Earliest-Deadline First Scheduling of Multiple Independent Dataflow Graphs

Adnan Bouakaz[1], Thierry Gautier[2] and Jean-Pierre Talpin[2]
[1]University of Rennes 1/IRISA, [2] INRIA
Campus de Beaulieu, 35042 Rennes Cedex, France
Email: adnan.bouakaz, thierry.gautier, jean-pierre.talpin@inria.fr

***Abstract*—Static dataflow graphs are widely used in design of concurrent real-time streaming applications on multiprocessor systems-on-chip. The increasing complexity of these systems advocates using real-time operating systems and dynamic scheduling to manage applications and resources. Providing timing guarantees (e.g. minimum throughput, deadlines) and minimizing the required amount of resources (e.g. number of processors, buffer capacities) are crucial aspects of these systems.**

This paper addresses uniprocessor and partitioned multiprocessor earliest-deadline first scheduling of multiple concurrent applications, each designed as an independent dataflow graph. Our scheduling approach maps each actor to a periodic real-time task and computes the appropriate buffer sizes and timing and scheduling parameters (i.e. periods, processor allocation, etc.). The proposed parametric schedulability analysis aims at maximizing the overall processor utilization, and hence allows for reducing the required number of processors.

I. Introduction and related work

Static dataflow graphs, such as synchronous dataflow (SDF) graphs [1] and cyclo-static dataflow (CSDF) graphs [2], are widely used in design and analysis of digital signal processing and concurrent real-time streaming applications on multiprocessor systems-on-chip (MPSoCs). A static dataflow graph is a directed graph that consists of a set of computation nodes (called actors) communicating through one-to-one FIFO channels. When it fires, an actor consumes a *predefined* number of tokens from its input channels and produces a predefined number of tokens on its output channels. Dataflow models naturally express the parallelism of applications which make them very suitable to exploit the parallelism offered by MPSoCs. SDF and CSDF models are restrictions of dataflow process networks that aim at realizing systems with predictable performance properties (e.g. throughput, channel capacities). Furthermore, they are amenable to compile-time construction of bounded and live static-order schedules.

Static-order scheduling consists in firing actors of the dataflow graph in sequence, one after another. If the sequence is periodic, then the schedule is a static-periodic schedule. In the past decades, static-periodic scheduling of (C)SDF graphs has been extensively addressed w.r.t. many performance metrics (e.g. code size minimization, throughput maximization, etc.). The main drawback of this scheduling approach is that the application consists of a monolithic, inflexible, and difficult to maintain code. Furthermore, there is no support for running independent and concurrent applications on the same platform, dynamic admission of new incoming applications, or handling aperiodic tasks.

Nowadays real-time streaming applications on MPSoCs are increasingly complex; and runtime systems are more needed to handle resource sharing, task priorities, etc. In [3], authors present a MPSoCs real-time operating system (RTOS) which takes a parametrized cyclo-static directed *acyclic* (PCSDA) graph as an input and dynamically dispatches the actors of the graph to the cores of a MPSoCs. The operational semantics is data-driven; i.e. an actor is activated when enough tokens accumulate on its input channels. The dataflow management step consists in parameterizing the graph at each millisecond, transforming the PCSDA graph into a single rate directed acyclic graph, and then a RTOS task is created for each actor in this latter graph. Unfortunately, the transformation of the PCSDA graph into a single rate graph may result in exponential increase in the number of actors. While the main objective of the scheduling step is load balancing and reduction of computation latency, no timing guarantees are provided.

Real-time streaming applications may have hard timing requirements that must be met; e.g. minimum throughput, maximum latency, deadlines of individual tasks, etc. Therefore, it is reasonable to exploit the existing real-time schedulability theory [4], [5] to provide strong mathematical guarantees that all tasks will meet their deadlines when using a given scheduling policy (e.g. fixed-priority scheduling, earliest-deadline first (EDF) scheduling, etc.). Recent works (such as [6], [7]) have shown that (C)SDF graphs can be scheduled under the real-time periodic task model (i.e. each actor is mapped to a periodic real-time task) w.r.t. EDF or fixed-priority preemptive scheduling policies. These approaches aim at computing the timing and scheduling characteristics of each task (e.g. period, phase, priority, processor allocation, etc.) such that all tasks meet their deadlines and no task attempts to write to a full channel (i.e. exclusion of overflow exceptions) or to read from an empty one (i.e. exclusion of underflow exceptions).

Bamakhrama et al. [7] have addressed partitioned multiprocessor EDF scheduling of ***acyclic*** CSDF graphs with implicit deadlines (i.e. deadlines of actors equal to their periods). Partitioned scheduling, in contrast to global scheduling, does not allow for a task to migrate from one processor to another. Since this problem is equivalent to bin-packing (and hence is NP-hard), authors consider a first-fit allocation heuristic, and compute the minimum number of processors needed for scheduling the graph. They show in [8] that implicit deadlines do not give the minimum latency, and hence propose a technique to constrain deadlines (i.e. deadlines less than or equal to periods) to achieve better latency. Recently, this approach has been extended to consider semi-partitioned EDF scheduling of acyclic CSDF graphs with implicit deadlines [9].

