Partitioning and Interface Synthesis in Hierarchical Multiprocessor Real-Time Systems

Alessandro Biondi
Scuola Superiore Sant'Anna
Pisa, Italy
alessandro.biondi@sssup.it

Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
giorgio.buttazzo@sssup.it

Marko Bertogna
University of Modena and Reggio Emilia
Modena, Italy
marko.bertogna@unimore.it

ABSTRACT

Hierarchical scheduling is an effective approach developed to support the integration of independently developed applications on the same computing platform. In particular, the M-BROE framework has been recently proposed and analyzed to efficiently support component-based development on multiprocessor platforms through the virtual multiprocessor abstraction implemented by reservation servers, in the presence of shared resources. However, the problems of partitioning applications to virtual processors and defining reservation parameters were not addressed.

This paper fills this gap by proposing a design methodology as an optimization problem for partitioning applications to virtual processors, performing a synthesis of the component interface and allocating virtual processors to physical processors. Experimental results are also presented to evaluate the proposed methodology.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION BASED SYSTEMS**]: Real-time and embedded systems; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

1. INTRODUCTION

In several applications domains, the exponential growth of software functions imposed by the market is pushing software developers towards a hierarchial design approach, where multiple independently developed applications are executed on the same hardware platform. For example, in automotive systems such a hierarchical approach has already begun with the objective of containing the total number of electronic control units (ECUs) installed in a car, which implies a significant reduction of used space, weight, energy, and cost [24].

On the other hand, the concurrent execution of multiple applications on the same hardware generates new problems that must be solved in order to make the hierarchical approach effectively usable by the industry. One of the most serious problems occurring when multiple components share the same resources is caused by the reciprocal interference among them, which may introduce unbounded delays and cause unpredictable performance degradation [25].

An efficient kernel method for reducing the interference among concurrent applications is the resource reservation mechanism [1,23], according to which each application is executed within a dedicated processor partition, implemented by a *reservation server*. A reservation server S_k is a time provisioning mechanism that allocates a budget Q_k for the application every period P_k. In this case, the bandwidth reserved to an application results to be $\alpha_k = Q_k/P_k$. The reservation mechanism must guarantee that the served application receives a given fraction of the processor bandwidth, but at the same time it cannot consume more than the allocated amount, thus protecting the other applications from possible overruns (*temporal isolation*).

When application tasks make use of mutually exclusive resources shared between reservations (such as I/O devices or global memory buffers) the isolation property could be broken when the server budged expires within a critical section. In this case, in fact, an extra delay would be added to the tasks blocked on the same resource to wait not only for the release of the lock, but also for the next budget replenishment.

To solve this problem, various approaches have been proposed in the literature [8,9,11,18]. Among them, thanks to an improved schedulability analysis [15], the BROE protocol [11] proposed by Bertogna, Fisher and Baruah has been found to perform best.

The BROE protocol, originally developed for uniprocessor systems, has been recently extended by Biondi et al. [14] into M-BROE to support the development of component-based hierarchial systems on multiprocessor platforms in the presence of shared resources. In the M-BROE framework (reviewed in Section 2), the tasks of a software component are statically allocated to virtual processors (implemented via reservation servers), which are in turn allocated to the physical processors at component-integration time. The resulting infrastructure relies on partitioned hierarchical scheduling and non-preemptive FIFO spin locks to regulate the access to shared resources.

RTNS '16 October 19-21, 2016, Brest, France

ACM 978-1-4503-4787-7/16/10 ...$15.00.
http://dx.doi.org/10.1145/2997465.2997489

Although the authors fully characterized the M-BROE protocol and provided the schedulability analysis of components on given reservation servers, the problems of partitioning applications on virtual processors and defining reservation parameters were not addressed.

Other authors solved task partitioning in multicore systems using *Integer Linear Program* (ILP) formulations [5, 7], but without considering reservations, nor resource sharing. Bini et al. [12] addressed the problem of partitioning parallel applications upon reservation servers for platform virtualization and Khalilzad et al. [21] studied the problem allocating component interfaces in multiprocessor systems under partitioned EDF scheduling, but in both the works no resource sharing has been considered. Wieder and Brandenburg [27] presented an optimal ILP-based partitioning strategy for fixed priority scheduling with shared resources and Al-Bayati et al. [2] addressed the same problem using heuristic approaches.

However, none of these works addressed partitioning methodologies under reservation servers taking resource sharing into account. Resource sharing further complicates task partitioning problems (an example will be shown in Section 4), determining the need for ad-hoc approaches that explicitly take into account blocking times.

This paper fills this gap by proposing the following contributions.

Contributions.

- A methodology based on a *Mixed-Integer Linear Program* (MILP) formulation is proposed for partitioning applications on virtual processors taking shared resources into account. Once an allocation is found, the synthesis of component interfaces is performed to find the optimal reservation parameters that guarantee the schedulability of the task set.
- An MILP formulation is presented for allocating virtual processors to physical processors depending on the component interfaces.

Paper structure. The rest of the paper is organized as follows: Section 2 presents the system model and the M-BROE framework; Section 3 summarizes the schedulability results derived for the M-BROE framework; Section 4 describes the methodology to address task partitioning and virtual processors design; Section 5 reports an MILP formulation for integrating the virtual processors of all components upon the physical platform; Section 6 presents some experimental results for evaluating the proposed methodology; Section 7 concludes the paper.

2. FRAMEWORK AND MODELING

2.1 System model

We consider a system composed of N software components $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$, also referred to as applications. Each *software component* Γ_k consists of a set $\mathcal{T}(\Gamma_k)$ of n_k real-time periodic or sporadic tasks. Each task $\tau_i \in \mathcal{T}(\Gamma_k)$ is characterized by a worst-case execution time (WCET) C_i, a period (or minimum interarrival time) T_i, and a relative deadline D_i.

The system runs on a multiprocessor platform consisting of M identical processors, each denoted as $\mathcal{P}_m$, with $m = 1, \ldots, M$. Each component Γ_k is statically partitioned over $\mathcal{M}$ virtual processors $S_j^k, j = 1, \ldots, \mathcal{M}$, each implemented by a reservation server characterized by a budget Q_j^k and a period P_j^k. The ratio $\alpha_j^k = Q_j^k / P_j^k$ is referred to as the reservation bandwidth. The virtual processor at which a task $\tau_i \in \mathcal{T}(\Gamma_k)$ is assigned is denoted as $\mathcal{S}(\tau_i)$.

Each component Γ_k is abstracted through a *component interface* consisting of $\mathcal{M}$ pairs (Q_j^k, P_j^k), with $j = 1, \ldots, \mathcal{M}$, representing budget and period of the reservation server associated with S_j^k. Note that, to keep the interface simple, no information related to tasks and resources is exported.

A *component integrator* is responsible for admitting or rejecting applications and statically assigning each reservation server to a specific physical processor. The processor on which server S_j^k is assigned is denoted as $\mathcal{P}(S_j^k)$. For the sake of simplicity, this paper assumes that $\mathcal{M} = M$. The framework described above is illustrated in Figure 1 for a platform of 4 processors.

