A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation

Giuseppe Lipari, Enrico Bini
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {g.lipari,e.bini}@sssup.it

***Abstract*—Hierarchical scheduling is a promising methodology for designing and deploying real-time applications, since it enables component-based design and analysis, and supports temporal isolation among competing applications. In hierarchical scheduling an application is described by means of a temporal interface. The designer faces the problem of how to derive the interface parameters so to make the application schedulable, at the same time minimizing the waste of computational resources. The problem is particularly relevant in multiprocessor systems, where it is not clear yet how the interface parameters influence the schedulability of the application and allocation on the physical platform.**

In this paper we present three novel contributions to hierarchical scheduling for multiprocessor systems. First, we propose the *Bounded-Delay Multipartition* (BDM), a new interface specification model that allows the designer to balance resource usage versus flexibility in selecting the virtual platform parameters. Second, we explore the schedulability region of a real-time application on top of a generic virtual platform, and derive the interface parameter. Finally, we propose *Fluid Best-Fit*, an algorithm that takes advantage of the extra degree of flexibility provided by the BDM to compute the virtual platform parameters and allocate it on the physical platform. The performance of the algorithm is evaluated by simulations.

I. Introduction

Multiprocessor systems are becoming increasingly commonplace, not only in desktop/laptop PCs and in servers, but also in embedded systems [1], [2]. This trend is expected to increase in the near future. Following the trend, real-time researchers focused on multiprocessor scheduling and schedulability analysis, in some cases extending existing techniques proposed for single processors to multiprocessors. It is the case of *hierarchical scheduling methodologies* [3], [4], [5], [6], which are regarded as useful tools to handle the complexity of medium to large-sized applications and enable a component-based approach to schedulability analysis; also, such techniques are helpful for providing temporal isolation and timing guarantees in open systems, and for enabling application-specific schedulers (also called *local schedulers*).

In the hierarchical scheduling model, the computational requirement of an application $\mathcal{A}$ is described by a *temporal interface* $\mathcal{I}$. After the application is admitted into the system, a *root-level scheduler* is in charge to accommodate all interfaces onto the available physical resources. The design of a hierarchical scheduling system involves the following phases summarized in Figure 1.

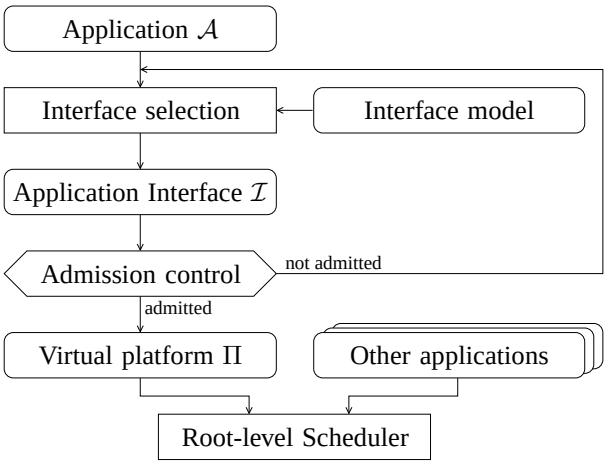

Figure 1: Design phases of a hierarchical scheduling system.

1) Interface Specification: At design time, the application designer must characterize the temporal requirements of the application, and derive an appropriate temporal interface specification that summarizes the requirements. We distinguish two aspects here: the choice of an *interface model*, and the selection of the parameters in accordance with the chosen model.

Given an interface model, the designer needs to instantiate an interface $\mathcal{I}$ so that the application is guaranteed on it. When selecting the "optimal" interface parameters for an application, the designer must trade-off different goals. For example, in [7], we proposed a methodology for single processor systems that, starting from the worst-case requirements of the application tasks, derives an interface based on the bounded partition model (α, Δ) from the application requirements, trading off the maximum delay Δ (that we want as large as possible to reduce overhead) versus the maximum bandwidth α, (that we want as little as possible to reduce usage of resource). When the application is a set of tasks scheduled by EDF, the optimal design problem has

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7 under grant agreement n.248465 "S(o)OS Service-oriented Operating Systems."

been analytically solved [8].

2) Run-time allocation: In this paper we assume an *open system*, where applications can dynamically join and leave. When an application joins the system, it presents its interface to the *admission control* policy, which performs a feasibility analysis to check if the application can be safely admitted without compromising the guarantees of the existing applications. If the answer is positive, the system instantiates a *virtual platform* Π that respects the temporal interface. The virtual platform is then scheduled on the physical platform together with the other virtual platforms of already running applications.

If the application cannot be admitted, the designer must go back to the interface specification and derive new interface parameters that enables a wider search in the design space.

In this paper, we investigate how the selection of the interface parameters influence the schedulability of the application on the virtual platform on one side, and the problem of allocating the virtual platforms on the physical processors, on the other side.

A. Related Work

In single processors, Mok et al. [9] proposed the *bounded-delay partition*, Shin and Lee [6] proposed the *periodic resource* model, Easwaran et al. [10] extended the periodic resource model by allowing deadline different from period.

Recently, some authors have addressed the problem of how to specify the application interface for an application to be executed on multiprocessor systems, and provide appropriate schedulability analysis to check if the application is schedulable on the interface.

Leontyev and Anderson [11] proposed to use the only overall bandwidth requirement w as interface for soft real-time applications. 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, Easwaran and Lee [12] proposed the multiprocessor periodic resource model (MPR): each application is assigned a set of periodic m reservations $\{(Q_i, P)\}$ all with the same period. This interface model is quite intuitive, but it has a drawback: it implicitly requires the synchronization between reservations running on different processors that is difficult to implement in a real system; when periods are not synchronized, it does not exist a worst-case scenario of the resource allocation, as explained in Section IV-A.

Chang et al. [13] 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. [14] proposed the Parallel Supply Function (PSF) interface of a virtual multiprocessor and developed a global EDF test developed on top of it. However the assignment of the parameters of the virtual platforms is not investigated.

B. Contributions of this paper

In this paper we propose a framework for designing hierarchical scheduling systems that covers both phases of the design. First, we propose a novel interface model, called *bounded-delay multipartition* (BDM) interface that allows the designer to balance the amount of consumed bandwidth vs. the flexibility of the interface, making easier the admission control problem. Second, rather than simply checking the schedulability of an application, as all past works did, we mostly focus on the derivation of the interface starting from the application requirement. For this purpose, we propose a schedulability test from which the impact of the interface is more apparent.