In our previous works [6], [10]–[13], we have proposed an abstraction-refinement framework for priority-driven scheduling of (***cyclic***) ultimately cyclo-static dataflow (UCSDF) graphs. This schedule construction approach is briefly presented in the next section. In [13], we have addressed uniprocessor and partitioned multiprocessor fixed-priority scheduling of UCSDF graphs with constrained deadlines. We have also presented different priority assignment policies that aim at maximizing the throughput or reducing the buffering requirements (i.e. the total sum of channel capacities). In [10], we have addressed uniprocessor, partitioned, and global multiprocessor EDF scheduling of UCSDF graphs. We have shown that implicit deadlines give the best throughput; and we have proposed a deadline adjustment technique that constrains deadlines to ensure some precedence constraints among tasks in order to improve the buffering requirements.

All the above presented works address hard real-time scheduling of weakly or strongly ***connected*** dataflow graphs. In case there are multiple independent and concurrent applications (each modeled as a connected dataflow graph), these approaches could handle this case by assuming that each graph has a *predefined* processing load budget. However, this solution may not give the best overall throughput for the whole set of independent graphs. In this paper, we extend our previous work to handle uniprocessor and partitioned multiprocessor EDF scheduling of multiple independent weakly connected UCSDF graphs. The proposed solution aims at maximizing the throughput and hence allows to minimize the number of processors needed for scheduling the graphs.

The following works are less related to our approach since they are based on a different operational semantics. Hausmans et al. [14], [15] have presented a temporal analysis for fixed-priority scheduling of (cyclic) SDF graphs. Assuming user-provided priorities and processor mapping, and the existence of a source actor with a predefined period, their analysis computes an upper bound on the worst-case response time of each actor. Unlike the clock-triggered operational semantics in our approach, they consider a data-driven operational semantics; i.e. apart from the periodic source actor, all the other actors are enabled by the arrival of data tokens on their input ports. These approaches follow the same scheme : the response time of an actor equals its execution time plus the maximum interference due to actors with higher priorities. Those approaches however differ in how to compute that interference. They may use a period-jitter characterization of the graph [14], an enabling rate characterization [15], a load characterization, etc.

Uniprocessor EDF schedulability analysis of acyclic computation graphs with a chain topology and constrained deadlines was addressed in [16]. It checks whether each actor meets its deadline or not assuming user-provided period of the source actor and data-driven enabling for the remaining actors.

The rest of the paper is organized as follows. Section II presents the background material on static dataflow graphs and the abstraction-refinement scheduling framework, needed for understanding the proposed algorithms. Section III presents both uniprocessor and partitioned multiprocessor parametric EDF schedulability analyses of multiple independent dataflow graphs. Section IV evaluates the performance of the proposed algorithms. Section V ends the paper with some conclusions.

II. Background

A. Static dataflow graphs

A static dataflow graph is a directed graph $G = (P, E)$ that consists of a set of actors $P = \{p_1, \ldots, p_N\}$ and a set of channels E. The worst-case execution time of actor p_i is denoted by C_i. Channel $e = (p_i, p_k, x, y)$ connects the producer p_i with the consumer p_k. The production and consumption rates of channel e are denoted by the two infinite integer sequences $x, y : \mathbb{N}_{>0} \to \mathbb{N}$, respectively. For instance, actor p_i writes $x(j)$ tokens on channel e *during* its j^{th} firing. The number of initial tokens in channel e is denoted by $\theta(e)$; while its size is denoted by $\delta(e)$.

Production and consumption rates are constant in SDF graphs, periodic in CSDF graphs, and ultimately periodic in UCSDF graphs. An infinite integer sequence s is *ultimately periodic* if $\exists \pi, j_0 \in \mathbb{N}_{>0}\ \forall j \geq j_0 : s(j) = s(j + \pi)$; we then write $s = uv^\omega$ for two finite sequences u and v. The cumulative function of an infinite sequence $s : \mathbb{N}_{>0} \to \mathbb{N}$ is an infinite sequence $\oplus s : \mathbb{N}_{>0} \to \mathbb{N}$ such that $\oplus s(j) = \sum_{i=1}^{j} s(i)$. For a finite sequence v, we denote its length by $|v|$ and the sum of all its elements by $\|v\|$.

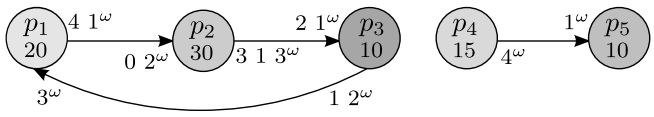

Fig. 1. Two independent UCSDF graphs.

