

# The Parallel Supply Function Abstraction for a Virtual Multiprocessor

**Enrico Bini, Marko Bertogna**  
*Scuola Superiore Sant'Anna*  
*Pisa, Italy*

**Sanjoy Baruah**  
*The University of North Carolina*  
*Chapel Hill, NC, USA*

## Abstract

*A new abstraction — the Parallel Supply Function (PSF) — is proposed for representing the computing capabilities offered by virtual platforms implemented atop identical multiprocessors. It is shown that this abstraction is strictly more powerful than previously-proposed ones, from the perspective of more accurately representing the inherent parallelism of the provided computing capabilities. Sufficient tests are derived for determining whether a given real-time task system, represented as a collection of sporadic tasks, is guaranteed to always meet all deadlines when scheduled upon a specified virtual platform using the global EDF scheduling algorithm.*

## 1 Introduction

There has been an increasing trend in embedded real-time systems design and implementation towards *open* environments [8], in which multiple independently-developed applications can be implemented upon a single shared platform. The typical approach towards providing scheduling support in such open environments is through the use of a two-level scheduler: the top level scheduler allocates resources to the various co-implemented applications, and each application's local scheduler then schedules the jobs comprising the application during the time allocated by the top-level scheduler. Over the past decade or so, sophisticated frameworks and architectures have been proposed for implementing such open environments upon preemptive uniprocessor platforms.

Along with this trend towards open environments, there is an increasing move towards implementing embedded real-time systems upon *multiprocessor* (and multicore) platforms. The use of such parallel architectures yields many benefits — great increases in computing capabilities at lower cost; greater energy efficiency; etc. However, these multiprocessor platforms present a programming model that is far more complex than those used in the classical uniprocessor context. In order to make it easier to build open environments which can offer support for provably correct applications upon multiprocessor platforms, it is desirable to design *abstractions* that conceal much of this additional complexity from the application designers and implementers, instead providing them with *interfaces* that are easy to use and to formally reason about. That is, the resources allocated by the top-level scheduler should be succinctly abstracted out into, and described by means of, an interface; each local scheduler would, in effect, be designed to execute upon a “virtual platform” that behaves as described in this interface. (Such an approach has the added benefit of de-linking application implementation from the platform upon which it will reside, and of allowing for an easier migration of applications among platforms: as hardware is upgraded to a more powerful platform, it is sufficient to ensure that the virtual processor provided by the global scheduler on the new hardware is compliant with the interface previously established.) This paper reports on our recent research towards designing such an abstraction and interface, under the assumptions that (i) the underlying multiprocessor platform is fully preemptive and supports global scheduling; (ii) the top-level scheduler provides each application with zero or more identical (and hence indistinguishable) processors at each instant in time; and (iii) each individual application can be modeled as a collection of sporadic tasks (see Section 2).

In proposing an abstraction, there is typically a tension between the degree of detail that is abstracted away, and the loss of accuracy that results from such information-hiding. The challenge is to come up with the appropriate abstraction that hides enough information so that it is relatively easy to build provably correct applications upon the resources provided by the interface, while minimizing the resulting loss of accuracy. For open systems implemented on uniprocessor platforms,

the parameters in the proposed interfaces that appear to have been most effective have been indicators of (i) the long-term average *computing capacity* that is offered; and (ii) the *time granularity* at which this computing capacity is made available. Specific examples of such interface implementations include the various budget-period servers (e.g., [14]), and the virtual processor abstractions (e.g., [13, 15]) — additional examples are listed in Section 5.

Upon multiprocessors, Shin et al. [16] proposed an extension of the budget-period abstraction to multiprocessor platforms, by adding a third parameter — the *maximum degree of parallelism* — to the interface specification. This is a wonderful idea since it explicitly recognizes the critical role of the degree of parallelism in multiprocessor schedulability: informally and intuitively speaking, the lesser the degree of parallelism in the provided budget, the better guaranteed use the local scheduler can make of it. However, we feel that [16] did not go far enough in exposing underlying parallelism — by representing all parallelism-related information by a single parameter, they, in our opinion, erred too much in favor of simplicity by abstracting away too much information at a cost of a considerable loss of accuracy. Bini et al. [6] proposed to remedy this shortcoming by representing the interface built upon an underlying  $m$ -processor platform with (the supply functions of)  $m$  distinct virtual processors; by using the knowledge of these virtual processors’ supply functions, it is possible to deduce additional information bounding the degree of parallelism in the budget supplied via the interface.

**Our contributions** In this paper, we propose the *Parallel Supply Function* (PSF) abstraction and associated interface for use upon multiprocessor platforms. We show (in Section 5) that this is a superior abstraction to the one in [6], in the sense that even more information can be deduced regarding the degree of parallelism in the budget represented by a PSF while not being any more complex to represent or reason with than the abstraction proposed in [6]. We present, and prove the correctness of, sufficient schedulability tests for determining whether a given application, represented as a collection of sporadic tasks, can be scheduled upon a specified interface when the local scheduling algorithm used is Earliest Deadline First (EDF).

## 2 Terminology and notation

In this section we describe the formal models we use to represent both the virtual multiprocessor platforms and the applications.

### 2.1 Modelling virtual platforms

Each individual application is scheduled onto a dedicated virtual platform  $\Pi$ , which may provide computing capacity upon multiple processors in parallel. Our formalisms do not set any constraints on the techniques used by the virtual platform  $\Pi$  to provide execution cycles to the application  $\Gamma$  — the platform  $\Pi$  could, for instance, be implemented by many sequential servers, static partitions of the processors over time, Pfair or other global schedulers, etc. We will take a closer look at virtual multiprocessor platforms in Section 3.

Figure 1 illustrates a virtual platform that supplies time according to two static partitions: one that provides 2 time units every 4, and another one that provides 4 every 8. In Section 3 we will use this example partition to illustrate some of the definitions.

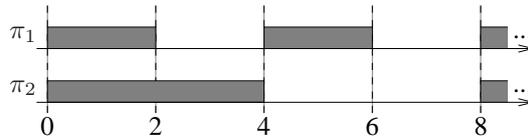
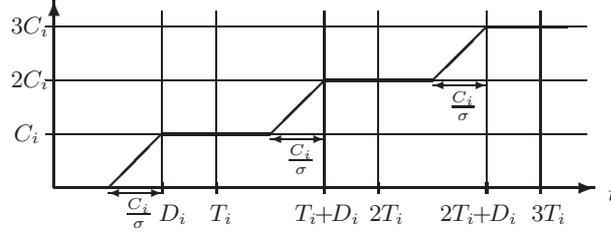


Figure 1. Example of a periodic static partition.

