Buffer Minimization in Earliest-Deadline First Scheduling of Dataflow Graphs

Adnan Bouakaz
University of Rennes 1 / IRISA
adnan.bouakaz@irisa.fr

Jean-Pierre Talpin
INRIA / IRISA
jean-pierre.talpin@inria.fr

Abstract

Symbolic schedulability analysis of dataflow graphs is the process of synthesizing the timing parameters (i.e. periods, phases, and deadlines) of actors so that the task system is schedulable and achieves a high throughput when using a specific scheduling policy. Furthermore, the resulted schedule must ensure that communication buffers are underflow- and overflow-free. This paper describes a (partitioned) earliest-deadline first symbolic schedulability analysis of dataflow graphs that minimizes the buffering requirements.

Our scheduling analysis consists of three major steps. (1) The construction of an abstract affine schedule of the graph that excludes overflow and underflow exceptions and minimizes the buffering requirements assuming some precedences between jobs. (2) Symbolic deadlines adjustment that guarantees precedences without the need for lock-based synchronizations. (3) The concretization of the affine schedule using a symbolic, fast-converging, processor-demand analysis for both uniprocessor and multiprocessor systems. Experimental results show that our technique improves the buffering requirements in many cases.

Categories and Subject Descriptors C.3 [*Special-Purpose and Application-Based Systems*]: Real-Time and Embedded Systems; F.1.1 [*Theory of Computation*]: Models of Computation

General Terms Theory, Algorithms, Design

Keywords Dataflow graphs, Symbolic schedulability analysis, Buffer minimization, Affine relation, Earliest-deadline first scheduling

1. Introduction

Dataflow models of computation are commonly used in embedded system design to describe stream processing or control applications. Their simplicity allows waiving part of the difficult and error-prone tasks of programming real-time schedules for computations and communications from the engineering process by implementing automated code generation techniques.

Modern multicore architectures, as well as embedded real-time operating systems, such as RTEMS for aerospace, OSEK for automotive, or the SCJ virtual machine, enable multi-task implementations of dataflow specifications. Most of these operating systems implement a bunch of different priority-driven scheduling policies such as earliest-deadline first (EDF) policy. For that reason, we argue the need for automatic synthesis techniques that produce EDF schedules of dataflow specifications while guaranteeing functional determinism and EDF schedulability. Functional determinism means that, for a given sequence of inputs, the system will always produce the same sequence of outputs. This implies that neither overflow nor underflow exceptions over communication buffers may occur at runtime [8].

Unlike with the classical static-periodic scheduling of dataflow graphs, this paper considers a fixed-job-priority preemptive scheduling policy: the EDF policy. Real-time scheduling theory for both uniprocessor and multiprocessor systems has already provided a lot of algorithms to check whether a task system meets its timing requirements, even in the worst-case scenario [11,18]. However, most algorithms assume that timing parameters of tasks are known a priori. Our approach to the problem is rather to *synthesize* the timing parameters of actors in dataflow graphs so as to ensure functional determinism and uniprocessor/multiprocessor schedulability. For the multiprocessor case, we will consider the partitioned EDF policy on identical multiprocessors where each task is allocated to a processor and no migration is permitted [4]. The current state of the art favors the partitioned scheduling over the global one for reasons that concern the run-time cost of inter-processor migration and the optimality of schedulability tests.

In a dataflow model of computation, an application is usually specified as a set of actors that communicate through one-to-one FIFO channels (i.e. streams). When an actor fires, it consumes tokens from its input channels and produces tokens on its output channels. In this paper, we target the design of software applications where software tasks interact with each other via software buffers. Among dataflow models of computation, synchronous data-flow (SDF) [15] and cyclo-static dataflow (CSDF) [6] are popular in the embedded system community. An actor has constant production and consumption rates in SDF and periodic rates in CSDF. Figure 1(a) shows a cyclic dataflow graph with ultimately periodic production and consumption rates. Channels are annotated by production and consumption rates. Sequence $1(2,0)$ on the output channel of actor p_1 means that the first firing (i.e. job) of p_1 produces one token; and then, alternatively, even jobs produce two tokens and odd jobs produce nothing. A number inside a node represents the worst-case execution time of an actor.

In order to execute such a dataflow specification on a real-time operating system with an EDF scheduler, we map each actor to a periodic task that has appropriate timing parameters; i.e. period, phase (first start time), and deadline. Recently, techniques that schedule dataflow graphs as implicit-deadline periodic (IDP) task systems have been proposed with an aim of achieving a high processor utilization factor (i.e. throughput) [1,8]. An IDP task has a deadline equal to its period. Such scheduling approach gives the maximum achievable processor utilization for large set of dataflow

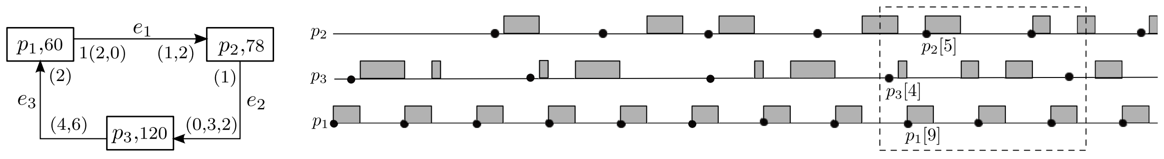

Figure 1. (a) A cyclic dataflow graph with ultimately periodic rates, and (b) its EDF scheduling on one processor.

graphs, called matched I/O rates graphs [1]. Furthermore, EDF symbolic schedulability of IDP task systems is quite simple with a polynomial time complexity [8]. However, the IDP model may increase the latency and the buffering requirements of the graphs. In [2], authors have shown that the IDP model increases the latency for a class of graphs called unbalanced graphs. In this paper, we will investigate the impact of the IDP model on the buffering requirements and we will propose a technique of deadlines adjustment that aims at reducing buffer sizes. Indeed, some embedded systems have strong memory constraints; therefore favoring buffer minimization over throughput maximization could be a legitimate choice. We will use a *constrained* task model where deadlines are less or equal to periods. Our technique is an extension of the affine dataflow approach [8] because, and unlike [1,2], that approach handles cyclic graphs.

1.1 Motivational example

Applying the scheduling technique proposed in [8] on the graph of Figure 1(a) results in the following timing parameters: periods $[\pi_1 = 160, \pi_2 = 240, \pi_3 = 400]$, phases $[r_1 = 0, r_2 = 360, r_3 = 40]$, and, of course, deadlines are equal to periods (i.e. implicit deadlines). For instance, based on the production and consumption rates of actors p_3 and p_1, actor p_1 must be exactly $\frac{5}{2}$ as fast as actor p_3 in order to implement channel e_3 as a bounded buffer. Figure 1(b) sketches the EDF scheduling of the graph on one processor where the j^{th} job of an actor p is denoted by $p[j]$. Black dots represent release times of jobs. Now, we will investigate the impact of preemption on the computation of buffer sizes.

Actor p_3 is the producer of channel e_3, while actor p_1 is its consumer. As depicted in Figure 1(b), jobs $p_1[9]$ and $p_1[10]$ preempt job $p_3[4]$. Since the technique proposed in [8] does not assume any knowledge about the implementation code of actors (i.e. it is unknown when exactly the actor reads and writes tokens), it considers the worst-case scenarios when computing the minimum number of initial tokens on the channel that prevents underflow exceptions and the minimum buffer size that prevents overflow exceptions. As a safe approximation, the worst-case scenario for the underflow analysis is when the producer p_3 writes its results at the end of its execution; while the worst-case scenario for the overflow analysis is when p_3 writes at the beginning.

To get rid of the over-approximation, we may write the implementation code of actors so that they produce tokens only at the end of their firings. However, firstly, a job may complete before its *worst-case* execution time. Secondly, if jobs $p_1[8], p_1[9]$, and $p_1[10]$ consume many tokens before the producer writes something to channel e_3, this is likely to increase the minimum number of initial tokens on the channel and so the buffering requirements. Our solution is rather to keep total freedom on how to write the implementation code and to adjust the deadline of actor p_3 instead. Indeed, if the deadline of job $p_3[4]$ is less or equal to the deadline of job $p_1[9]$, then this latter will not preempt job $p_3[4]$ according to the EDF policy. Hence, we enforce the precedence between jobs $p_3[4]$ and $p_1[9]$ without using lock-based mechanisms.

In a dual way, actor p_3 is the consumer of channel e_2; while actor p_2 is its producer. As depicted in Figure 1(b), job $p_2[5]$ also preempts job $p_3[4]$. If the deadline of job $p_3[4]$ is less or equal to that of $p_2[5]$, then this latter will not preempt job $p_3[4]$. This technique of deadlines adjustment can be used to play on priorities of jobs and enforce other kinds of precedences.