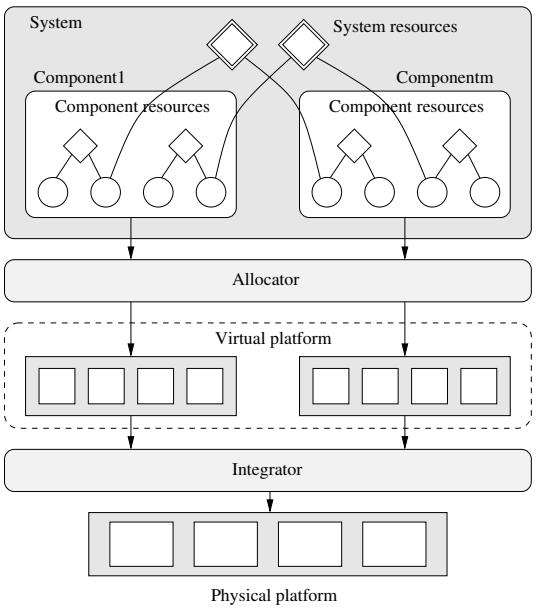

Figure 1: Proposed hierarchical framework.

Tasks can share resources through mutually exclusive critical sections. At a component level, we distinguish between *component resources*, accessed only by tasks belonging to the same component and *system resources*, accessed by tasks belonging to different components.

For each resource R_ℓ, $\delta_{i,\ell}$ denotes the length of the longest critical section of task τ_i related to R_ℓ, while $\eta_{i,\ell}$ denotes the number of critical sections used by τ_i on R_ℓ. We assume to have $\eta_{i,\ell} = 0$ if a task τ_i does not access resource R_ℓ.

For a resource R_ℓ accessed by a task τ_i, the **Resource Holding Time** $H_{i,\ell}$ is defined as the maximum budget consumed from the lock of R_ℓ until its unlock in the τ_i's code. Note that if a resource is accessed non preemptively, its resource holding time is equal to the critical section length

(i.e., $H_{i,\ell} = \delta_{i,\ell}$), otherwise it must include all possible preemption delays occurring within the critical section [10]. A component Γ_k can only be admitted in the system if all its tasks have a resource holding time bounded by $\mathcal{H}$, that is, if

$$\forall \tau_i \in \mathcal{T}(\Gamma_k), \forall R_\ell,\ H_{i,\ell} \leq \mathcal{H}, \quad (1)$$

For *component resources* R_q accessed by tasks allocated to different virtual processors, the sum of the maximum resource holding times of R_q from each virtual processor must be bounded by $M\mathcal{H}$. Formally, we require that

$$\forall \Gamma_k, \forall R_q \in \mathcal{R}_k,\ \sum_{j=1}^{M} \max\{H_{i,q} \mid \mathcal{S}(\tau_i) = S_j^k\} \leq M\mathcal{H}, \quad (2)$$

where $\mathcal{R}_k$ denotes the set of *component resources* accessed by tasks executing on different virtual processors, formally defined as

$$\mathcal{R}_k = \{R_q \mid \exists \tau_1, \tau_2 \text{ using } R_q \wedge \mathcal{S}(\tau_1) \neq \mathcal{S}(\tau_2)\}. \quad (3)$$

2.2 Scheduling infrastructure

Tasks allocated to a virtual processor are handled by a *local scheduler*, which can be any fixed-priority (FP) algorithm or Earliest Deadline First (EDF) [22]. Tasks may include non-preemptive regions in which preemption is disabled for the local scheduler. Each virtual processor is implemented by an M-BROE server [14] and the various M-BROE servers are scheduled under partitioned EDF scheduling on the M processors.

Once reservation servers are mapped to physical processors, three types of shared resources can be distinguished: *Local resources*, shared only by tasks handled by the same server; *Processor-local resources*, shared only by tasks executing on the same processor, but on different servers; *Global resources*, shared by tasks executing on different processors.

Local resources are accessed through the SRP [3] protocol, while *processor-local resources* are accessed by the H-SRP [18] protocol in conjunction with M-BROE, in a *local non-preemptive manner*. Finally, *global resources* are accessed by the MSRP [20] protocol in conjunction with M-BROE.

3. SUMMARY OF SCHEDULABILITY RESULTS

To make this work self consistent, this section summarizes the schedulability results derived for the M-BROE framework. In particular, after recalling how to derive blocking factors, we summarize the schedulability tests used for local (tasks guarantee upon virtual processors) and global (virtual processors guarantee upon the physical platform) analysis. Please refer to [14] for further details.

3.1 Resource sharing

Under the M-BROE framework, global resources are protected by non-preemptive FIFO spinlocks. If a task τ_i wants to use a global resource locked from a task on another processor, τ_i starts spinning non-preemptively until the resource is granted. Critical sections on global resources are also executed non-preemptively and simultaneous requests from different processors are served in FIFO order.

To analyze blocking times related to spinlocks we rely on the MSRP [20] analysis: please note that although an improved analysis for spinlocks has been proposed in [26], it cannot be directly used for the M-BROE framework for several reasons as explained in [14] (page 4). According to the MSRP analysis, a bound on the maximum spinning time (denoted as *remote blocking*) is computed for each global resource R_ℓ accessed from a given processor and used to inflate the WCET of the tasks using R_ℓ. Also, non-preemptive spinning and non-preemptive access to global resources introduce a *non-preemptive blocking* factor that must be accounted for each task. The access to *local* and *processor local* resources is regulated by the SRP [3] and the H-SRP [16] protocols, generating *local blocking* and additional *non-preemptive blocking*, respectively.

In the following, we first provide an upper-bound for the spinning time and then report the expressions for computing *remote blocking*, *non-preemptive blocking* and *local blocking*.

Upper-bound for the spinning time. According to Lemma 1 in [14], a safe upper bound on the spinning time $\xi_{\ell,j}$ related to *system* resources R_ℓ can be computed as

$$\xi_{\ell,j} \leq (M-1)\mathcal{H}.$$

For *component resources*, critical section lengths are known, since they belong to tasks of the same component. However, when tasks are assigned to different virtual processors, it is not possible to infer the physical processor on which they will be executed. For this reason, a safe bound on the spinning time can be computed by assuming that all virtual processors of a component will be assigned to different physical processors. In this case, an upper bound on the spinning time $\xi_{\ell,j}$ for a *component resource* R_ℓ in Γ_k can be computed as

$$\xi_{\ell,j} \leq \sum_{S_j^k \neq S_m^k} \max\{\delta_{i,\ell} \mid \mathcal{S}(\tau_i) = S_m^k\}. \quad (4)$$

This fact imposes also that *processor-local resources* have to be always accounted as *global resources* to capture the worst-case in the local analysis.

