. 9. Since $t^* \in [-\frac{t}{2}, \frac{t}{2}]$, such a transformation can only move the resource out of the time interval $[-\frac{t}{2}, \frac{t}{2}]$, so that

$$\text{supply}_k(S, t) \ge \text{supply}_k(S', t).$$

To analyze the resource supply over S', we now employ the auxiliary function $s_k(t)$ introduced in Lemma 1. From Fig. 9, it follows that

Fig. 10 Scenario S_k for Case 1b: $0 \le \frac{t}{2} \le t^* \le \frac{\Pi}{2}$



$$\text{supply}_k(S', t) = s_k\left(\frac{t}{2} + t^*\right) + s_k\left(\frac{t}{2} - t^*\right).$$

Applying condition (3) to the RHS of the equation above, we get that

$$\text{supply}_k(S', t) \ge 2s_k\left(\frac{t}{2}\right) = \text{supply}_k\left(S^{\text{even}}, t\right),$$

where S^{even} is the resource pattern depicted in Fig. 8.

Case 1b $0 \le \frac{t}{2} \le t^* \le \frac{\Pi}{2}$. Let us transform the scenario S into S' by moving out of the time interval $[-\frac{t}{2}, \frac{t}{2}]$ as much resource as possible (see Fig. 10), so that

$$\text{supply}_k(S, t) \ge \text{supply}_k(S', t).$$

From Fig. 10, it follows that

$$\text{supply}_k(S', t) \ge s_k(t) = s_k(t) + s_k(0) \ge 2s_k\left(\frac{t}{2}\right) = \text{supply}_k\left(S^{\text{even}}, t\right),$$

where the inequality holds due to Lemma 1.

The proof for $t^* \le 0$ is done by analogy to Cases 1a, 1b. Thus, S^{even} is the worst-case scenario for any $t \le \Pi$.

Case 2 $t > \Pi$. From any scenario S of resource supply, let us transform it into S' by moving left any resource provided before time instant 0 and by moving right any resource provided after 0. Since such a transformation can only move the resource out of the interval, it must again be that

$$\text{supply}_k(S, t) \ge \text{supply}_k(S', t).$$

For S', let us decompose the interval $[-\frac{t}{2}, \frac{t}{2}]$ into the three sub-intervals $[-\frac{t}{2}, t^*]$, $[t^*, t^* + p\Pi]$, and $[t^* + p\Pi, \frac{t}{2}]$ as shown in Fig. 11, where t^* denotes the first replenishment instant after $-\frac{t}{2}$, and $p \in \mathbb{N}$ is the number of full replenishment cycles in $[-\frac{t}{2}, \frac{t}{2}]$.

It follows that

$$p \in \left\{\left\lfloor \frac{t - \Pi}{\Pi} \right\rfloor, \left\lfloor \frac{t}{\Pi} \right\rfloor\right\},$$

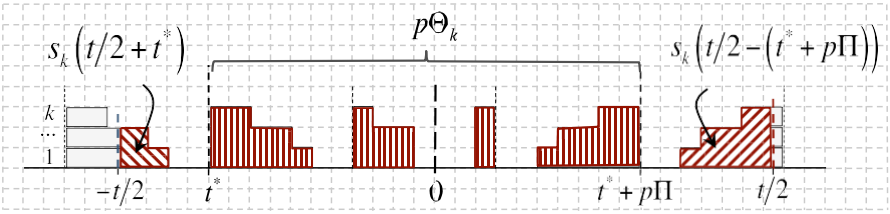


Fig. 11 Scenario S' for Case 2: $t > \Pi$

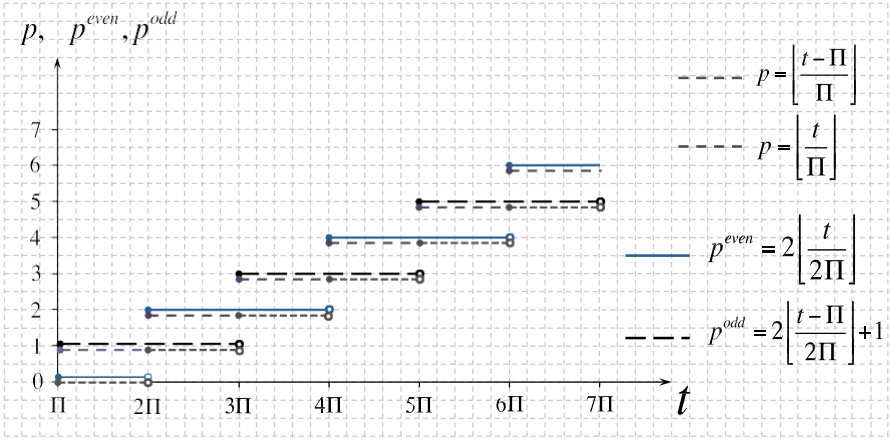


Fig. 12 Comparison of $p,\ p^{\text{even}}, p^{\text{odd}}$

which can also be written as $p \in \{p^{\text{even}}, p^{\text{odd}}\}$ (see Fig. 12 for a graphical interpretation), with

$$p^{\text{even}} = 2\left\lfloor\frac{t}{2\Pi}\right\rfloor \qquad p^{\text{odd}} = 2\left\lfloor\frac{t-\Pi}{2\Pi}\right\rfloor + 1.$$

The resource $\text{supply}_k(S', t)$ in the interval $[-\frac{t}{2}, \frac{t}{2}]$ is the sum of resource available over the three considered sub-intervals (see Fig. 11), so that

$$\begin{aligned}\text{supply}_k(S', t) &= s_k\left(\frac{t}{2}+t^*\right) + p\,\Theta_k + s_k\left(\frac{t}{2}-(t^*+p\Pi)\right)\\ &\geq p\,\Theta_k + 2s_k\left(\frac{t-p\Pi}{2}\right),\end{aligned}$$

where the inequality holds due to Lemma 1. In case $p = p^{\text{even}}$, then the equation above turns into

$$\text{supply}_k(S', t) \geq \text{supply}_k(S^{\text{even}}, t),$$

otherwise, if $p = p^{\text{odd}}$, then

$$\text{supply}_k(S', t) \geq \text{supply}_k(S^{odd}, t).$$

Thus, we conclude that no other scenario S exists providing less resource than S^{even} and S^{odd}. nu

Theorem 1 determines that the worst-case pattern for the resource supply of a GMPR interface is either S^{odd} or S^{even}. The next corollary uses such a result to compute the PSF of a GMPR interface.

Corollary 1 *The PSF function $Y_k(t)$ for GMPR is computed as*

$$Y_k(t) = \min\left(Y_k^{\text{even}}(t), Y_k^{\text{odd}}(t)\right), \tag{4}$$

where Y_k^{even} and Y_k^{odd} denote the resource provided by the patterns S^{even} and S^{odd} depicted in Fig. 8, computed as

$$Y_k^{\text{even}}(t) = p^{\text{even}}\Theta_k + 2\sum_{i=1}^{k}\left(r^{\text{even}} - \Pi + \Theta_i - \Theta_{i-1}\right)_0 \tag{5}$$

$$p^{\text{even}} = 2\left\lfloor \frac{t}{2\Pi} \right\rfloor \tag{6}$$

$$r^{\text{even}} = \frac{1}{2}\left(t - p^{\text{even}}\Pi\right) \tag{7}$$

and

$$Y_k^{\text{odd}}(t) = p^{\text{odd}}\Theta_k + 2\sum_{i=1}^{k}\left(r^{\text{odd}} - \Pi + \Theta_i - \Theta_{i-1}\right)_0 \tag{8}$$

$$p^{\text{odd}} = 2\left\lfloor \frac{t-\Pi}{2\Pi} \right\rfloor + 1 \tag{9}$$

$$r^{\text{odd}} = \frac{1}{2}\left(t - p^{\text{odd}}\Pi\right). \tag{10}$$

As an example, in Fig. 13 we illustrate the 4 parallel supply functions $\{Y_1(t), \ldots, Y_4(t)\}$ of the GMPR interface $(7, \{6, 11, 15, 17\})$. At the bottom of the figure we also represent the worst-case resource patterns that originate the parallel supply functions.