From the experimental results, the average improvement of the buffering requirements is up to 39% compared to the technique proposed in [8], and of 11% compared to the technique proposed in [1]. Furthermore, our technique may also reduce the context switching overhead since it results in fewer preemptions. However, in some cases, it is not possible to adjust deadlines without jeopardizing the throughput, as we will show in Section 5.

1.2 Paper contributions

The contributions of this paper are twofold.

• The IDP task model, used in [1, 8], is too restrictive since deadlines can be constrained in realistic applications to values less than periods. While standard uniprocessor EDF schedulability analysis of IDP task systems has a polynomial time complexity, existing exact schedulability tests for constrained tasks systems have a pseudo-polynomial time complexity. The main contribution of this a paper is a pseudo-polynomial EDF *symbolic* schedulability analysis of dataflow graphs with constrained deadlines called "Symbolic Quick-convergence Processor-demand Analysis" (SQPA). It is based on the well-known QPA algorithm [24,25]. Instead of testing schedulability for all possible timing parameters, our algorithm does that *incrementally* and so maximizes the processor utilization factor. For partitioned EDF scheduling of dataflow graphs, the allocation problem is known to be NP-hard. We hence propose an allocation heuristic that aims at reducing the buffering requirements. The SQPA algorithm will be used to synthesize the timing parameters for which each partition fits on one processor. In the appendix, we will propose a global EDF symbolic schedulability analysis of dataflow graphs which is similar to the SQPA technique but based on (an improved version of) the speedup-optimal schedulability analysis presented in [3, 5].

• Once we have a symbolic schedulability analysis for constrained task systems, the deadlines adjustment technique, as illustrated in the previous example, can be used to reduce the buffering requirements. The construction of the abstract affine schedule is similar to that of [8] (reviewed separately in Section 3) but also considers the deadlines adjustment as shown in Section 4. One benefit of adding precedence constraints is that the linear safe approximations used to solve the buffer minimization problem are less pessimistic than the ones obtained in [8].

The rest of the paper is organized as follows. Section 2 discusses additional related work. Section 3 introduces the background material on the affine model needed for understanding the contributions of this paper. Our deadlines adjustment technique is then presented in Sections 4 and 5. Sections 6 and 7 present our symbolic EDF schedulability analysis for both uniprocessor and multiprocessor systems. An empirical evaluation of the algorithms is presented in Section 8. Finally, Section 9 ends the paper with conclusions.

2. Related work

Schedules of (C)SDF graphs that minimize buffer sizes under throughput constraints have been investigated in many studies, for instance in [23]. Those works aim at creating static cyclic schedules of actors for which hard real-time scheduling theories are not applicable. It is well known that the buffer sizes influence the maximum throughput that can be achieved. Recently, a technique that computes the trade-offs between the throughput and buffer sizes for (C)SDF graphs was proposed in [20,22]. Our approach rather aims at creating EDF schedules that achieve a high throughput while trying to reduce the total amount of buffer storage capacities. It also considers a more general model than CSDF where production and consumption rates can be ultimately periodic.

EDF scheduling of data-dependent tasks was tackled in [12] by adjusting deadlines and release times and without using lock-based synchronizations. That approach, implemented in the Prelude compiler [16], differs from the present one in three respects. 1) In Prelude, deadlines are adjusted so as to ensure data-dependencies w.r.t. multi-rate communication patterns; while precedences in the present mode may also impose an order of reads before writes. 2) Communications in the Prelude compiler are not flow-preserving: when the producer is faster than the consumer, an adapter is added allowing some produced tokens not be buffered; and when the consumer is faster, an adapter is added to allow for reading the same value several times. 3) Prelude adjusts deadlines and checks schedulability based on user-provided timing parameters while the present proposal performs symbolically both the deadlines adjustment and the schedulability analysis.

Few works had addressed the symbolic schedulability problem. Cimatti et al. [10] used parametric timed automata to symbolically compute the schedulability region (i.e. the region of the parameter space that corresponds to feasible designs) of a task system scheduled using *fixed* priorities. In [13], the Inverse method for parametric timed automata is used to synthesize zones of the timing parameter space where the system is schedulable. Our solution is tailored to a specific scheduling policy and to specific constraints; it has hence less complexity and implementation overhead.

Regarding affine relations, a subclass of affine relations between abstract clocks was used in [14] to address time requirements of streaming applications on multiprocessor systems on chip. The whole class of affine relations is used in this paper. The affine model is also used in [7] to compute fixed-priority schedules of dataflow graphs with constrained deadlines.

3. Background

This section outlines the generic approach for strictly (as opposite to static) periodic scheduling of dataflow graphs proposed in [8]. The method consists of two steps: construction of an abstract schedule of the graph (i.e. a set of scheduling constraints) and then the concretization of the schedule (i.e. computing the concrete timing parameters).

3.1 Affine dataflow graphs

A dataflow graph is a weakly connected directed graph $G = (P, E)$ which consists of a finite set of actors P, and a set of FIFO channels E. The worst-case execution time of each actor $p_i \in P$ is denoted by $C_i \in \mathbb{N}^*$ where $\mathbb{N}^*$ is the set of strictly positive integers. Each channel $e = (p_i, p_k, x, y)$ has exactly one producer p_i and one consumer p_k, and it is associated with two integer functions $x, y : \mathbb{N}^* \to \mathbb{N}$. The function x denotes the production rate; i.e. the producer writes $x(j)$ tokens on channel e during its j^{th} firing. Similarly, the function y denotes the consumption rate; the consumer reads $y(j)$ tokens from channel e during its j^{th} firing. Rate functions must be bounded since an actor cannot produce (or consume) an infinite number of tokens during a finite number of firings. Rate functions are constant in SDF, periodic in CSDF; but they can also be ultimately periodic as we will show in the sequel.

The number of initial tokens on channel $e \in E$ is denoted by $\theta(e)$ s.t. $\theta : E \to \mathbb{N}$, while its size is denoted by $\delta(e)$ s.t. $\delta : E \to \mathbb{N}^*$. Both the number of initial tokens and the size are assumed to be unknown at design time (but they can be user-imposed). Channels are implemented as separated storage spaces (i.e. empty space in one channel cannot be used to store tokens of other channels); as cyclic arrays for example. The buffering requirements of a dataflow graph is hence given by $\sum\limits_{e \in E} F(e)\delta(e)$ where $F(e)$ is the size of a token communicated over channel e.

The cumulative function of a rate function x is a monotone function $X : \mathbb{N} \to \mathbb{N}$ such that $X(j) = \sum\limits_{k=1}^{j} x(k)$. Hence, $X(j)$ denotes the total number of produced (or consumed) tokens until and including the j^{th} firing. In the next section, we will use an ILP formalism; let us therefore take functions $X^l, X^u : \mathbb{N} \to Q$ to be the linear lower bound and the linear upper bound of the cumulative function X, respectively. In the dataflow model, we may allow any kind of rate functions as long as their cumulative functions are linearly bounded.

Example 1 (Ultimately periodic rates)**.** A rate function x is ultimately periodic if and only if $\exists j_0, \pi \in \mathbb{N}^* : \forall j \geq j_0 : x(j + \pi) = x(j)$. The notation $x = u(v)$, for two finite integer sequences u and v, means that $x(j) = u[j]$ if $j \leq |u|$, and $x(j) = v[(j - |u| - 1) \mod |v| + 1]$ otherwise; such that $|u|$ denotes the length of a finite sequence u, $\|u\|$ denotes the sum of its elements, and $u[j]$ denotes its j^{th} element. It is easy to compute the linear bounds of the cumulative function X which increases by $\|v\|$ every $|v|$ steps. So, we have that $\exists \lambda_x^l, \lambda_x^u \in \mathbb{Q} : \forall j \in \mathbb{N} : \frac{\|v\|}{|v|} j + \lambda_x^l \leq X(j) \leq \frac{\|v\|}{|v|} j + \lambda_x^u$.

In time-triggered scheduling of dataflow graphs, each actor p has an activation clock $\hat{p}$ (i.e. an infinite ordered set of ticks). An actor instance, or job, is released at each tick of $\hat{p}$ and its execution must complete before the next tick (i.e. auto-concurrency is disabled) and cannot self-suspend. The j^{th} job of actor p is denoted by $p[j]$ for $j \in \mathbb{N}^*$.