In this paper, we are interested in preemptive EDF scheduling of a set of $\mathcal{N}$ independent weakly connected UCSDF graphs $G_i = (P_i, E_i)$ on a set of $M \geq 1$ homogeneous processors. In case of partitioned multiprocessor scheduling (i.e. $M \geq 2$), each actor will be allocated to one processor. Timing requirements are expressed in two ways: (1) the minimum throughput $\Theta^l(G_i)$ that must be achieved by each UCSDF graph G_i, and (2) an individual constrained deadline d_i of each actor p_i. For more expressive specifications, deadlines are not just constants but monotone functions in terms of periods. We take the simple example where $\forall p_i : d_i = a_i \pi_i + b_i$ such that π_i is the period of actor p_i, $a_i \in \mathbb{Q}_{\geq 0}$, and $b_i \in \mathbb{Z}$. Hence, for an implicit deadline, we have that $a_i = 1$ and $b_i = 0$; while for a constant deadline, we have that $a_i = 0$.

Example. We will illustrate all the proposed algorithms using the following example. Figure 1 shows an application that consists of two independent UCSDF graphs where $P_1 = \{p_1, p_2, p_3\}$ and $P_2 = \{p_4, p_5\}$. Edges are annotated by production and consumption rate sequences, while numbers inside nodes represent the worst-case execution times. Hence, the vector of worst-case execution times is $C = [20, 30, 10, 15, 10]$. We suppose that the user-provided timing requirements are $\Theta^l(G_1) = 0$ (i.e. unspecified), $\Theta^l(G_2) = 2.8 \times 10^{-3}$, and $d = [\frac{3}{4}\pi_1, \frac{\pi_2}{2} - 5, \pi_3 - 2, \frac{7}{24}\pi_4 - 4, \pi_5]$.

B. Affine scheduling

Since each actor will be mapped to a periodic real-time task, it is necessary to compute the appropriate channel capacities and timing and scheduling characteristics of each actor p_i: a period π_i, a phase r_i, and a processor allocation. These

characteristics should ensure that: overflow and underflow exceptions over communication channels are statically excluded, timing requirements are satisfied, and the overall throughput is maximized.

In our previous work, we have proposed an abstraction-refinement schedule construction approach. It consists of two steps. The first step is physical-time independent; it computes some relations between the logical order of activations of every two adjacent actors. Since actors are mapped to periodic tasks, those activation relations are affine relations [17]. An affine relation between the activation clocks of two actors p_i and p_k is described by three integer parameters $n, d \in \mathbb{N}_{>0}$ and $\varphi \in \mathbb{Z}$, as illustrated in Figure 2. Parameters n and d encode the relation between the speeds of the actors (i.e. for every d activations of p_i, there are n activations of p_k) while parameter φ encodes the difference between their phases.

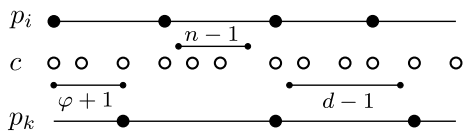

Fig. 2. A (n, φ, d)-affine relation. Clock c is a fictional abstract clock.

The set of computed affine relations should satisfy two constraints: exclusion of overflow and underflow exceptions and consistency.

a) Overflow and underflow analysis: Excluding overflow and underflow exceptions implies that the number of accumulated tokens on every channel and at each step of the execution is greater than or equal to zero (i.e. no underflow) and less than or equal to the buffer size (i.e. no overflow).

Let $e = (p_i, p_k, x = u_1 v_1^\omega, y = u_2 v_2^\omega)$ be a channel between the two (n, φ, d)-affine-related actors p_i and p_k. When the j^{th} firing of p_i and the j'^{th} firing p_k complete, the number of accumulated tokens on channel e is given by $\theta(e) + \oplus x(j) - \oplus y(j')$; i.e. the number of initial tokens plus the number of all produced tokens by all firings of p_i until the j^{th} firing minus all the consumed tokens. Obviously, indices j and j' are related to each other according to the affine relation. Hence,

- No overflow exception over channel e means that:

$$\forall j : \theta(e) + \oplus x(j) - \oplus y(j') \leq \delta(e) \tag{1}$$

- No underflow exception over channel e means that:

$$\forall j : \theta(e) + \oplus x(j') - \oplus y(j) \geq 0 \tag{2}$$

We have shown in [6], [10] that parameters n and d of affine relations can be obtained using the boundedness criterion (Equation 3), while parameters φ can be obtained using integer linear programming after sound linear over/under-approximations of Equations 1 and 2.

$$\frac{n}{d} = \frac{\|v_1\|}{|v_1|} \frac{|v_2|}{\|v_2\|} \tag{3}$$

b) Consistency: According to the overflow and underflow analysis, the affine relation between two adjacent actors which excludes exceptions over channels between them can be computed independently of the other relations. However, the overall abstract affine schedule could be inconsistent in case there are undirected cycles in the graph. For instance, since parameter φ encodes the difference between phases, we must ensure that the accumulated difference on a cycle is null. Proposition 1 in [6] is a sufficient condition for consistency of affine schedules.