Third, we propose an allocation policy, called *fluid best-fit* that performs admission control and, at the same time, instantiates the virtual platform parameters from the interface specification so as to optimize the underlying resource allocation. In addition, our model is suitable for a number of extension and modifications, so to cope with additional constraints and goals. We demonstrate by experiments that, thanks to the extra level of flexibility allowed by the interface model, our allocation policy performs better that existing policies.

II. System model

The overall system is composed of a set of *real-time applications* $\{\mathcal{A}^\ell\}$ that run concurrently onto a multiprocessor constituted by M processors. Some applications are always running, while some others dynamically join and leave the system. To enable composability and isolation, each application $\mathcal{A}^\ell$ is executed onto a dedicated *virtual platform* Π^ℓ. The real-time requirements of the application $\mathcal{A}^\ell$ are guaranteed onto the platform Π^ℓ by a *guarantee test* $\mathcal{T}^\ell$.

In the rest of this section we provide a more detailed model of each notion we introduced above. Since we focus on each application in isolation, from now on we drop the index ℓ of the application. We denote $\max\{0, \cdot\}$ with $(\cdot)_0$.

A. Application model

The application $\mathcal{A}$ is composed of a set of n *independent sporadic tasks* $\{\tau_1, \ldots, \tau_n\}$. Every time a task is activated, a *job* must be executed. The *minimum interarrival time* T_i is the minimum separation between two consecutive jobs of τ_i. Each job of τ_i has a *computation time* C_i and must be completed within a *deadline* D_i from its activation.

B. Platform model

The virtual platform Π is modeled by a set of virtual processors $\{\pi_1, \ldots, \pi_m\}$ that we also call *virtual multiprocessor*.

A platform interface $\mathcal{I}$ is a predicate on the values that the virtual platform parameters may have. Hence, an interface $\mathcal{I}$ yields naturally the subset of all virtual platforms that are compliant with it. We denote this subset of platforms by $\mathbf{\Pi}(\mathcal{I})$. Examples of interface specifications are: "all the platforms with an overall bandwidth of 2.5", or "all the platforms in which one virtual processor has a bandwidth of at least 0.8", etc. An interface that specifies many constraints yields a small set $\mathbf{\Pi}(\mathcal{I})$. On the other hand, if we specify only loose constraints in $\mathcal{I}$, the set $\mathbf{\Pi}(\mathcal{I})$ becomes larger.

In this paper, we assume that the virtual platform is deployed on the physical platform using partitioning. Once the application is admitted and the virtual platform is created, each one of the virtual processors is allocated on one of the physical processors. We choose the partitioning approach for practical and theoretical issues. From a practical point of view, partitioning is easier to implement in a multicore operating system, as it reduces the amount of shared data structures. Also, it can be advantageous for applications. In facts, many multiprocessor systems have a non-uniform memory architecture (NUMA machines), and it is desirable that tasks belonging to the same application do not arbitrarily migrate across the entire physical platform, but only on the subset of processors that have the same latency in accessing the memory. In our model, this can be achieved by allocating all the virtual processors of the same virtual platform on a group of "neighbor" processors.

Also, as we will show in Section VI, the partition approach coupled with our the flexible interface specification that we will introduce in Section IV allows a high density of *packing* for the virtual platforms. This property leads to a better utilization of the overall system, as well as to the possibility to power-off or put in stand-by mode non used processors.

C. Model of the guarantee test

We model a guarantee test $\mathcal{T}(\mathcal{A}, \Pi)$ as a boolean function that returns TRUE if the application $\mathcal{A}$ is *guaranteed* on platform Π, FALSE otherwise. We also extend a guarantee test to an interface $\mathcal{I}$ as follows

$$\mathcal{T}(\mathcal{A}, \mathcal{I}) = \bigwedge_{\Pi \in \mathbf{\Pi}(\mathcal{I})} \mathcal{T}(\mathcal{A}, \Pi). \tag{1}$$

Guaranteeing an application over an interface $\mathcal{I}$ would require to test it over all the platforms in $\mathbf{\Pi}(\mathcal{I})$ unless it exists a *worst-case platform*.

Definition 1: Given an interface $\mathcal{I}$ and a test $\mathcal{T}$, we say that Π^{wc} is the worst-case platform of $\mathcal{I}$ when:

$$\Pi^{\text{wc}} \in \mathbf{\Pi}(\mathcal{I}) \tag{2}$$

$$\forall \mathcal{A} \qquad \mathcal{T}(\mathcal{A}, \Pi^{\text{wc}}) \Rightarrow \mathcal{T}(\mathcal{A}, \mathcal{I}). \tag{3}$$

For example, in the uni-processor case the worst-case platform Π^{wc} of a periodic interface $\mathcal{I}$ that provides a budget Q every period P occurs when there is an idle interval $[0, 2(P-Q)]$, and the budget Q is provided at the end of the server period [7], [6]. The existence of the worst-case platform Π^{wc} for an interface $\mathcal{I}$ justifies the advantages of an interface-based analysis, since if the application is guaranteed on Π^{wc} then it is guaranteed on any platform in $\mathbf{\Pi}(\mathcal{I})$, so that during the allocation phase the designer can freely select any platform in $\mathbf{\Pi}(\mathcal{I})$.

In the context of real-time applications, the guarantee test $\mathcal{T}$ is also called *schedulability test*: if $\mathcal{T}(\mathcal{A}, \Pi)$ returns TRUE then no task deadline will be missed when $\mathcal{A}$ runs on Π. On the other hand the test $\mathcal{T}$ can also encode other kinds of requirements: the minimum throughput of an MPEG decoder, an average response time with some confidence level, etc.

III. Schedulability test on a virtual platform

Applications must be guaranteed on the corresponding platforms. In the next section we recall a tight description of a platform that is well suited for schedulability tests.

A. The Parallel Supply Functions of a platform

To introduce the minimum possible pessimism in abstracting the amount of resource provided by a platform, we first adopt the Parallel Supply Function (PSF) abstraction, recently introduced by Bini et al. [14]. Without entering all the details of the definition (that can indeed be found in [14]), we recall here the basic concepts.

Definition 2: Given a virtual platform Π composed by the m virtual processors $\{\pi_1, \ldots, \pi_m\}$, its PSF is composed by the set of functions $\{Y_k\}_{k=1}^{m}$, where $Y_k(t)$ is the *minimum* amount of resource provided in *any* interval of length t with a parallelism *at most* k.

To clarify this definition we propose an example (please refer to Figure 2). Suppose that in the interval $[0, 11]$ the three virtual processors $\{\pi_1, \pi_2, \pi_3\}$ composing the virtual platform, provides resource in accordance to the schedule drawn in gray.