Remote blocking. An upper-bound ξ_i on the *remote blocking* for task τ_i is computed by accounting for the maximum spinning time on each critical section of τ_i, with $\mathcal{P}(\mathcal{S}(\tau_i)) = \mathcal{P}_j$, that is:

$$\xi_i = \sum_{R_\ell} \{\eta_{i,\ell} \cdot \xi_{\ell,j} \mid R_\ell \text{ used by } \tau_i\}. \quad (5)$$

Non-preemptive blocking. Such a blocking is bounded by the longest non-preemptive section causing arrival blocking. Under local EDF scheduling, assuming that tasks τ_k and τ_i execute on processor $\mathcal{P}_j$, it can be computed as

$$B_i^{\text{NP}} = \max_{k, R_\ell} \{\xi_{\ell,j} + \delta_{k,\ell} \mid R_\ell \text{ used by } \tau_k \wedge D_k > D_i\}, \quad (6)$$

Local blocking. The SRP blocking factor for task τ_i due to local resources is denoted by B_i^L and is given by critical sections of resources that are locked by tasks τ_L with deadlines greater than D_i and shared with tasks τ_H with

deadlines less than or equal to D_i. Formally,

$$B_i^L = \max\{\delta_{L,\ell} \mid D_H \leq D_i < D_L \wedge \tau_L,\tau_H \text{ use } R_\ell\}. \quad (7)$$

Since *non-preemptive blocking* and *local blocking* occur at the task's release, the resulting blocking is called *arrival blocking* and can be computed as

$$B_i = \max\{B_i^{\text{NP}}, B_i^L\}. \quad (8)$$

3.2 Local analysis

Using the processor demand criterion extended to include resource sharing [4], a task set $\mathcal{T}$ is schedulable by EDF under the M-BROE server if:

$$\forall t > 0 \quad B(t) + \mathsf{dbf}(t) \leq \mathsf{sbf}(t) \quad (9)$$

where

$$\mathsf{dbf}(t) = \sum_{\tau_i \in \mathcal{T}} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 (C_i + \xi_i), \quad (10)$$

$$B(t) = \max\{B_i \mid D_i \leq t\}, \quad (11)$$

and $\mathsf{sbf}(t)$ is the *supply bound function* for the M-BROE server reported in [14] and $(x)_0$ denotes $\max(0, x)$. Note that the $\mathsf{dbf}(t)$ in Equation (10) takes into account the computation time inflation ξ_i due to *remote blocking*.

3.3 Component integration analysis

The *component integrator* has to ensure the schedulability of the reservation servers assigned to each processor. Each component provides a set of reservation servers to the *component integrator* according to the specified interface; such reservation servers will be scheduled under partitioned scheduling on the M physical processors.

The schedulability test for guaranteeing the execution of virtual processors upon physical processors is reported in the following equation:

$$\forall S_j^k : \mathcal{P}(S_j^k) = \mathcal{P}_m, \sum_{\substack{r,l: P_r^l \leq P_j^k \\ \wedge \\ \mathcal{P}(S_r^l) = \mathcal{P}_m}} \frac{Q_r^l}{P_r^l} + \frac{B_j^k}{P_j^k} \leq 1. \quad (12)$$

Since the simple interface (Q, P) is used for each component, no information is available about shared resources, hence, by Equations (2) and (3), the upper bound $B_j^k = M\mathcal{H}$ can be used. Note that a more accurate upper bound can be computed by adopting a more complex component interface, as the one proposed in [14].

4. TASK PARTITIONING AND SERVER DESIGN

This section addresses the problem of partitioning application tasks on a set of virtual processors implemented by M-BROE reservation servers, and determining their configuration parameters in terms of budgets and periods.

A partitioning methodology for the M-BROE framework must take into account three aspects simultaneously: (i) the computational demand of the application; (ii) the parameters (budget and period) of the reservation servers; and (iii) the blocking times related to resource sharing (also including the effect of spinlocks). In particular, the dependency of partitioning on resource sharing is better illustrated by the following example.

Example. Consider a component composed of 3 tasks with periods $T_1 = 10$, $T_2 = 20$ and $T_3 = 50$ ms and implicit deadlines, to be executed on a platform composed of two processors. Tasks τ_1 and τ_3 share a component resource R. If tasks τ_1 and τ_3 are assigned to different processors, then the access to R will be regulated by a spinlock. This solution penalizes the schedulability by generating non-preemptive blocking and increasing the execution times of τ_1 and τ_3 due to spinning time. Conversely, if the two tasks are allocated on the same processor, the blocking factor related to R is smaller (due to only uniprocessor SRP blocking). While it seems convenient allocating tasks on the same processor, it is easy to see that allocating τ_2 together with the other tasks may not be appropriate, due to the local blocking experienced by τ_2 (because $D_1 < D_2 < D_3$, from Equation (7)). If τ_2 is allocated to the other processor, it will experience no blocking. Note however, that τ_2 can experience a different blocking if it has a different period (e.g., $T_2 = 5$ ms).

The simple example presented above shows that resource sharing further complicates partitioning (which is intrinsically NP-HARD, being similar to a bin-packing problem), and must be taken into account to identify a "good" task allocation.

In this paper, task partitioning is formulated as an optimization problem. Unfortunately, however, given any performance objective (e.g., minimizing the overall bandwidth for the reservation servers), searching for an optimal solution is practically intractable for the following reasons:

- The exact EDF schedulability test requires the specification of a pseudo-polynomial number of check points [6] that depends on the parameters of the server upon which tasks execute (as explained in [11, 17]). Being the server parameters part of the output of the optimization problem (hence, unknown), upper bounds on them must be used, thus obtaining a potentially large set of constraints and variables.
- As shown in [17], it is possible to formulate an optimization problem to compute optimal BROE server parameters, minimizing the server bandwidth still ensuring the task schedulability. However, the exact problem formulation involves non-linear constraints, because the supply bound function of the server is non-linear.
- In the M-BROE framework, the access to global resources is protected by FIFO non-preemptive spinlocks, but an exact analysis for spinlocks is still missing, thus preventing the search for an optimal partitioning.

Given such limitations, this paper proposes a sub-optimal methodology for task partitioning and virtual processors design by splitting the problem in two phases: First, an MILP optimization is performed for partitioning tasks to virtual processors; then the optimal server parameters are computed through the approach presented in [17].

To overcome the problems highlighted above, the following approximations are proposed to express partitioning as an MILP optimization problem:

- The EDF schedulability is carried out by the Fully Polynomial Time Approximation Scheme (FPTAS), proposed by Fisher, Baker, and Baruah [19]. According to this approach, the workload of a task is described by the exact demand bound function for the first λ steps, and by a *linear* upper-bound for the remaining steps. Formal details about this approximation will be reported in Section 4.1.3.
- The reservation servers are approximated as ideal (fluid) virtual processors, running at a given speed α, which represents the server bandwidth. Note that using a classical bounded-delay (α-Δ) approximation, the optimization problem results non-linear.
- As done by Wieder and Brandenburg [27], blocking times in the presence of FIFO non-preemptive spinlocks are computed by the original MSRP (sufficient) analysis proposed by Gai et al. [20].

Please, note that approaching the problem through an MILP formulation guarantees that the achieved partitioning is optimal with regard to the assumed approximations.

4.1 Optimization problem formulation

This section presents an MILP formulation to solve the problem of task partitioning upon virtual processors. The formulation is inspired by the one proposed by Wieder and Brandenburg [27] in the context of classical partitioned fixed-priority scheduling without reservation mechanisms.

As stated in Section 3, the worst-case blocking related to *component* resources is computed assuming that all the virtual processors of a component are assigned to different physical processors: without loss of generality, the index $k = 1, \ldots, M$ will be used for both physical processors and virtual processors. In the following we refer to a single component, hence the component index is removed from all the terms used below. All the real variables used in the optimization problem are implicitly constrained as greater than or equal to zero. For such variables, lower-bounds expressing the minimum (safe) value will be used to ensure schedulability.

4.1.1 Decision variables for task allocation

The following decision (binary) variables are defined to decide on the task's allocation.

- $A_{i,k}$: binary variable that is set to 1 if and only if task τ_i is assigned to server S_k.