Example. By applying the boundedness criterion to our dataflow example, we obtain the following affine relations: $p_1 \overset{(2,\varphi_1,4)}{\longrightarrow} p_2 \overset{(6,\varphi_2,2)}{\longrightarrow} p_3 \overset{(4,\varphi_3,6)}{\longrightarrow} p_1$ and $p_4 \overset{(8,\varphi_4,2)}{\longrightarrow} p_5$.

The second step in our abstraction-refinement approach is called parametric schedulability analysis. This step refines the obtained abstract affine schedule by computing the physical timing characteristics (i.e. periods and phases). These characteristics should satisfy the following constraints.

c) Affine relations: If actors p_i and p_k are (n, φ, d)-affine-related, then their timing characteristics must satisfy

$$d\pi_i = n\pi_k \tag{4}$$

$$r_k - r_i = \frac{\varphi}{n}\pi_i \tag{5}$$

From Equation 4, we deduce that the period and deadline of every actor in a weakly connected UCSDF graph can be expressed in terms of the period of one actor in that graph. Hence, we can put $\forall p_i \in G_j : \pi_i = \alpha_i T_j$ and $d_i = \beta_i T_j + b_i$ where $\alpha_i \in \mathbb{Q}_{>0}$, $\beta_i \in \mathbb{Q}_{\geq 0}$ and $b_i \in \mathbb{Z}$. Furthermore, since the timing characteristics are non-negative integers, variable T_j should be a multiple of some integer factor B_j.

Example. By applying Equations 4 and 5, we have that $4\pi_1 = 2\pi_2$, $r_2 - r_1 = \frac{\varphi_1}{2}\pi_1$, $d_1 = \frac{3}{4}\pi_1$, etc. Hence, we can put $\pi = [T_1, 2T_1, \frac{2}{3}T_1, T_2, \frac{1}{4}T_2]$ and $d = [\frac{3}{4}T_1, T_1 - 5, \frac{2}{3}T_1 - 2, \frac{7}{24}T_2 - 4, \frac{1}{4}T_2]$. We also have that $B_1 = 12$ and $B_2 = 24$.

d) Schedulability: The computed timing characteristics must guarantee the timing requirements. Hence, the second step of our approach must explore the $\mathcal{T}$-space (i.e. all the possible values of variables T_j), to find a point which satisfies the timing requirements and optimizes the throughput. Standard EDF schedulability tests (Section III) could be used at each point to check whether all tasks will meet their deadlines. However, such enumerative solution could be very time consuming. In this paper, we present more efficient solutions in case of multiple independent dataflow graphs.

C. EDF schedulability

This section presents the standard uniprocessor EDF schedulability tests of a set of N periodic real-time tasks. It is based on the processor-demand approach as given by the following lemma [18].

Lemma 1: A synchronous[1] periodic task set is schedulable if and only if $U \leq 1$ and $\forall t \leq L : h(t) \leq t$.

[1] An asynchronous task set is schedulable if its corresponding synchronous task set is schedulable.

$U = \sum_{i=1}^{N} \frac{C_i}{\pi_i}$ is the processor utilization factor of the task set. L is called the feasibility bound. It is equal to the length of the synchronous busy period, which can be computed by the following recurrence.

$$L^0 = \sum_{i=1}^{N} C_i \qquad L^{m+1} = \sum_{i=1}^{N} \left\lceil \frac{L^m}{\pi_i} \right\rceil C_i \tag{6}$$

The processor demand function $h(t)$ calculates the maximum processor demand of all jobs which have their arrival times and deadlines in a contiguous interval of length t. So, $h(t)$ is given by

$$h(t) = \sum_{i=1}^{N} \max\{0, 1 + \left\lfloor \frac{t - d_i}{\pi_i} \right\rfloor\} C_i \tag{7}$$

The value of the demand function does not change from one point t to another one t' unless there is at least one absolute deadline in the interval $]t, t']$. Therefore, it is necessary to check condition $h(t) \leq t$ only for the set of absolute deadlines which are less than the feasibility bound L. This set can be large and a technique like the Quick convergence Processor-demand analysis (QPA) [19] is needed. Listing 1 represents the QPA algorithm where d denotes an absolute deadline and $\min\{d\}$ is the smallest absolute deadline. So, instead of checking all deadlines in the increasing order, the QPA algorithm starts from the last deadline and moves backward, skipping many intermediate deadlines.