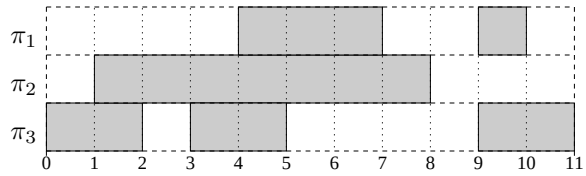

Figure 2: From a resource schedule to the PSF.

In this case $Y_1(11) = 10$ because there is always at least one processor available in $[0, 11]$ except in $[8, 9]$. Then $Y_2(11) = 16$, that is found by summing up all the resource except one with parallelism 3 (provided only in $[4, 5]$). Finally $Y_3(11) = 17$ that is achieved by summing all the resources provided in $[0, 11]$.

In general, the parallel supply functions are computed also by sliding the time window of length t and by searching for the most pessimistic scenario of resource allocation. This

minimization is somehow equivalent to the one performed on uni-processor hierarchical scheduling [7], [6].

B. A schedulability test on the PSF interface

Since we aim at describing all possible interfaces $\mathcal{I}$ that can guarantee the given application $\mathcal{A}$, we find it useful to propose a schedulability condition that is equivalent to Theorem 2 in [14]. We choose this condition because it applies to several different local schedulers such as global EDF or global FP, but it applies to constrained deadline tasks, i.e. for all tasks $\tau_i \quad D_i \leq T_i$. While choosing other tests is possible [15], the proposed (equivalent) formulation has the advantage of highlighting the constraint on the interface.

Theorem 1: An application $\mathcal{A} = \{\tau_i\}_{i=1}^n$ is schedulable on a virtual platform Π modeled by the PSF $\{Y_k\}_{k=1}^m$, if

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} k\,C_i + W_i \leq Y_k(D_i), \tag{4}$$

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

$$W_i = \sum_{j=1, j\neq i}^{n} \left\lfloor \frac{D_i}{T_j} \right\rfloor C_j + \min\left\{ C_j, D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right\}, \tag{5}$$

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

$$W_i = \sum_{j \in \mathsf{hp}(i)} W_{ji}, \tag{6}$$

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

$$W_{ji} = N_{ji} C_j + \min\{C_j, D_i + D_j - C_j - N_{ji} T_j\} \tag{7}$$

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

Proof: The interfering workload W_i is an upper bound to the amount of work that can be requested in $[0, D_i]$ by tasks with priority higher than τ_i [16]. The workload W_i interferes on τ_i only if it occupies all the available processors (otherwise τ_i could execute). The *interference* I_i is maximized when the work W_i is executed at the lowest possible parallelism. In the example of Figure 3, the workload $W_i = 8$ causes the maximum interference when it is executed for one time unit on a single processor, for 2 units at parallelism of 2 and for one last unit at parallelism 3. To reach the interference $I_i = 6$ we must also account for the two time units with no resource available.

If we call k^* the highest degree of parallelism that is occupied by W_i, then

$$k^* D_i - k^* I_i = Y_{k^*}(D_i) - W_i. \tag{8}$$

This relationship can be explained in Figure 3, by expressing the work represented in the dashed box by $k^* D_i - k^* I_i$ and by $Y_{k^*}(D_i) - W_i$. In the figure $k^* = 3$, on top of the figure a legend explains how $Y_k(D_i)$ and W_i are depicted. Equation (8) can be rewritten as

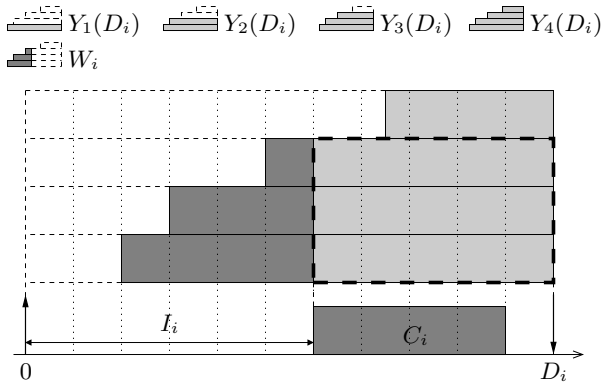

Figure 3: Schedulability of task τ_i onto the virtual platform.

$$I_i = D_i - \frac{Y_{k^*}(D_i) - W_i}{k^*}. \tag{9}$$

By observing that the evaluation of the RHS of (9) for any other index $k \neq k^*$ is not smaller than I_i, it follows

$$\begin{aligned} I_i &= \min_{k=1,\ldots,m} \left\{ D_i - \frac{Y_k(D_i) - W_i}{k} \right\} \\ &= D_i - \max_{k=1,\ldots,m} \left\{ \frac{Y_k(D_i) - W_i}{k} \right\}. \end{aligned} \tag{10}$$

Hence the classic interference-based schedulability test [16]

$$\forall i = 1, \ldots, n \qquad C_i + I_i \leq D_i,$$

becomes

$$\forall i = 1, \ldots, n \qquad C_i \leq \max_{k=1,\ldots,m} \left\{ \frac{Y_k(D_i) - W_i}{k} \right\},$$

which can be rewritten with the AND ($\wedge$) and OR ($\vee$) as

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1,\ldots,m} C_i \leq \frac{Y_k(D_i) - W_i}{k},$$

from which the Theorem follows. ■

IV. The Bounded-Delay Multipartition model

The PSF could be indeed used a tight interface model. However it is too detailed to be intuitively handled by the designers, whereas it is often highly desirable to provide a simpler and more manageable interface.

A. Inappropriateness of the periodic interface

A natural candidate for a simple interface is the specification of a common period P among all the virtual processors $\{\pi_1, \ldots, \pi_m\}$ and an overall budget Q that is shared by all the π_k's. Following this idea, Shin et al. [12] proposed the multiprocessor periodic resource model (MPR).

According to the MPR interface the virtual multiprocessor is abstracted by three parameters: a period P, an overall

budget Q, and a maximum parallelism $m \leq M$. In [12], the authors implicitly assumed a tight synchronization among the virtual processors π_k that ensures that all virtual processor implementation (that we call servers) are activated simultaneously on all the processors. Unfortunately, due to the difficulty of synchronizing clocks among different processors, this hypothesis often cannot be guaranteed. If this hypothesis is removed, the periodic interface becomes inappropriate for a very subtle reason that may however cause a deadline miss: The worst-case platform Π^{wc} does not exist for the MPR interface. We show this by an example.