### 2.2 Modeling applications

We model an application as a set of  $n$  sporadic tasks  $\tau = \{\tau_i\}_{i=1}^n$ . Each task  $\tau_i = (C_i, T_i, D_i)$  is characterized by a worst-case computation time  $C_i$ , a minimum inter-arrival time  $T_i$  (also referred to as period), and a relative deadline  $D_i$ . Each task  $\tau_i$  releases a sequence of jobs  $\tau_{i,k}$ , where each job is characterized by an arrival time  $r_{i,k}$ , an absolute deadline  $d_{i,k}$ , a computation time  $c_{i,k}$ . We have that  $c_{i,k} \leq C_i$ ,  $r_{i,k} \geq r_{i,k-1} + T_i$ , and  $d_{i,k} = r_{i,k} + D_i$ . In this paper, we assume a



**Figure 2. Illustrating**  $\text{FF-DBF}(\tau_i, t, \sigma)$ .

*constrained deadline* model, where  $D_i \leq T_i$  for all  $i$ . We also set  $D_{\min} = \min_i D_i$ . Time is continuous and time variables are represented by real numbers.

**The forced-forward demand bound function** Let  $\tau_i$  denote a sporadic task,  $t$  any positive real number, and  $\sigma$  any positive real number  $\leq 1$ . The *forced forward demand bound function*  $\text{FF-DBF}(\tau_i, t, \sigma)$  is defined as follows:

$$\text{FF-DBF}(\tau_i, t, \sigma) \stackrel{\text{def}}{=} q_i C_i + \begin{cases} C_i & \text{if } r_i \geq D_i \\ C_i - (D_i - r_i)\sigma & \text{if } D_i > r_i \geq D_i - \frac{C_i}{\sigma} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where

$$q_i \stackrel{\text{def}}{=} \left\lfloor \frac{t}{T_i} \right\rfloor \quad \text{and} \quad r_i \stackrel{\text{def}}{=} t \bmod T_i,$$

Informally speaking,  $\text{FF-DBF}(\tau_i, t, \sigma)$  can be thought of as a bound on the demand of  $\tau_i$  for interval-length  $t$ , when execution *outside* the interval occurs on a speed- $\sigma$  processor. This function is illustrated for an example task in Figure 2.

The FF-DBF concept is easily extended from individual tasks to applications that are modeled as collections of sporadic tasks: for any such application  $\tau$

$$\text{FF-DBF}(\tau, t, \sigma) \stackrel{\text{def}}{=} \sum_{\tau_\ell \in \tau} \text{FF-DBF}(\tau_\ell, t, \sigma)$$

It is evident from the definition of FF-DBF (Equation (1)) that  $\text{FF-DBF}(\tau, s, t)$  can be computed very efficiently, in polynomial time — see, e.g., [3] for further details.

Some additional notation that we will use. Let  $\tau$  denote an application that is modeled as a collection of sporadic tasks, and  $\tau_k$  any task in  $\tau$ :

$$\begin{aligned} \text{density } \delta_k &\stackrel{\text{def}}{=} C_k / D_k \\ \text{utilization } U_k &\stackrel{\text{def}}{=} C_k / T_k \\ \text{maximum density } \bar{\delta} &\stackrel{\text{def}}{=} \max_{\tau_i \in \tau} \delta_i \\ \text{total utilization } \bar{U} &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i \end{aligned}$$

Finally, we will use  $(x)_0$  as a short for  $\max(x, 0)$ .

### 3 The parallel supply function abstraction

The need of developing the applications independently of the underlying hardware strongly motivates the investigation of interfaces for multiprocessor platforms. As stated above, however, it is important that the interfaces used retain, as much as possible, information regarding the degree of parallelism in which execution capacity (the “budget”) is supplied by the interface. In [16], such information was communicated via the *maximum parallelism* parameter. (The example virtual platform of Figure 1 is thus represented in the Shin et al. model [16] by a budget of 8, a period of 8, and a maximum parallelism of 2. Hence, this formalism abstracts away the potentially useful information that only 4 of the 8 units of the budget occurs upon parallel processors, and that some processor is available for 6 units of time out of every 8.) In [6],

more parallelism information could be communicated via an interface called the *Multi Supply Function* (MSF). The MSF is described in detail in Section 5, where it is shown that even the MSF interface has some shortcomings with regard to retaining parallelism information.

To overcome the limitations of the MSF, we start by generalizing the concept of time partition to the multiprocessor case. Recall from [15] that this concept was introduced to formally represent the availability of a processor that is not necessarily continually available; a *time partition* represents the availability of such a processor by a collection of time-intervals, denoting the times when the processor is available. Since there are multiple processors in a multiprocessor platform, the extension of time partitions to multiprocessors must be able to represent the aggregation of the time partitions of all the processors.

**Definition 1** A time multi-partition  $\mathcal{P}$  is a countable multiset<sup>1</sup> of intervals, formally

$$\mathcal{P} \stackrel{\text{def}}{=} \{[a_i, b_i)\}_{i \in \mathbb{N}}.$$

Intuitively,  $\mathcal{P}$  is the aggregation (the “multi-union”) over all the processors in the platform, of the time partitions of each processor. Without loss of generality we set the instant when the virtual platform is created equal to 0. Hence we have  $a_i \geq 0, \forall i \in \mathbb{N}$ .

A time multi-partition represents the instants over time when the virtual platform allocates time to the application. For example, the multi-partition of Figure 1 is

$$\mathcal{P} = \{[4k, 2 + 4k), [8k, 4 + 8k)\}_{k \in \mathbb{N}}. \quad (2)$$

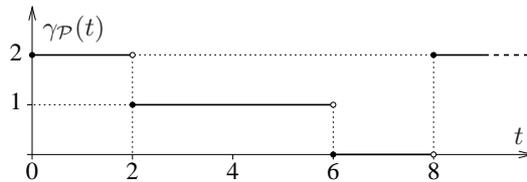
For a given multi-partition, our objective is to define a suitable abstraction that represents the execution capacity supplied by this multi-partition, while retaining information about the degree of parallelism present in this supply. We start by defining the characteristic function  $\gamma_A$  of any subset  $A \subseteq \mathbb{R}$

$$\gamma_A(t) \stackrel{\text{def}}{=} \begin{cases} 1 & t \in A \\ 0 & t \notin A \end{cases}, \quad (3)$$

and the characteristic function of a multi-partition

$$\gamma_{\mathcal{P}}(t) \stackrel{\text{def}}{=} \sum_{[a_i, b_i) \in \mathcal{P}} \gamma_{[a_i, b_i)}(t). \quad (4)$$

The characteristic function of the multi-partition  $\mathcal{P}$  of Figure 1 is depicted in Figure 3.



