Real-Time Fixed Priority Scheduling Synthesis using Affine DataFlow Graphs: from Theory to Practice

Alexandre Honorat, Hai Nam Tran, Thierry Gautier, Loïc Besnard, Shuvra S. Bhattacharyya, Jean-Pierre Talpin

Real-Time Fixed Priority Scheduling Synthesis using Affine DataFlow Graphs: from Theory to Practice

ALEXANDRE HONORAT, Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, France
HAI NAM TRAN, University of Brest, CNRS, Lab-STICC - UMR 6285, France
THIERRY GAUTIER, INRIA Research Centre at the Univ. of Rennes, France
LOÏC BESNARD, CNRS, Univ. of Rennes Research Centre, France
SHUVRA S. BHATTACHARYYA, University of Maryland, College Park, USA; INSA/IETR, Rennes, France; INRIA, Rennes, France
JEAN-PIERRE TALPIN, INRIA Research Centre at the Univ. of Rennes, France

The major drawback of using static schedules to execute dataflow applications is their high inflexibility. In real-time systems, periodic schedules make it easier to assert safety guarantees and to decrease the schedule size, but their characteristics remain hard to compute. This article presents an approach to automatically generate fixed priority schedules from a dataflow specification. To do so, precedence dependencies between actors in the dataflow graphs are abstracted, as well as the task periods, by using *affine relations*. This abstraction allows us to synthesize schedules efficiently considering two main objectives: the maximization of throughput and the minimization of buffer sizes. Given a dataflow graph to execute in a real-time environment, we transform it into an Affine DataFlow Graph (ADFG) and compute the task priorities, their mapping, the number of delays in the buffers, and the buffer sizes. This article is the first to present an overview of both theoretical and practical aspects of ADFG. On the theoretical side, it presents corrections and improvements on the fixed priority case. On the practical side, benchmark evaluations demonstrate the robustness and maturity of the approach that our scheduling synthesizer implements. Synthesized schedules are evaluated by using scheduling simulation and real-time implementation. Last but not least, the synthesized periods reach the optimal throughput if enough processors are available, and most of the time the periods reach the maximal processor utilization factor in the uni-processor case. Moreover, execution time of the synthesis is about only one second for the main proposed algorithms.

CCS Concepts: • **Computer systems organization → Real-time system specification**.

Additional Key Words and Phrases: real-time systems, periodic scheduling, fixed priority, dataflow

ACM Reference Format:
Alexandre Honorat, Hai Nam Tran, Thierry Gautier, Loïc Besnard, Shuvra S. Bhattacharyya, and Jean-Pierre Talpin. 2023. Real-Time Fixed Priority Scheduling Synthesis using Affine DataFlow Graphs: from Theory to Practice. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2023), 31 pages. https://doi.org/10.1145/3615586

Authors' addresses: Alexandre Honorat, Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, France, alexandre.honorat@inria.fr; Hai Nam Tran, University of Brest, CNRS, Lab-STICC - UMR 6285, France, hai-nam.tran@univ-brest.fr; Thierry Gautier, INRIA Research Centre at the Univ. of Rennes, France, thierry.gautier@inria.fr; Loïc Besnard, CNRS, Univ. of Rennes Research Centre, France, loic.besnard@irisa.fr; Shuvra S. Bhattacharyya, University of Maryland, College Park, USA; and INSA/IETR, Rennes, France; and INRIA, Rennes, France, ssb@umd.edu; Jean-Pierre Talpin, INRIA Research Centre at the Univ. of Rennes, France, jean-pierre.talpin@inria.fr.

1539-9087/2023/1-ART1 $15.00
https://doi.org/10.1145/3615586

1 INTRODUCTION

The dataflow programming paradigm is widely used in embedded system design to describe Digital Signal Processing (DSP) control [49] and stream processing [77] applications. Synchronous DataFlow (SDF) graphs [48], for instance, provide automated code generation services to limit the most problematic and error-prone tasks of programming parallel applications. However, DSP applications are usually subject to additional constraints, such as throughput requirements or limited available memory, which also need to be tackled.