Suppose an MPR interface $\mathcal{I}$ specifies a virtual platform composed by 2 virtual processors that provide an overall budget of $Q = 8$ time units with a period of $P = 8$. The interface does not specify how the budget is split between Q_1 and Q_2 on the two virtual processors π_1 and π_2, respectively. In Figure 4 we show some possible scenarios of distribution of the budget $Q = 8$. At the bottom of the figure we report the worst-case resource schedule as Q_1 ranges from 8 to 4 (and Q_2 varies accordingly from 0 to 4). These schedules are worst in the sense that the overall resource provided in $[0, t]$ is minimal. In the resource schedule, a vertical thick black line is drawn at each server period. In the upper part we show the cumulative supply function Y_2 that measures the amount of resource provided in each scenario.

It can be noticed that *almost always* the worst case of the parallel supply function Y_2 happens when the budget is evenly divided between the two virtual processors ($Q_1 = Q_2 = 4$). This result would be in accordance to well-known results on uniform multiprocessor scheduling, where the worst-case speed distribution over a multiprocessor is the case when all the speeds are equal to each other [17]. Unexpectedly, assigning the two budgets $Q_1 = 6$ and $Q_2 = 2$ leads to the most pessimistic condition (minimum value of Y_2) for an interval of length 12. Hence, an application that is schedulable on a "more difficult" platform (the one with $Q_1 = Q_2 = 4$) may be not schedulable on an apparently "easier" platform (the one with $Q_1 = 6$ and $Q_2 = 2$). It follows that there is no worst-case platform Π^{wc} for the non-synchronized MPR interface.

In the next Section we propose an interface that does not suffer this drawback, and we formalize the concept of *concavity* of the platform which measures the intuitive concept of "difficulty" of schedulability on the platform.

B. The proposed interface model

The problem highlighted in Section IV-A happens because the supply functions grow discontinuously. If the supply functions of the virtual processors are linear, this phenomenon does not happen. This observation leads us to formulate the following interface model of a multiprocessor, based on an extension of the bounded-delay time partition [9].

Definition 3: An interface $\mathcal{I}$ is a *bounded-delay multipartition (BDM)* interface $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$ with

$$\Delta \geq 0,$$
$$\forall k = 1, \ldots, m \quad 0 \leq \beta_k - \beta_{k-1} \leq 1, \tag{11}$$
$$\forall k = 1, \ldots, m \quad \beta_k - \beta_{k-1} \geq \beta_{k+1} - \beta_k, \tag{12}$$

when the following two statements are equivalent

- $\Pi \in \mathbf{\Pi}(\mathcal{I})$
- the PSFs $\{Y_k\}$ of Π are

$$\forall k = 1, \ldots, m, \ \forall t \geq 0 \quad Y_k(t) \geq \beta_k (t - \Delta)_0. \tag{13}$$

For notational convenience, we define $\beta_0 = 0$ and $\beta_k = \beta_m$ for all $k > m$.

The BDM offers a greater simplicity compared with the PSF interface. However, it certainly introduces some resource waste similarly to what happens with the uniprocessor bounded-delay time partition.

The main difference between the BDM interface and the MPR interface is that the time granularity is specified by a common delay Δ (that represents the length of the longest interval with no resource) rather than by a common period P among the virtual processors. However this small difference enables the statement of the following Theorem that would be otherwise impossible to prove.

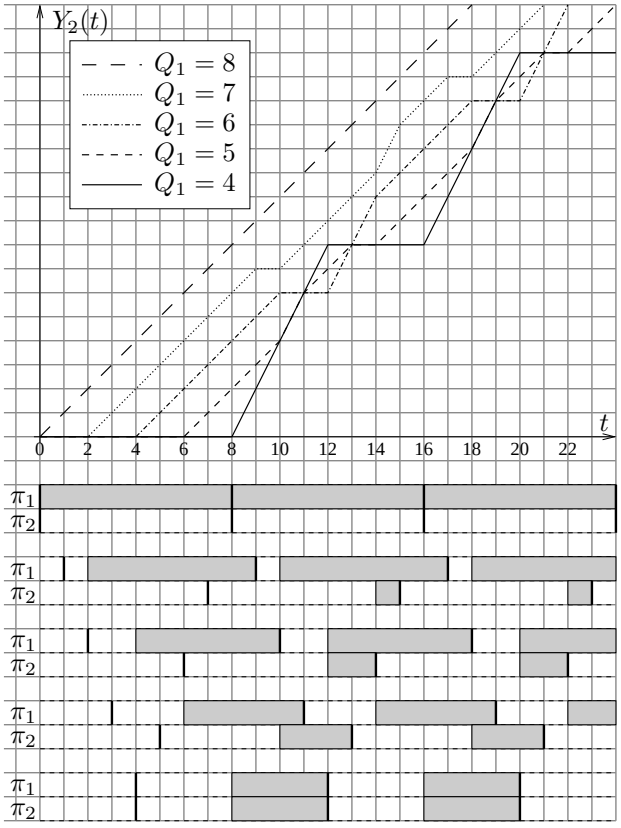

Figure 4: A periodic interface is inappropriate for multiprocessor.

Theorem 2: Let $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$ be a BDM interface. Its worts-case virtual platform Π^{wc} is the set of m bounded-delay virtual uni-processors [9]

$$\Pi^{\text{wc}} = [(\alpha_1, \Delta), \ldots, (\alpha_m, \Delta)] \tag{14}$$

with

$$\forall k = 1, \ldots, m \quad \alpha_k = \beta_k - \beta_{k-1}. \tag{15}$$

Proof: From (11) and (12) it follows that

$$1 \geq \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_m \geq 0. \tag{16}$$

Because of the ordering (16), the PSFs $\{Y_k^{\text{wc}}\}$ of the platform Π^{wc} defined by (15), are

$$Y_k^{\text{wc}}(t) = \sum_{i=1}^{k} \alpha_i (t - \Delta)_0 = \beta_k (t - \Delta)_0,$$

from which it follows directly that $\Pi^{\text{wc}} \in \mathbf{\Pi}(\mathcal{I})$.

If a real-time application $\{\tau_1, \ldots, \tau_n\}$ is schedulable over Π^{wc} by Theorem 1, we have:

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} kC_i + W_i \leq \beta_k (D_i - \Delta)_0,$$

from which it follows the schedulability on any other platform $\Pi' \in \mathbf{\Pi}(\mathcal{I})$, by the property (13). Hence Π^{wc} is the worst-case platform. ∎

As explained in Section IV-A, Theorem 2 cannot be stated for the MPR interface, since it is not possible to define a worst-case platform for it.