Let $p_i, p_k \in P$ be two actors of the graph, and let us take the monotone function $\Delta_{i,k} : \mathbb{N}^* \to \mathbb{N}$ such that $\forall j \in \mathbb{N}^* : \Delta_{i,k}(j) = \max\{0, j' \in \mathbb{N}^* |$ job $p_k[j']$ is released strictly before job $p_i[j]\}$. The *relative positioning* of ticks of clocks $\hat{p}_i$ and $\hat{p}_k$ can be described entirely by the two functions $\Delta_{i,k}$ and $\Delta_{k,i}$. Those functions do not however describe the physical duration between clock ticks.

In the first step of the scheduling approach, activations clocks are regarded as abstract clocks in the sense that durations between ticks do not matter, and the most important thing is the *relative positioning* of ticks. Affine relations, defined below, between activation clocks are expressive enough to describe strictly periodic schedules of dataflow graphs. Affine transformations of abstract clocks were introduced in [19] and they enjoy a canonical form and other useful mathematical properties.

Definition 1 (Affine relation)**.** A (n, φ, d)-affine relation between two clocks $\hat{p}_i$ and $\hat{p}_k$ has three parameters $n, d \in \mathbb{N}^*$ and $\varphi \in \mathbb{Z}$. In case φ is positive (resp. negative), clock $\hat{p}_i$ is obtained by counting each n^{th} instant on a referential abstract clock $\hat{c}$ starting from the first (resp. $(-\varphi + 1)^{th}$) instant; while clock $\hat{p}_k$ is obtained by counting each d^{th} instant on $\hat{c}$ starting from the $(\varphi + 1)^{th}$ (resp. first) instant.

Figure 2 depicts a $(3, -4, 5)-$affine relation between two actors. If actors p_i and p_k are (n, φ, d)-affine-related, then there is a

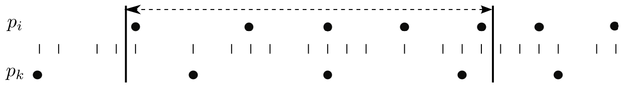

Figure 2. A $(3, -4, 5)-$affine relation. There is a positioning pattern which consists of 5 activations of p_i and 3 activations of p_k.

positioning pattern of ticks that will repeat infinitely so that for every $\frac{d}{\gcd(n,d)}$ activations of actor p_i, there are $\frac{n}{\gcd(n,d)}$ activations of actor p_k. The affine relation describes therefore the relation between the rates of activations of actors. Indeed, a $(1, \varphi, 2)$-affine relation means that the first actor is twice as fast as the second actor; while the difference between their phases is expressed by the parameter φ.

An affine relation can be also described by the two monotone functions $\Delta_{i,k}$ and $\Delta_{k,i}$ such that $\forall j \in \mathbb{N}^*$

$$\Delta_{i,k}(j) = \max\{0, \left\lceil \frac{n(j-1)-\varphi}{d} \right\rceil\}$$

$$\Delta_{k,i}(j) = \max\{0, \left\lceil \frac{d(j-1)+\varphi}{n} \right\rceil\}$$

The sign $\lceil x \rceil$ refers to the smallest integer not less than x. From these two equations, if p_i and p_k are (n, φ, d)-affine-related, then equivalently (1) p_k and p_i are $(d, -\varphi, n)$-affine related; and (2) p_i and p_k are $(cn, c\varphi, cd)$-affine-related for $c \in \mathbb{N}^*$.

3.2 Affine relation synthesis

An affine abstract schedule of the graph consists of all the affine relations between adjacent actors. In order to construct the abstract schedule, we need to find the appropriate affine relation between every two adjacent actors so that: (1) Overflow and underflow exceptions are *statically* excluded to ensure functional determinism. (2) The set of affine relations is consistent. (3) The buffering requirements are minimized.

3.2.1 Overflow and underflow analyses

An underflow exception occurs when an actor attempts to read from an empty channel; while an overflow exception occurs when an actor attempts to write into a full channel. Excluding overflow and underflow exceptions implies that the number of accumulated tokens on every channel and at each step of the execution is greater or equal to zero (i.e. no underflows) and less or equal to the buffer size (i.e. no overflows).

Let p_i be the producer and p_k be the consumer of channel $e = (p_i, p_k, x, y)$ such that actors p_i and p_k are $(n, \varphi, d)-$affine-related. When jobs $p_i[j]$ and $p_k[j']$ complete, the number of accumulated tokens on channel e is given by $\theta(e) + X(j) - Y(j')$. Of course, indices j and j' are affine-related as shown in details in Section 4.

- No overflow exception over channel e means that:

$$\forall(j, j'), \theta(e) + X(j) - Y(j') \leq \delta(e) \tag{2}$$

- No underflow exception over channel e means that:

$$\forall(j', j), \theta(e) + X(j) - Y(j') \geq 0 \tag{3}$$

3.2.2 Consistency

Consistency is an important property of (C)SDF graphs [6, 15]. A non-consistent graph requires unbounded memory to execute or it deadlocks during its execution. Consistency is related to cyclic graphs since bounded and deadlock-free schedules of acyclic graphs always exist.

Example 2. Let us take the cycle $\{p_1, p_2, p_3\}$ in the graph of Figure 1(a). Using Equations 2 and 3, we may compute three affine relations $p_1 \overset{(n_1,\varphi_1,d_1)}{\longrightarrow} p_2 \overset{(n_2,\varphi_2,d_2)}{\longrightarrow} p_3 \overset{(n_3,\varphi_3,d_3)}{\longrightarrow} p_1$ so that their parameters n and d are deduced from the rate functions. For every d_i activations of p_i, there are $n_{(i \bmod 3)+1}$ activations of $p_{(i \bmod 3)+1}$. Therefore, if $n_1 n_2 n_3 \neq d_1 d_2 d_3$, then the graph is non-consistent and cannot execute in a bounded memory.

A second condition for consistency considers the parameters φ (constrained by the number of initial tokens on the channels). Since the difference between phases of two successive actors in the cycle is encoded with the parameter φ, Equation 5 ensures that the difference between the phases of each actor and itself is null. The affine relations of the EDF schedule in Figure 1(b) are $p_1 \overset{(4,9,6)}{\longrightarrow} p_2 \overset{(3,-4,5)}{\longrightarrow} p_3 \overset{(10,-1,4)}{\longrightarrow} p_1$ which satisfy both conditions.

The graph of affine relations is an undirected graph where nodes are actors and edges represent affine relations (recall that an affine relation can be reversed). Consistency of the affine abstract schedule is given by the following proposition.

Proposition 1 (from [8]). *The graph is consistent if for every fundamental cycle* $p_1 \overset{(n_1,\varphi_1,d_1)}{\longrightarrow} p_2 \rightarrow \cdots \rightarrow p_m \overset{(n_m,\varphi_m,d_m)}{\longrightarrow} p_1$ *in the graph of affine relations, we have that*

$$\prod_{i=1}^{m} n_i = \prod_{i=1}^{m} d_i \quad (4) \qquad \sum_{i=1}^{m}\left(\prod_{j=1}^{i-1} d_j\right)\left(\prod_{j=i+1}^{m} n_j\right)\varphi_i = 0 \quad (5)$$

If Proposition 1 is satisfied for fundamental cycles, then it will be also satisfied for any cycle in the graph since fundamental cycles are a cycle basis of the the cycle space. Fundamental cycles can be easily obtained from the spanning tree of the graph of affine relations. A cycle basis can be found in polynomial time.

Remark If the number of initial tokens on channels are imposed by the user, then underflow-free affine schedules may not exist in case of cyclic graphs. Therefore, it is better to let the scheduling algorithm decides the appropriate number of initial tokens.

3.3 EDF symbolic schedulability analysis

The computed affine schedule imposes some scheduling constraints on actors, but it does not indicate how actors will be concretely scheduled. We opt for an EDF scheduling of the dataflow graph, in which each actor p_i is mapped to a periodic task with a period π_i, a phase r_i, and a relative deadline d_i with $\pi_i, d_i \in \mathbb{N}^*$ and $r_i \in \mathbb{N}$ such that $C_i \leq d_i \leq \pi_i$. We also allow the user to impose lower and upper bounds on periods in response to some extra requirements so that $\pi_i^l \leq \pi_i \leq \pi_i^u$.

Symbolic schedulability analysis is the process of computing the timing parameters (π, r, d) of each actor so that

- They respect the affine abstract schedule, the bounds on periods, and other constraints if any.
- They ensure uniprocessor/multiprocessor EDF schedulability.
- They maximize the throughput, or equivalently they maximize the processor utilization factor $U = \sum_{p_i \in P} \frac{C_i}{\pi_i}$.

The first requirement implies that each $(n, \varphi, d)-$affine relation between actors p_i and p_k is concretized as follows.