Constraints on throughput or memory must be respected by the application *schedule*. While many schedule types exist, some are easier to analyze in order to assert that all constraints are respected. Assertions are easily checked on static schedules but on the other side static schedules eventually require a high memory to store all execution times. Periodic schedules are a trade-off often used in real-time systems: efficient algorithms exist to assert their schedulability [2, 43] and they have a limited memory cost. Periodic scheduling is then divided in two complementary problems: first to compute the periods of tasks, and second to check that all constraints are respected. In periodic real-time systems, constraints also include the fact that every task meets its periodic deadline. Although periodic schedules are still hard to synthesize in the general case, dataflow graphs provide a programming abstraction well adapted to study throughput and memory constraints under periodic real-time scheduling.

In this article, we describe and evaluate a method to solve the two aforementioned problems at once: period synthesis and constraints assessment. Our method uses several abstractions and Integer Linear Programming (ILP) formulations, and thus it is still limited by the NP-hard complexity of ILP, just as in the general case of task scheduling [47]. However, and despite this computational complexity limit, we demonstrate that our method runs in about one second for a scheduling problem with hundred tasks. In contrast to many scheduling approaches, the Affine DataFlow Graph (ADFG) tool [14] synthesizes near-optimal and always valid periods of the real-time tasks to execute, without any simulation of the complete schedule. We say that the ADFG tool *synthesizes a schedule*, which actually means that it computes all needed characteristics of the tasks to schedule: period, priority, mapping… The periods are *near-optimal* in the sense that they are computed with the objective to maximize the throughput of the scheduled application; they are *valid* since the schedulability of the system is guaranteed by formal methods adapted from synchronous languages [72]. Regarding optimality, the ADFG tool does *not* guarantee throughput optimality especially if also considering the objective to minimize the total buffer size; nevertheless the experiments show that the optimal periods are often reached. Regarding validity, the ADFG tool guarantees schedulability in the considered periodic execution model, but other timing metrics such as preemption costs are *not* taken into account in the ADFG synthesizer itself.

Yet, the scheduling can be highly impacted by extra factors such as preemption cost. So we also present how the ADFG tool can be used in conjunction with other software to simulate and to execute an application, for example in order to derive the amount of preemption needed. Moreover, we highlight specific details that a designer has to be aware of to make effective use of the scheduler. Simulations reveal in a few cases both pessimistic or optimistic behaviors, that a designer has to deal with when using the ADFG tool. Pessimistic schedules may occur due to the Worst Case Execution Times (WCETs) of actors in Cyclo-Static DataFlow (CSDF) graphs [9], while optimistic schedules may occur due to underestimated preemption costs. In any case, the synthesized schedules are inherently valid in the model of ADFG theory.

Concretely, given a Ultimately Cyclo-Static DataFlow (UCSDF) graph to execute in a periodic real-time environment with known WCETs, the following schedule characteristics are computed by the ADFG tool: the task priorities and periods, the mapping, the number of delays in the buffers,

and the buffer sizes. The application to schedule is modeled thanks to an UCSDF graph. The UCSDF model is an extension of the CSDF model, being itself an extension of SDF. The tasks are periodic with implicit deadline; they are scheduled by a Fixed Priority (FP) preemptive partitioned scheduler. Synthesis algorithms are available for the uni-processor and the multi-processor cases; in the multi-processor case, the number of homogeneous processors is given as an input by the designer. Most of the provided algorithms try to maximize the throughput of the scheduled application, although two of them allows to find trade-offs between throughput and total buffer size. Numerous aspects of the ADFG theory have already been published over the past 12 years: while it started with affine equations for synchronous languages [72], the main theory has been formalized in the Bouakaz' PhD thesis [14] for both FP scheduling [15, 18, 19] and Earliest Deadline First (EDF) scheduling [16, 17, 40]. As a tool, ADFG has been used to validate a subset of Architecture Analysis and Design Language (AADL) [7, 32] and to generate code compatible with the Real-Time Executive for Multiprocessor Systems (RTEMS) real-time operating system [79]. This article is the first to provide a general overview of both theoretical and practical aspects (up to the execution of the synthesized code) and it describes and evaluates so far unpublished details of the synthesis algorithms. In a decade a few bugs have been fixed, and updated evaluations still show the efficiency of our approach.