Since from now on we will only consider platforms with the same delay Δ, for notational convenience we identify a virtual platform Π only by the array of bandwidths $[\alpha_1, \ldots, \alpha_m]$. Without loss of generality we assume the α_k to be sorted non-increasingly.

For the purpose of an intuitive design space exploration, the BDM model enables a simple description of $\mathbf{\Pi}(\mathcal{I})$.

Corollary 1: Let $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$ be a BDM interface. A platform $\Pi = [\alpha_1, \ldots, \alpha_m]$ belongs to $\mathbf{\Pi}(\mathcal{I})$ when

$$\forall k = 1, \ldots, m \quad \sum_{i=1}^{k} \alpha_i \geq \beta_k. \tag{17}$$

Proof: The corollary follows from the observation that a platform with bandwidths defined in accordance to (17) has the PSFs $\{Y_k\}$ that respect (13). ∎

In practice, if we test our application on the platform Π^{wc}, then at run-time we can choose any other virtual platform $\Pi \in \mathbf{\Pi}(\mathcal{I})$ by simply moving bandwidth from a virtual processor π_k to another one π_ℓ with $\alpha_k \leq \alpha_\ell$. This can be viewed as a sort of compacting the bandwidth on the "heaviest" virtual processors. This adjustment of the platform allows a degree of flexibility at run-time that can be exploited by the allocation algorithm. In Section VI we will present an algorithm that will take advantage of this flexibility.

The proposed interface has several additional advantages: it does not rely on synchronization of the virtual resources; it does not rely on a specific underlying mechanisms (i.e. periodic servers), and can be applied to any bounded-delay partition (e.g. P-fair [18], static time partition [9]).

C. Example

To better clarify the BDM interface model and the application of Theorem 2 and Corollary 1, we present a simple example. Let us suppose that our application presents a BDM interface $\mathcal{I} = (3, 6, [0.7, 1.2, 1.4])$.

From (15) it follows that the worst-case platform is $\Pi^{\text{wc}} = [0.7, 0.5, 0.2]$ (we recall that these values are the bandwidths of the virtual processors $\pi_k \in \Pi$). Thanks to Corollary 1 it is possible to move some bandwidth from π_3 to π_2 achiving, for example, a platform $\Pi' = [0.7, 0.7]$, compatible with $\mathcal{I}$. Moreover it is possible to move again bandwidth from π_2 to π_1 achieving another compatible platform $\Pi'' = [1, 0.4]$. If instead, starting from Π^{wc} we move bandwidth from π_2 to π_3 we can find a platform incompatible with $\mathcal{I}$ such as $\Pi''' = [0.7, 0.4, 0.3]$. In fact, in this case the constraint (17) for $k = 2$ is violated since $\alpha_1''' + \alpha_2''' = 1.1 < \beta_2 = 1.2$.

D. Application schedulability vs. allocation flexibility

The amount of resource consumed by an interface can be roughly summarized by overall bandwidth β_m. It is however convenient also to formally represent the accuracy of an interface $\mathcal{I}$. For this reason we introduce the following index.

Definition 4: The *concavity index* (or simply concavity) of virtual platform $\Pi = [\alpha_1, \ldots, \alpha_m]$, is defined as:

$$c(\Pi) = \max_{k=1,\ldots,m-1} (\alpha_k - \alpha_{k+1}). \tag{18}$$

Definition 5: The *concavity index* (or simply concavity) of interface $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$ is defined as:

$$c(\mathcal{I}) = c(\Pi^{\text{wc}}) = \max_{k=1,\ldots,m-1} (2\beta_k - \beta_{k-1} - \beta_{k+1}), \tag{19}$$

where Π^{wc} is the worst-case platform of the interface $\mathcal{I}$.

As an example, the concavity of the interface of the example in Section IV-C is $c(\mathcal{I}) = \max\{0.7 - 0.5, 0.5 - 0.2\} = 0.3$, while the concavity of the more compact platform Π'' is $c(\Pi'') = 0.6$.

For any interface, the minimum value that the concavity index can assume is 0, and it corresponds to an interface with $\forall k, \beta_k = \frac{k}{m}\beta_m$ and a worst-case platform with $\alpha_k = \frac{\beta_m}{m}, \forall k$ (from Theorem 2). Therefore, we can say that a smaller concavity implies a more "difficult" interface for the application, since the application must be schedulable on a platform where all virtual processors have similar bandwidth. Notice also that when concavity is zero, the number of virtual platforms compliant with the interface is the largest (from Corollary 1). Hence this interface corresponds to a scenario with minimum schedulability and maximum platform flexibility.

When the concavity index is large, instead, the virtual platform is unbalanced, having some virtual processors with large bandwidth, and others with small bandwidth. Consider the case of an interface with total bandwidth $\beta_m \notin \mathbb{N}$, distributed as follows: $\mathcal{I} = \{\lceil \beta_m \rceil, \Delta, [1, 2, \ldots, \lfloor \beta_m \rfloor, \beta_m]\}$. It is easy to show that the concavity of this interface is $c(\mathcal{I}) = \lfloor \beta_m \rfloor - \beta_m + 1$. In this case, no platform other than $\Pi^{\mathsf{wc}} = [1, 1, \ldots, \lceil \beta_m \rceil - \beta_m]$ is compliant with the interface. The interface is "easier" from a schedulability point of view (larger values of $Y_k(t)$), however its flexibility is minimal, since it requires the availability of $\lfloor \beta_m \rfloor$ empty processors.

Summarizing, the concavity index is a measure of the trade-off between "schedulability" and "flexibility" of the interface. A small concavity index is more difficult from a schedulability point of view (thus it might imply more wasted bandwidth), but it allows a high number of compliant virtual platforms and possibly a more effective allocation; a large concavity index implies a higher degree of schedulability of the application (allowing to spare some bandwidth) but a minor number of compliant virtual platforms.

V. From the application to the interface

To enable an allocation phase onto larger exploration spaces, it is always convenient to select the interface $\mathcal{I}$ with the largest possible set of platforms $\mathbf{\Pi}(\mathcal{I})$, among the ones that can guarantee the application. For this reason we introduce the following definition.

Definition 6: Given an application $\mathcal{A}$ and a guarantee test $\mathcal{T}$, we say that an interface $\mathcal{I}$ is *maximal* when:

$$\begin{aligned} &\mathcal{T}(\mathcal{A}, \mathcal{I}), \\ &(\mathcal{T}(\mathcal{A}, \mathcal{I}') \ \wedge \ \mathbf{\Pi}(\mathcal{I}) \subseteq \mathbf{\Pi}(\mathcal{I}')) \Rightarrow \mathcal{I} = \mathcal{I}'. \end{aligned} \tag{20}$$

Hence the interface selection should be performed onto maximal interfaces. For real-time applications, the schedulability test of Theorem 1 can be used to readily derive the maximal BDM interfaces $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$ that can guarantee the real-time requirement of the application.