Algorithm 1: QPA algorithm

```
t = max{d | d ≤ L};
while h(t) ≤ t ∧ h(t) > min{d} do
    if h(t) < t then t = h(t);
    else t = max{d | d < t};
if h(t) ≤ min{d} then the task set is schedulable;
else the task set is not schedulable;
```

Our parametric EDF schedulability analysis, presented in the next section, is based on the two following observations. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two vectors in the $\mathcal{T}$-space such that $\mathcal{T}_1 \leq \mathcal{T}_2$. Hence, for every actor p_i, its period at point $\mathcal{T}_1$ is smaller than its period at point $\mathcal{T}_2$. Furthermore, since deadlines are monotone functions in terms of periods, the deadline of a given actor at point $\mathcal{T}_1$ is smaller than its deadline at point $\mathcal{T}_2$. Therefore and according to Equations 6 and 7, we have that

Observation 1. The feasibility bound L at point $\mathcal{T}_1$ is larger than the feasibility bound at point $\mathcal{T}_2$.

Observation 2. For every t, the processor demand $h(t)$ at point $\mathcal{T}_1$ is larger than the processor demand at point $\mathcal{T}_2$. Hence, if $h(t) \leq t$ at point $\mathcal{T}_1$, then we also have that $h(t) \leq t$ at point $\mathcal{T}_2$.

III. Parametric EDF schedulability analysis

In this section, we present the parametric uniprocessor and partitioned multiprocessor EDF schedulability analysis for multiple independent dataflow graphs.

A. Uniprocessor scheduling

We first compute the boundaries of the explored $\mathcal{T}$-space; i.e. we compute a lower bound vector $\mathcal{T}^l = [T_1^l, \ldots, T_{\mathcal{N}}^l]$ and an upper bound vector $\mathcal{T}^u = [T_1^u, \ldots, T_{\mathcal{N}}^u]$. Hence, we will search for the optimal solution in the interval $[\mathcal{T}^l, \mathcal{T}^u]$. Those bounds are deduced for the following constraints: (1) the necessary EDF schedulability test $U \leq M$ (M is the number of processors), (2) the constrained deadlines $\forall p_i : C_i \leq d_i \leq \pi_i$, and (3) the minimum throughput requirements.

The throughput of an actor p_i is the average number of firings of p_i per unit of time and hence equals to its frequency $\frac{1}{\pi_i}$. The throughput of a (weakly) connected graph G is equal to $\frac{1}{\pi_i r(i)}$. The repetition vector r can be easily obtained from the well-known balance equation [1] or equivalently from parameters n and d of affine relations. Since $\forall p_i : \pi_i = \alpha_i T$, maximizing the throughput is equivalent to minimizing T and hence to maximizing the processor utilization factor U. Thus, from the minimum throughput requirement of the graph (i.e. $\Theta(G) \geq \Theta^l(G)$), we can obtain an upper bound T^u. In the literature, there is no definition for the overall throughput of a set of independent dataflow graphs. Therefore, we have chosen, as in the first case, to maximize the processor utilization factor.

The lower bound vector $\mathcal{T}^l$ can be improved by performing the parametric EDF schedulability analysis on each graph G_j assuming the minimum throughput for other graphs. The best processor utilization factor obtained by this step is denoted by U_0, while the corresponding vector is denoted by $\mathcal{T}_0$.

Example. The UCSDF example consists of two independent graphs G_1 and G_2 with minimum throughput requirements $\Theta(G_1)^l = 0$ (i.e. unspecified) and $\Theta^l(G_2) = 2.8 \times 10^{-3}$. The repetition vectors are $r_1 = [2, 1, 3]$ and $r_2 = [1, 4]$. We also have that $B_1 = 12$ and $B_2 = 24$. From constraint $C_i \leq d_i \leq \pi_i$, we deduce $\mathcal{T}^l = [36, 72]$. From constraint $U = \frac{50}{T_1} + \frac{55}{T_2} \leq 1$, we deduce $\mathcal{T}^l = [60, 72]$. From constraint $\Theta(G_j) \geq \Theta(G_j)^l$, we obtain $\mathcal{T}^u = [\infty, 336]$. By assigning the minimum throughput to graph G_2 (i.e. $T_2 = 336$), we found that the task set is not schedulable neither for $T_1 = 60$ (deadline miss at $t = 55$) nor for $T_1 = 72$ (deadline miss at $t = 94$). It is however schedulable for $T_1 = 84$ with $U_0 = 0.759$. Similarly and by assigning the minimum throughput to graph G_1 (i.e. $T_1 = \infty$), we found that the task set is not schedulable neither for $T_2 = 72$ (deadline miss at $t = 18$) nor for $T_2 = 96$ (deadline miss at $t = 24$). Therefore, the new lower bound is $\mathcal{T}^l = [84, 120]$.