- $n\pi_k = d\pi_i$.
- $r_k - r_i = \frac{\varphi}{n}\pi_i$.

In words, the concretization imposes constant time intervals between ticks of every activation clock. Indeed, at this stage of the scheduling, physical time does matter. For a weakly connected graph and from the first equation, the periods of all actors can be expressed in terms of the period of a fixed one. Therefore,

$\forall p_i \in P : \pi_i = \alpha_i T$ such that $\alpha_i \in \mathbb{Q}$ and $T^l \leq T \leq T^u$. The bounds on T, if any, are deduced from the bounds on periods imposed by the user. Since timing parameters are integers, T should be multiple of some integer B so that the previous equations may have a solution.

The second and third requirements of symbolic schedulability analysis depend on what approach is used for schedulability analysis: processor-demand approach, processor utilization approach, etc. In this paper, we will present a symbolic processor-demand analysis.

To sum up, this paper proposes a heuristic, that consists of two steps and favors buffer minimization over throughput maximization, to solve the following problem. Let $\theta_t(e)$ be the number of tokens in channel e at instant t and $f_i[j]$ be the finish time of job $p_i[j]$ during an EDF execution of the dataflow graph. The problem consists in finding for every actor $p_i \in P$ the timing parameters (π_i, r_i, d_i) and for every channel $e \in E$ the characteristics $(\theta(e), \delta(e))$ that maximizes the processor utilization U and minimizes $\sum_{e \in E} F(e)\delta(e)$ such that for every EDF execution of the graph we have that $\forall e \in E : \forall t : 0 \leq \theta_t(e) \leq \delta(e)$ and $\forall p_i \in P : \forall j \in \mathbb{N}^* : f_i[j] \leq r_i + (j-1)\pi_i + d_i$. The problem is further complicated in partitioned scheduling where every actor must be assigned to a processor.

4. Affine Schedules with deadlines adjustment

The first step of the approach consists in computing an affine relation between every two adjacent actors by means of an integer linear program (ILP). Therefore, we need to safely approximate Equations 2 and 3. If actors p_i and p_k are (n, φ, d)-affine-related and $e = (p_i, p_k, x = u_1(v_1), y = u_2(v_2))$ is a channel between them, then linear approximations are obtained as follows.

4.1 Overflow analysis

A linear upper bound of the number of accumulated tokens in the channel is given by $\theta(e) + X^u(j) - Y^l(j')$ which must be less or equal to the size $\delta(e)$. We have that (see Example 1) $X^u(j) = \frac{\|v_1\|}{|v_1|} j + \lambda_x^u$ and $Y^l(j') = \frac{\|v_2\|}{|v_2|} j' + \lambda_y^l$. It remains to compute the linear lower bound of j' in terms of j according to the affine relation and assuming that deadlines will be adjusted to enforce the needed precedences.

Let us suppose that p_i and p_k are allocated to the same processor. Figure 3 shows, for different cases, what job of p_k precedes immediately job $p_i[j]$. It shows hence what kind of precedences will be ensured by the deadlines adjustment. Case (c) corresponds to what has been explained in the introduction. In case (e), we have to look at the actors' ID to break the tie. According to these precedences, the lower bound of j' is equal to $\Delta_{i,k}(j)$. Thus, the linear lower bound of j' is $\frac{n}{d} j - \frac{n+\varphi}{d}$. By substituting all the linear bounds in Equation 2, we obtain the following linear constraint. $\forall j \in \mathbb{N}^*$:

$$\forall j \in \mathbb{N}^* : \theta(e) - \delta(e) + \frac{\|v_1\|}{|v_1| n}\varphi + \xi j \leq \lambda_y^l - \lambda_x^u - \frac{\|v_1\|}{|v_1|} \quad (6)$$

such that $\xi = \frac{\|v_1\|}{|v_1|} - \frac{\|v_2/|n}{|v_2|d}$.

Since j tends to infinity, it is a requirement for an execution free of overflows and underflows that ξ equals zero. Consequently, the boundedness criterion is:

$$\frac{n}{d} = \frac{\|v_1\|}{|v_1|} \frac{|v_2|}{\|v_2\|} \quad (7)$$

This boundedness criterion is equivalent to the balance equation in static-periodic scheduling of (C)SDF graphs.

In case actors p_i and p_k are allocated to different processors, precedences will be more conservative since deadlines does not ensure, for instance in case (c), that job $p_k[j']$ will complete before job $p_i[j]$ starts. Indeed, they can execute in parallel with each other.

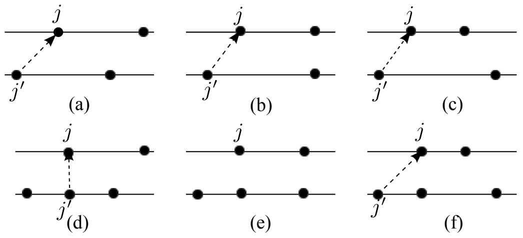

Figure 3. Precedences in the overflow analysis.

4.2 Underflow analysis

The underflow analysis is dual to the overflow analysis. A linear lower bound of the number of accumulated tokens on the channel is given by $\theta(e) + X^l(j) - Y^u(j')$ which must be greater or equal to zero. Again, we need to compute the linear lower bound of j in terms of j'. Without depicting the precedences, the lower bound of j is equal to $\Delta_{k,i}(j')$. Thus, the linear lower bound of j is $\frac{d}{n} j' + \frac{\varphi - d}{n}$. By substituting all the linear bounds in Equation 3, we obtain the following linear constraint.

$$\forall j' \in \mathbb{N}^* : \theta(e) + \frac{\|v_1\|}{|v_1| n}\varphi + \frac{d}{n}\xi j' \geq \lambda_y^u - \lambda_x^l + \frac{\|v_2\|}{|v_2|} \quad (8)$$

Equations 6 and 8 are less pessimistic than the ones obtained without imposing precedence constraints.

4.3 Algorithm

The input of this algorithm is a dataflow graph with ultimately periodic rates; while the output is the set of all affine relations. In the first place, the boundedness criterion (Equation 7) is used to compute the parameters n and d of all the affine relations. Then, Equation 4 is used to check the first requirement of the consistency of the graph. To compute the parameters φ, we construct an integer linear program by applying Equations 6, and 8 on channels; and Equation 5 on fundamental cycles. The objective function of the linear program is to minimize the buffering requirements. Integer linear programming is NP-hard; however solutions for real-life benchmarks, used in Section 8, can be computed in few seconds.

Sizes obtained by the solution of the linear program are a safe approximation of the actual sizes. Therefore, they may be recomputed after obtaining the affine relations.

5. Deadlines adjustment

This section shows how deadlines are adjusted to enforce the needed precedences so that if the task system is EDF-schedulable, then precedences are guaranteed without the need for locks. Let us denote by $R_i[j]$ the release time of job $p_i[j]$. We have that $R_i[j] = r_i + (j-1)\pi_i$. Let us also denote by $D_i[j]$ and $d_i[j]$ the absolute and relative deadlines of job $p_i[j]$, respectively, such that $D_i[j] = R_i[j] + d_i[j]$.

Let p_i and p_k be two $(n, \varphi, d)-$affine-related adjacent actors; and let us take $\Gamma_{i,k}(j) = \min\{j' | \; D_i[j]$ must be less or equal to $D_k[j']\}$. Figure 3 gives an intuition on how to compute $\Gamma_{i,k}(j)$. For instance, in case (c), $\Gamma_{i,k}(j) = j' + 1$ and $\Gamma_{k,i}(j') = j$.

The absolute deadline of job $p_i[j]$ of an actor p_i can be now computed as follows. Let us put $\mathcal{S}_i$ to be the set of neighbors of actor p_i. We have that

$$D_i[j] = \min_{p_k \in \mathcal{S}_i} \{R_i[j+1], D_k[j'] - 1 | j' = \Gamma_{i,k}(j)\} \quad (9)$$

That is, the absolute deadline of a job $p_i[j]$ is less or equal to its implicit deadline (i.e. $R_i[j+1]$) and less than the absolute deadline $D_k[j']$ of any job $p_k[j']$ that must be preceded by $p_i[j]$. In Equation 9, $D_i[j]$ cannot depend on itself according to the way precedences are computed. If we consider relative deadlines, then Equation 9 can be written as

$$d_i[j] = \min_{p_k \in \mathcal{S}_i} \{\pi_i, d_k[j'] - 1 + R_k[j'] - R_i[j] | j' = \Gamma_{p_i,p_k}(j)\}$$

Recall that each affine relation contains a positioning pattern that repeats infinitely. Thus, relative deadlines of jobs of an actor will be ultimately periodic. From the concretization of affine relations, we know that periods and phases can be expressed in terms of the period of a fixed actor ($\forall p_i \in P : \pi_i = \alpha_i T$). Thus, the deadlines can be also expressed in terms of T.