**Figure 3. Example of characteristic function  $\gamma_{\mathcal{P}}$ .**

For a given multi-partition  $\mathcal{P}$ , it is useful to define the maximum degree of parallelism as follows.

**Definition 2** Given a multi-partition  $\mathcal{P}$ , we define the maximum degree of parallelism as

$$M(\mathcal{P}) \stackrel{\text{def}}{=} \max_{t \geq 0} \gamma_{\mathcal{P}}(t) \quad (5)$$

For the multi-partition depicted in Figure 1, the maximum degree of parallelism is equal to two.

Definition 1 provides a formal notation for the exact representation of virtual multiprocessors that are not continually available. However, it is often not desirable in practice to represent such virtual multiprocessor in an exact manner, for

<sup>1</sup>In set theory, a *multiset* is a generalization of a set, in which individual elements may occur multiple times. Each such occurrence counts as a separate element of the multiset.

several reasons. First, too much information is not always useful and can render programming and analysis cumbersome — indeed, concealing some detail is the very idea behind abstraction and information-hiding. More critically, it is possible that all the knowledge is simply not available at design and specification time; more typically, the exact availability of the virtual processors depends on run-time events such as contention with other virtual multiprocessors that are sharing the same physical platform, and hence only becomes known during run-time. The best we can do during specification and design time is specify bounds on the supplied computing capacity. Such bounds are conveniently modeled by characteristic *supply functions*, as follows.

**Definition 3** Given a multi-partition  $\mathcal{P}$ , we define the level- $j$  supply function  $Y_{j,\mathcal{P}}(t)$  as the minimum amount of time provided by the multi-partition in every interval of time of length  $t \geq 0$  by at most  $j$  intervals in parallel. That is

$$Y_{j,\mathcal{P}}(t) \stackrel{\text{def}}{=} \min_{t_0 \geq 0} \int_{t_0}^{t_0+t} \min\{j, \gamma_{\mathcal{P}}(x)\} dx. \quad (6)$$

We believe that this definition captures properly the amount of resource provided by a multi-partition, by investigating the number of processors that supply the resource simultaneously.

Below we provide some simple properties of the level- $j$  supply functions  $Y_{j,\mathcal{P}}$ . Notice that when comparing any two functions  $f, g : \mathbb{R} \rightarrow \mathbb{R}$ , when we write  $f \leq g$  we mean  $\forall t f(t) \leq g(t)$ .

**Lemma 1** For any multi-partition  $\mathcal{P}$ , we have

$$Y_{0,\mathcal{P}} = 0, \quad (7)$$

$$\forall j \geq 0, \quad Y_{j+1,\mathcal{P}} \geq Y_{j,\mathcal{P}}, \quad (8)$$

$$\forall j \geq M(\mathcal{P}), \quad Y_{j,\mathcal{P}} = Y_{j+1,\mathcal{P}}, \quad (9)$$

$$\forall j \geq 1, \quad Y_{j,\mathcal{P}} - Y_{j-1,\mathcal{P}} \geq Y_{j+1,\mathcal{P}} - Y_{j,\mathcal{P}}. \quad (10)$$

$$\forall j \geq 0, \forall s, t \geq 0, \quad Y_{j,\mathcal{P}}(s+t) \geq Y_{j,\mathcal{P}}(s) + Y_{j,\mathcal{P}}(t) \quad (11)$$

**Proof** All the properties follow from Definition 3.

When  $j = 0$ , the minimum of Eq. (6) is constantly zero, because  $\gamma_{\mathcal{P}} \geq 0$ . Hence  $Y_{0,\mathcal{P}} = 0$  for any  $\mathcal{P}$ , proving Eq. (7).

For any integer  $k$ , we have  $\min\{j+1, k\} \geq \min\{j, k\}$  that proves Eq. (8).

Proof of Eq. (9).

$$\forall t \geq 0, \quad j \geq M(\mathcal{P}) \geq \gamma_{\mathcal{P}}(t) \Rightarrow \min\{j, \gamma_{\mathcal{P}}(t)\} = \gamma_{\mathcal{P}}(t),$$

Hence, when  $j \geq M(\mathcal{P})$ , we have

$$\forall t \geq 0, \quad Y_{j,\mathcal{P}}(t) = Y_{j+1,\mathcal{P}}(t) = \min_{t_0 \geq 0} \int_{[t_0, t_0+t]} \gamma_{\mathcal{P}}(x) dx.$$

Proof of Eq. (10). Equation (10) is equivalent to

$$\forall j \geq 1, \quad 2Y_{j,\mathcal{P}} \geq Y_{j+1,\mathcal{P}} + Y_{j-1,\mathcal{P}}.$$

We prove it by showing that

$$\forall k \in \mathbb{N}, \quad 2 \min\{j, k\} \geq \min\{j+1, k\} + \min\{j-1, k\}.$$

In fact, when  $k \geq j+1$ , then

$$\begin{aligned} 2 \min\{j, k\} &= 2j \\ \min\{j+1, k\} + \min\{j-1, k\} &= j+1 + j-1 = 2j; \end{aligned}$$

when  $k \leq j-1$ ,

$$\begin{aligned} 2 \min\{j, k\} &= 2k \\ \min\{j+1, k\} + \min\{j-1, k\} &= 2k; \end{aligned}$$

finally, when  $k = j$ ,

$$\begin{aligned} 2 \min\{j, k\} &= 2j \\ \min\{j+1, k\} + \min\{j-1, k\} &= j + j - 1 = 2j - 1, \end{aligned}$$

which proves the desired property.

We conclude by proving that  $Y_{j,\mathcal{P}}$  is *superadditive* (Equation (11)). For any function  $f : \mathbb{R} \rightarrow \mathbb{R}$ , we have

$$\begin{aligned} \min_{t_0} \int_{t_0}^{t_0+s+t} f(x)dx &= \min_{t_0} \left( \int_{t_0}^{t_0+s} f(x)dx + \int_{t_0+s}^{t_0+s+t} f(x)dx \right) \\ &\geq \min_{t_0} \int_{t_0}^{t_0+s} f(x)dx + \min_{t_0} \int_{t_0}^{t_0+t} f(x)dx \end{aligned}$$

from which it follows Eq. (11), when  $f(x) = \min\{j, \gamma_{\mathcal{P}}(x)\}$ .  $\square$

Definition 3 requires the knowledge of the exact time multi-partition  $\mathcal{P}$  corresponding to the virtual multiprocessor platform under discussion. As discussed above (prior to Definition 3), such information is often known only at run-time (and not at design time) since the actual allocation typically depends on events (such as contention with other VPs) that cannot always be predicted during design time. In the following, we extend Definition 3 by removing the need for such a knowledge.