3.3 The lower and the upper bounds for $Y_k(t)$

We now propose a lower and an upper bound to $Y_k(t)$. These bounds will be later exploited in Sect. 5.3 to reduce the time required to compute a GMPR interface for a given task set.

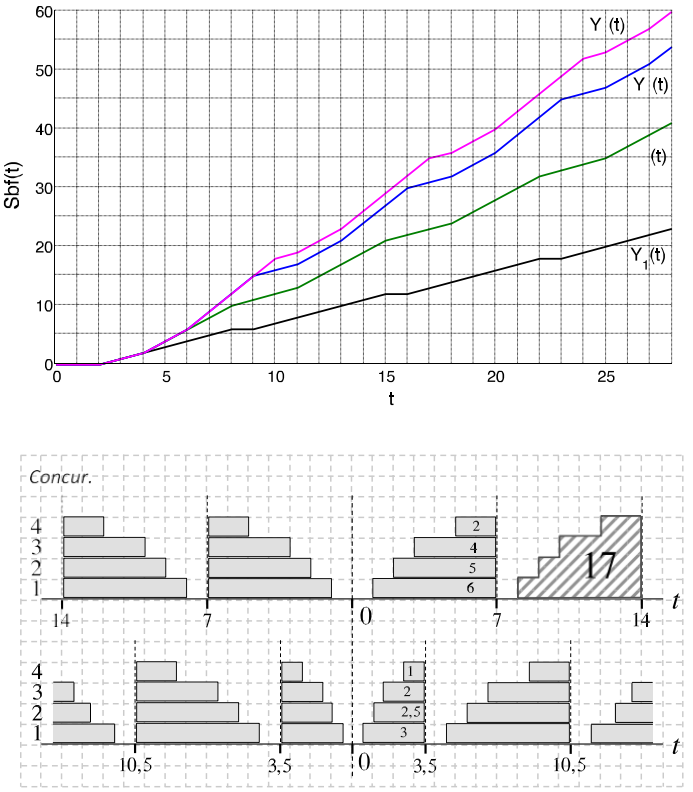


Fig. 13 The PSF (*top*) and the worst-case supply patterns (*bottom*) of the GMPR interface (7, {6, 11, 15, 17}). The *bold points* indicate the slope change of the PSF functions

The supply functions $Y_k^{\text{even}}(t)$, $Y_k^{\text{odd}}(t)$ defined by Eqs. (5), (8) can be equally expressed as

$$Y_k^{\text{even}}(t) = p^{\text{even}}\,\Theta_k + 2s_k(r^{\text{even}})$$
$$Y_k^{\text{odd}}(t) = p^{\text{odd}}\,\Theta_k + 2s_k(r^{\text{odd}}), \qquad (11)$$

with $s_k(t)$ defined by (2), and $p^{\text{even}}, r^{\text{even}}, p^{\text{odd}}, r^{\text{odd}}$ defined by (6), (7), (9), and (10), respectively.

We now observe that the function $s_k(t)$ can be lower bounded by the function $\underline{s}_k(t)$ defined as (see also Fig. 14)

$$s_k(t) \geq \underline{s}_k(t) = (\Theta_k - k\,(\Pi - t))_0\,. \qquad (12)$$

Substituting (12) into (11), we derive the following lower bounds for $Y_k^{\text{even}}(t)$, $Y_k^{\text{odd}}(t)$ denoted as $\underline{Y}_k^{\text{even}}(t)$, $\underline{Y}_k^{\text{odd}}(t)$:

$$\underline{Y}_k^{\text{even}}(t) = p^{\text{even}}\,\Theta_k + 2\,(\Theta_k - k\,(\Pi - r^{\text{even}}))_0 \qquad (13)$$

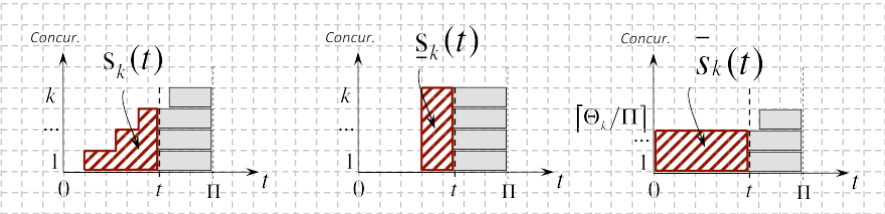
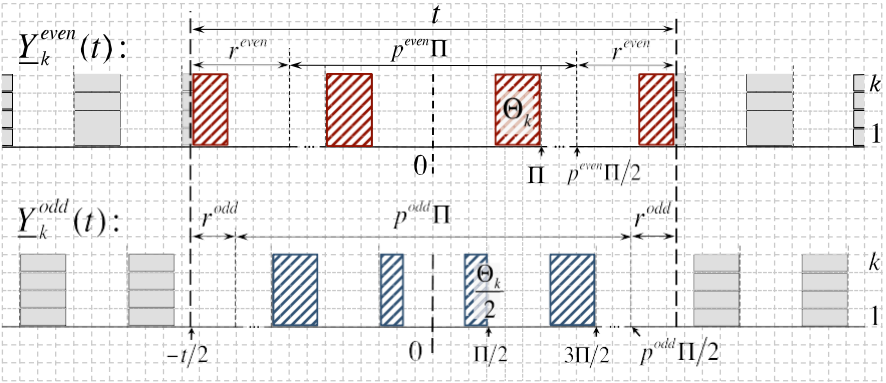


Fig. 14 The lower and upper bounds $s_{\underline{k}}(t)$, $s_{\overline{k}}(t)$ for the supply function $s_k(t)$. The overall supply allocated over $[0;\Pi]$ is Θ_k



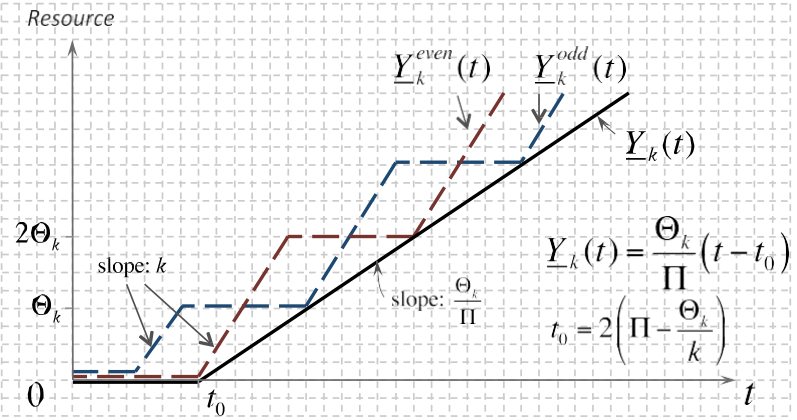


Fig. 15 The lower bound for $Y_k(t)$

$$\underline{Y}_k^{\text{odd}}(t) = p^{\text{odd}}\,\Theta_k + 2\left(\Theta_k - k\left(\Pi - r^{\text{odd}}\right)\right)_0. \qquad (14)$$

The bounds $\underline{Y}_k^{\text{odd}}(t)$, $\underline{Y}_k^{\text{even}}(t)$ are plotted in Fig. 15. Considering Eq. (4) and Fig. 15, we conclude that a valid lower bound for $Y_k(t)$ is $\underline{Y}_k(t)$ defined as

$$\underline{Y}_k(t) = \frac{\Theta_k}{\Pi}t - 2\frac{\Theta_k}{\Pi}\left(\Pi - \frac{\Theta_k}{k}\right). \qquad (15)$$