Algorithm 2: $\texttt{QPA}^*(\mathcal{T}, t_0)$

```
if U(T) > 1 then return t0;
t = max{d | d ≤ L(T) ∧ d ≤ t0};
while h(T,t) ≤ t ∧ h(T,t) > min{d} do
    if h(T,t) < t then t = h(T,t);
    else t = max{d | d < t};
return h(T,t);
```

Let $U(\mathcal{T})$, $L(\mathcal{T})$, and $h(\mathcal{T}, t)$ denote the processor utilization, the feasibility bound, and the processor demand function at a given point $\mathcal{T}$ in the $\mathcal{T}$-space, respectively. As shown in Algorithm 2, function $\texttt{QPA}^*(\mathcal{T}, t_0)$ performs QPA (i.e. testing

$h(t) \leq t$ in a backward manner) for a given value $\mathcal{T}$ in the $\mathcal{T}$-space starting from point $\min\{t_0, L(\mathcal{T})\}$. It ends when there is a deadline miss or if the task set is schedulable. If t is the returned value, then there is no deadline miss in the interval $[t, \min\{L(\mathcal{T}), t_0\}]$. In case $U(\mathcal{T}) > 1$, the procedure returns immediately.

Algorithm 3: DF−B&B PQPA

```
Procedure main() begin
    T_bst = T_0;  U_bst = U_0;
    T_cur = T^l;
    t = QPA*(T_cur, ∞);
    if t ≤ min{d} then return T_cur ;
    else
        VisitTree(T_cur, t); return T_bst;
Procedure visitTree(T, t) begin
    for j = 1, 2, ..., N do
        T_cur ← increase T_j in T;
        if T_j > T_j^u or U(T_cur) ≤ U_bst then
            prune this node;
        else
            t = QPA*(T_cur, t);
            if t ≤ min{d} then
                T_bst = T_cur;  U_bst = U(T_cur);
            else VisitTree(T_cur, t);
```

Algorithm 3 (DF−B&B PQPA) is a depth-first branch and bound parametric EDF schedulability analysis. It is based on the QPA* function and Observations 1 and 2. We will illustrate this algorithm using the UCSDF example.

An initial feasible solution is $\mathcal{T}_0$ which is obtained when computing the lower bound $\mathcal{T}^l$. Hence, the optimal solution is initialized at $\mathcal{T}_{\text{bst}} = \mathcal{T}_0$ and $U_{\text{bst}} = U_0$. The algorithm explores the $\mathcal{T}$-space starting from point $\mathcal{T}^l$, and at each time increments one component of the vector (i.e. $T_j = T_j + B_j$). The constructed tree, obtained by this process, is illustrated in Figure 3 where nodes are numbered in order of their appearance.

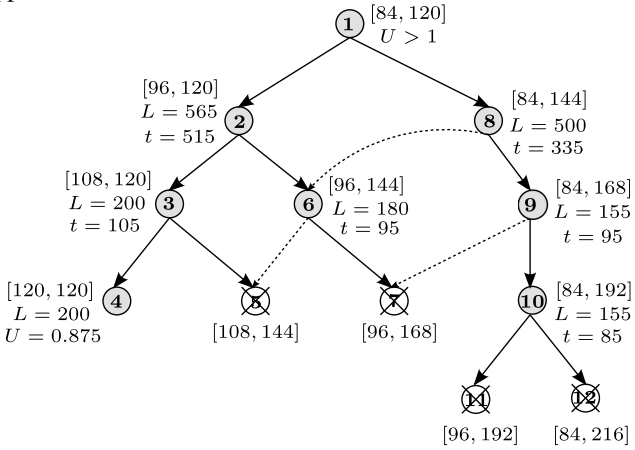

Fig. 3. Illustration of the DF−B&B PQPA algorithm.

Initially, the current node is $\mathcal{T}_{\text{cur}} = [84, 120]$. Since $U(\mathcal{T}_{\text{cur}}) > 1$, function QPA* returns immediately. Component T_1 is incremented first by B_1 (i.e. node number 2 with $\mathcal{T}_{\text{cur}} = [96, 120]$). Function QPA* checks deadlines starting from $L(\mathcal{T}_{\text{cur}}) = 565$. It detects a deadline miss (at $d = 511$) and returns $t = 515$. Component T_1 is incremented again (node number 3 with $\mathcal{T}_{\text{cur}} = [108, 120]$). Thanks to Observations 1 and 2, we do not have to recheck the already checked deadlines, and hence QPA* starts from $\min\{L(\mathcal{T}) = 200, t = 515\}$. At node number 4, QPA* starts from $\min\{L(\mathcal{T}) = 200, t = 105\}$, and hence skips many deadlines. Since the task set is schedulable at node number 4, the best solution is updated (i.e. $\mathcal{T}_{\text{bst}} = [120, 120]$ and $U_{\text{bst}} = 0.875$).