**Definition 4** Given a virtual multiprocessor platform  $\Pi$ , we define  $\text{legal}(\Pi)$  as the set of multi-partitions  $\mathcal{P}$  that can be allocated by  $\Pi$ .

The maximum degree of parallelism, and the level- $j$  supply functions, of a virtual multiprocessor platform are defined generalizing the analogous concepts for individual multi-partitions.

**Definition 5** Given a virtual platform  $\Pi$ , we define its maximum degree of parallelism as

$$m \stackrel{\text{def}}{=} \max_{\mathcal{P} \in \text{legal}(\Pi)} M(\mathcal{P}) \quad (12)$$

**Definition 6** Given a virtual platform  $\Pi$ , its level- $j$  supply function  $Y_j(t)$  is the minimum amount of time provided, with parallelism at most  $j$ , by the server  $\Pi$  in every time interval of length  $t \geq 0$ ,

$$Y_j(t) \stackrel{\text{def}}{=} \min_{\mathcal{P} \in \text{legal}(\Pi)} Y_{j,\mathcal{P}}(t). \quad (13)$$

Notice that the properties of Lemma 1 hold also for the  $Y_j$  level- $j$  supply functions, because they hold for the  $Y_{j,\mathcal{P}}$  functions, for any multi-partition  $\mathcal{P}$ .

We are now ready to define the Parallel Supply Function (PSF) of any virtual platform  $\Pi$ .

**Definition 7** We define the Parallel Supply Function (PSF) interface of the virtual platform  $\Pi$  as the set  $\{Y_j(t)\}_{j=1}^m$  of the level- $j$  supply functions.

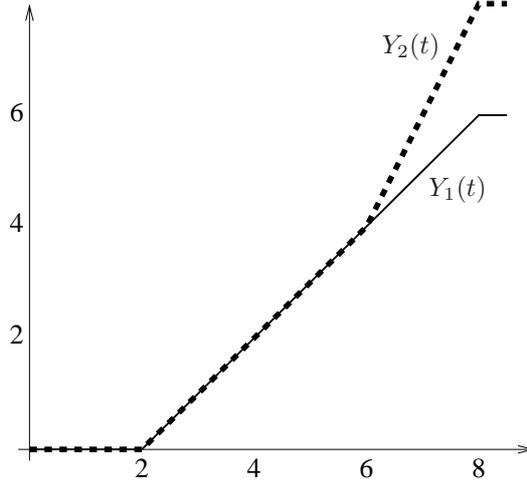
The introduction of the PSF allows a more precise characterization of the time supplied by a virtual platform. We illustrate this on the simple example of Figure 1. In the virtual platform  $\Pi$  corresponding to this figure, the time is allocated statically to the two servers, hence  $\text{legal}(\Pi)$  is composed of one single multi-partition  $\mathcal{P}$  (the one given by Eq. (2)). For this multi-partition  $\mathcal{P}$ , the corresponding characteristic function  $\gamma_{\mathcal{P}}$  is depicted in Figure 3. If we compute the level-1 and level-2 supply functions from Definition 3, we can find the two functions  $Y_1(t)$  and  $Y_2(t)$  reported in Figure 4.

Similarly to what is done for single processor hierarchical scheduling [13, 15, 17], we find it useful to lower bound the parallel supply functions  $Y_j(t)$  with a linear function  $\alpha_j(t - \Delta_j)_0$ . Since  $Y_j$  is superadditive (Equation (11)), a result attributed to Fekete [10] ensures that the following limit exists:

$$\alpha_j \stackrel{\text{def}}{=} \lim_{t \rightarrow +\infty} \frac{Y_j(t)}{t} = \sup_t \frac{Y_j(t)}{t} \quad (14)$$

Notice also that  $\alpha_j \leq j$ , from Eq. (6). Hence, by defining

$$\Delta_j \stackrel{\text{def}}{=} \sup_t \left\{ t - \frac{Y_j(t)}{\alpha_j} \right\} \quad (15)$$



**Figure 4. The level- $j$  supply functions  $Y_1(t)$  and  $Y_2(t)$  for the example of Figure 1.**

the level- $j$  parallel supply function can be conveniently lower bounded by

$$Y_j(t) \geq \alpha_j(t - \Delta_j)_0. \quad (16)$$

The PSF is an *abstraction* of the computing capabilities of the virtual platform, rather than its exact representation. One of the consequences of this fact is that none of the multi-partitions that could be generated by a particular virtual platform  $\Pi$  may correspond exactly to the characterization of  $\Pi$  by its  $Y_j(t)$  functions. We can nevertheless assert lower bounds on the durations for which individual processors must be made available over any time interval in any multi-partition that could be generated by  $\Pi$ , as follows.

Let us arbitrarily assign a total ordering to the processors in the physical platform upon which  $\Pi$  is implemented, so that it makes sense to talk of the  $j$ 'th processor  $P_j$ ,  $1 \leq j \leq m$ . Consider an arbitrary multi-partition  $\mathcal{P}$  of  $\Pi$ , and some interval of length  $L$ ; at any instant in this interval at which  $\mathcal{P}$  makes fewer than  $m$  processors available, we will *rename*<sup>2</sup> the processors in order to choose which of the  $m$  processors  $P_1, \dots, P_m$  are available, in the following manner:

- by definition,  $\Pi$  makes  $\geq Y_1(L)$  units of non-parallel execution available over the interval. Let us “assign” exactly  $Y_1(L)$  of this execution to the first processor  $P_1$ , in the sense that we will name the processor(s) on which this execution has occurred for  $Y_1(L)$  time units as  $P_1$ ;
- similarly,  $\Pi$  makes at least  $Y_2(L)$  units of execution with parallelism at most two available over the interval. Let us again assign  $(Y_2(L) - Y_1(L))$  of this execution to the second processor  $P_2$ ;
- in a similar vein, we can assign exactly  $(Y_j(L) - Y_{j-1}(L))$  units of execution to the  $j$ 'th processor  $P_j$ , for each  $j$ ,  $1 \leq j \leq m$ ;
- observe that since the  $Y_j(L)$ 's denote *lower bounds* on the amount of computing capacity that must be available in the partition, the actual availability of execution capacity in  $\mathcal{P}$  may exceed the amount assigned in the steps above. Once all these assignments have been done, therefore, the remaining execution can be arbitrarily assigned among the processors (over time durations when they have not already been assigned execution during the above steps).