The upper bound $\overline{Y}_k(t)$ for $Y_k(t)$ is derived in a similar way. First, we observe that the function $s_k(t)$ is upper bounded by the function $\overline{s}_k(t)$ depicted in Fig. 14. Then, substituting the expression for $\overline{s}_k(t)$ into (11), we derive the upper bounds $\overline{Y}_k^{\text{even}}(t)$, $\overline{Y}_k^{\text{odd}}(t)$ for $Y_k^{\text{even}}(t)$, $Y_k^{\text{odd}}(t)$, and in the end we determine that

$$\overline{Y}_k(t) = \frac{\Theta_k}{\Pi} t, \tag{16}$$

as

$$\overline{Y}_k(t) \geq \max\left(\overline{Y}_k^{\text{even}}(t),\ \overline{Y}_k^{\text{odd}}(t)\right).$$

4 Schedulability over the GMPR interface

The GMPR interface describes the amount of computing resources provided to an application. We can then formulate a schedulability test over the GMPR.

As schedulability test for the application, we choose the extension of the test by Bertogna et al. (2009) to the PSF interface developed by Bini et al. (2009). We choose this condition because it applies to several different application schedulers such as global EDF or global FP, although it assumes constrained deadline tasks, i.e. for all tasks τ_i, $D_i \leq T_i$. While choosing other tests like the one derived in Baruah et al. (2010) would be possible, the proposed formulation has the advantage of highlighting the constraint on the interface. Thanks to the lossless transformation of a GMPR interface into a PSF (see Sect. 3.2), we can apply directly the schedulability condition developed over PSF. Below we report, for completeness, the schedulability condition in the simpler expression proposed in Lipari and Bini (2010).

Theorem 2 (Theorem 1 in Lipari and Bini 2010) *A set of sporadic tasks* $T = \{\tau_1, \ldots, \tau_n\}$ *is schedulable on a resource modeled by the PSF functions* $Y_1(t), \ldots, Y_m(t)$, *if*

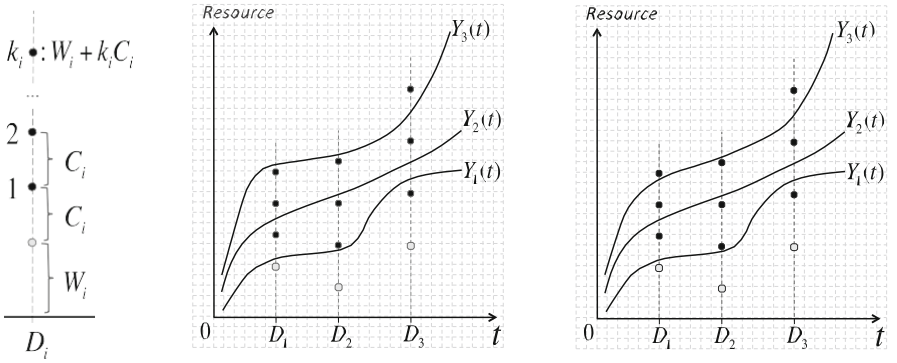
$$\bigwedge_{i=1,\ldots,n} \bigvee_{k_i=1,\ldots,m} k_i C_i + W_i \leq Y_{k_i}(D_i), \tag{17}$$

where W_i *is the maximum interfering workload that can be experienced by task* τ_i *in the interval* $[0, D_i]$, *defined as*

$$W_i = \sum_{j=1, j\neq i}^{n} \left(\left\lfloor \frac{D_i}{T_j} \right\rfloor C_j + \min\left\{ C_j, D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right\} \right), \tag{18}$$

if the application tasks are scheduled by global EDF. Instead if the application tasks are scheduled by global FP

$$W_i = \sum_{j \in \text{hp}(i)} W_{ji}, \tag{19}$$



(a) Notation **(b)** Test is true **(c)** Test is false

Fig. 16 Graphical interpretation of the PSF-based schedulability test

where $\mathrm{hp}(i)$ *denotes the set of indices of tasks with higher priority than* i*, and* W_{ji} *is the amount of interfering workload caused by* τ_j *on* τ_i*, that is*

$$W_{ji} = N_{ji} C_j + \min\{C_j, D_i + D_j - C_j - N_{ji} T_j\} \tag{20}$$

with $N_{ji} = \left\lfloor \frac{D_i + D_j - C_j}{T_j} \right\rfloor$.

To better understand the schedulability test over PSF of Theorem 2, we illustrate it graphically in Fig. 16. In this example we consider a task set T composed by $n = 3$ tasks. Each task τ_i has an amount of interference W_i, properly determined according to the local scheduling algorithm. For each task τ_i, we draw a dashed vertical line at $t = D_i$. Along this line we represent the quantity W_i denoted as a white dot, and the quantities $W_i + k_i C_i$, with $k_i \in \{1, 2, 3\}$, denoted as black dots. These dots represent the LHS of (17). Then we draw the PSF functions $Y_1(t)$, $Y_2(t)$, $Y_3(t)$ as bold continuous lines. In accordance to condition (17), task τ_i is schedulable if the kth dot is not above the Y_k, for some k.

Now consider the case depicted in Fig. 16. In that case T is schedulable as the condition (17) turns valid for $k_1 \in \{3\}$, $k_2 \in \{2, 3\}$, and $k_3 \in \{1\}$. In Fig. 16, instead, we show a case when τ_1 cannot be guaranteed by the test of Theorem 2.

Later we exploit such a schedulability condition to compute the GMPR parameters $\Theta_1, \ldots, \Theta_m$ for a given task set.

4.1 Simplification of the schedulability condition

The schedulability condition of Theorem 2 has the complexity of $O(nm)$ since it requires to check if for each task $\tau_i \in T$ exists any value $k_i \in \{1, \ldots, m\}$ satisfying the inequality (17). However, we can shrink the set of values of k_i to be tested without making any pessimistic assumption, by exploiting by the linear upper bounds of the PSF functions.

The PSF function $Y_k(t)$ can be bounded from above by

$$Y_k(t) \leq k\,t. \tag{21}$$

Substituting Eq. (21) into the condition (17), we get $k_i\,C_i + W_i \leq k_i\,D_i$ and thus

$$k_i \geq \frac{W_i}{D_i - C_i}. \tag{22}$$

Considering that k_i is integer and by defining $\bar{k}_i$ as

$$\bar{k}_i = \left\lceil \frac{W_i}{D_i - C_i} \right\rceil, \tag{23}$$

the schedulability condition (17) turns into

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k_i=\bar{k}_i,\ldots,m} k_i\,C_i + W_i \leq Y_{k_i}(D_i). \tag{24}$$

5 Determining the GMPR interface of an application

When an application $T = \{\tau_1, \ldots, \tau_n\}$ is given, it is of key importance to select an interface that can guarantee the timing constraints of the application and, at the same time, requires the minimal amount of resource. In Burmyakov et al. (2012) we proposed an algorithm to generate a GMPR interface for T assuming integer resource parameters. However, this assumption made the problem hardly tractable even for a task set with a low utilization. If instead, the interface parameters are assumed real-valued, the problem can be attacked and solved more efficiently.

Consider a set of sporadic tasks $T = \{\tau_1, \ldots, \tau_n\}$ locally scheduled by the global EDF or the global FP scheduler. In this section we describe a method to compute a GMPR interface for T: For a specified period Π and a parallelism m we find the minimal real-valued resources $\Theta_1, \ldots, \Theta_m$ such that T is schedulable over the GMPR $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$, according to Theorem 2.