Nodes are pruned in two cases: (1) if the value of one component exceeds its upper bound (i.e. $T_j > T_j^u$), and (2) if $U(\mathcal{T}_{\text{cur}}) \leq \mathcal{T}_{\text{bst}}$ since increasing periods will result in smaller processor utilization. For instance, node number 5 is pruned because $U([108, 144]) < U_{\text{bst}} = 0.875$.

The number of explored nodes depends on factors B_j. It is larger for small factors since components are incremented each time by small quantities which are not enough to resolve the deadline miss. Algorithm DF−B&B PQPA can be easily turned to a heuristic that increments a component T_j by $k_j B_j$ for a given constant $k_j \in \mathbb{N}_{>0}$. This way, the number of explored nodes is reduced but at a cost of less accurate results.

B. Partitioned multiprocessor scheduling

One advantage of partitioned scheduling compared with global scheduling is that, once an allocation of actors to processors has been achieved, it is possible to apply uniprocessor schedulability analyses [5]; for instance, the technique proposed in the previous section. Since partitioning the tasks on M identical processors is NP-hard, we propose a best-fit heuristic.

If P is the set of all actors and $S \subseteq P$, then function DF−B&B PQPA*$(S, \mathcal{T}_0)$ performs the previous parametric uniprocessor EDF schedulability analysis on just actors of subset S starting from point $\mathcal{T}_0$ (not from $\mathcal{T}^l$). We note that this function aims at maximizing U as it is defined for set P (and not only for subset S). Let $(V_i)_{i=1,M}$ be the set of, initially empty, M partitions. Our heuristic consists of the following steps:

Algorithm 4: Best-fit allocation heuristic

```
T_cur = T^l;
while P ≠ ∅ do
    1. Select actor p ∈ P with the smallest deadline at
    point T_cur;
    2. for i = 1, ..., M do
        T_i = DF−B&B PQPA*(V_i ∪ {p}, T_cur);
    3. Assign actor p to partition V_k with maximum
    U(T_k), put T_cur = T_k, and remove p from P;
```

At each iteration, DF−B&B PQPA* explores the $\mathcal{T}$-space starting from the solution obtained in the previous iteration. This ensures that assigning one actor to a partition does not jeopardize the schedulability of other partitions.

Example. We illustrate this algorithm on the UCSDF example for $M = 2$. As illustrated in the following table, the obtained

partitions are $V_1 = \{p_4, p_1, p_2\}$ and $V_2 = \{p_5, p_3\}$, while the overall processor utilization is $U = 1.458$.

$\mathcal{T}_{\text{cur}}$	actor	$\mathcal{T}_1$	$\mathcal{T}_2$	assignment
$[36, 72]$	p_4	$[36, 72]$	$[36, 72]$	V_1
$[36, 72]$	p_5	$[36, 120]$	$[36, 72]$	V_2
$[36, 72]$	p_3	$[36, 120]$	$[36, 72]$	V_2
$[36, 72]$	p_1	$[48, 72]$	$[48, 168]$	V_1
$[48, 72]$	p_2	$[72, 72]$	$[48, 216]$	V_1

IV. Experimental results

In this section, we will evaluate the performance of the `DF-B&B PQPA` algorithm in comparison with that of a basic enumerative solution which proceeds as follows: starting from $\mathcal{T}_{\text{cur}} = \mathcal{T}^l$, test condition $h(\mathcal{T}, t) \leq t$ at each absolute deadline in $[0, L(\mathcal{T}_{\text{cur}})]$. If there is a deadline miss, then put $\mathcal{T}_{\text{cur}} =$ `getNext()` and repeat the same process until reaching the first feasible task set. Function `getNext()` returns the next unexplored point in the $\mathcal{T}$-space with maximum utilization. Algorithms are compared in terms of the number of checked deadlines. This experiment will also demonstrate that our algorithm computes the same solutions as the basic enumerative algorithm.

We randomly generate $\mathcal{N}$ task sets with equal number of nodes, each task set corresponds to an independent dataflow graph and is generated as follows: for each actor p_i, we generate three parameters (C_i, α_i, β_i) such that $\pi_i = \alpha_i T_j$ and $d_i = \beta_i T_j$. The `UUniFast` algorithm is used to generate uniformly distributed $\frac{C_i}{\alpha_i}$ values. Worst-case execution times are uniformly distributed in the interval $[100, 1000]$. Parameters α_i and β_i are uniformly generated by fixing the value of factor B_j and the value of an experimental parameter $D \in]0, 1]$ so that $\forall p_i : \beta_i \in [D\alpha_i, \alpha_i]$. Hence, deadlines are more constrained for small values of parameter D. As a throughput requirement, we suppose that the lower bound on the processor utilization of each graph is equal to 0.1.