5.1 Approximate deadlines

It is sometimes required, as in the schedulability analysis, that each actor has a single deadline. Thus, we may take $\forall p_i \in P : d_i = \min_j d_i[j]$. The deadlines computation will be much simpler as shown in the following.

$$\begin{aligned} d_i &= \min_j d_i[j] = \min_j \min_{p_k \in \mathcal{S}_i} \{\pi_i, d_k[j'] - 1 + R_k[j'] - R_i[j]\} \\ &= \min_{p_k \in \mathcal{S}_i} \{\pi_i, (\min_j d_k[j']) - 1 + \min_j (R_k[j'] - R_i[j])\} \\ &\geq \min_{p_k \in \mathcal{S}_i} \{\pi_i, d_k - 1 + \min_j (R_k[j'] - R_i[j])\} \\ &\quad (\text{because } \min_j d_k[j'] \geq \min_j d_k[j] = d_k) \end{aligned}$$

$\min_j (R_k[j'] - R_i[j])$ can be equal to zero as in case (f) of Figure 3 or can be equal to $\frac{\pi_i}{n}$ in other cases. If we put $X = (d_1, d_2, \ldots)^\top$, the deadlines computation can be written as $X = F(X)$. So, we need to compute the greatest fixed point of the function F. Since this latter is a monotone function, the fixed point, *if any*, can be found by computing the sequence $X^0, X^1 = F(X^0), X^2 = F(X^1), \ldots$ until stabilization such that X^0 is the vector where $\forall p_i \in P : d_i = \pi_i$.

Example 3. If we apply the affine relation synthesis described in Section 4 on the graph in Figure 1(a), we obtain the following affine relations: $p_1 \xrightarrow{(2,3,3)} p_2 \xrightarrow{(3,-3,5)} p_3 \xrightarrow{(5,0,2)} p_1$. So, we have that $p_1 = \frac{2}{3}T, p_2 = T$, and $p_3 = \frac{5}{3}T$. If we consider approximate deadlines, we obtain the following equations. $d_1 = \min\{\pi_1, d_2 - 1, d_3 - 1\}$, $d_2 = \min\{\pi_2, d_1 - 1 + \frac{\pi_2}{3}, d_3 - 1\}$, and $d_3 = \min\{\pi_3, d_1 - 1 + \frac{\pi_3}{5}, d_2 - 1 + \frac{\pi_3}{5}\}$. Hence, $d_1 = \pi_1, d_2 = \pi_2 - 2$, and $d_3 = \frac{3}{5}p_3 - 1$.

The new task system is schedulable for $T \geq 261$ as illustrated in Figure 4. As one can notice, no job is preempted by other jobs and all precedences as described above are satisfied. The safe buffering requirements of the schedule in Figure 1(b) equal 20; while they equal only 12 in the new schedule. This buffering improvement comes at the price of a processor utilization decrease from 1.0 to 0.92.

5.2 Impact on the processor utilization factor

The processor utilization factor is given by $U = \sum_{p_i \in P} \frac{C_i}{\alpha_i T} = \frac{\sigma}{T}$ with $\sigma = \sum_{p_i \in P} \frac{C_i}{\alpha_i}$. A necessary condition for EDF schedulability on m identical processors ($m \geq 1$) is that $U \leq m$; which implies that $T \geq \frac{\sigma}{m}$. For $T \simeq \frac{\sigma}{m}$, we obtain the maximum utilization factor when using an IDP model.

Suppose that after the deadlines computation, we have that $d_i[j] = \beta T - \beta'$ with $\beta \in \mathbb{Q}^+$ and $\beta' \in \mathbb{N}$. But $d_i[j] \geq C_i$ implies that $T \geq \frac{C_i + \beta'}{\beta}$. Therefore, if the bound $\frac{C_i+\beta'}{\beta}$ is much larger than $\frac{\sigma}{m}$, then the utilization factor will be definitely worse than the one obtained by the IDP model.

One solution is to exclude from the set $\mathcal{S}_i$ in Equation 9 each actor p_k for which imposing a precedence between a job $p_i[j]$ and a job $p_k[j']$ may jeopardize the utilization factor. When it comes to buffer sizes computation of channels between p_i and p_k, if a preemption occurs, then we will consider the worst-case scenarios as in [8].

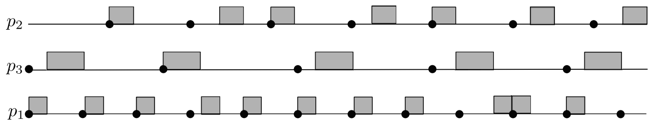

Figure 4. EDF scheduling after deadlines adjustment.

6. Uniprocessor schedulability analysis

This section describes the symbolic uniprocessor EDF schedulability analysis of dataflow graphs. We have the following constraints that are deduced from the affine relation synthesis, deadlines adjustment, and concretization of affine relations.

- $\forall p_i \in P : \pi_i = \alpha_i T$ and $d_i = \beta_i T - \beta'_i$ (we take a single minimum deadline for each task).
- $T^l \leq T \leq T^u$ and T must be a multiple of some integer B since timing parameters must be integers. The bounds are obtained from the user-provided bounds on periods and from the constraint $C_i \leq d_i \leq \pi_i$.
- $U = \frac{\sigma}{T} < 1$.

The input of the symbolic schedulability analysis is the previous constraints while its output is the minimum value of T (which gives the maximum utilization factor) that guarantees uniprocessor EDF schedulability. All the timing parameters can be deduced once T is known. From the previous constraints, we have that $T \in \{T_k | \sigma' = \max\{T^l, \lfloor\sigma\rfloor + 1\} \leq T_k = kB \leq T^u \wedge k \in \mathbb{N}\}$. The sign $\lfloor x \rfloor$ refers to the largest integer not greater than x. Hence, $T_k = \left\lceil \frac{\sigma'}{B} \right\rceil B + kB$ for $k \geq 0$. If we want to bound the processor utilization factor to a value less than one (e.g. in case we want to add an aperiodic server that handles some aperiodic tasks in the system), we need just to change the lower bound on T.

The enumerative solution consists in checking the EDF schedulability of the task system for each T_k in an increasing order, staring from T_0 and until reaching an appropriate value or exceeding the upper bound T^u. However, this is a time-consuming approach. In the rest of this section, we present a faster solution based on the following observation. When T is increased, periods and deadlines are stretched while execution times remain constant. Hence, if a deadline is not missed for the previous value of T, then it will not be missed for the new one.

6.1 Standard schedulability analysis

The exact schedulability analysis of constrained periodic task systems is based on the processor-demand approach as given by the following lemma.

Lemma 1 (from [17]). *A synchronous periodic task set is schedulable if and only if $U < 1$ and $\forall l \leq L^* : h(l) \leq l$ where $L^* = \frac{\sum_{p_i \in P} (\pi_i - d_i) U_i}{1-U}$ and $U_i = \frac{C_i}{\pi_i}$.*

L^* is called the feasibility bound. As proposed in [17], another feasibility bound consists in the the length of the synchronous busy period that can be computed by the following recurrence.

$$w^0 = \sum_{p_i \in P} C_i \qquad w^{m+1} = \sum_{p_i \in P} \left\lceil \frac{w^m}{\pi_i} \right\rceil$$

When the recurrence stops (i.e. $w^{m+1} = w^m$), then $L^* = w^m$.

The processor demand function $h(l)$ calculates the maximum processor demand of all jobs which have their arrival times and deadlines in a contiguous interval of length l. So, $h(l)$ is given by $h(l) = \sum_{p_i \in P} \max\{0, 1 + \left\lfloor \frac{l-d_i}{\pi_i} \right\rfloor\} C_i$. The value of the demand function does not change from one point l to another one l' unless there is at least one absolute deadline in $]l, l']$. Therefore, it is necessary to check condition $h(l) \leq l$ only for the set of absolute deadlines which are less than L^*. This set can be large and a technique like the Quick convergence Processor-demand analysis (QPA) [24] is needed. In the next paragraphs, $d, d', d_m,$ and d^* denote some absolute deadlines.

Lemma 2 (from [24]). *For an unschedulable periodic task set, if $h(d_m) \leq d_m$, then $d^* < h(d^*) \leq d'$, where $d_m = \max\{d | d \leq L^*\}, d^* = \max\{d | 0 < d < L^* \wedge h(d) > d\}$, and $d' = \min\{d | d > d^*\}$.*

From Lemma 2, we can easily deduce that $\forall l \in [h(d^*), L^*]$, $h(l) \leq l$. Based on this result, Listing 1 represents the QPA algorithm (from [24]).