Below, in Sect. 5.1 we compute the minimal necessary parallelism for a GMPR for a given application. Then, in Sect. 5.2 we compute the GMPR resource Θ_m, and in Sect. 5.3 we derive a set of techniques to reduce the computation time for Θ_m. Finally, in Sects. 5.4 and 5.5 we generalize our approach by iteratively computing the resources $\Theta_1, \ldots, \Theta_m$ for all levels of parallelism.

5.1 Minimal necessary parallelism for GMPR

No valid GMPR interface may exist for an arbitrary small parallelism. Hence, in Theorem 3 we propose a necessary and sufficient condition for the parallelism of a GMPR, assuming Theorem 2 as schedulability test.

Theorem 3 *Consider a set of sporadic tasks* $T = \{\tau_1, \ldots, \tau_n\}$ *locally scheduled by the global EDF or the global FP. Then there always exists a feasible GMPR interface for* T *with a parallelism* $m \geq \max(\bar{k}_1, \ldots, \bar{k}_n)$, *with* $\bar{k}_i$ *as in (23). However, no GMPR can satisfy the schedulability condition (17) if* $m < \max(k_1, \ldots, k_n)$.

Proof To prove the existence of a GMPR with a parallelism at least $m = \max(k_1, \ldots, k_n)$, we show that $\mu = (\Pi, \{\Pi, 2\Pi, \ldots, m\Pi\})$ is a valid GMPR interface for T. According to Eq. (4), the PSF functions for μ are

$$Y_k(t) = k\,t, \quad k = 1, \ldots, m.$$

The schedulability condition (24) over μ turns into

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k_i=\bar{k}_i,\ldots,m} k_i C_i + W_i \leq k_i D_i.$$

For each τ_i we set $k_i = \bar{k}_i$, and check that the schedulability of T over μ holds:

$$\begin{aligned}
&\left\lceil \frac{W_i}{D_i - C_i} \right\rceil D_i \geq \left\lceil \frac{W_i}{D_i - C_i} \right\rceil C_i + W_i \\
&\left\lceil \frac{W_i}{D_i - C_i} \right\rceil (D_i - C_i) \geq W_i \\
&\left\lceil \frac{W_i}{D_i - C_i} \right\rceil \geq \frac{W_i}{D_i - C_i}.
\end{aligned}$$

Thus, $\mu = (\Pi, \{\Pi, 2\Pi, \ldots, m\Pi\})$ is a valid GMPR for T.

To prove the other direction of the implication, let us denote, without loss of generality, by $\bar{k} = \max(\bar{k}_1, \ldots, \bar{k}_n)$ and by Π the task index such that $\bar{k} = \bar{k}_\Pi$. If $m < \bar{k}$, then the task τ_Π can never be guaranteed by (24). ∎

According to Theorem 3, we can only compute a GMPR interface for T with a parallelism $m \geq \max(k_1, \ldots, k_n)$.

5.2 Minimization of the overall resource

When designing an interface of a given application, our primary target is the minimization of the overall resource consumption Θ_m. Before formulating the interface design as an optimization problem, let us denote D_Θ all feasible resources $\Theta_1, \ldots, \Theta_m$ satisfying the constraints in Definition (3) of GMPR, so that:

$$(\Theta_1, \ldots, \Theta_m) \in D_\Theta \iff \begin{cases} 0 \leq \Theta_{k+1} - \Theta_k \leq \Pi \\ \Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1} \\ \Theta_k = 0, \quad k \leq 0. \end{cases} \tag{25}$$

Then we compute Θ_m subject to the schedulability test (24):

minimize Θ_m

subject to

$$(\Theta_1, \ldots, \Theta_m) \in D_\Theta$$
$$\forall i = 1, \ldots, n,\ \exists k_i \in \{\overline{k}_i, \ldots, m\} :\ Y_{k_i}(D_i, \Theta_1, \ldots, \Theta_{k_i}) \geq k_i C_i + W_i. \tag{26}$$

To solve the optimization problem (26), we first have to exclude the $\exists$-quantifiers from it. Therefore, we propose to solve (26) for each possible combination $(k_1, \ldots, k_n)$, with $k_i \in \{\overline{k}_i, \ldots, m\}$, and then to choose the minimal Θ_m over all cases. Below we provide a detailed description of this approach.

Let us denote possible combinations $(k_1, \ldots, k_n)$ as K_m so that

$$K_m = \{(k_1, \ldots, k_n) |\ k_i = \overline{k}_i, \ldots, m\}.$$

For a specific choice of $(k_1, \ldots, k_n) \in K_m$ the optimization problem (26) turns into

minimize Θ_m

subject to

$$(\Theta_1, \ldots, \Theta_m) \in D_\Theta$$
$$\forall i = 1, \ldots, n \quad Y_{k_i}(D_i, \Theta_1, \ldots, \Theta_{k_i}) \geq k_i C_i + W_i \tag{27}$$

To solve (27), we employ the Matlab optimization toolbox. Let us denote the solution of (27) as $\Theta_m(k_1, \ldots, k_n)$, if any exists. Then we choose the minimal Θ_m over K_m as

$$\Theta_m = \min_{(k_1, \ldots, k_n) \in K_m} \Theta_m(k_1, \ldots, k_n). \tag{28}$$

For some combination $(k_1, \ldots, k_n)$ the optimization problem (27) may have no feasible solution. However, from Theorem 3, there exists at least one case $(k_1, \ldots, k_n) \in K_m$ such that (26) becomes feasible. Hence, the minimum of (28) is well defined.

Next, in Sect. 5.3 we propose a method to reduce the run-time of the optimization problem (28) by reducing the search space for the resources $\Theta_1, \ldots, \Theta_m$ and shrinking the enumeration space K_m.

5.3 Search space for the GMPR resources

To reduce the search space for the GMPR resources $\Theta_1, \ldots, \Theta_m$, we first formulate a set of preliminary constraints in Lemma 2.

Lemma 2 *All feasible GMPR resources $(\Theta_1, \ldots, \Theta_m) \in D_\Theta$ defined by (25) satisfy the following constraints:*

$$j < k \quad \Rightarrow \quad \Theta_k \leq \frac{k}{j}\Theta_j \tag{29}$$

Proof Let us decompose Θ_k as

$$\Theta_k = \sum_{\ell=1}^{k-1} (\Theta_\ell - \Theta_{\ell-1}) + (\Theta_k - \Theta_{k-1}). \tag{30}$$

From (25), each feasible case $(\Theta_1, \ldots, \Theta_m) \in D_\Theta$ satisfies the constraint

$$\Theta_\ell - \Theta_{\ell-1} \geq \Theta_k - \Theta_{k-1} \qquad \ell < k. \tag{31}$$