The article is organized as follows. Next Section 2 introduces related work. Section 3 recalls the main concepts of SDF graphs and FP scheduling. Then the Section 4 provides an overview of the ADFG theory and the UCSDF graph model; and previously unpublished contributions are also detailed. Regarding practice, the generation of the scheduling code of an application thanks to the results provided by the ADFG tool is explained in Section 5. The evaluation of the schedules, by metrics obtained directly or from simulations, is presented in Section 6 and precedes closing remarks.

2 RELATED WORK

The ADFG tool computes the periods of actors in a UCSDF graph, ensuring that all implicit deadlines will be met, and that there is no buffer underflow or overflow when executing the graph. Thus the ADFG tool can be seen as both an analysis and a synthesis tool. Both categories of tools, dedicated to SDF or CSDF graphs, are reviewed in this section.

Many other models than SDF exist to check real-time properties but, to our knowledge, those models make the analysis and especially the synthesis of buffer sizes more tedious. For example, a real-time system implementation using Timed Automata may need one clock per buffer slot [76], manually set for each possible buffer size. Conversely, the ADFG theory is not adapted to dynamic real-time systems where, for example, some tasks could be enabled/disabled by other tasks results.

2.1 Generic analysis of SDF graphs

SDF graphs and their extensions, as CSDF [9], have been widely analyzed on several aspects including: latency [34, 44, 60], throughput [25, 33], buffer size [35, 83], or liveness [57]. These analyses often relax the available processors constraint (their number is considered as potentially infinite) and use most often either static periodic scheduling (for [12, 35, 80]) or self-timed scheduling[1] (for [33, 60, 83]). Several combinations of the aforementioned aspects have been studied, such as the minimization of the total buffer size under a throughput constraint [5, 12]. For instance the SDF3 tool [74] provides trade-offs analysis between buffer size and throughput for SDF graphs [73] and CSDF graphs [75]. The authors of SDF3 also proved that both metrics, throughput and buffer

[1]In self-timed scheduling, each task is fired whenever available resources allow it, regarding to the consumed and produced tokens to avoid buffer underflows and overflows, and regarding to the processors availability if not unlimited.

size, cannot be optimized at the same time. However most of these analytic methods are purely symbolic and do not compute a real-time schedule, including mapping and periods, or the number of delays of the graph.

Besides, many tools are able to analyze SDF graphs, to derive a few schedule characteristics (e.g. mapping and buffer size), and finally to generate the glue code of the schedule automatically: DIF-GPU [53], PREESM [67], LabVIEW [1], MAPS [21], Diplomat [13], Gaspard [31], Ocarina [42], PeaCE [37], and Ptolemy [29][2] which includes the Metronomy tool [36] for timing verification. But these tools either do not consider real-time executions (e.g. PREESM and LabVIEW), or do not perform all the syntheses automatically (e.g. delay or periods are an input of the user). For example, periods usually are an input of the problem [39, 45], instead of the output. Moreover, most tools computing a schedule consider only one optimization objective, usually throughput maximization. Periodic scheduling for throughput maximization [65] has been studied pretty early by Parhi and Messerschmitt, and it is possible to derive the period achieving best throughput [4], but not taking into account FP scheduling, nor preemption, nor the number of processors available.

2.2 Real-Time Schedule Synthesis of SDF graphs

To our knowledge, only the DARTS [3] and ADFG [14, 40] tools consider buffer size minimization and throughput maximization at once, while being able to derive the periods of CSDF graph. Whereas the DARTS tool considers unlimited available processors, the ADFG tool takes the number of processors as an input. Other methods synthesize real-time characteristics such as periods [46], but not for SDF graphs.