From Theorem 1 and the PSF of an interface $\mathcal{I}$ (reported in (13)), it immediately follows that the application is guaranteed when

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} kC_i + W_i \leq \beta_k (D_i - \Delta)_0$$

or, by writing the constraint on the bandwidths α_k, equivalently

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k=1\ldots,m} \sum_{j=1}^{k} \alpha_j (D_i - \Delta)_0 \geq kC_i + W_i \tag{21}$$

keeping in mind that the α_k are sorted decreasingly.

To provide a deeper understanding of the space of possible selections for the interface $\mathcal{I}$, we illustrate (21) by an example. Suppose we have the real-time application $\mathcal{A}$ whose parameters are reported in Table I. Let us schedule this application by local fixed priority. In this case the interfering workload W_i of each task τ_i can be computed according to (6).

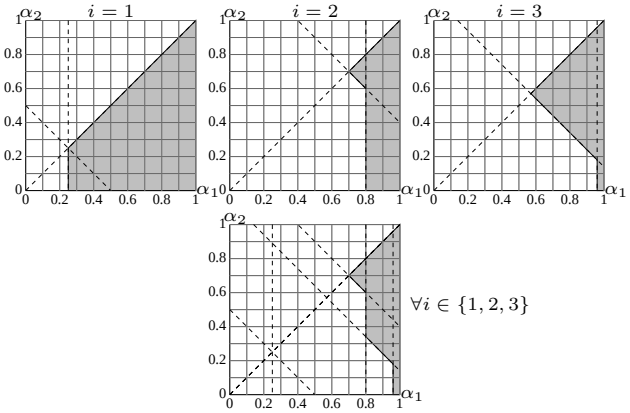

Figure 5: The space of α_1 and α_2.

i	C_i	T_i	D_i	W_i
1	1	6	6	0
2	15	27	27	5
3	9	52	52	39

Table I: An example of application.

First of all, we set $m = 2$ and $\Delta = 2$. By writing explicitly (21), we find the following relationships on α_1 and α_2

$$\begin{array}{lll} i = 1 & k = 1 & \alpha_1 \geq \frac{1}{4} \\ & k = 2 & \alpha_1 + \alpha_2 \geq \frac{1}{2} \\ i = 2 & k = 1 & \alpha_1 \geq \frac{4}{5} \\ & k = 2 & \alpha_1 + \alpha_2 \geq \frac{1}{5} \\ i = 3 & k = 1 & \alpha_1 \geq \frac{48}{50} \\ & k = 2 & \alpha_1 + \alpha_2 \geq \frac{37}{50} \end{array}$$

Moreover we assume that all α_k's are sorted decreasingly. Hence $\alpha_1 \geq \alpha_2$. In Figure 5 we draw in gray the space of all feasible selections of the array $[\alpha_1, \alpha_2]$ of an interface with $m = 2$ and $\Delta = 2$.

$\beta_1 = \alpha_1$	β_2	$\alpha_2 = \beta_2 - \beta_1$	$c(\Pi)$
0.7	1.4	0.7	0
0.8	1.14	0.34	0.46
0.96	0.96	0	0.96

Table II: Candidate interfaces of the application.

In this case there are three maximal candidate interfaces to represent the application requirement whose parameters are reported in Table II. In Figure 5 the maximal interfaces are graphically represented as the left corners of the bottom figure. The first choice consumes the maximum bandwidth, although it leaves to the allocation phase the maximum degree of freedom. On the other hand, the last one consumes indeed the minimum amount of bandwidth, however it may be harder to allocate, since it requires a bandwidth of 0.96 on a single processor.

We highlight that (21) defines the design space of the bandwidths of virtual processors belonging to the platform. Instead we let the designer to freely choose the value of Δ to balance between schedulability and overhead cost. In the future we plan to solve analytically this step as well as it was done in the uniprocessor case [8].

VI. From interface to allocation

In this section, we present an on-line algorithm called FLUIDBESTFIT that, given an interface specification $\mathcal{I}$, selects one platform $\Pi \in \mathbf{\Pi}(\mathcal{I})$ and allocates it on the physical platform.

The algorithm is described in Figure 6. In short, the algorithm performs a Best-Fit Decreasing partitioning of the Π^{wc} platform of the interface $\mathcal{I}$. Then, it tries to find a more compact platform $\Pi \in \mathbf{\Pi}(\mathcal{I})$, by moving bandwidth from virtual processors with higher index to the ones with lower index (thanks to Theorem 2 this move always preserves the schedulability of the application).

We assume that the real platform consists of M identical processors, and we denote by U_k the amount of bandwidth allocated on the **physical** processor k. Initially the physical multiprocessor is empty, so all $U_k = 0$. The procedure takes as parameters the interface specification $\mathcal{I}$ and a boolean variable ALLOCATED. The procedure is called with ALLOCATED set to FALSE when the application joins the system, and it sets ALLOCATED to TRUE if the application has been successfully allocated onto the available physical platform. The array $[\mathsf{cpuID}_1, \ldots, \mathsf{cpuID}_m]$ contains the processor index where π_i has been allocated.

Initially, the procedure makes a local copy of all U_k (line 2). Then, if the application is joining the system for the first time, it computes the worst-case platform Π^{wc} and sets all elements of the array cpuID_i to -1 (lines 3–7). We assume that all π_i are ordered in non-decreasing order of α_i.

Then, for each virtual processor π_h, it first tries to allocate it on one of the physical processors, using the best-fit strategy (line 11). If the bandwidth α_h does not fit in any of the physical processors, the allocation fails, and sets ALLOCATED to FALSE (lines 12–15). Otherwise, let k be the physical processor on which the virtual processor has been allocated.

It may happen that after the allocation, some free space is left on processor k. At this point we try to modify the virtual platform by increasing the α_h and decreasing the bandwidth of successive virtual processors, until we fill processor k (while cycle in lines 20–30). This transformation preserves the platform $\Pi \in \mathbf{\Pi}(\mathcal{I})$: in fact, if the condition $\forall j, \sum_{i=1}^{j} \alpha_i \geq \beta_j$ is respected before the transformation, it is still respected after the transformation (see Corollary 1).