Substituting (31) into (30) gives us

$$\Theta_k \leq \frac{k}{k-1}\Theta_{k-1}. \tag{32}$$

Applying mathematical induction to the expression above, we get (29):

$$\begin{aligned}\Theta_k &\leq \frac{k}{k-1}\Theta_{k-1} \leq \frac{k}{k-1}\left(\frac{k-1}{k-2}\Theta_{k-2}\right) \\ &\leq \frac{k}{k-1}\frac{k-1}{k-2}\cdots\left(\frac{k-i+1}{k-i}\Theta_{k-i}\right) = \frac{k}{k-i}\Theta_{k-i}\end{aligned} \tag{33}$$

with $i = 1, \ldots, k-1$. ∎

Let T be a schedulable task set over a GMPR interface $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ according to condition (24). For each task τ_i, let us denote by k^*_i the smallest k_i, in $\{k_i, \ldots, m\}$, for which the condition (24) is true. Below, we compute a reduced search space for the GMPR resources $\Theta_1, \ldots, \Theta_m$ by exploiting the lower and the upper bounds for $Y_k(t)$ derived in Sect. 3.3:

$$Y_k(t) \geq \underline{Y}_k(t) = \frac{\Theta_k}{\Pi}t - 2\frac{\Theta_k}{\Pi}\left(\Pi - \frac{\Theta_k}{k}\right) \tag{34}$$

$$Y_k(t) \leq \overline{Y}_k(t) = \frac{\Theta_k}{\Pi}t. \tag{35}$$

Consider a task τ_i. The test (24) is false for any $k_i < k^*_i$:

$$Y_{k_i}(D_i) < k_i C_i + W_i.$$

Substituting the lower bound (34) for $Y_k(t)$ into the condition above, we get the quadratic inequality

$$\left(\frac{2}{k\Pi}\right)\Theta_k^2 + \left(\frac{D_i}{\Pi} - 2\right)\Theta_k - (k C_i + W_i) < 0$$

with a solution

$$\overline{\Theta}_k{}^* = \frac{k\Pi}{4}\left(\sqrt{\left(\frac{D_i}{\Pi}-2\right)^2+\frac{8}{k\Pi}(k\,C_i+W_i)}-\left(\frac{D_i}{\Pi}-2\right)\right),$$
$$\Theta_k < \overline{\Theta}_k{}^*. \tag{36}$$

By applying Lemma 2, the constraint above yields the following upper bound for the resource Θ_k denoted as $\overline{\Theta}_k$:

$$\overline{\Theta}_k = \begin{cases} \Pi, & k = k_i^* = 1, \\ \min\left(\overline{\Theta}_k{}^*, \Pi\right), & k < k_i^*,\ k = 1, \\ \min\left(\overline{\Theta}_k{}^*, \frac{k}{k-1}\overline{\Theta}_{k-1}\right), & k < k_i^*,\ k \neq 1, \\ \frac{k}{k-1}\overline{\Theta}_{k-1}, & k \geq k_i^*,\ k \neq 1. \end{cases} \tag{37}$$

with $\overline{\Theta}_k{}^*$ defined by (36).

The test (24) is true for $k = k_i^*$. Applying the upper bound (35) for $Y_k(t)$ in (24), we get

$$\Theta_{k_i^*} \geq \frac{\Pi}{D_i}\left(k_i^*\,C_i + W_i\right), \tag{38}$$

that, together with Lemma 2, yields the following lower bound for the resource Θ_k denoted as $\underline{\Theta}_k$:

$$\underline{\Theta}_k = \begin{cases} \frac{k}{k_i^*}\Theta_{k_i^*}, & \text{if } k < k_i^*, \\ \frac{\Pi}{D_i}\left(k_i^* C_i + W_i\right), & \text{otherwise.} \end{cases} \tag{39}$$

Let us denote the search space for task τ_i as $S_\Theta(\tau_i, k_i^*)$ so that

$$S_\Theta(\tau_i, k_i^*) = \left\{(\Theta_1, \ldots, \Theta_m) \mid \underline{\Theta}_k \leq \Theta_k \leq \overline{\Theta}_k\right\},$$

where $\underline{\Theta}_k$, $\overline{\Theta}_k$ are computed according to (39), (37). The resulting search space for a task set T is then defined as

$$S_\Theta(T, k_1^*, \ldots, k_n^*) = D_\Theta \bigcap_{i=1,\ldots,n} S_\Theta(\tau_i, k_i^*), \tag{40}$$

where D_Θ denotes all feasible GMPR resources $\Theta_1, \ldots, \Theta_m$ satisfying the constraint (25).

Consequently, a case $(k_1, \ldots, k_n) \in K_m$ is feasible if it results in a non-empty search space

$$S_\Theta(T, k_1, \ldots, k_n) \neq \emptyset, \tag{41}$$

otherwise it can be excluded from K_m. According to our experiments, this approach drastically reduces the size of K_m: the reduction is by more than 99,99 % in an average case.

5.4 Iterative computation of the supply at lower parallelism

In Sect. 5.2 we computed the GMPR overall resource Θ_m, only. To complete the GMPR specification, we now need to compute the remaining resources $\Theta_{m-1}, \ldots, \Theta_1$, which should be provided at lower concurrencies.

We propose to compute the resource Θ_k recursively, after computing the resources $\Theta_m, \ldots, \Theta_{k+1}$. To do so, we simply update the optimization problem (26) by setting the objective function to minimize Θ_k, and by placing the previously found values for $\Theta_m, \ldots, \Theta_{k+1}$ into the optimization constraints.

In this case, rather than repeating the enumeration of K_m to solve the optimization problem (26) for Θ_k, we can further shrink the enumeration space by considering among the feasible cases $(k_1, \ldots, k_n)$ only those ones, which yield the minimal value for Θ_{k+1}. Hence the reduced enumeration space K_k for Θ_k is given by the equation

$$\begin{aligned} &K_k \subseteq K_{k+1} : \\ &\forall (k_1, \ldots, k_n) \in K_{k+1} : \Theta_{k+1}(k_1, \ldots, k_n) = \Theta^*_{k+1} \rightarrow (k_1, \ldots, k_n) \in K_k, \end{aligned} \quad (42)$$

where Θ^*_{k+1} denotes the found minimal value for Θ_{k+1}.

The computation time for Θ_k is significantly lower compared to Θ_{k+1}, what is due to a shrunk enumeration space K_k, and a lower number of optimization variables.

5.5 Algorithm to compute GMPR

Finally, we conclude by proposing an algorithm that assigns the minimal GMPR resources $\Theta_1, \ldots, \Theta_m$ such that a given task set T is schedulable over an interface. As a schedulability condition, we choose the one in (24). We recall that the period Π and the parallelism m for a searching GMPR are given.

Step 1: For each task τ_i compute $\overline{k}_i$ as defined in (23).

Step 2: Check whether the necessary condition for m (Theorem 3) is met:

$$m \geq \max(\overline{k}_1, \ldots, \overline{k}_n).$$

If the condition above is violated, report the nonexistence of a valid GMPR interface for T with a specified m, and terminate the algorithm.

Step 3: Generate the enumeration space K_m such that

$$K_m = \{(k_1, \ldots, k_n) | \; k_i = \overline{k}_i, \ldots, m\}$$

satisfying the condition (41).

Step 4: Compute Θ_m: for each case $(k_1, \ldots, k_n) \in K_m$ determine the search space according to (40), solve the optimization problem (26), and then choose the minimal Θ_m over K_m.

Step 5: Compute Θ_k recursively after computing $\Theta_m, \ldots, \Theta_{k+1}$:

(a) Define K_k from Eq. (42) so that any $(k_1, \ldots, k_n) \in K_{k+1}$ resulted in the optimal Θ_{k+1} is included into K_k.

(b) Substitute the computed values for $\Theta_m, \ldots, \Theta_{k+1}$ into the optimization constraints of (26), and minimize Θ_k subject to these constraints. Solve the resulting optimization problem over K_k, and then choose the minimal Θ_k.

Step 6: Follow the Step 5 to compute all the resources $\Theta_{m-1}, \ldots, \Theta_1$. In the end, $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ is the sought-for interface for T having the minimized resources $\Theta_1, \ldots, \Theta_m$.

Algorithm complexity. The complexity of the algorithm to compute a GMPR interface depends on the complexity of the optimization problem (27). Due to the presence of the PSF function $Y_k(t)$, which is non-convex, the optimization problem (27) is non-convex. Although the complexity of such problems remains to be an open problem in the literature, it is generally considered as exponential, until the opposite is proved (Ausiello et al. 2008). Thus, the resulting complexity of the proposed algorithm is exponential.