Algorithm 1: QPA algorithm

```
$l = d_m$;
while $h(l) \leq l \wedge h(l) > \min\{d\}$ do
    if $h(l) < l$ then $l = h(l)$;   else $l = \max\{d | d < l\}$;
if $h(l) \leq \min\{d\}$ then the task set is schedulable;
else the task set is not schedulable;
```

6.2 Symbolic QPA

Our symbolic schedulability analysis of dataflow graphs consists in incorporating the search of the minimum T that ensures EDF schedulability into the QPA algorithm. Listing 2 represents the symbolic QPA algorithm. Let $L^*(T), U(T)$, and $h^T(t)$ denote respectively the values L^*, U, and $h(t)$ for a given T.

Starting from the minimum value of T (i.e. T_0), SQPA performs the QPA analysis in the interval $[0, L^*(T_0)]$. This first iteration leads either to $h^{T_0}(l) \leq \min\{d\}$ or to a deadline miss, i.e. $h^{T_0}(l) > l$ (Figure 5). In the first case, the task system is schedulable and the algorithm returns T_0.

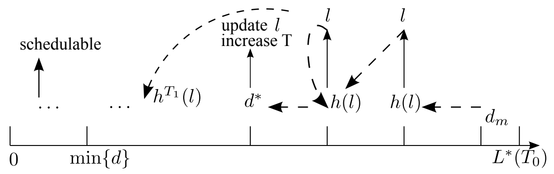

Figure 5. Illustration of SQPA.

In the second case, assume that the deadline miss occurs at d^* (i.e. $h^{T_0}(d^*) > d^*$). In this case, T must be increased to T_1 (instruction `k++;`). According to Lemma 3, $L^*(T_1) \leq L^*(T_0)$. Hence, the verification can restart from $L^*(T_1)$ instead of $L^*(T_0)$. But, according to Lemma 2, we have that $\forall l \in [h^{T_0}(d^*), L^*(T_0)] : h^{T_0}(l) \leq l$. Since Lemma 4 implies that $\forall l \in [h^{T_0}(d^*), L^*(T_0)] : h^{T_1}(l) \leq h^{T_0}(l) \leq l$, the verification process for T_1 can restart from $\min\{L^*(T_1), h^{T_0}(d^*)\}$. This process is repeated until the termination condition is reached or T exceeds the upper bound T^u.

Algorithm 2: SQPA algorithm

```
$k = 0$; $l = \max\{d | d \leq L^*(T_k)\}$;
while $h(l) > \min\{d\}$ do
    if $h(l) < l$ then $l = h(l)$;
    else if $l == h(l)$ then $l = \max\{d | d < l\}$;
    else
        k++;
        if $T_k > T^u$ then return task set not schedulable;
        $l = \min\{h^{T_{k-1}}(l), \max\{d | d \leq L^*(T_k)\}\}$;
return $T_k$;
```

Lemma 3. *If $T \leq T'$, then $L^*(T) \geq L^*(T')$.*

Proof: The proof follows easily from the definition of the synchronous busy period. ☐

Lemma 4. *If $T \leq T'$, then $\forall l, h^T(l) \geq h^{T'}(l)$.*

Proof: We have that $h^T(l) = \sum_{p_i \in P} h_i^T(l)$ such that $h_i^T(l) = \sum_{D_i[j] \leq l} C_i$. Recall that any absolute deadline $D_i[j]$ can be written as $(\alpha_i(j-1) + \beta_i)T - \beta'_i$. But α_i and β_i are positive; which implies that a given absolute deadline occurs earlier for T than for T'. Therefore, $\forall l, h_i^T(l) \geq h_i^{T'}(l)$. ☐

SQPA algorithm has a pseudo-polynomial complexity since it checks in the worst-case scenario all the deadlines in interval $[0, L^*(T_0)]$ as a standard processor-demand analysis may do.

7. Multiprocessor schedulability analysis

One of the major advantages of using partitioned EDF over global EDF scheduling is that, once an allocation of actors to processors has been achieved, real-time symbolic schedulability analysis for uniprocessor systems (e.g. the SQPA technique) can be applied on each partition. The main objective of the allocation heuristic proposed in this section is to minimize the buffering requirements of dataflow graphs scheduled on an architecture that consists of m identical processors with a shared memory where channels are implemented as software buffers. The scheduling approach consists of the following steps.

1) Parameters n and d of every affine relation are computed using the boundedness criterion. So, we have that $\forall p_i \in P : \pi_i = \alpha_i T$.

2) Let $G = (V, E)$ be an undirected graph where nodes represent actors and edges represent affine relations. Each node v_i is associated with a weight that represents the utilization of actor p_i; i.e. $w(v_i) = \frac{C_i}{\alpha_i}$. The weight of edge $e_{i,j} = (v_i, v_j)$, denoted by $w(e_{i,j})$, is equal to zero if enforcing precedences between p_i and p_j may jeopardize the processor utilization factor, and equal to the approximate gain that comes from enforcing the precedences otherwise.

3) The graph G is partitioned into m balanced partitions $(V_i)_{i=1,m}$; that is, $\forall i, j : |w(V_i) - w(V_j)|$ is minimal such that $w(V_i) = \sum_{v \in V_i} w(v)$. The second objective of this partitioning is to minimize the total weight of edges connecting different partitions. The rational behind this requirement is that adjusting deadlines will not ensure precedences between actors allocated on different processors. This m-partitioning problem is well known in graph theory; for instance, we use the SCOTCH tool [9] to solve the problem.

4) Once each actor is assigned to a processor, overflow and overflow analysis can be performed to compute parameter φ of each affine relation such that the worst-case scenarios are considered for edges that have a weight equal to zero and edges connecting partitions.

5) Computation of symbolic deadlines of tasks in each partition.
6) The SQPA algorithm is then applied on each partition V_i. The algorithm will return the minimum value T_i that ensures EDF schedulability of the set V_i on a single processor. We need just to take $T = \max_{i=1,m} T_i$.

8. Experimental validation

We evaluate our scheduling technique w.r.t. buffering requirements, processor utilization factor, and performance by performing an experiment on a set of 18 real-life applications which come from different domains (see [1] for more details on the benchmarks) and some randomly generated SDF graphs and task sets.

8.1 Throughput and buffering requirements

Figures 6 and 7 present the results obtained by three tools on a single processor system: `DARTS` represents the results obtained by the `DARTS` tool [1, 2], `ADF` represents the results obtained by the `ADF` tool [8], and `ADF2` represents the results obtained by the technique presented in this paper. Figure 6 shows the ratios of the buffering requirements to the minimum (regardless of the throughput) achievable buffering requirements. The minimum buffer sizes are obtained by the `SDF`^{3} tool [21] which computes static-periodic schedules of (C)SDF graphs (auto-concurrency disabled). Figure 7 shows the processor utilization factor obtained by each tool.

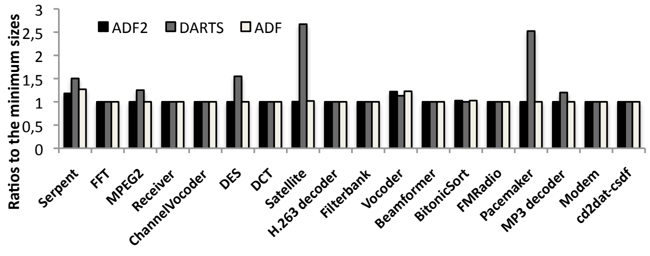

Figure 6. Buffering requirements comparison.

A channel $e = (p_i, p_k, u_1(v_1), u_2(v_2))$ is said to be a matched I/O channel if the values $\frac{\|v_1\|}{|v_1|}$ and $\frac{\|v_2\|}{|v_2|}$ are close to each other. It is said to be a perfectly matched I/O channel if $\frac{\|v_1\|}{|v_1|} = \frac{\|v_2\|}{|v_2|}$. The three tools seem to be able to achieve results close to the minimum buffering requirements for graphs that consist of perfectly matched I/O channels (e.g. `FFT`, `DCT`, `FMRadio`, `Serpent`). Our tool slightly outperforms the other tools, in terms of buffering requirements, for this kind of graphs (e.g. `Serpent`, `DES`, `MPEG2`). A $(1, \varphi, 1)$-affine relation is associated to each perfectly matched I/O channel. Hence, all the reads and writes are already ordered without the need for any deadlines adjustment. This is why, for graphs with perfectly matched I/O channels, the `ADF2` tool gives similar throughput and buffer storage capacities to the ones obtained by the `ADF` tool.