As a consequence of the above argument, it follows that

**Lemma 2** *Let  $\Pi$  be a virtual platform characterized by the supply functions  $\{Y_j(t)\}_{j=1}^m$ . For any multi-partition  $\mathcal{P}$  in  $\text{legal}(\Pi)$  and any interval of length  $L$ , there exists a dynamic renaming of the processors over the interval such that the  $j$ 'th processor is available for at least  $(Y_j(L) - Y_{j-1}(L))$  time units over this interval in  $\mathcal{P}$ .*

<sup>2</sup>We point out that we are not actually requiring that the virtual platform be implemented to make allocations in a manner that corresponds to our renaming — this is a mere notational convenience. Since we are restricting our attention here to virtual platforms implemented upon identical multiprocessors, we can always rename processors for the purposes of reasoning about the schedule, without loss of generality.

```

for  $i \leftarrow 2, 3, \dots$  do
  let  $J_i$  denote a job that
    – arrives at some time-instant  $t_i < t_{i-1}$ ;
    – has a deadline after  $t_{i-1}$ ;
    – has not completed execution by  $t_{i-1}$ ; and
    – has executed for strictly less than  $(t_{i-1} - t_i) \bar{\delta}$ 
      units over the interval  $[t_i, t_{i-1})$ .
  if there is no such job then
     $k \leftarrow (i - 1)$ 
    break (out of the for loop)
  end if
end for

```

Figure 5. Proof of Theorem 1: defining the  $J_i$ 's, the  $t_i$ 's and  $k$ .

## 4 Schedulability analysis

In this section, we derive two sufficient tests for determining whether a given application, modeled as a collection of sporadic tasks, can be scheduled to meet all deadlines when scheduled upon a virtual platform  $\Pi$  using global EDF as the local scheduling algorithm. The first method borrows the idea of forced forward demand bound function [3] and allows deriving a schedulability condition with pseudopolynomial complexity. The second test, inspired by Bertogna et al. [5] has polynomial complexity. It derives an upper bound on the interfering workload generated over the scheduling window of each task, and checks whether it is sufficient to cause a deadline miss. Since none of them is proved to dominate the other, both can be used for admission control.

The tests consider an application  $\tau$  composed of  $n$  sporadic tasks  $\tau_1, \dots, \tau_n$ . The virtual multiprocessor platform, denoted  $\Pi$ , has its maximum parallelism  $m$ , and is characterized by its parallel supply function (PSF) abstraction, denoted  $\{Y_j(t)\}_{j=1}^m$ .

### 4.1 FF-DBF based schedulability test

In this section, we present a sufficient schedulability condition based on the concept of FF-DBF, as defined in Section 2.2.

**Theorem 1** Any constrained-deadline sporadic task system  $\tau$  satisfying

$$\forall L \geq D_{\min}, \quad \text{FF-DBF}(\tau, L, \bar{\delta}) \leq \max_k \{Y_k(L) - (k-1)\bar{\delta}L\} \quad (17)$$

is guaranteed to be EDF-schedulable upon  $\Pi$ .

**Proof** Let us suppose that sporadic task system  $\tau$  is not EDF-schedulable on  $\Pi$ , and let us consider a minimal sequence of jobs of  $\tau$  upon which EDF misses deadlines when implemented on  $\Pi$ . Let  $t_o$  denote the (first) instant at which a deadline miss occurs in such an EDF schedule. Let  $J_1$  denote a job that misses its deadline at  $t_o$ , and let  $t_1$  denote  $J_1$ 's arrival-time. (Observe that  $(t_o - t_1) \geq D_{\min}$ .)

We define a sequence of jobs  $J_i$ , time-instants  $t_i$ , and an index  $k$ , according to the pseudo-code in Figure 5.

Let  $L$  denote the length of the interval  $[t_k, t_o)$ :  $L \stackrel{\text{def}}{=} (t_o - t_k)$ . For each  $i$ ,  $1 \leq i \leq k$ , let  $W_i$  denote the total amount of execution that occurs over the interval  $[t_i, t_{i-1})$ .

**Lemma 1.1**  $\text{FF-DBF}(\tau, L, \bar{\delta}) \geq \sum_{i=1}^k W_i$ .

**Proof** All jobs that execute in  $[t_k, t_o)$  (and hence contribute to  $\sum_{i=1}^k W_i$ ) have their deadlines within the interval  $[t_k, t_o)$ . Some of them will also have arrived within this interval, while others may not.

Now it may be verified that the amount of execution that jobs of any task  $\tau_\ell$  contribute to  $\sum_{i=1}^k W_i$  is bounded from above by the scenario in which a job of  $\tau_\ell$  has its deadline coincident with the end of the interval, and prior jobs have arrived exactly  $T_\ell$  time-units apart. Under this scenario, the jobs of  $\tau_\ell$  that may contribute to  $\sum_{i=1}^k W_i$  include

- at least  $q_\ell \stackrel{\text{def}}{=} \lfloor L/T_\ell \rfloor$  jobs of  $\tau_\ell$  that lie entirely within the interval  $[t_k, t_o]$ ; and
- (perhaps) an additional job that has its deadline at time-instant  $t_k + r_\ell$ , where  $r_\ell \stackrel{\text{def}}{=} L \bmod T_\ell$ .

We now consider two separate cases:

1.  $r_\ell \geq D_\ell$ ; i.e., the additional job with deadline at  $t_k + r_\ell$  arrives at or after  $t_k$ . In this case, its contribution is  $C_\ell$ .
2.  $r_\ell < D_\ell$ ; i.e., the additional job with deadline at  $t_k + r_\ell$  arrives prior to  $t_k$ . From the exit condition of the for-loop, it must be the case that this job has completed at least  $\bar{\delta}(D_\ell - r_\ell)$  units of execution prior to time-instant  $t_k$ ; hence, its remaining execution is at most  $\max(0, C_\ell - \bar{\delta}(D_\ell - r_\ell))$ .