The bandwidth of which virtual processors can we decrease? In our algorithm, we choose to decrease the bandwidths $[\alpha_{h+1}, \ldots, \alpha_l]$, with $l > h$, taking care of preserving the decreasing order (lines 21–28). While it would be possible to select differently the indexes of virtual processors from which we steal bandwidth, our choice has the advantage of reducing the maximum bandwidth across all following virtual processors, thus favoring their allocation with the best-fit strategy.

```
1:  procedure FLUIDBESTFIT(I, ALLOCATED)
2:      ∀k  U'_k ← U_k                                  ▷ make a copy
3:      if ALLOCATED is FALSE then
4:          compute Π^wc = {α_i} from I                 ▷ Eq. (15)
5:          ∀i  cpuID_i ← −1        ▷ all π_i are unallocated
6:          ALLOCATED ← TRUE
7:      end if
8:      l ← −∞                                          ▷ initialization
9:      for h ∈ {1, ..., m} do                          ▷ π_h to be allocated
10:         if cpuID_h = −1 then
11:             cpuID_h = BESTFIT({U'_k}, α_h)
12:             if cpuID_h = −1 then
13:                 ALLOCATED ← FALSE
14:                 return
15:             end if
16:         end if
17:         k ← cpuID_h
18:         U'_k ← U'_k + α_h
19:         l ← max(l, h + 1)
20:         while 1 − U'_k > 0 and l ≤ m do
21:             δ ← min(1 − U'_k, (l − h)(α_l − α_{l+1}))
22:             U'_h ← U'_h + δ
23:             for j ∈ {h + 1, ..., l} do
24:                 α_j ← α_j − δ/(l−h)
25:                 if cpuID_j ≠ −1 then
26:                     U'_{cpuID_j} ← U'_{cpuID_j} − δ/(l−h)
27:                 end if
28:             end for
29:             l ← l + 1
30:         end while
31:     end for
32:     ∀k  U_k = U'_k
33: end procedure
```

Figure 6: The FLUIDBESTFIT algorithm

When an application leaves the system, it is possible to further compact the existing applications enabling them to fill the available bandwidth caused by the departure of applications. This can be made by invoking the algorithm with ALLOCATED set to TRUE. In this case the algorithm does not perform the initial allocation anymore, but just tries to fill the available bandwidth.

In the case of a new application entering the system, the complexity of the algorithm is $O(m \log M)$, where m is the number of virtual processors (due to the for cycle at line 9), and M is the number of physical processors (due to the BESTFIT algorithm invoked at line 11). In the latter case

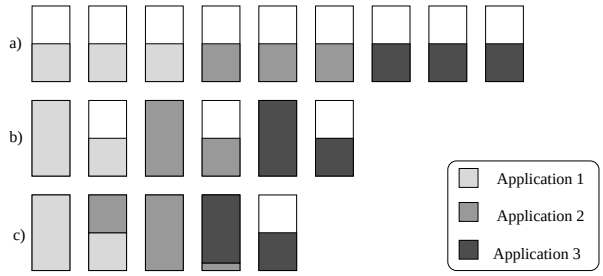

Figure 7: Different strategies for allocating interfaces.

(ALLOCATED set to true), the complexity reduces to $O(m)$.

A. Example of allocation

We now present a very simple example. Consider three interfaces, each one presents a total bandwidth of 1.53 equally distributed on 3 different virtual processors: $\mathcal{I}^1 = \mathcal{I}^2 = \mathcal{I}^3 = (3, \Delta, [0.51, 1.02, 1.53])$. If we apply a simple best-fit strategy to the 3 interfaces, we end up with requiring 9 processors (see Figure 7a, where each rectangle represent a processor, and the filled part represent the percentage of allocated bandwidth).

Inspired by the work by Leontyev and Anderson [11], for any interface $\mathcal{I}$ with an overall bandwidth requirement of β_m, we could use a platform with bandwidths

$$[\underbrace{1, \ldots, 1}_{\lfloor \beta_m \rfloor}, \beta_m - \lfloor \beta_m \rfloor].$$

This chosen Π is always, by construction, in $\mathbf{\Pi}(\mathcal{I})$. In this example, for each interface $\mathcal{I}$ we must prepare a virtual platform $\Pi^i = [1, 0.53]$. However, it is easy to see that a partitioned scheme cannot use less than 6 physical processors (see Figure 7b).

Finally, we apply our algorithm to the same example. For all the three interfaces the worst-case platform Π^{wc} is $[0.51, 0.51, 0.51]$. The algorithm is first called on $\mathcal{I}^1$. After the first iteration of the loop of lines 9–31 with $h = 1$, it allocates $\alpha_1^1 = 1$, leaving $\alpha_2^1 = \alpha_3^1 = 0.265$. Then, it *compacts* the third virtual processor on the second, obtaining $\alpha_2^1 = 0.53$ and $\alpha_3^1 = 0$. Now, the algorithm is called on the second interface $\mathcal{I}^2$: 1) it allocates $\alpha_1^2 = 1$ on the third physical processor, leaving $\alpha_2^2 = \alpha_3^2 = 0.265$, and $U = [1, 0.53, 1]$; 2) since $U_2 = 0.53$, it allocates the second virtual processor on the second physical processor, and compacts it, obtaining $\alpha_2^2 = 0.47$, $\alpha_3^2 = 0.06$ and $U = [1, 1, 1, 0.06]$; 3) Finally, $\alpha_3^2 = 0.06$ is allocated on the fourth processor. It is easy to see that the last interface is allocated on physical processor 4 and 5, with $\alpha_1^3 = 0.94$ and $\alpha_2^3 = 0.59$ and $U = [1, 1, 1, 1, 0.59]$. The final situation is depicted in Figure 7c. In total, our algorithm uses only 5 processors (which is the minimal possible), against 9 of the best-fit and 6 of the strategy inspired by [11].

VII. SIMULATIONS

In this section, we compare our proposed algorithm FLUIDBESTFIT with BESTFIT and FIRSTFIT.

In the experiments, we assumed a physical processor with an unlimited amount of processors. The goal of the three algorithms is simply to use the least number of processors. We have performed two sets of simulations, one with "light" interfaces, the other one with "heavy" interfaces.

Every interface was randomly generated as follows. We first extracted the maximum interface parallelism m as an random number uniformly distributed in $[2, 5]$. We also set the overall bandwidth requirement of the interface $\beta_m = r \cdot m$, where r is a random number uniformly distributed between $[0.2, 0.5]$ in the "light" experiment, and $[0.3, 0.7]$ in the "heavy" experiment. The other interface parameters β_i were calculated so that the interface had a certain concavity index[1]. We defined the *Concavity ratio*, a parameter of the random generation algorithm, as the ratio between the actual concavity and the maximum possible concavity.