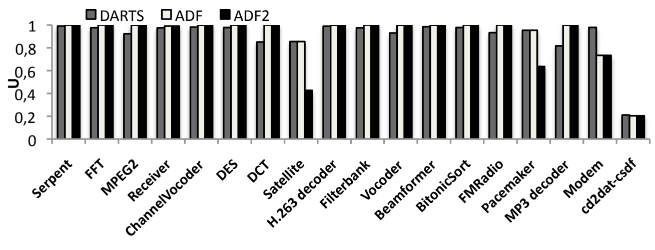

Figure 7. Throughput comparison.

For graphs with mis-matched I/O channels (e.g. `Pacemaker`, `Satellite`), the `DARTS` tool gives the worst buffering requirements. The average improvement obtained by our tool is of 11.43% compared to `DARTS`. The `ADF2` tool slightly outperforms the `ADF` tool but at the price of a processor utilization decrease. Clearly, the IDP task model (i.e. `ADF` tool) results in the best throughput.

Most of the previous benchmarks consist of perfectly matched I/O channels; they unfortunately do not allow us to measure the benefit that comes from adjusting deadlines and using a constrained task model instead of the IDP model. We will use hence a set of randomly generated SDF graphs. The graphs are generated by the SDF^3 tool with the following setting. Production and consumption rates follow a normal distribution with a mean equal to 5 and a variance equal to 4. Worst-case execution times follow a normal distribution with a mean equal to 1000 and a variance equal to 300. The graphs are generated with different number of nodes (from 6 to 80) such that the average degree of each node is 2. Figure 8 shows the obtained results where `Imp` denotes the improvement in buffering requirements obtained by the `ADF2` tool compared to `ADF`; while `Dec` denotes the decrease in the processor utilization.

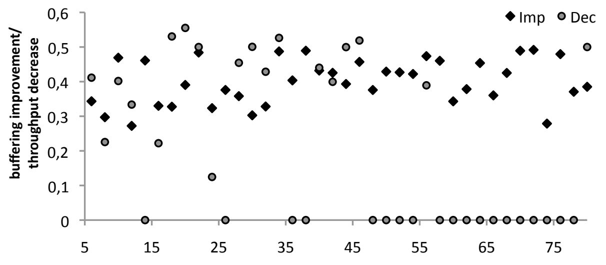

Figure 8. Impact of deadlines adjustment.

The average improvement is equal to 39.99% with a low standard deviation (0.05). This improvement comes at the price of a throughput decline with an average equal to 20.95%. In 50% of the graphs, the deadlines adjustment improves the buffering requirements without affecting the throughput. These are graphs with highly mis-matched I/O channels. Indeed, when factor B is too large (compared to σ), adjusting deadlines may not impact the achievable throughput. Compromised solutions can be obtained by excluding some precedences in the deadlines adjustment step.

8.2 Schedulability analysis

Since QPA algorithm outperforms the standard processor demand-analysis (see [25] for experiments), it is expected that SQPA algorithm also outperforms the enumerative solution. We will compare the number of checked points by both approaches on a huge set of randomly generated task sets. For each task p_i, we generate three parameters (C_i, α_i, β_i) such that $\pi_i = \alpha_i T$ and $d_i = \beta_i T$. 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 B (T must be a multiple of B) and the value of an experimental parameter $D \in]0, 1]$ so that $\forall p_i : \beta_i \in [D\alpha_i, \alpha_i]$. Figure 9 shows the obtained results for different configurations of N, B, and D. Each point on the diagram is the average number of checked points by the enumerative solution for 2000 task sets divided by the average number of checked points by the SQPA algorithm. For each configuration, we denote by E the average number of checked points per task (obtained by the enumerative solution) to indicate the complexity of the problem.

The complexity of the symbolic schedulability problem increases inversely with the factor B. Indeed, if B has a small value, then the number of checked deadlines will be large since each time a deadline miss occurs, T is increased only by a small quantity. For dataflow graphs with matched I/O channels, factor B is generally small, and the schedulability problem is hence more complicated.

The SQPA algorithm outperforms the enumerative solution in all cases except cases with small N and cases with very small D. When deadlines are too constrained, deadline misses could be detected earlier by a forward search than by a backward search. Thus, for dataflow graphs with few actors or highly constrained deadlines, it is better to use the enumerative solution than the SQPA algorithm.

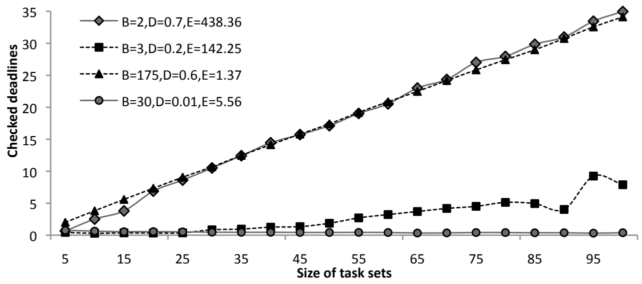

Figure 9. Performance of the SQPA algorithm.

9. Conclusion

We have presented a technique to reduce buffer sizes in uniprocessor/multiprocessor EDF scheduling of dataflow graphs. The technique consists in enforcing some precedences between jobs and encoding them by adjusting deadlines. After constructing an abstract affine schedule of the graph, our symbolic schedulability analysis computes all the necessary timing parameters so it maximizes the processor utilization factor. The technique has significantly improved the buffering requirements in many cases as shown by the experimental evaluation. Our ongoing work aims at further improving the processor utilization factor by looking to actors in a more fine-grained way. We are also investigating how to use deadlines adjustment to allow deterministic multi-producer/multi-consumer channels by imposing total order on writes and reads.

Acknowledgments

We would like to thank Mohamed Bamakhrama for his support with benchmarks and the DARTS tool. We would like also to thank the anonymous reviewers for their valuable comments.

References

[1] M. Bamakhrama and T. Stefanov. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *Proceedings of the 9th ACM International Conference on Embedded Software*, pages 195–204, 2011.

[2] M. A. Bamakhrama and T. Stefanov. Managing latency in embedded streaming applications under hard-real-time scheduling. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 83–92, 2012.

[3] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Implementation of speedup-optimal global EDF schedulability test. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, pages 259–268, 2009.

[4] S. K. Baruah and N. W. Fisher. The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Syst.*, 36(3):199–226, 2007.

[5] M. Bertogna and S. Baruah. Tests for global EDF schedulability analysis. *J. Syst. Archit.*, 57(5):487–497, 2011.

[6] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44:397–408, 1996.

[7] A. Bouakaz and J.-P. Talpin. Design of safety-critical Java Level 1 applications using affine abstract clocks. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems*, 2013.

[8] A. Bouakaz, J.-P. Talpin, and J. Vitek. Affine data-flow graphs for the synthesis of hard real-time applications. In *Proceedings of the 12th International Conference on Application of Concurrency to System Design*, 2012.

[9] C. Chevalier and F. Pellegrini. PT-Scotch: a tool for efficient parallel graph ordering. *Parallel Comput.*, 34(6-8):318–331, 2008.

[10] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric automata. In *the 29th IEEE Real-Time Systems Symposium*, pages 80–89, 2008.

[11] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2001.

[12] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 301–310, 2010.

[13] L. Fribourg, R. Soulat, D. Lesens, and P. Moro. Robustness analysis for scheduling problems using the inverse method. In *19th International Symposium on Temporal Representation and Reasoning*, pages 73–80, 2012.

[14] A. Gamatié. Design of streaming applications on MPSoCs using abstract clocks. In *Design, Automation and Test in Europe Conference*, pages 763–768, 2012.

[15] E. A. Lee and D. G. Messerchmitt. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Trans. Comput.*, 36:24–35, 1987.

[16] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete event dynamic systems*, 21(3):307–338, 2011.

[17] I. Ripoll, A. Crespo, and A. K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Syst.*, 11(1):19–39, 1996.

[18] L. Sha, T. Abdelzaher, K.-E. Arzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: a historical perspective. *Real-Time Syst.*, 28(2-3):101–155, 2004.

[19] I. M. Smarandache, T. Gautier, and P. L. Guernic. Validation of mixed signal-alpha real-time systems through affine calculus on clock synchronisation constraints. In *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems*, volume 2, pages 1364–1383, 1999.

[20] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd annual Design Automation Conference*, pages 899–904, 2006.

[21] S. Stuijk, M. Geilen, and T. Basten. Sdf^3: Sdf for free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*, pages 276–278, 2006.