In either case, it may be seen that the upper bound on the total contribution of  $\tau_\ell$  to  $\sum_{i=1}^k W_i$  is equal to  $\text{FF-DBF}(\tau_\ell, L, \bar{\delta})$  (see Equation 1). The lemma follows, by summing over all tasks  $\tau_\ell \in \tau$ .  $\square$

**Lemma 1.2** *Some execution occurs at all the time-instants in  $[t_k, t_o]$  during which the virtual platform  $\Pi$  makes one or more processors available.*

**Proof** Consider each interval  $[t_i, t_{i-1})$ . By definition of  $j_i$ , it arrives at  $t_i$  and has not completed execution by  $t_{i-1}$ ; hence, EDF will execute it whenever processors are available. This rules out the existence of a time-instant over  $[t_i, t_{i-1})$  during which some processor is available, but no job — not even  $j_i$  — is executing. The lemma follows by summing over all  $i$ ,  $1 \leq i \leq k$ ,  $\square$

**Lemma 1.3** *The total duration of all time-intervals over  $[t_k, t_o]$  during which processors are made available by the virtual platform  $\Pi$ , but are not being used in the EDF schedule, is strictly less than  $\bar{\delta}L$ .*

**Proof** For each  $i$ ,  $1 \leq i \leq k$ , let  $x_i$  denote the total length of the time-intervals over  $[t_i, t_{i-1})$  during which job  $J_i$  executes. Since job  $J_i$ , by its definition, arrives at  $t_i$  and has not completed execution by  $t_{i-1}$ , all the processors that  $\Pi$  makes available over this interval must be executing some job whenever  $J_i$  is not. Furthermore,  $J_i$  is chosen such that  $x_i < \bar{\delta}(t_{i-1} - t_i)$ ; hence, the total duration of all the time-intervals over  $[t_i, t_{i-1})$  during which processors are made available by the virtual platform  $\Pi$ , but are not being used in the EDF schedule, is strictly less than  $\bar{\delta}(t_{i-1} - t_i)$ . The lemma follows by summing over all  $i$ ,  $1 \leq i \leq k$ , and using the fact that  $L \stackrel{\text{def}}{=} \sum_{i=1}^k (t_i - t_{i-1})$ .  $\square$

Recall from Lemma 2 that the  $j$ 'th processor is allocated in any multi-partition of  $\Pi$  for at least  $(Y_j(L) - Y_{j-1}(L))$  time units over the interval  $[t_k, t_o]$ . As a consequence of this fact and Lemma 1.3, the  $j$ 'th processor therefore completes at least

$$\max\left(0, ((Y_j(L) - Y_{j-1}(L)) - \bar{\delta}L)\right)$$

units of execution over  $[t_k, t_o]$ , for each  $j > 1$ ; while, by Lemma 1.2, the first processor completes at least  $Y_1(L)$  units of execution. By Lemma 1.1, we therefore have

$$\begin{aligned} & \text{FF-DBF}(\tau, L, \bar{\delta}) \\ & > Y_1(L) + \sum_{j=2}^m \max\left(0, ((Y_j(L) - Y_{j-1}(L)) - \bar{\delta}L)\right) \\ & = Y_1(L) + \sum_{j=2}^m \left( (Y_j(L) - Y_{j-1}(L)) - \right. \\ & \quad \left. \min((Y_j(L) - Y_{j-1}(L)), \bar{\delta}L) \right) \\ & = Y_m(L) - \sum_{j=2}^m \min((Y_j(L) - Y_{j-1}(L)), \bar{\delta}L). \end{aligned}$$

We have thus shown that in order for  $\tau$  to not be EDF-schedule on  $\Pi$ , it is necessary that

$$\text{FF-DBF}(\tau, L, \bar{\delta}) > Y_m(L) - \sum_{j=2}^m \min(Y_j(L) - Y_{j-1}(L), \bar{\delta}L) \quad (18)$$

for some  $L \geq D_{\min}$ .

The RHS can be further simplified. From Eq. (10) it follows that the values  $Y_j(L) - Y_{j-1}(L)$  are decreasing with  $j$ . Let  $k^*$  be the greatest index in the summation where the min of the RHS is given by  $\bar{\delta}L$  (when the minimum is always given by  $Y_j(L) - Y_{j-1}(L)$ , we set  $k^* = 1$ ). Then, the RHS becomes

$$\begin{aligned} Y_m(L) - \sum_{j=k^*+1}^m (Y_j(L) - Y_{j-1}(L)) - \sum_{j=2}^{k^*} \bar{\delta}L &= \\ Y_{k^*}(L) - (k^* - 1)\bar{\delta}L. \end{aligned}$$

Since for any other index  $k \neq k^*$ ,

$$Y_k(L) - (k - 1)\bar{\delta}L \leq Y_{k^*}(L) - (k^* - 1)\bar{\delta}L,$$

it follows

$$\max_k (Y_k(L) - (k - 1)\bar{\delta}L) = Y_{k^*}(L) - (k^* - 1)\bar{\delta}L.$$

Eq. (18) can then be rewritten as

$$\text{FF-DBF}(\tau, L, \bar{\delta}) > \max_k (Y_k(L) - (k - 1)\bar{\delta}L) \quad (19)$$

for some  $L \geq D_{\min}$ . Theorem 1 immediately follows, as the contrapositive of the above statement.  $\square$

**A schedulability test** Theorem 1 suggests the following strategy for checking whether a given sporadic task system is not EDF-schedulable on a specified virtual platform  $\Pi$ : determine whether there is any  $L \geq D_{\min}$  satisfying Inequality (19). If not, then  $\tau$  is guaranteed to be EDF-schedulable on  $\Pi$ .

While  $D_{\min}$  represents a lower bound on the range of values of  $L$  for which Inequality (17) must be tested, we do not yet have an upper bound. Hence, while we could start Inequality (17) with  $L \leftarrow D_{\min}$  and repeatedly increase the value of  $L$  being tested, it is not immediately evident when it would be safe to stop and conclude that  $\tau$  is in fact schedulable. To determine an upper bound for  $L$  that allows stopping the check of Inequality (17), we extend a technique previously used in uniprocessor [4] or multiprocessor [3] EDF schedulability tests.

A linear lower bound of each level- $j$  parallel supply function is given by Eq. (16). It is evident from the definition of FF-DBF (Definition 1; also see Figure 2) that  $(C_i + tU_i)$  is an upper bound on  $\text{FF-DBF}(\tau_i, t, \bar{\delta})$  for any  $t$ . Then, an upper bound for the LHS of Inequality (19) is

$$\text{FF-DBF}(\tau, L, \bar{\delta}) \leq L\bar{U} + \sum_{\tau_i \in \tau} C_i. \quad (20)$$

For the RHS of Inequality (19), an obvious lower bound is obtained applying Equation (16) for all values of  $k$ , that is

$$\max_k \{\alpha_k(L - \Delta_k)_0 - (k - 1)\bar{\delta}L\}. \quad (21)$$

Substituting all these bounds into Inequality (19), we conclude that in order for  $\tau$  to not be EDF-schedulable, it is necessary that some  $L \geq D_{\min}$  satisfies

$$\begin{aligned} \forall k, \quad L\bar{U} + \sum_{\tau_i \in \tau} C_i &> \alpha_k(L - \Delta_k) - (k - 1)\bar{\delta}L \\ \forall k, \quad L &< \frac{\alpha_k \Delta_k + \sum_{\tau_i \in \tau} C_i}{\alpha_k - \bar{U} - (k - 1)\bar{\delta}} \\ L &< \min_k \frac{\alpha_k \Delta_k + \sum_{\tau_i \in \tau} C_i}{\alpha_k - \bar{U} - (k - 1)\bar{\delta}}. \end{aligned} \quad (22)$$

Hence if Inequality (19) is to be satisfied for any value of  $L$ , it will be satisfied for some  $L$  no larger than the bound in Equation (22) above. Equivalently, if we have verified that Inequality (19) evaluates to true for all values of  $L$  up to the bound in Equation (22), we can safely conclude that task system  $\tau$  is indeed schedulable on the virtual platform  $\Pi$  when global EDF is being used as the local scheduling algorithm.