Customized computation of GMPR. We proposed an algorithm to compute a GMPR interface having the minimized resources $\Theta_1, \ldots, \Theta_m$. At the same time, our approach is easily extendable for computing a customized GMPR interface, which meets specific user requirements (e.g. a constraint on the maximum resource fraction to be provided at each concurrency), rather than simply having the minimized consumed resources. In this case the custom constraints should be incorporated in the optimization problem (27).

6 Scheduling GMPR interfaces

Once the resource demand of each component is abstracted by an interface, these interfaces should be scheduled upon a hardware platform. To schedule GMPR interfaces, we now introduce a notion of interface tasks. A set of *interface* tasks for a GMPR interface $(\Pi, \{\Theta_1, \ldots, \Theta_m\})$ is comprised of m implicit-deadline $(D = T)$ periodic tasks such that:

$$T' = \{\tau'_1 = (C'_1, \Pi), \ldots, \tau_m = (C'_m, \Pi)\}, \tag{43}$$

where the execution time equals to

$$C'_k = (\Theta_k - \Theta_{k-1}).$$

(We set $\Theta_0 = 0$ for convenience).

The interface tasks in T' have an identical period T equal to the period of a GMPR interface Π. Clearly, the overall resource demand of T' over a period Π is $\sum_{k=1}^{m} C'_k = \Theta_m$.

To schedule GMPR interfaces, we first transform each one into interface tasks following (43), and then we employ any suitable policy to schedule the resulting periodic tasks.

The notion of interface tasks supports another important property for hierarchical systems, which is called *composability:* by the given GMPR interfaces of child components we can compute a GMPR interface of a parent component.

7 Evaluation of GMPR

In this section, we compare the amount of resource used by GMPR and MPR to feasibly schedule randomly generated task sets. For each experiment setting, we compute the minimal GMPR and MPR interfaces by employing the algorithm described in Sect. 5.5.

The algorithm to compute interfaces and the scenarios of the experiments have been implemented in Matlab, and they are publicly available at https://sites.google.com/site/artemburmyakov/home/papers.

7.1 Task set generation

Synthetic task sets $\mathcal{T} = \{\tau_i = (C_i, T_i)\}$ are randomly generated by specifying the total task set utilization $U_{\mathcal{T}}$, the maximum individual task utilization $U_{\max}$, and the ratio between the maximum and the minimum periods $T_{\max}/T_{\min}$. In our random generation method, the number of tasks in $\mathcal{T}$ is not fixed. Instead, it is implicitly determined as the total utilization of $\mathcal{T}$ reaches the specified value $U_{\mathcal{T}}$.

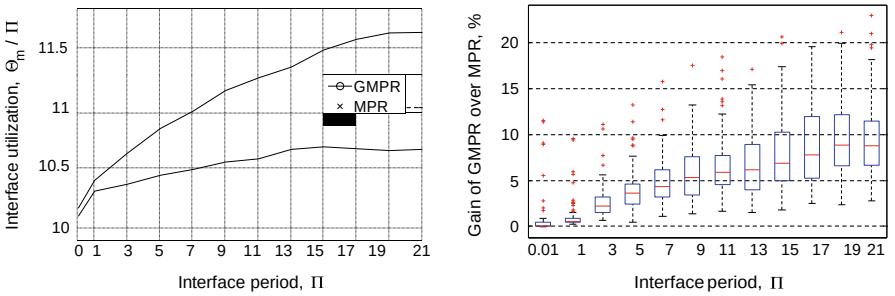
The minimum period $T_{\min}$ is set to 20 and all task periods are randomly generated so that the specified ratio $T_{\max}/T_{\min}$ is not violated.

7.2 Experiments: the resource gain

We evaluate the resource gain of GMPR over MPR for the parameters listed in Table 3. In each experiment, we compare the interfaces utilization as one parameter varies, while the rest are left equal to the default values reported in Table 3.

Table 3 Key parameters: default values

Parameter	Default value
Task set utilization, $U_{\mathcal{T}}$	2.5
Maximum individual task utilization, $U_{\max}$	0.3
Minimum task period, $T_{\min}$	20
Ratio between the maximum and the minimum task periods, $T_{\max}/T_{\min}$	10
Interface period, Π	20
Parallelism increment, Δm	3



(a) Interface utilizations **(b)** GMPR gain over MPR

Fig. 17 Case: $U_\mathcal{T} = 2.5$, $U_{\max} = 0.3$, $\frac{T_{\max}}{T_{\min}} = 10$, and $\Delta m = 3$

In each experiment, we randomly generate at least 200 task sets, and then we plot the average interface utilizations $\frac{\Theta_m}{\Pi}$ among these task sets, as well as the relative GMPR gain.

For each generated task set, the interface parallelism is set to

$$m = m_{\min} + \Delta m,$$

where $m_{\min}$ is the minimal parallelism defined by Theorem 3, and the increment Δm is varied through the experiments.

The gain of GMPR over MPR is computed as

$$\text{gain}_{\text{GMPR}} = \frac{U_{\text{MPR}} - U_{\text{GMPR}}}{U_{\text{GMPR}}},$$

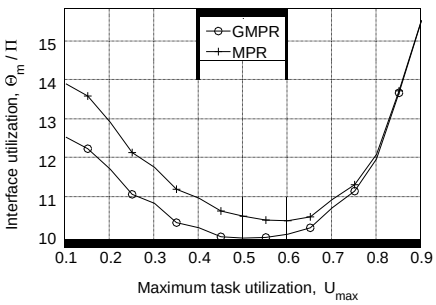
where U_{MPR} denotes the MPR utilization $\frac{\Theta}{\Pi}$, and U_{GMPR} is the GMPR utilization $\frac{\Theta_m}{\Pi}$.

7.2.1 Varying interface period Π

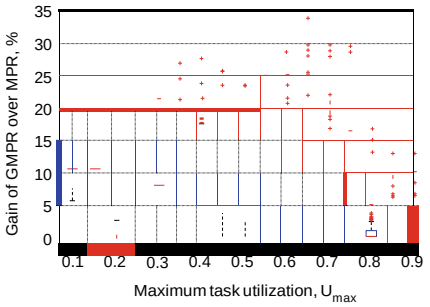
First, we analyze the GMPR gain for a varying interface period Π. The resulting utilizations of both GMPR and MPR interfaces are plotted in Fig. 17a. For such settings the average GMPR gain is in the order of 5–10 %, and it increases for the increasing Π.

The observed trend for gain increase is justified by an expanding search space for the GMPR resources together with Π, which results in a higher degree of freedom for GMPR over MPR.

In Fig. 17b we also illustrate the gain variability using a boxplot diagram (McGill et al. 1979). In this diagram, the central horizontal mark on each box is the median for the observed gain, the horizontal edges of the box are the 25th and the 75th percentiles, the dashed lines extend to the most extreme gains covering 99.3 % observed cases, and the outliers are depicted individually as crosses.