[22] S. Stuijk, M. Geilen, and T. Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. Comput.*, 57:1331–1345, 2008.

[23] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Efficient computation of buffer capacities for cyclo-static datatflow graphs. In *Proceedings of the 44th annual Design Automation Conference*, pages 658–663, 2007.

[24] F. Zhang and A. Burns. improvement to quick processor-demand analysis for EDF-scheduled real-time systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, pages 76–86,

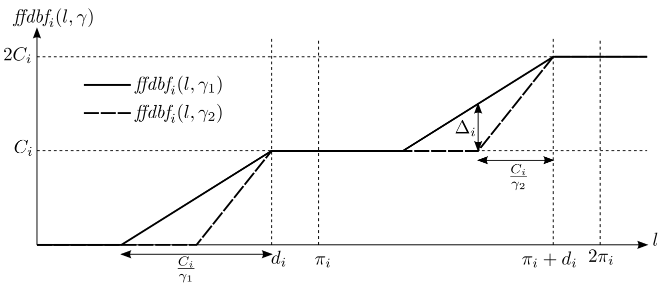

Figure 10. Illustration of *ffdbf*

2009.

[25] F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58:1250–1258, 2009.

A. Global EDF symbolic schedulability analysis

In this section, we will present a global EDF symbolic schedulability analysis of dataflow graphs where a single job can migrate to and execute on different processors. Interest in global EDF has been recently rekindled by the advancement in multiprocessor technology that reduces the migration penalties [5].

A.1 Standard schedulability analysis

Our symbolic schedulability analysis is based on the speedup-optimal schedulability analysis presented in [3, 5] which has a pseudo-polynomial complexity.

Lemma 5 (from [5]). *A task set is schedulable if* $\exists\gamma_{\min} \le \gamma < \gamma_{\max} - \epsilon$ *(with an arbitrary small* ϵ*) such that* $\forall l \le L^* : h(l,\gamma) \le l$ *where*

$L^* = \frac{\sum_{p_i\in P}(\pi_i - d_i)U_i}{m-(m-1)\gamma-U}$, $\gamma_{\min} = \max_{p_i\in P}\frac{C_i}{D_i}$, $\gamma_{\max} = \frac{m-U}{m-1}$, $h(l,\gamma) = \frac{ffdbf(l,\gamma)}{m-(m-1)\gamma}$, and $ffdbf(l,\gamma) = \sum_{p_i\in P} ffdbf_i(l,\gamma)$ which is the forced-forward demand bound function. We have that: $q_i = \left\lfloor \frac{l}{\pi_i} \right\rfloor$, $r_i = (l \mod \pi_i)$, and

$$ffdbf_i(l,\gamma) = q_iC_i + \begin{cases} C_i & \text{if } r_i \ge d_i \\ C_i - (d_i - r_i)\gamma & \text{if } d_i > r_i \ge d_i - \frac{C_i}{\gamma} \\ 0 & \text{otherwise} \end{cases}$$

Figure 10 depicts the forced-forward demand bound function for two values γ_1 and γ_2 such that $\gamma_1 < \gamma_2$. It is sufficient to check condition of Lemma 5 for only absolute deadlines in the interval $[0, L^*]$. Furthermore, the technique of QPA can be used to reduce the number of points to be checked. Listing 3 represents the QPA-FFDBF algorithm proposed in [5].

Algorithm 3: QPA-FFDBF algorithm

```
$\gamma = \gamma_{\min}$;
while $\gamma < \gamma_{\max}$ do
    $l = L^*$;
    while $\min\{d\} < h(l,\gamma) \le l$ do
        $l = \min\{h(l,\gamma), \max\{d | d < l\}\}$;
    if $h(l,\gamma) \le \min\{d\}$ then return the task set is schedulable ;
    $\gamma = \gamma + \epsilon$;
return the task set is unschedulable;
```

A.2 Symbolic QPA-FFDBF

Firstly, we improve the QPA-FFDBF algorithm by exploiting the following lemma.

Lemma 6. *if* $\gamma_1 < \gamma_2$ *and* $ffdbf(l,\gamma_1) \ge (\frac{m-(m-1)\gamma_1}{(m-1)\gamma_2}) \sum_{p_i\in P} C_i$, *then* $h(l,\gamma_1) \le h(l,\gamma_2)$.

Proof: As shown in Figure 10, we have clearly that $\forall l : 0 \le \Delta_i(l) = ffdbf_i(l,\gamma_1) - ffdbf_i(l,\gamma_2) \le \Delta_i$. Using basic geometry, we have that $\Delta_i = C_i(1 - \frac{\gamma_1}{\gamma_2})$. So,

$$h_i(l,\gamma_2) - h_i(l,\gamma_1) = \frac{ffdbf_i(l,\gamma_1) - \Delta_i(l)}{m-(m-1)\gamma_2} - \frac{ffdbf_i(l,\gamma_1)}{m-(m-1)\gamma_1}$$

Hence,

$$h(l,\gamma_2) - h(l,\gamma_1) = \frac{x}{(m-(m-1)\gamma_2)(m-(m-1)\gamma_1)}$$

s.t. $x = (m-1)(\gamma_2 - \gamma_1)\, ffdbf(l,\gamma_1) - (m-(m-1)\gamma_1) \sum_{p_i\in P} \Delta_i(l)$.

Therefore, if $x \ge 0$; i.e.

$$ffdbf(l,\gamma_1) \ge \frac{m-(m-1)\gamma_1}{(m-1)(\gamma_2-\gamma_1)} \sum_{p_i\in P} \Delta_i(l)$$

then $h(l,\gamma_1) \le h(l,\gamma_2)$. But, $\forall l : \sum_{p_i\in P} \Delta_i(l) \le \sum_{p_i\in P} \Delta_i$. □

Lemma 6 is used as follows. For a given γ_1, if $h(l,\gamma_1) > l$ and $ffdbf(l,\gamma_1) \ge F^* = (\frac{m-(m-1)\gamma_1}{(m-1)\gamma_1}) \sum_{p_i\in P} C_i$, then it does not matter if we increase γ_1 to γ_2 since $h(l,\gamma_2)$ will be also greater than l. Note that we used γ_1 in the dominator of F^* instead of γ_2.

Algorithm

Listing 4 represents the symbolic QPA-FFDBF algorithm. We have the same hypotheses as in SQPA; i.e. $\forall p_i \in P : \pi_i = \alpha_i T_k$ and $d_i = \beta_i T_k - \beta'_i$ such that $C_i \le d_i \le \pi_i$. In addition, $U = \frac{\sigma}{T_k}$ and $T_k = \left\lceil \frac{\sigma'}{B} \right\rceil B + kB \le T^u$ such that $\sigma' = \max\{T^l, \lfloor \frac{\sigma}{m} \rfloor + 1\}$. Let $\gamma_{\min}(T)$, $\gamma_{\max}(T)$, and $F^*(T)$ denote respectively $\gamma_{\min}$, $\gamma_{\max}$, and F^* for a given T.

If $h(l,\gamma) > l$, then we have to increase either γ or T according to Lemmas 6 and 5. As for SQPA, we note that

- If $T \le T'$, then $\forall l : h^T(l,\gamma) \ge h^{T'}(l,\gamma)$. Hence, if d^* is the first deadline for which $h^T(d^*,\gamma) > d^*$, then we have that $\forall l \in [h^T(d^*,\gamma), L^*(T)] : h^{T'}(l,\gamma) \le h^T(l,\gamma) \le l$. For given values T_{k+1} and γ, we take Prev$(\gamma) = h^{T_k}(d^*,\gamma)$.
- If $T \le T'$, then $L^*(T) \ge L^*(T')$.

Thanks to these observations, for a given value γ, it not necessary to recheck deadlines in the interval $[L^*(T'), \text{Prev}(\gamma)]$ when T is increased to T'.

Algorithm 4: SQPA-FFDBF algorithm

```
$k = 0$; $\gamma = \gamma_{\min}(T_k)$; $l = L^*(T_k)$;
while $h(l,\gamma) > \min\{d\}$ do
    if $h(l,\gamma) \le l$ then $l = \min\{h(l,\gamma), \max\{d | d < l\}\}$;
    else
        if $ffdbf(l,\gamma) \ge F^*(T_k) \vee \gamma + \epsilon > \gamma_{\max}(T_k)$ then
            k++;
            if $T_k > T^u$ then return task set unschedulable;
            $\gamma = \gamma_{\min}(T_k)$;
        else $\gamma = \gamma + \epsilon$;
        $l = \min\{L^*(T_k), \text{Prev}(\gamma)\}$;
return $T_k$;
```