## 4.2 Demand based schedulability test

The second schedulability condition we present is based on the concept of interfering workload  $W_k$ , defined as the sum of the execution times of higher priority jobs interfering on  $\tau_k$ . Bertogna et al. proposed the following bound [5]:

$$W_k \leq \bar{W}_k = \sum_{\substack{i=1 \\ i \neq k}}^n \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min \left\{ C_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i \right\} \quad (23)$$

The interference  $I_k$  denotes instead the total duration in  $[0, D_k)$  in which  $\tau_k$  is ready to execute but it cannot be scheduled due to higher priority jobs or unavailable supply.

In [6], a method is presented to bound the interference  $I_k$  as a function of the interfering workload  $W_k$  when the virtual multiprocessor is abstracted through the MSF interface. The next theorem adapts this method to the PSF model adopted in this paper.

**Theorem 2** Consider a task set  $\tau$  that is scheduled on a virtual platform  $\Pi$  with maximum degree of parallelism  $m$ , that is characterized by the PSF  $\{Y_j(t)\}_{j=1}^m$ . Then, for each task  $\tau_k$ , the interference  $I_k$  is upper bounded by

$$I_k \leq \bar{I}_k = L_0 + \sum_{\ell=1}^m \min \left( L_\ell, \frac{\left( W_k - \sum_{p=0}^{\ell-1} p L_p \right)_0}{\ell} \right) \quad (24)$$

with  $\{L_\ell\}_{\ell=0}^m$  equal to

$$\begin{aligned} L_0 &= D_k - Y_1(D_k) \\ L_\ell &= 2Y_\ell(D_k) - Y_{\ell-1}(D_k) - Y_{\ell+1}(D_k) \\ L_m &= Y_m(D_k) - Y_{m-1}(D_k). \end{aligned} \quad (25)$$

**Proof** In [6], a similar theorem is proved for the case in which the platform is specified by means of a set of  $m$  individual supply functions  $\{Z_j(D_k)\}_{j=1}^m$ , where  $Z_j(D_k)$  represents the minimum supply granted by the  $j$ -th virtual processor in any interval of length  $D_k$ . The only difference lies in the values  $\{L_\ell\}_{\ell=0}^m$ , which were defined as

$$\begin{aligned} L_0 &= D_k - Z_1(D_k) \\ L_\ell &= Z_\ell(D_k) - Z_{\ell+1}(D_k) \\ L_m &= Z_m(D_k). \end{aligned} \quad (26)$$

We need to adapt this result to the case in which the platform is specified with the parallel supply functions  $\{Y_j(t)\}_{j=1}^m$ .

By Lemma 2, we know that for each platform represented by  $\{Y_j(t)\}_{j=1}^m$ , we can dynamically rename the processors over any interval of length  $D_k$ , such that the  $j$ -th processor is available for at least  $(Y_j(D_k) - Y_{j-1}(D_k))$  time units over this interval. This means that the platform can be represented as well by a set of  $m$  individual supply functions  $\{Z_j(D_k)\}_{j=1}^m$ , such that

$$Z_j(D_k) = (Y_j(D_k) - Y_{j-1}(D_k)).$$

The theorem follows replacing  $Z_j(D_k)$  in Equation (26) with the above expression. Note that, by Lemma 1, each  $L_\ell$  is non-negative, for all  $\ell$ .  $\square$

The next theorem easily follows, considering that a necessary condition for a deadline miss is that a task  $\tau_k$  should be interfered for more than its slack  $D_k - C_k$ .

**Theorem 3** A task set  $\tau = \{\tau_i\}_{i=1}^n$  is schedulable with EDF on a PSF platform modeled by  $\{Y_j\}_{j=1}^m$ , if

$$\forall k = 1, \dots, n \quad C_k + \bar{I}_k \leq D_k, \quad (27)$$

where  $\bar{I}_k$  is computed from Eq. (24).

It is possible to prove that Theorem 3 is superior, in terms of number of schedulable task sets detected, to the corresponding theorem in [6] based on MSF, because of the superiority of the PSF abstraction over the MSF.

We highlight that the bound on  $W_k$  expressed by Equation (23) can be refined using an iterative method described in [5]. However we do not report the details here, due to space limitations.

## 5 Related works

The virtualization of a resource is the process of providing a view that is independent of the physical implementation of the resource itself. One notable example of virtualization of computing devices is certainly the Java Virtual Machine [11] that provides an abstraction of the machine through a machine independent instruction set. This allows the portability of code from processor to processor without the need of re-compiling on the new architecture.

In real-time systems, the interface of a virtual platform describes the amount of computing resource that is provided. The virtualization of computing resource was extensively applied to uniprocessors in the past. Mercer et al. [14] proposed a resource reservation mechanism based on a required budget and period to provide an abstraction of a uniprocessor with reduced speed. Abeni and Buttazzo [1] proposed the Constant Bandwidth Server (CBS) to isolate an application requiring a varying amount of computation on a virtual processor with reduced speed.

Mok, Feng, and Chen [15] introduced the concept of “supply function” of a static time partition to measure the minimum amount of computing resource provided. This paper set the root of later research. Almeida and Pedreiras [2] applied similar techniques to schedule messages over the FTT-CAN network. Lipari and Bini [13] derived the set of supply functions that can feasibly schedule a given application. Shin and Lee [17] introduced the periodic resource model (that is a special class of supply functions) also deriving a utilization bound, later extended by Easwaran et al. [9] to account for a server deadline possibly different than the period.

Very recently, there has been an increasing interest in proposing interfaces for the computing power available on a multiprocessor. Leontyev and Anderson [12] proposed to abstract the amount of resource provided by a virtual multiprocessor using one single parameter: the bandwidth  $w$ . The authors propose 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 interface was provided.