Since each task is assigned to exactly one server, the following constraint can be derived:

Constraint 1.

$$\forall \tau_i, \sum_{k=1}^{M} A_{i,k} = 1. \tag{13}$$

- $V_{i,j}$: binary variable that is set to 1 if and only if both tasks τ_i and τ_j are assigned to the same server S_k (i.e., $A_{i,k} = A_{j,k}$, for the same S_k).

This variable can be derived from variables $A_{i,k}$ by using the following constraint:

Constraint 2.

$$\begin{aligned} &\forall \tau_i, \forall \tau_i \neq \tau_j, \forall k = 1, \ldots, M \\ &V_{i,j} \geq 1 - (2 - A_{i,k} - A_{j,k}). \end{aligned} \tag{14}$$

Proof. If tasks τ_i and τ_j are assigned to the same server S_k, then $A_{i,k} = A_{j,k} = 1$, so obtaining $V_{i,j} \geq 1$. If τ_i and τ_j are assigned to different servers, we have $A_{i,k} \neq A_{j,k}, \forall k = 1, \ldots, M$, leading to $(2 - A_{i,k} - A_{j,k}) \geq 1, \forall k = 1, \ldots, M$ so degenerating the constraint to $V_{i,j} \geq 0$. □

4.1.2 Resource sharing - variables and constraints

Blocking times related to resource sharing are crucial to express the task schedulability as a constraint of the optimization problem.

First of all, decision variables are provided to identify which resource can cause arrival blocking on tasks.

- $Z_{i,\ell}$: binary variable that is set to 1 if and only if resource R_ℓ can cause arrival blocking on task τ_i.

To define these variables it is necessary to distinguish between *component* and *system* resources. If R_ℓ is a *component* resource, it can results in a *local* or a *global* resource depending on the allocation of tasks using R_ℓ. If a *component* resource R_ℓ results in a *local* resource, the arrival blocking is equal to the classical case of uniprocessor SRP, leading to

Constraint 3.

$$\begin{aligned} \forall \tau_i, \forall R_\ell, \quad & \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0, \\ & \forall \tau_H \mid D_H \leq D_i \wedge \eta_{H,\ell} > 0 \\ & Z_{i,\ell} \geq 1 - (2 - V_{i,L} - V_{i,H}). \end{aligned} \tag{15}$$

Proof. This constraint derives directly from the expression of local blocking for SRP reported in Equation (7). Being a blocking related to local resources, tasks τ_i, τ_L and τ_H are involved only if they are allocated on the same virtual processor: hence, the term $(2 - V_{i,L} - V_{i,H})$ is designed to be zero if and only if all the three tasks are allocated on the same virtual processor (as done in Constraint 2); otherwise it results in a positive value, thus degenerating the constraint to $Z_{i,\ell} \geq 0$. □

If a *component* resource R_ℓ results in a *global* resource, it will be protected by a FIFO non-preemptive spinlock generating non-preemptive blocking on tasks. Such a kind of blocking is accounted through the following constraint:

Constraint 4.

$$\begin{aligned} \forall \tau_i, \forall R_\ell, \quad & \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0, \\ & \forall \tau_H \mid \eta_{H,\ell} > 0 \\ & Z_{i,\ell} \geq 1 - (1 - V_{i,L}) - V_{i,H}. \end{aligned} \tag{16}$$

PROOF. A task τ_i can incur in arrival blocking if a task τ_L with deadline greater than τ_i ($D_L > D_i$) is accessing a global resource R_ℓ: this is because τ_L will use non-preemptive spinning and a non-preemptive critical section for R_ℓ. First of all, this is possible only if τ_i and τ_L are allocated to the same virtual processor: to this purpose, the term $-(1 - V_{i,L})$ will cause a degeneration of the constraint to $Z_{i,\ell} \geq 0$ if this is not true. At the same time, there must exist a task τ_H accessing R_ℓ that is *not* allocated on the virtual processor of τ_i and τ_L, otherwise R_ℓ would not be a global resource: to this purpose, the term $-V_{i,H}$ is provided to not have arrival blocking if such a task τ_H does not exist. □

Consider now *system* resources. These resources are accessed from all the components; hence, a safe bound for their blocking is computed by assuming that they will always result in *global* resources, independently of the task allocation on the other processors. Overall, a *system* resource R_ℓ leads to definition of the following constraint:

Constraint 5.

$$\forall \tau_i, \forall R_\ell, \quad \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0, \quad Z_{i,\ell} \geq 1 - (1 - V_{i,L}). \tag{17}$$

PROOF. As discussed in Section 3, since *system* resources are accessed by all the components, we have to assume that they will always result in global resources (thus involving non-preemptive spinning and non-preemptive critical sections). Hence, to have arrival blocking on a task τ_i, it is sufficient to have a task τ_L executing on the same virtual processor of τ_i, having $D_L > D_i$ and accessing a *system* resource R_ℓ. The term $-(1 - V_{i,L})$ is provided to degenerate the constraint to $Z_{i,\ell} \geq 0$ if such a task τ_L does not exist; otherwise $Z_{i,\ell} \geq 1$, thus expressing the presence of arrival blocking. □

We now provide constraints to quantify the arrival blocking on tasks, expressed through the following variables for the optimization problem:

- $B_{i,\ell}$: real variable expressing a lower-bound on the arrival blocking imposed on τ_i by critical sections on resource R_ℓ.
- $B_{i,\ell,k}$: real variable expressing a lower-bound on the arrival blocking imposed on τ_i by critical sections on resource R_ℓ, executed by tasks running on processor P_k.

The blocking $B_{i,\ell}$ can be computed as

Constraint 6.

$$\forall \tau_i, \forall R_\ell, \quad B_{i,\ell} = \sum_{k=1}^{M} B_{i,\ell,k}. \tag{18}$$

The key part consists in providing constraints for the contributions of arrival blocking $B_{i,\ell,k}$ coming from each processor. Also in this case, we have to distinguish between *component* and *system* resources. Considering *component* resources R_ℓ, the blocking contribution $B_{i,\ell,k}$ coming from the same virtual processor on which task τ_i is allocated can be expressed as follows:

Constraint 7.

$$\begin{aligned} &\forall \tau_i, \forall R_\ell, \forall P_k, \\ &\quad \forall \tau_L \mid D_L > D_i \wedge \eta_{L,\ell} > 0, \\ &\quad B_{i,\ell,k} \geq \delta_{L,\ell} - \delta_{L,\ell} \cdot (1 - Z_{i,\ell}) - \delta_{L,\ell} \cdot (1 - A_{L,k}) \\ &\quad -\delta_{L,\ell} \cdot (1 - A_{i,k}). \end{aligned} \tag{19}$$

PROOF. This constraint expresses the blocking contribution for τ_i coming from the virtual processor at which such a task is allocated. Concerning arrival blocking, a task τ_i can be blocked by critical sections of tasks τ_L (i.e., $\delta_{L,\ell}$), either in the case R_ℓ is a local resource (imposing SRP blocking) or a global resource (imposing non-preemptive blocking).

Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,\ell,k} \geq 0$. First of all, the blocking has to be zero if R_ℓ cannot impose arrival blocking on τ_i: this is enforced by the term $-\delta_{L,\ell} \cdot (1 - Z_{i,\ell})$ which becomes equal to $-\delta_{L,\ell}$ if $Z_{i,\ell} = 0$. Without loss of generality we assume that τ_i is allocated on S_k. Then, the arrival blocking from such a server exists only if τ_i and τ_L are allocated on S_k: to this purpose, the terms $-\delta_{L,\ell} \cdot (1 - A_{L,k})$ and $-\delta_{L,\ell} \cdot (1 - A_{i,k})$ are provided to degenerate the constraint to $B_{i,\ell,k} \geq 0$ if this is not true. □

If a *component* resource R_ℓ results in a *global* resource, there is a blocking contribution coming from other virtual processors (i.e., the ones at which τ_i is not allocated). These blocking contributions are reflected as non-preemptive spinning that prevents τ_i from executing. A lower-bound on such a blocking $B_{i,\ell,k}$ can be formulated as follows:

Constraint 8.

$$\begin{aligned} &\forall \tau_i, \forall R_\ell, \forall P_k, \\ &\quad \forall \tau_x \mid \eta_{x,\ell} > 0, \\ &\quad B_{i,\ell,k} \geq \delta_{x,\ell} - \delta_{x,\ell} \cdot (1 - Z_{i,\ell}) - \delta_{x,\ell} \cdot (1 - A_{x,k}) \\ &\quad -\delta_{x,\ell} \cdot A_{i,k}. \end{aligned} \tag{20}$$

PROOF. Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,\ell,k} \geq 0$. Like Constraint 7, the blocking has to be zero if R_ℓ cannot impose arrival blocking on τ_i: this is enforced by the term $-\delta_{L,\ell} \cdot (1 - Z_{i,\ell})$. If a component resource R_ℓ results in a global resource, there exists a remote task τ_x accessing R_ℓ that is allocated on a virtual processor different from the one of τ_i. Without loss of generality, assume that S_k is the virtual processor at which τ_i is allocated. The remote task τ_x has to be allocated on a virtual processor different from S_k, hence the term $-\delta_{x,\ell} \cdot (1 - A_{x,k})$ is provided to have zero blocking if this is not true. Similarly, the term $-\delta_{x,\ell} \cdot A_{i,k}$ enforces zero blocking from the same virtual processor S_k at which τ_i is allocated. □

When blocking on a *system* resource R_ℓ is considered, we assume that a critical section of maximum length $\mathcal{H}$ is present on each virtual processor (please refer to Section 3), so obtaining the following constraint:

Constraint 9.

$$\forall \tau_i, \forall R_\ell, \forall P_k, \quad B_{i,\ell,k} \geq \mathcal{H} - \mathcal{H} \cdot (1 - Z_{i,\ell}) - \mathcal{H} \cdot A_{i,k}. \tag{21}$$

Proof. Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,\ell,k} \geq 0$. First of all, the term $-\mathcal{H} \cdot (1 - Z_{i,\ell})$ is provided to have zero blocking if task τ_i cannot incur in arrival blocking related to R_ℓ. Then, we have a blocking contribution equal to the maximum critical section length $\mathcal{H}$ on each remote processor, that is, all processors except the one at which τ_i is allocated, which is excluded through the term $-\mathcal{H} \cdot A_{i,k}$. □

As stated in Section 3, the use of non-preemptive FIFO spinlocks is accounted by inflating tasks WCETs with the spinning time ξ_i generated by *remote blocking*. We decompose the spinning time ξ_i of a task τ_i by using the following variables:

- $\xi_{i,k}$: real variable expressing a lower-bound on the spinning time for task τ_i originated from processor P_k.
- $\xi_{i,k,\ell}$: real variable expressing the contribution of the spinning time ξ_i in accessing the resource R_ℓ, originated from processor P_k.

The per-processor spinning time $\xi_{i,k}$ can be expressed as

Constraint 10. $\forall \tau_i, \forall P_k, \quad \xi_{i,k} = \sum_{R_\ell} \xi_{i,k,\ell}$.

Similarly, the overall spinning time ξ_i of a task τ_i is formulated as

Constraint 11. $\forall \tau_i, \quad \xi_i = \sum_{k=1}^{M} \xi_{i,k}$.

Then, the key part consists in identifying a lower-bound on the spinning time $\xi_{i,k,\ell}$. Again, it necessary to distinguish between *component* and *system* resources. For a *component* resource R_ℓ, the following constraint is used:

Constraint 12.

$$\forall \tau_i, \forall P_k, \forall R_\ell, \quad \forall \tau_x \mid \tau_i \neq \tau_x, \quad \xi_{i,k,\ell} \geq \delta_{x,\ell} \cdot \eta_{i,\ell} \cdot A_{x,k} - \mathcal{B} \cdot A_{i,k}. \tag{22}$$

Proof. This constraint derives directly from the computation of the spinning time for component resources, as defined in Equations (4) and (5). The constraint collects the maximum critical section on R_ℓ of tasks τ_x allocated on virtual processor S_k. To account for the overall spinning time, the critical section length is multiplied for the number of critical sections on R_ℓ for τ_i (see Equation (5)). Thanks to the decision variable $A_{x,k}$, the first term becomes zero if task τ_x is not allocated on server S_k. The remaining term $-\mathcal{B} \cdot A_{i,k}$ is provided to have zero spinning time contribution from the same processor at which τ_i is allocated (critical sections executed on the same processor of τ_i do not cause spinning). In this case, $\mathcal{B}$ represents a numerically large constant that dominates all possible values for the term $\delta_{x,\ell} \cdot \eta_{i,\ell}$, and can be formally defined as $\mathcal{B} = \max_{x,\ell}\{\delta_{x,\ell}\} \cdot \max_{i,\ell}\{\eta_{i,\ell}\}$. □

When a *system* resource R_ℓ is considered, the spinning time can be expressed as follows:

Constraint 13.

$$\forall \tau_i, \forall P_k, \forall R_\ell, \quad \forall \tau_x \mid \tau_i \neq \tau_x, \quad \xi_{i,k,\ell} \geq \mathcal{H} \cdot \eta_{i,\ell} \cdot A_{x,k} - \mathcal{B} \cdot A_{i,k}. \tag{23}$$

Proof. The proof follows from the one of Constraint 12, assuming critical sections of length $\mathcal{H}$. □

4.1.3 Schedulability - variables and constraints

This section presents the constraints for the optimization problem expressing the schedulability of a task set upon a reservation server. As stated in Section 3, the local schedulability upon an M-BROE server can be checked by Equation (9); however, as expressed at the beginning of Section 4, the exact test is not easily tractable in an optimization problem. To solve this problem, the workload of a task set is approximated using the FPTAS [19] approach. According to the FPTAS, the demand bound function $\mathsf{dbf}_i(t)$ of a task τ_i is exact for the first λ steps and then approximated with a linear upper-bound. Depending on the chosen value of λ, function $\mathsf{dbf}_i(t)$ can be approximated with any desired degree of accuracy. Formally, the FPTAS for the demand bound function is expressed as

$$\mathsf{dbf}_i^{(\lambda)}(t) = \begin{cases} \mathsf{dbf}_i(t), & \text{if } t \leq (\lambda - 1)T_i + D_i \\ C_i + \xi_i + (t - D_i)U_i, & \text{otherwise,} \end{cases} \tag{24}$$

where $U_i = (C_i + \xi_i)/T_i$, to account for the WCET inflation related to the use of spinlocks.