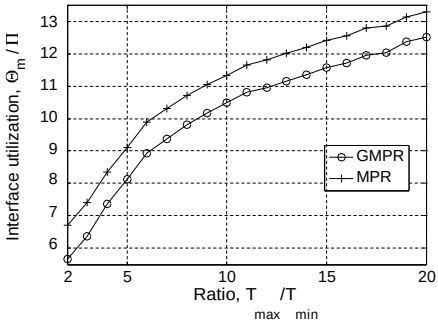


(a) Interface utilizations

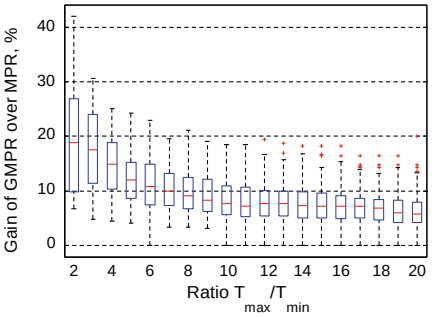


(b) GMPR gain over MPR

Fig. 18 Case: $U_T = 2.5$, $\frac{T_{max}}{T_{min}} = 10$, $\Delta m = 3$, $\Pi = 20$



(a) Interface utilizations



(b) GMPR gain over MPR

Fig. 19 Case: $U_T = 2.5$, $U_{max} = 0.3$, $\Delta m = 3$, $\Pi = 20$

7.2.2 Varying maximum task utilization U_{max}

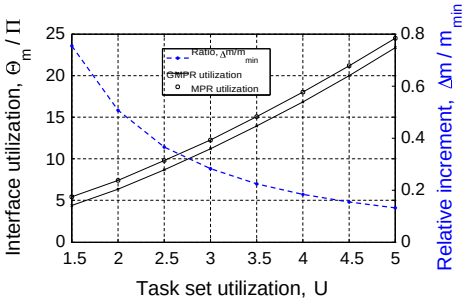
In the next experiment, we explore the dependency of the interface utilization on the weight of individual tasks, by varying the maximum task utilization U_{max}. The results are reported in Fig. 18. The interface utilization is minimal for U_{max} closer to 0.5–0.6, and it drastically increases for U_{max} tending to 0 or 1. We believe that this behavior is influenced by our choice of schedulability test (Lipari and Bini 2010) used to compute interfaces.

The GMPR gain itself is maximized for lower U_{max}, reaching up to 10–15 %, and the gain vanishes as U_{max} tends to one.

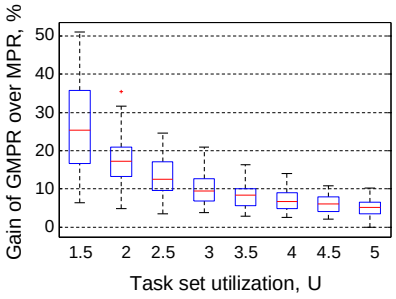
7.2.3 Varying ratio $\frac{T_{max}}{T_{min}}$

In Fig. 19 we provide the experimental results for a varying ratio T_{max}/T_{min}. The interface utilization significantly increases together with the ratio T_{max}/T_{min}, but the GMPR gain is maximized for lower T_{max}/T_{min}, reaching up to 15–25 %.

The observed utilization increase for both GMPR and MPR interfaces with respect to T_{max} is justified by the nature of the chosen schedulability test, described in

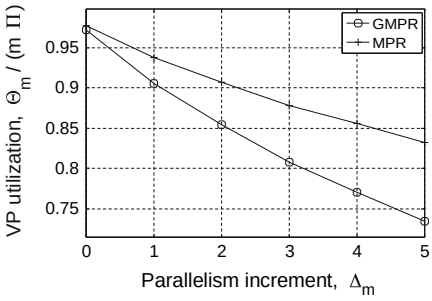


(a) Interface utilizations

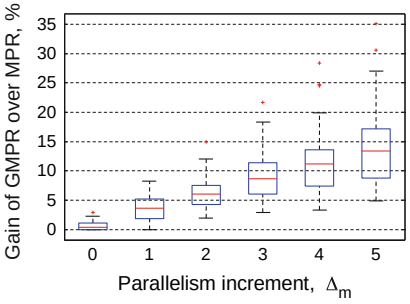


(b) GMPR gain over MPR

Fig. 20 Case: $U_{\max} = 0.3$, $\frac{T_{\max}}{T_{\min}} = 10$, $\Delta m = 3$, $\Pi = 20$



(a) Average utilization of a virtual processor



(b) GMPR gain over MPR

Fig. 21 Case: $U_T = 2.5$, $U_{\max} = 0.3$, $\frac{T_{\max}}{T_{\min}} = 10$, $\Pi = 20$

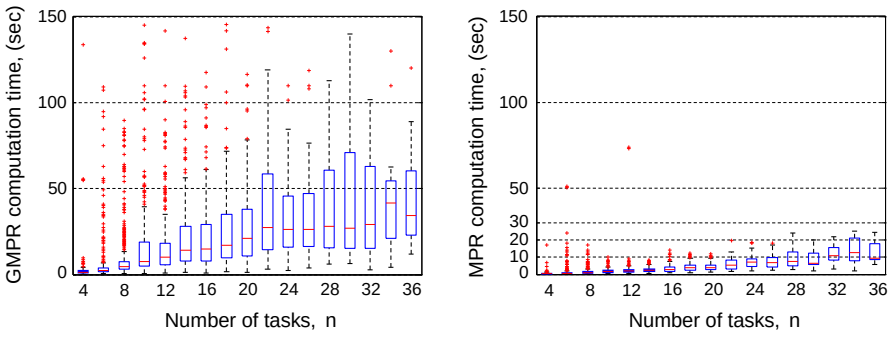
Theorem 2. In fact, for fixed parameters U and $U_{\max}$, increasing task periods result in a higher interference of jobs accross the deadline window (so called "carry-in", defined by Eq. (18)), increasing the overall utilization of an interface.

7.2.4 Varying task set utilization U_T

We also analyze the gain of GMPR over MPR as the task set utilization U_T varies. The results are depicted in Fig. 20. In this case the gain decreases for increasing U_T. A reason for such behavior is that, although the absolute parallelism increment Δm remains constant, its relative proportion $\Delta m / m_{\min}$ decreases (see Fig. 20a), due to $m_{\min}$ increasing with U_T, resulting in a reduced scope for parallelism.

7.2.5 Varying parallelism increment Δm

In the last experiment we analyze the relation between an average utilization of a virtual processor, $\frac{\Theta_m}{m\Pi}$, and the parallelism increment Δm. The results are provided in Fig. 21. As expected, an average utilization of a virtual processor reduces for increasing parallelism of an interface.



(a) GMPR **(b)** MPR

Fig. 22 Computation time for GMPR and MPR. $\Delta m = 3$, the percentiles are 25 and 75 %

The GMPR gain itself increases together with m. Such a dependency is expectable since an increased m leads to a higher degree of freedom for GMPR over MPR, allowing a larger margin to minimize the consumed resource.

We also notice that the utilization of both GMPR and MPR is minimal for $m = 0$, and it increases with m. This observation confirms the result of Shin et al. (2008) regarding the minimum utilization of a multiprocessor interface, and moreover, this result looks to be independent of the schedulability test used to compute an interface.

7.3 Analysis of the run-time for the interface generation

In this experiment we analyze a set of performance metrics for the algorithm to compute a GMPR interface, based on the solution of the resource minimization problem, as described in Sect. 5. The algorithm has been implemented in the Matlab 2010 environment. The experiment has been performed on a hardware platform with the following specifications:

- Processor: Intel(R) Core(TM) i7-3630QM CPU @ 2.40 GHz
- Operating memory (RAM): 8,00 GB
- System type: 64-bit

In Tables 4 and 5 and in Fig. 22 we report the measured run-time for the GMPR computation, for a varying number of tasks n and the parallelism m. Although the proposed algorithm to compute GMPR is considered to have an exponential complexity, the results show a linear increase of the algorithm run-time over n and m. This result confirms the effectiveness of the search space reduction mechanism derived in Sect. 5.3.

The computation time for MPR is 2–5 times lower compared to GMPR, what is due to a simpler PSF function in the optimization constraints of (27).