Shin et al. [16] proposed a multiprocessor periodic resource model to describe the computational power supplied by a parallel machine. They modeled a virtual multiprocessor by the triplet  $\langle \Pi, \Theta, m' \rangle$ , meaning that an overall budget  $\Theta$  is provided by  $m'$  processors every period  $\Pi$ . The big advantage of this interface is that it is simple and captures the most significant features of the platform. Nonetheless, the aggregation of all the computing resource by a unique number  $\Theta$  leads to a more pessimistic analysis.

Chang et al. [7] 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. [6] proposed to abstract any parallel machine by associating a supply function [9, 13, 15, 17] to each sequential server, suggesting the Multi-Supply-Function (MSF) interface.

**Definition 8 (Def. 1 in [6])** A Multi Supply Function (MSF) of a set  $\Pi = \{\pi_j\}_{j=1}^m$  of VPs is a set of  $m$  supply functions  $\{Z_j\}_{j=1}^m$ , one for each VP  $\pi_j$ , respectively.

However, when tasks are allowed migrating from one virtual processor to another, this model can be too pessimistic because all the supply functions are derived assuming the worst-case condition for each virtual processor in isolation. We show this pessimism by the example of Figure 1, where the virtual platform provides time according to two static partitions: one that provides 2 time units every 4, and another one that provides 4 every 8. Following the approach suggested by Bini et al. [6] this platform should be modeled by two supply functions  $Z_1$  and  $Z_2$  each one associated to each of the two servers. Figure 6 reports the two supply functions.

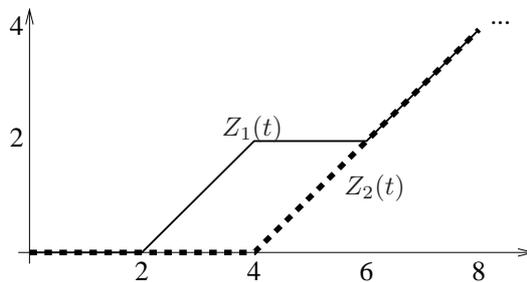


Figure 6. Example of two static partitions.

Suppose we have to schedule an aperiodic job that has a deadline  $D = 6$  from its arrival and a computation time of  $C = 4$ . If we abstract the platform by the two supply functions  $\{Z_1, Z_2\}$ , the job is not schedulable because none of the two supply functions can provide 4 time units in a 6 units interval. In fact  $Z_1(6) = Z_2(6) = 2 < 4$ . However, from Figure 1, it is clear that in any interval of length 6 there are always 4 time units provided by one processor. Note that the PSF abstraction can capture this notion. In fact, the definition of  $Y_1$  can take advantage of the time available on both processors (see Figure 4), so that in any interval of length  $D$ , there are at least  $Y_1(D) = Y_1(6) = 4$  time units provided by at most one processor. The schedulability of the aperiodic job is therefore assured.

## 6 Conclusions

In order to be able to build open environments — environments that provide support for multiple independently-developed applications — upon multiprocessor platforms, it is necessary that appropriate abstractions be devised for representing the computing capabilities of *parts* of the underlying multiprocessor platform. If these open environments are to be capable of hosting safety-critical applications, such abstractions must be strictly enforceable (in the sense that they correspond to strict guarantees of computing capability), and they must be formal enough and expressive enough that it is possible to formally establish the correctness (in particular, the timeliness) of real-time applications implemented upon these abstractions.

Over the past few years, a series of such abstractions, of increasing generality and expressiveness, have been proposed. The major insight that the community seems to have gathered in performing this work is that the critical information which needs to be represented in the abstraction is the degree of parallelism in the provided computing capability. Accordingly, these abstractions have aimed to maximize the amount of such information that is communicated. The Parallel Supply Function (PSF) abstraction proposed in this paper continues this trend. We have shown that the PSF abstraction is strictly more powerful than the prior ones — from [6, 16] — that support similar interfaces, in the sense that it preserves more of the parallelism information. To demonstrate the usability of this abstraction for building provably-correct real-time systems, we have derived sufficient schedulability tests that are able to determine whether a given sporadic task systems is schedulable by EDF upon the computing capabilities guaranteed by such an abstraction.

## References

- [1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19<sup>th</sup> IEEE Real-Time Systems Symposium*, pages 4–13, Madrid, Spain, December 1998.
- [2] Luís Almeida, Paulo Pedreiras, and José Alberto G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, December 2002.
- [3] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dublin, July 2008. IEEE Computer Society Press.
- [4] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11<sup>th</sup> IEEE Real-Time Systems Symposium*, pages 182–190, Lake Buena Vista (FL), U.S.A., December 1990.
- [5] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 2008.
- [6] Enrico Bini, Giorgio C. Buttazzo, and Marko Bertogna. The multy supply function abstraction for multiprocessors. In *Proceedings of the 15<sup>th</sup> IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 294–302, Beijing, China, August 2009.
- [7] Yang Chang, Robert Davis, and Andy Wellings. Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning. Technical Report YCS 432, University of York, 2008. available at <http://www.cs.york.ac.uk/ftplib/reports/2008/YCS/432/YCS-2008-432.pdf>.
- [8] Zhong Deng and Jane win-shih Liu. Scheduling real-time applications in Open environment. In *Proceedings of the 18<sup>th</sup> IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, U.S.A., December 1997.

- [9] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the 28<sup>th</sup> IEEE International Real-Time Systems Symposium*, pages 129–138, Tucson, AZ, USA, 2007.
- [10] Michael Fekete. Über die verteilung der wurzeln bei gewissen algebraischen gleichungen mit ganzzahligen koeffizienten. *Mathematische Zeitschrift*, 17:228–249, 1923.
- [11] James Gosling and Henry McGilton. The java language environment: A white paper. Technical report, Sun Microsystems, 1996. available at <http://java.sun.com/docs/white/langenv/>.
- [12] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the 20<sup>th</sup> Euromicro Conference on Real-Time Systems*, pages 191–200, Prague, Czech Republic, July 2008.
- [13] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the 15<sup>th</sup> Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.
- [14] Clifford W. Mercer, Stefan Savage, and Hydeuyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, Boston, MA, U.S.A., May 1994.
- [15] Aloysius K. Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings of the 7<sup>th</sup> IEEE Real-Time Technology and Applications Symposium*, pages 75–84, Taipei, Taiwan, May 2001.
- [16] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering multiprocessors. In *Proceedings of the 20<sup>th</sup> Euromicro Conference on Real-Time Systems*, pages 181–190, Prague, Czech Republic, July 2008.
- [17] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24<sup>th</sup> Real-Time Systems Symposium*, pages 2–13, Cancun, Mexico, December 2003.