Using this approximation, the EDF schedulability has to be considered in $\lambda + 1$ time points for each task. The resulting sufficient EDF schedulability test for a set of tasks $\mathcal{T}$ is expressed as follows:

$$\forall p = 0, 1, \ldots, \lambda, \ \forall t \in \mathrm{tSet}(p), \quad B(t,p) + \sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i^{(\lambda)}(t) \leq \mathsf{sbf}(t), \tag{25}$$

where $\mathrm{tSet}(p)$ is the set of schedulability check-points [19] defined as

$$\mathrm{tSet}(p) = \bigcup_{\tau_i \in \mathcal{T}} \{pT_i + D_i\}, \tag{26}$$

and $B(t,p)$ is defined to approximate the blocking term of

Equation (11) as

$$B(t,p) = \begin{cases} B(t), & \text{if } 0 \leq p < \lambda \\ \max_i\{B_i\}, & p = \lambda. \end{cases} \quad (27)$$

As stated by Baruah in [4], the blocking term $B(t)$ is zero for values of t larger than the maximum relative deadline of the tasks under analysis. As a consequence, the exact blocking function $B(t)$ can be used (i.e., $B(t,p) = B(t),\ \forall p \geq 0$) in the MILP formulation if a sufficiently large number λ of check-points is used for the FPTAS.

We now define a set of variables and constraints to express the EDF schedulability according to the FPTAS approach. All the variables contain the processor index k, since the schedulability has to be checked for each processor addressing a partitioned scheduling scheme. First of all, we introduce the following variables to account for the WCET inflation related to spinlocks:

- $J_{i,k}$: real variable expressing the inflated WCET of a task τ_i executing on virtual processor S_k.

Such a variable can be defined by using the following constraint:

Constraint 14.

$$\forall \tau_i, \forall P_k, \quad J_{i,k} = C_i + \xi_i - \mathcal{B} \cdot (1 - A_{i,k}). \quad (28)$$

Proof. This constraint simply adds C_i to the overall spinning time ξ_i. Term $-\mathcal{B} \cdot (1 - A_{i,k})$ is provided to have a null inflated WCET if τ_i is not allocated to server S_k (i.e., when $A_{i,k} = 0$). $\mathcal{B}$ represents a numerically large constant that dominates all possible values of the term $C_i + \xi_i$, and can be defined as $\mathcal{B} = \max_i\{C_i\} \cdot (M-1)\mathcal{H} \cdot \max_{i,\ell}\{\eta_{i,\ell}\}$. □

Now note that Equations (25) involves the blocking time $B(t)$, defined in Equation (27). Having to express the schedulability in a limited number of time points, we introduce variables to quantify the blocking time at the p^{th} time point of a task:

- $PB_{k,p,j}$: real variable expressing the blocking time on virtual processor S_k at schedulability check-point $pT_j + D_j$ of task τ_j, with $p = 0, 1, \ldots, \lambda$.

Such a blocking time is expressed by the following constraint, making use of the blocking time $B_{i,\ell}$ expressed in Constraint 6:

Constraint 15.

$$\begin{aligned} &\forall S_k, \forall p = 0, 1, \ldots, \lambda, \forall \tau_j, \\ &\quad \forall R_\ell \\ &\quad \forall \tau_i \mid D_i \leq pT_j + D_j \wedge p < \lambda \\ &\quad PB_{k,p,j} \geq B_{i,\ell} - \mathcal{B} \cdot (1 - A_{i,k}). \end{aligned} \quad (29)$$

Proof. According Equations (11) and (27), the blocking at the schedulability check-point t involves blocking times of tasks having deadlines less than or equal to t for $p = 0, 1, \ldots, \lambda$. The p^{th} check-point originated from τ_j is $pT_j + D_j$ (see Equation (26)): this constraint considers all tasks τ_i having $D_i \leq pT_j + D_j$ (thus contributing to blocking) excluding the ones that are not allocated on virtual processor S_k. Such tasks are excluded through the term $-\mathcal{B} \cdot (1 - A_{i,k})$, where $\mathcal{B}$ is a numerically large constant that dominates all possible values for the term $B_{i,\ell}$ and can be formally defined as $\mathcal{B} = M\mathcal{H}$. Similarly, all tasks τ_i allocated to S_k are considered for $p = \lambda$, accounting for the maximum blocking on S_k. □

The computational supply provided by each virtual processor S_k is approximated by assuming ideal (fluid) virtual processors with bandwidth α_k. The following variables are introduced to support this choice:

- α_k: real variable representing the bandwidth of virtual processor S_k, with $0 \leq \alpha_k \leq 1$.

At this point we have all the variables and the constraints to express the EDF schedulability on each virtual processor S_k:

Constraint 16.

$$\forall S_k,\ \forall p = 0, \ldots, \lambda,\ \forall \tau_j \quad (30)$$
$$PB_{k,p,j} + \sum_{\tau_i \in \Gamma} \mathsf{dbf}_{i,k}^{(\lambda)}(pT_j + D_j) \leq \alpha_k \cdot (pT_j + D_j),$$

where

$$\mathsf{dbf}_{i,k}^{(\lambda)}(t) = \begin{cases} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right)_0 J_{i,k}, & \text{if } t \leq (\lambda - 1)T_i + D_i \\ J_{i,k} + (t - D_i) \left(\frac{J_{i,k}}{T_i} \right), & \text{otherwise.} \end{cases}$$

Proof. This constraint derives directly from the FPTAS schedulability test in Equation (25), for each virtual processor S_k. Thanks to the definition of variables $J_{i,k}$, the contribution in terms of demand bound function of tasks τ_i is null if τ_i is not assigned to virtual processor S_k. □

4.1.4 Objectives

We now propose two alternative allocation strategies for the optimization problem, aiming at different objectives. The first one, denoted as (A), aims at allocating the tasks of a component on a small set of virtual processors having high bandwidth; the second one, denoted as (B), aims at distributing tasks among a larger set of virtual processors with lower bandwidth. Clearly, each allocation strategy leads to a different instance of the component interface. We now formalize the objectives for both strategies.

- Strategy **(A)**: minimize the overall bandwidth required by a software component, that is the sum of the bandwidths required by its virtual processors:

$$\textbf{minimize} \sum_{k=1}^{M} \alpha_k. \quad (31)$$

- Strategy **(B)**: minimize the maximum bandwidth required by the virtual processor of a component:

$$\textbf{minimize } \max\{\alpha_k\} = \textbf{minimize } \Lambda, \quad (32)$$

where Λ is an additional real variable of the optimization problem defined by the following constraint:

Constraint 17. $\forall k = 1, \ldots, M \quad \Lambda \geq \alpha_k$.

4.2 Interface synthesis

Given the allocation of the component tasks to the virtual platform, produced by the MILP solution, the design of the reservation server parameters is performed using the approach presented in [17]. Such an approach computes the *optimal* budget and period (under the assumed scheduling infrastructure) that guarantee the application schedulability while minimizing the bandwidth for each virtual processor, taking server context switch overhead into account. The resulting server parameters for each virtual processor constitute the component interface exported to the component integrator.

5. VIRTUAL PROCESSOR ALLOCATION

The goal of this section is to propose a methodology to partition the virtual processors of all the components to the M physical processors, that is the task performed by the component integrator. This is done by an MILP optimization problem formulation that is able to find an *exact solution* for the allocation problem, based on the interfaces exported by the components.