In addition, we have evaluated the performance of several optimization solvers available in the Matlab, as they significantly affect the overall run-time of the GMPR computation. Although the interior-point algorithm finds a more precise solution for (27), we have chosen the active-set algorithm for its 5–100 times faster performance, and

Table 4 The performance metrics for $m = 5$

n	GMPR time (s)	MPR time (s)	Size of K_m	Size reduction (times)
1–10	<10	<1	1–25	1–50
11–18	1–20	1–10	10–50	10^2–10^4
19–25	10–50	1–15	50–120	10^3–10^6
26–30	25–100	5–25	100–200	10^4–10^7
31–35	50–150	10–50	100–300	10^5–10^{10}

Table 5 The performance metrics for $m = 10$

n	GMPR time (s)	MPR time (s)	Size of K_m	Size reduction (times)
1–10	<10	<1	5–50	10–1,000
11–18	5–100	1–10	10–100	10^2–10^6
19–25	50–250	10–50	50–150	10^5–10^9
26–30	100–300	20–100	<400	10^8–10^{13}
31–35	100–500	25–150	<400	10^8–10^{16}

its acceptable error which is at most 0.05 %, and the failure ratio of at most 2 % (in case the active-set fails, we employ the interior-point instead).

In Tables 4 and 5 we report the size of the reduced search space K_m defined by Eq. (41). In each case, this value corresponds to the number of optimization problems (27) to be resolved in order to determine the minimal GMPR interface. To analyze the efficiency of the search space reduction algorithm, proposed in Sect. 5.3, we also indicate the relative size reduction of K_m compared to the original search space defined by the schedulability test (24). We observe an exponential size reduction of K_m over a number of tasks n and an interface parallelism m.

8 Conclusion

Motivated by the need to save resource, we introduced the GMPR model, as an interface of a multiprocessor virtual platform, and proposed a schedulability test for a set of sporadic tasks over GMPR.

Since GMPR is a generalization of the previously proposed MPR model (Shin et al. 2008), it can consume at most as much as MPR. Our evaluation confirmed that the resource gain of GMPR over MPR increases together with the period and the parallelism of an interface. The GMPR gain is especially noticable for task sets with smaller individual tasks' utilizations and a shorter range of tasks' periods.

We also addressed the problem of computing a GMPR interface for a given set of sporadic tasks, objecting to minimize the overall amount of resource required by an interface. This problem was modeled as an optimization problem, which turned to be efficiently solvable thanks to the derived tight lower and upper bounds for the solution search space. Such an approach is easily extendable to compute a customized

GMPR interface, which meets specific user requirements rather than simply has the minimized consumed resource.

For the future, our primary objective is to explore the flexibility of the GMPR model in deriving a tighter schedulability analysis, specifically dedicated for it. We also consider extending GMPR to a case of asynchronous virtual processors with different periods.

Acknowledgments This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and European Regional Development Fund (ERDF) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within Project Ref. FCOMP-01-0124-FEDER-022701; by FCT and COMPETE (ERDF), within REHEAT and REGAIN Project, Ref. FCOMP-01-0124-FEDER-010045 and FCOMP-01-0124-FEDER-020447 respectively; by FCT and the EU ARTEMIS JU funding, within RECOMP project—ref. ARTEMIS/0202/2009, JU Grant Number 100202; and by FCT and European Social Fund (ESFE) through Portuguese Human Potential Operational Program (POPH), under Ph.D. Grant SFRH/BD/71368/2010. The research leading to these results was supported by the Marie Curie Intra European Fellowship within the 7th European Community Framework Programme and by the Linneaus Center LCCC.

References

Almeida L, Pedreiras P, Fonseca JAG (2002) The FTT-CAN protocol: why and how. IEEE Trans Ind Electron 49(6):1189–1201

Ausiello G, Crescenzi P, Kann V, Gambosi G, Marchetti-Spaccamela A, Protazi M (2008) Complexity and approximation: combinatorial optimization problems and their approximability properties. Springer, Berlin

Baruah S, Bonifaci V, Marchetti-Spaccamela A, Stiller S (2010) Improved multiprocessor global schedulability analysis. Real Time Syst J 46:3–24

Bertogna M, Cirinei M, Lipari G (2009) Schedulability analysis of global scheduling algorithms on multiprocessor platforms. IEEE Trans Parallel Distrib Syst 20(4):553–566

Bini E, Bertogna M, Baruah S (2009) Virtual multiprocessor platforms: specification and use. In: Proceedings of the 30th IEEE real-time systems symposium, Washinghton, DC, USA, pp 437–446

Burmyakov A, Bini E, Tovar E (2012) The generalized multiprocessor periodic resource interface model for hierarchical multiprocessor scheduling. In: Proceedings of the 20th international conference on real-time and network systems (RTNS), pp 131–139

Chang Y, Davis R, Wellings A (2008) Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning. Technical report YCS 432. University of York. http://www.cs.york.ac.uk/ftpdir/reports/2008/YCS/432/YCS-2008-432.pdf

Deng Z, Liu JWS (1997) Scheduling real-time applications in open environment. In: Proceedings of the 18th IEEE real-time systems symposium, San Francisco, CA, USA, pp 308–319

Easwaran A, Anand M, Lee I (2007) Compositional analysis framework using EDP resource models. In: Proceedings of the 28th IEEE international real-time systems symposium. IEEE Computer Society, Tucson, pp 129–138. http://dx.doi.org/10.1109/RTSS.2007.17

Feng X, Mok AK (2002) A model of hierarchical real-time virtual resources. In: Proceedings of the 23rd IEEE real-time systems symposium, Austin, TX, USA, pp 26–35

Fisher N, Dewan F (2009) Approximate bandwidth allocation for compositional real-time systems. In: Proceedings of the 21st Euromicro conference on real-time systems, Dublin, Ireland, pp 87–96

Holman P, Anderson JH (2006) Group-based pfair scheduling. Real Time Syst 32(1–2):125–168

Khalilzad NM, Behnam M, Nolte T (2012) Exact and approximate supply bound function for multiprocessor periodic resource model: unsynchronized servers. In: Proceedings of CRTS 2012

Kuo TW, Li CH (1999) Fixed-priority-driven open environment for real-time applications. In: Proceedings of the 20th IEEE real-time systems symposium, Phoenix, AZ, USA, pp 256–267

Kuo TW, Lin K, Wang Y (2000) An open real-time environment for parallel and distributed systems. In: Proceedings of the 20th international conference on distributed computing systems, Taipei, Taiwan, pp 206–213

Leontyev H, Anderson JH (2008) A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In: Proceedings of the 20th Euromicro conference on real-time systems, Czech Republic, Prague, pp 191–200

Lipari G, Bini E (2003) Resource partitioning among real-time applications. In: Proceedings of the 15th Euromicro conference on real-time systems, Porto, Portugal, pp 151–158

Lipari G, Bini E (2010) A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In: Proceedings of the IEEE real-time systems symposium

McGill R, Tukey JW, Larsen WA (1979) Variations of boxplots. Am Stat 32:12–16

Mercer CW, Savage S, Tokuda H (1994) Processor capacity reserves: operating system support for multimedia applications. In: Proceedings of IEEE international conference on multimedia computing and systems, Boston, MA, USA, pp 90–99

Moir M, Ramamurthy S (1999) Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In: Proceedings of the 20th IEEE real-time systems symposium, Phoenix, AZ, USA, pp 294–303

Parekh AK, Gallager RG (1993) A generalized processor sharing approach to flow control in integrated services networks: the single-node case. IEEE/ACM Trans Netw 1(3):344–357

Shin I, Lee I (2003) Periodic resource model for compositional real-time guarantees. In: Proceedings of IEEE real-time systems symposium (RTSS), pp 2–13

Shin I, Easwaran A, Lee I (2008) Hierarchical scheduling framework for virtual clustering of multiprocessors. In: Proceedings of the 20th Euromicro conference on real-time systems conference (ECRTS'08)

Stoica I, Abdel-Wahab H, Jeffay K, Baruah SK, Gehrke JE, Plaxton CG (1996) A proportional share resource allocation algorithm for real-time, time-shared systems. In: Proceeding of the 17th IEEE real time system symposium, Washington, DC, USA, pp 288–299