The DARTS [3][3] tool allows to compute the strictly periodic schedule characteristics achieving the best throughput under EDF or Rate Monotonic (RM) policies, with a maximum total buffer size as a constraint. The DARTS tool considers only acyclic CSDF graphs and a non-constrained number of available processors on the target system. The DARTS and ADFG tools differ in the input constraints: ADFG accepts cyclic and acyclic UCSDF graphs, constrained and non-constrained buffer sizes, and it requires the number of available processors in the target system. Indeed, throughput maximization and buffer size minimization are two antagonistic objectives [75] and rather than solving a difficult multi-objective optimization, both the ADFG and DARTS tools constrain one of the objectives and then optimize the other. Constrained objective of the DARTS tool is the total buffer size fixed by the user; buffers are used as much as possible in order to maximize the throughput first, and then the DARTS tool minimizes the number of processors. Constrained objective of the ADFG tool is the number of available processors fixed by the user; processors are used as much as possible in order to maximize the throughput first, and then the ADFG tool minimizes the total buffer size. Furthermore, the ADFG tool provides extra trade-offs between buffer size and throughput objectives thanks to its internal algorithms SP_UNI_UDH and SP_MULT_MBS.

Like ADFG, several tools use ILP formulations to synthesize the buffer sizes. Previous ILP formulations for CSDF graphs have been made in order to maximize the throughput and minimize the total buffer size: under self-timed scheduling [83], under static periodic scheduling with maximum number of processors constraint [80], or under static periodic scheduling with minimum throughput constraint [5], but not under EDF and FP scheduling. Also, a synthesis tool has been developed [28] for a mix of the periodic and self-timed scheduling policies, with priorities. Finally, buffer size minimization can be improved when data access pattern of the SDF actors are known [81].

[2] https://ptolemy.berkeley.edu/
[3] https://daedalus.liacs.nl/daedalus-rt.html

3 BACKGROUND

Common knowledge and notations of this article, about the SDF graph model and about FP scheduling, are reviewed in the following Section 3.1 and Section 3.2, respectively.

3.1 Synchronous DataFlow

A Synchronous DataFlow (SDF) graph [48] is a directed multi-graph $G = (V, E)$ consisting of a finite set of *actors* $V = \{v_1, ..., v_N\}$ and a finite set of one-to-one *buffers* E. A buffer $e_{1\to 2} = (v_1, v_2, p, q) \in E$ connects the data producer v_1 to the data consumer v_2 such that the production (resp. consumption) rate is given by an integer $p \in \mathbb{N}^*$ (resp. $q \in \mathbb{N}^*$). The data unit is called a *token*. Every time an actor is *fired* (i.e. is executed), it consumes as many tokens as specified by each input buffer connected to it, and produces as many tokens as specified by each output buffer connected to it. SDF graphs actually are a subset of weighted Petri nets, with only one incoming and one outgoing transition (i.e. actor) per place (i.e. buffer). Figure 1 depicts a simple SDF graph with 3 actors. Figure 2 depicts the h263decoder which is taken from a set of real applications provided by the SDF3 benchmark [74].

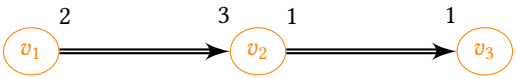


Fig. 1. A simple acyclic delayless SDF graph.

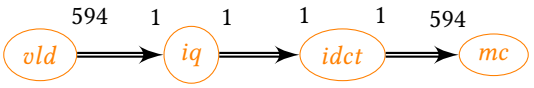


Fig. 2. SDF representation of h263decoder [74].

When implementing an SDF graph, it is common to impose statically-determined buffer sizes — that is denoted $\delta(e_{a\to b})$ for a buffer $e_{a\to b}$. If some buffer sizes are too small, the SDF graph could deadlock. For example, with the graph in Figure 1, the buffer $e_{1\to 2}$ connecting v_1 to v_2 needs a minimum size of 4. The reason is that v_1 must be fired at least 2 times before v_2 has enough input tokens to be fired. There can be a number of initial tokens present on each buffer. These initial tokens are called the *delays* of a buffer since they can induce an offset in the execution of the consumer in relation to the producer [8]. When scheduling an SDF graph, the buffer sizes are always bounded since the production and consumption rates are fixed, however *consistency* checks are needed when multiple buffers exist between two actors, and also for the undirected cycles in the graph (not detailed here). If the graph is *consistent*, it is possible to compute a *repetition vector*, defined in the following Definition 3.1. In this article, we study only consistent and weakly connected[4] graphs.