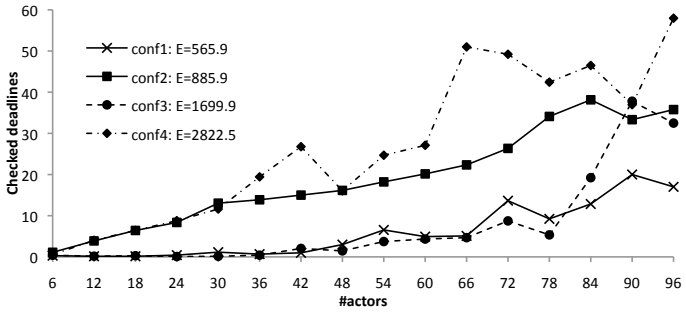

Fig. 4. Performance of the `DF-B&B PQPA` algorithm.

Figure 4 shows the obtained results for $\mathcal{N} = 2$ and 4 configurations of parameters B_j and D. Parameters B_j are uniformly distributed in the interval $[10, 15]$ for the first and second configurations (`conf1` and `conf2`), and in the interval $[5, 10]$ for `conf3` and `conf4`. Parameter D takes either value 0.3 (`conf1` and `conf3`) or value 0.8 (`conf2` and `conf4`). Each point in the diagram is the average (over 20 generated task sets) of ratios of the number of checked deadlines by the enumerative solution to the number of checked deadlines by `DF-B&B PQPA`. For each configuration, we denote by E the average number of checked deadlines per task (obtained by the enumerative solution) to indicate the complexity of the problem. As expected and explained in the previous section, the number of checked deadlines is larger for small values of factors B_j. It is also slightly larger when $D = 0.8$. Our algorithm largely outperforms the enumerative solution in most cases except when there is a considerable proportion of actors with highly constrained deadlines. When deadlines are too constrained, deadline misses could be detected earlier by a forward search than by a backward search (i.e. QPA).

V. Conclusion

As an extension to our abstraction-refinement framework for priority-driven schedule construction of UCSDF graphs, this paper presents a new parametric EDF schedulability analysis that can handle multiple independent dataflow graphs. Our future work will address the parametric fixed-priority schedulability analysis of multiple independent graphs.

References

[1] E. A. Lee and D. G. Messerchmitt, "Synchronous data flow," *Proceedings of the IEEE*, 1987.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *IEEE Transactions on Signal Processing*, 1996.

[3] Y. Oliva, M. Pelcat, J.-F. Nezan, J.-C. Prevotet, and S. Aridhi, "Building a RTOS for MPSoC dataflow programming," in *SOC*, 2011.

[4] L. Sha *et al.*, "Real time scheduling theory: a historical perspective," *Real-Time Syst.*, 2004.

[5] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, 2001.

[6] A. Bouakaz, J.-P. Talpin, and J. Vitek, "Affine data-flow graphs for the synthesis of hard real-time applications," in *ACSD*, 2012.

[7] M. Bamakhrama and T. Stefanov, "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications," in *EMSOFT*, 2011.

[8] M. A. Bamakhrama and T. Stefanov, "Managing latency in embedded streaming applications under hard-real-time scheduling," in *CODES*, 2012.

[9] E. Cannella, M. A. Bamakhrama, and T. Stefanov, "System-level scheduling of real-time streaming applications using a semi-partitioned approach," in *DATE*, 2014.

[10] A. Bouakaz and J.-P. Talpin, "Buffer minimization in earliest-deadline first scheduling of dataflow graphs," *SIGPLAN Not.*, 2013.

[11] A. Bouakaz, "Real-time scheduling of dataflow graphs," Ph.D. dissertation, University of Rennes 1, 2013.

[12] A. Bouakaz and T. Gautier, "An abstraction-refinement framework for priority-driven scheduling of static dataflow graphs," in *MEMOCODE*, 2014.

[13] A. Bouakaz and J.-P. Talpin, "Design of safety-critical Java level 1 applications using affine abstract clocks," in *M-SCOPES*, 2013.

[14] J. P. H. M. Hausmans, M. H. Wiggers, S. Geuns, and M. J. G. Bekooij, "Dataflow analysis for multiprocessor systems with non-starvation-free schedulers," in *M-SCOPES*, 2013.

[15] J. P. H. M. Hausmans *et al.*, "Temporal analysis flow based on an enabling rate characterization for multi-rate applications executed on MPSoCs with non-starvation-free schedulers," in *SCOPES*, 2014.

[16] S. Goddard and K. Jeffay, "Analyzing the real-time properties of a dataflow execution paradigm using a synthetic aperture radar application," in *RTAS*, 1997.

[17] I. M. Smarandache, T. Gautier, and P. Le Guernic, "Validation of mixed signal-alpha real-time systems through affine calculus on clock synchronisation constraints," in *FM*, 1999.

[18] I. Ripoll, A. Crespo, and A. K. Mok, "Improvement in feasibility testing for real-time tasks," *Real-Time Syst.*, 1996.

[19] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Transactions on Computers*, 2009.