As done for the task allocation problem presented in Section 4, we introduce a set of (binary) decision variables to decide on the virtual processors allocation:

- $A^l_{j,k}$: binary variable that is set to 1 if and only if the reservation S^l_j of component Γ_j is assigned to physical processor P_k.

Since a reservation must be assigned to only one processor, the following constraint can be derived:

Integration Constraint 1.

$$\forall S^l_j, \quad \sum_{k=1}^{M} A^l_{j,k} = 1. \tag{33}$$

To express the schedulability at the integration level in the optimization problem formulation, we have to derive constraints from the test reported in Equation (12). The idea is to provide variables and constraints representing the contribution of bandwidth on each physical processor; such variables will be defined to be equal to zero if a virtual processor is not allocated to a processor:

- $Q^l_{j,k}$: real variable representing the budget of server S^l_j executing on processor P_k. Such a budget will be zero if $\mathcal{P}(S^l_j) \neq P_k$.
- $B^l_{j,k}$: real variable representing the blocking imposed on server S^l_j, executing on processor P_k. Such a blocking will be zero if $\mathcal{P}(S^l_j) \neq P_k$.

The following constraints formulate the value for these variables.

Integration Constraint 2.

$$\forall k = 1, \ldots, M, \forall S^l_j, \quad Q^l_{j,k} \geq Q^l_j - \mathcal{B} \cdot (1 - A^l_{j,k}). \tag{34}$$

PROOF. The term $-\mathcal{B} \cdot (1 - A^l_{j,k})$ is provided to degenerate the constraint to $Q^l_{j,k} \geq 0$ if virtual processor S^l_j is not allocated to physical processor P_k, where $\mathcal{B}$ is a numerically large constant formally defined as $\mathcal{B} = \max_{l,j}\{Q^l_j\}$. □

Integration Constraint 3.

$$\forall k = 1, \ldots, M, \forall S^l_j, \quad B^l_{j,k} \geq B^l_j - \mathcal{B} \cdot (1 - A^l_{j,k}). \tag{35}$$

PROOF. It follows from the proof of Integration Constraint 2, with $\mathcal{B} = M\mathcal{H}$. □

Now it is possible to present the main constraint expressing the schedulability of the reservation servers.

Integration Constraint 4.

$$\forall k = 1, \ldots, M, \ \forall S^l_j, \quad \sum_{r,v: P^v_r \leq P^l_j} \frac{Q^v_{r,k}}{P^v_r} + \frac{B^k_{j,k}}{P^l_j} \leq 1. \tag{36}$$

PROOF. It follows directly from Equation (12) and Integration Constraints 2 and 6. If a reservation server is not assigned to processor P_k, its contribution to the sum will be zero. □

As stated in Section 3, the blocking factor B^l_j can be considered constant and equal to $B^l_j = M\mathcal{H}$. If a more complex component interface is used (e.g., the one proposed in Section VI of [14]), it is possible to setup additional variables and constraints to compute allocation-dependent blocking times, as done in Section 4 through Constraints 3-9. This extension is not addressed in this paper for space limitations.

6. EXPERIMENTAL RESULTS

This section presents some experimental results aimed at evaluating the proposed methodology. Experiments have been conducted to (i) evaluate the performance in terms of schedulability ratio for the whole methodology, and (ii) measure the run time of the procedure for partitioning an application upon a virtual processor. The proposed MILP formulations have been implemented with the IBM CPLEX solver running on an 8-core Intel Xeon at 2.8 GHz.

6.1 Workload generation

Given an overall system utilization U, the utilizations U_k of the N components have been generated using the UUnifast [13] algorithm, limiting their values in the range [0.15, 1.5]. All random variables were generated with uniform distribution in a given range. The task set $\mathcal{T}_k$ in each component Γ_k was generated by fixing a random number of tasks $n_k \in [4, 8]$, each with utilization $u_i \leq 0.8$ generated by UUnifast. Tasks periods T_i were randomly chosen in the set $\{5, 10, 20, 30, 50, 80, 100, 150, 200\}$ ms, and tasks

computation times were computed as $C_i = u_i T_i$. We assumed the presence of NR^S system resources and NR^C component resources. For each resource R_ℓ, a random number of tasks in the range $[1, \text{rsf} \cdot n_k]$ was selected to access R_ℓ. The rsf parameter (resource sharing factor) indicates how many tasks use a resource. For each task τ_i accessing R_ℓ, we generated $\eta_{i,\ell} \in [1, \eta^{\text{MAX}}]$ critical sections of length $\delta_{i,\ell} \in (0, \mathcal{H}]$. To have realistic task sets, we enforced $C_i \geq \sum_{R_\ell}(\eta_{i,\ell} \cdot \delta_{i,\ell}),\ \forall \tau_i \in \mathcal{T}_k$.

6.2 Experiment 1

This experiment was carried out to evaluate the performance of the proposed approach comparing the two allocation strategies described in Section 4 (here denoted as MILP-A and MILP-B), and a mixed strategy, denoted as A $\vee$ B, which selects the best of the two interfaces (resulting from strategy A and B), in terms of component integration schedulability. Simulations consider systems composed of $N = 5$ components to be executed on a physical platform including $M = 4$ processors. The performance of the proposed methodology was evaluated by measuring the schedulability ratio (i.e., the ratio of schedulable systems and the total number of generated systems) as a function of the system utilization U, varied from 0.75 to M with step 0.25. For each value of U, the schedulability ratio was computed over 250 systems, hence testing 1250 components and generating a number of tasks between 5000 and 10000.

Figure 2 reports the results of this experiment obtained with the following parameters: $NR^S = 2$, $NR^C = 2$, $\lambda = 30$, rsf= 0.3, $\eta^{\text{MAX}} = 2$, $\mathcal{H} = 100\mu s$.

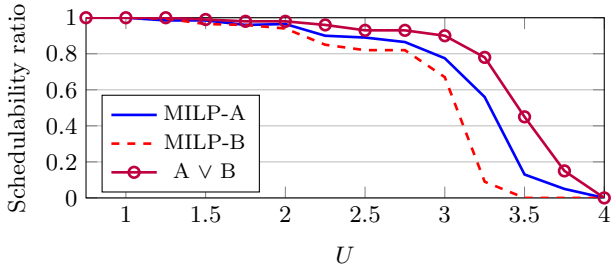

Figure 2: Schedulability ratio as a function of the system utilization U.

Note that, for this particular setting of parameters, MILP-A is slightly more effective than MILP-B for all system utilizations, and both of them are able to guarantee more than 80 percent of the generated systems for overall utilizations up to 3. As expected, the mixed strategy (A $\vee$ B) outperforms the others, being able to admit more than 40 percent of the generated systems for utilizations up to 3.5.

To evaluate the impact of resource sharing on the schedulability ratio, we performed another test with an increased number of resources ($NR^S = 3$, $NR^C = 3$). The result of this test is reported in Figure 3. As can be noted from the graphs, the performance of both MILP-A and MILP-B is slightly degraded, while the mixed strategy (A $\vee$ B) does not show a significant degradation.