For each value of the concavity ratio in $[0, 1]$, with steps of 0.1, we generated 500 interfaces, that were submitted to each of the three algorithms. A maximum of 5 applications were allowed in the system at any time. This means that, after the first 5 interfaces, whenever an interface was submitted one application was removed from the system. After the removal, algorithm FLUIDBESTFIT was run again with ALLOCATED set to TRUE, in order to compact existing applications.

After each allocation, we computed the *Compaction Index* for each algorithm, defined as the ratio between the number of processors used by the algorithm and the (theoretical) minimal number of processors $\lceil U_{tot} \rceil$.

In Figures 8 and 9 we show the average *Compaction Index* for the "light" and the "heavy" experiments, respectively. As expected, for small values of the concavity index Algorithm FLUIDBESTFIT (lines labeled with FBF in the figures) performs much better than BESTFIT and FIRSTFIT (labeled with BF and FF, respectively), because the extra flexibility allowed by the interface model can be used to obtain a more compact allocation. As the concavity ratio increases, the performance of FBF becomes similar to that of BF and FF, because when there is little or no flexibility, FBF is basically coincident with BF.

Notice that allocating interfaces according to the bandwidth distribution of [11] (and explained in the example of Section VI-A) consists simply in applying the best-fit algorithm to a platform $\Pi' \in \mathbf{\Pi}(\mathcal{I})$ with maximum concavity. Hence its performance is similar to the performance of BF at maximum concavity.

[1]For space constraints, we do not report here the algorithm for generating an interface with the desired concavity index. The interested reader can download the source code of the simulator from http://retis.sssup.it/~lipari/abf.tgz

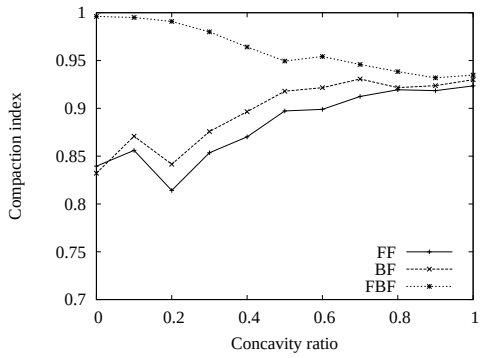

Figure 8: Compaction index for light interfaces

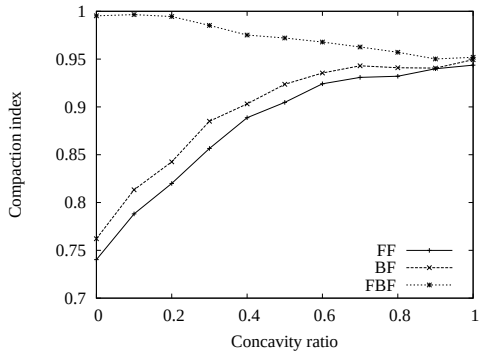

Figure 9: Compaction index for heavy interfaces

VIII. Conclusions and Future works

In this paper we presented a framework for the hierarchical scheduling of real-time applications onto multiprocessors. In particular, we addressed the problem of trading-off resource utilization versus flexibility in specifying the interface parameters.

Due to the generality of the methodology, we foresee significant space for improvements. First, we want to account for more realistic application model, including, for example, task dependencies. Also, we want to further explore the allocation problem, by devising an algorithm with a guaranteed approximation ratio w.r.t. an optimal algorithm.

Acknowledgements: We gratefully thank the anonymous reviewers for having contributed to greatly improve the paper with their keen and precise comments.

References

[1] "Automotive market overview," ARM website, http://www.arm.com/markets/automotive/index.html, Sep. 2004.

[2] Freescale semiconductors, "A smarter approach to multicore: Freescales next-generation communications platform," Freescale, http://www.freescale.com/files/32bit/doc/white_paper/MULTICOREFTFWP.pdf, Tech. Rep., 2008.

[3] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Boston, MA, U.S.A., May 1994, pp. 90–99.

[4] Z. Deng, J. w.-s. Liu, and J. Sun, "A scheme for scheduling hard real-time applications in open system environment," in *Proceedings of the* 9^{th} *Euromicro Workshop on Real-Time Systems*, Toledo, Spain, Jun. 1997, pp. 191–199.

[5] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems*, Porto, Portugal, Jul. 2003, pp. 151–158.

[6] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the* 24^{th} *Real-Time Systems Symposium*, Cancun, Mexico, Dec. 2003, pp. 2–13.

[7] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *Journal Embedded Computing*, vol. 1, no. 2, pp. 257–269, 2005.

[8] E. Bini, G. Buttazzo, and Y. Wu, "Selecting the minimum consumed bandwidth of an EDF task set," in 2^{nd} *Workshop on Compositional Real-Time Systems*, Washington (DC), USA, Dec. 2009, available at http://retis.sssup.it/~bini/publications/.

[9] A. K. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proceedings of the* 7^{th} *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, May 2001, pp. 75–84.

[10] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Proceedings of the* 28^{th} *IEEE International Real-Time Systems Symposium*, Tucson, AZ, USA, 2007, pp. 129–138.

[11] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *Proceedings of the* 20^{th} *Euromicro Conference on Real-Time Systems*, Prague, Czech Republic, Jul. 2008, pp. 191–200.

[12] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering multiprocessors," in *Proceedings of the* 20^{th} *Euromicro Conference on Real-Time Systems*, Prague, Czech Republic, Jul. 2008, pp. 181–190.

[13] Y. Chang, R. Davis, and A. Wellings, "Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning," University of York, Tech. Rep. YCS 432, 2008, available at http://www.cs.york.ac.uk/ftpdir/reports/2008/YCS/432/YCS-2008-432.pdf.

[14] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *Proceedings of the* 30^{th} *IEEE Real-Time Systems Symposium*, Washinghton, DC, USA, Dec. 2009, pp. 437–446.

[15] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Implementation of a speedup-optimal global EDF schedulability test," in *Proceedings of the EuroMicro Conference on Real-Time Systems*. Dublin: IEEE Computer Society Press, July 2009.

[16] E. Bini, G. C. Buttazzo, and M. Bertogna, "The multy supply function abstraction for multiprocessors," in *Proceedings of the* 15^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, Aug. 2009, pp. 294–302.

[17] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *Proceedings of the* 22^{nd} *IEEE Real-Time Systems Symposium*, London, United Kingdom, Dec. 2001, pp. 183–192.

[18] S. K. Baruah, N. K. Cohen, G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun. 1996.