6.3 Experiment 2

In this experiment we measured the run time needed to solve the MILP formulation for task partitioning as a function of

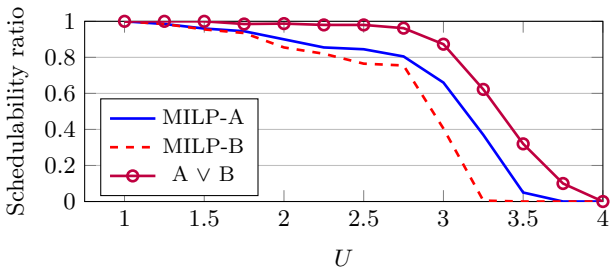

Figure 3: Schedulability ratio as a function of the system utilization U.

the number of tasks (n) in a component. For each value of n, the average and the maximum run time was measured over 500 randomly generated components. Figure 4 reports run time taken by strategy (A), while Figure 5 shows the run time for strategy (B). The results are collected under the same parameter configuration used in Figure 2. The maximum run time shows an exponential trend as the number of tasks increases. This result is expected since the number of the variables on the MILP formulation increases with the number of tasks.

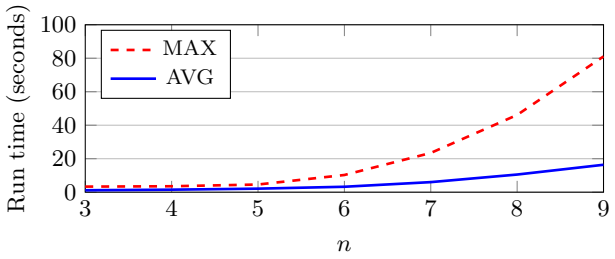

Figure 4: Average and maximum run time for solving the MILP formulation for task partitioning with strategy (A).

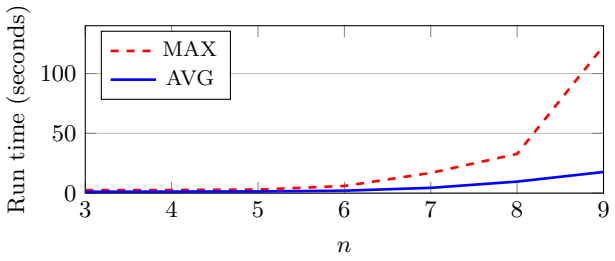

Figure 5: Average and maximum run time for solving the MILP formulation for task partitioning with strategy (B).

7. CONCLUSIONS AND FUTURE WORK

This paper presented a component-based design methodology for supporting the integration of independently developed real-time applications upon multiprocessor platforms in the presence of shared resources. Applications consists of a set of periodic and sporadic real-time tasks that can use both component and system-level resources. The physical platform is abstracted through a number of virtual platforms, one for each component, each consisting of a set of virtual processors implemented by reservation servers.

The proposed methodology uses an MILP formulation to

partition each application upon the virtual multiprocessor platform taking shared resources into account. Once an allocation is found for each component, the synthesis of the virtual processors is performed to find the optimal reservation parameters that can guarantee the schedulability of the applications. Then, a component integrator uses an MILP formulation for allocating all virtual processors to the physical processors to preserve the schedulability of the system.

Simulation experiments on synthetic applications have been carried out to validate the effectiveness of the approach. The achieved results showed that the proposed design methodology (in particular the mix of the proposed allocation strategies) is able to admit 90 percent of the generated systems having utilization up to 3 on a quad-processor platform, in the presence of shared resources and reservations.

As a future work we plan to investigate non-linear optimization problem formulations for the task partitioning, as well as resource sharing driven heuristics, hence proposing a comparison study through extensive simulation experiments.

Acknowledgements

This work has been partially supported by the RETINA Eurostars Project E10171 and received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688860. The authors like to thank Enrico Bini for the fruitful discussions that helped this work.

8. REFERENCES

[1] L. Abeni and G. Buttazzo. Resource reservations in dynamic real-time systems. *Real-Time Systems*, 27(2):123–165, 2004.

[2] Z. Al-bayati, Y. Sun, H. Zeng, M. D. Natale, Q. Zhu, and B. Meyer. Task placement and selection of data consistency mechanisms for real-time multicore applications. In *Proc. of the 21st IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2015)*, Seattle, WA, USA, 2015.

[3] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, April 1991.

[4] S. Baruah. Resource sharing in EDF-scheduled systems: a closer look. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, December 5-8, 2006.

[5] S. Baruah and E. Bini. Partitioned scheduling of sporadic task systems: an ILP-based approach. In *Proc. of the Conference on Design and Architectures for Signal and Image Processing*, Bruxelles, Belgium, November 24-26, 2008.

[6] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2, 1990.

[7] S. K. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *Proceedings of the International Conference on Parallel Processing (ICPP 2004)*, Montreal, Quebec, Canada, August 15-18, 2004.

[8] M. Behnam, T. Nolte, M. Sjödin, and I. Shin. Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. *IEEE Transactions on Industrial Informatics*, 6(1):93–104, February 2010.

[9] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems. In *Proc. of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT 2007)*, Salzburg, Austria, October 1-3, 2007.

[10] M. Bertogna, N. Fisher, and S. Baruah. Resource holding times: Computation and optimization. *Real-Time Systems*, 41(2):87–117, February 2009.

[11] M. Bertogna, N. Fisher, and S. Baruah. Resource-sharing servers for open environments. *IEEE Transactions on Industrial Informatics*, 5(3):202–219, August 2009.

[12] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Arzen, V. R. Segovia, and C. Scordino. Resource management on multicore systems: The ACTORS approach. *IEEE Micro*, 31(3):72–81, May-June 2011.

[13] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[14] A. Biondi, G. Buttazzo, and M. Bertogna. Supporting component-based development in partitioned multiprocessor real-time systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.

[15] A. Biondi, G. C. Buttazzo, and M. Bertogna. Schedulability analysis of hierarchical real-time systems under shared resources. *IEEE Transactions on Computers*, 65(5):1593 – 1605.

[16] A. Biondi, A. Melani, and M. Bertogna. Hard constant bandwidth server: Comprehensive formulation and critical scenarios. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, Pisa, Italy, June 18-20, 2014.

[17] A. Biondi, A. Melani, M. Bertogna, and G. Buttazzo. Optimal design for reservation servers under shared resources. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 9-11, 2014.

[18] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proc. of the IEEE Real-time Systems Symposium (RTSS 2006)*, pages 257–268, Rio de Janeiro, Brazil, Dec. 5-8, 2006.

[19] N. Fisher, T. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. In *Proceedings of the International Conference on Real-time Computing Systems and Applications (RTCSA)*, Sydney, Australia, August 2006.

[20] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of IEEE Real-Time Systems Symposium*, 2001.

[21] N. Khalilzad, M. Behnam, and T. Nolte. On component-based software development for multiprocessor real-time systems. In *Proc. 21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2015.

[22] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[23] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. In *Proceedings of IEEE international conference on Multimedia Computing and System*, May 1994.

[24] M. D. Natale and A. S. Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proc. of the IEEE*, 98(4):603–620, April 2010.

[25] L. Thiele. Model-based design of real-time systems. In *Keynote speeach given at the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014), Madrid, Spain*, July 10th, 2014.

[26] A. Wieder and B. Brandenburg. On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS'2013)*, pages 45–56, December 2013.

[27] A. Wieder and B. Brandenburg. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In *Proc. of the 8th IEEE International Symposium*

on Industrial Embedded Systems (SIES 2013), June 2013.