Definition 3.1 (Repetition Vector [8]). *The repetition vector of an SDF graph is an array of length equal to the number of actors in the graph, such that each element of the array is a positive integer, the greatest common divisor of all of the elements is* 1*, and if each actor is invoked for the number of times equal to its entry, then the number of tokens on each edge of the graph remains unchanged.*

[4]A directed graph is said to be weakly connected when its underlying undirected graph is connected.

When each actor has been fired as many times as required by the repetition vector, this defines a graph *iteration*, meaning that the whole application has been executed once. For the graph in Figure 1, the repetition vector is $\vec{z} = [3, 2, 2]$; actors v_2 and v_3 are fired the same number of times since the buffer $e_{2\to3}$ connecting them has a consumption rate equal to its production rate. The repetition vector can be computed efficiently by applying a depth-first search algorithm [8].

3.2 Fixed Priority Scheduling of Periodic Tasks

In this article, only partitioned preemptive fixed priority scheduling of asynchronous periodic tasks is considered[5]. Each SDF actor corresponds to a hard real-time periodic task with implicit deadline. The priority P_v of each actor v is unique and fixed over time; $P_v \in \mathbb{N}^*$ and the value 1 corresponds to the highest priority level. Auto-concurrency is not allowed: two firings of an actor cannot occur in parallel. Moreover, an actor v is permanently mapped to a single processor M_v on homogeneous multi-processor architectures. Extension of the developments of this article to more relaxed versions of the scheduling problem (e.g., considering heterogenous architectures) is a useful direction for future work.

The period of an actor v is denoted $T_v \in \mathbb{N}^*$, as in the model of Liu and Layland [54]. This period must be greater or equal to the WCET of v, denoted $C_v \in \mathbb{N}^*$. Each actor v is *released* every T_v time units and it must be executed completely before the next release; otherwise the implicit hard deadline is missed and the schedule is not valid. Each actor v may be initially released with an offset $O_v \in \mathbb{N}$; the offset is non negative and possibly greater than the period T_v. When an actor v is currently running, it is preempted by any released actor having a higher priority and mapped on the same processor; v resumes only when the actors of higher priority all finished their execution. The preemption overhead is not modeled in our work.

Finally, the processor utilization factor U is defined as follows in Equation (1); it must be lower than the number of processors m in the targeted homogeneous multi-processor [41].

$$U = \sum_{v \in V} \frac{C_v}{T_v} \leq m \tag{1}$$

4 FIXED PRIORITY SCHEDULING FROM AFFINE DATAFLOW GRAPHS

Before seeing the details of the theory, we recall the notations and depict the periodic scheduling thanks to the following example. Considering the Synchronous DataFlow (SDF) graph in Figure 1, and given the Worst Case Execution Times (WCETs) $C_{v_1} = 10$, $C_{v_2} = 6$ and $C_{v_3} = 7$, the Affine DataFlow Graph (ADFG) tool computes the smallest possible actor periods to ensure safety of the execution. For example, if the targeted system has 2 processors, the ADFG tool finds the periods $T_{v_1} = 10$, $T_{v_2} = 15$, $T_{v_3} = 15$ and offsets $O_{v_1} = 0$, $O_{v_2} = 20$, $O_{v_3} = 20$. Actor v_1 is mapped alone on the first processor, for a total processor utilization factor of $U = 1.87$ (over $m = 2$). Part of the resulting schedule is depicted in Figure 3: the two dashed red lines delimit the indefinitely repeated schedule, omitting the first two firings of v_1. Actors v_2 and v_3 are released at the same time but the ADFG tool assigns a higher priority to v_2 and thus v_2 is executed first.

The ADFG tool provides several algorithms to compute the schedule characteristics. Table 1 summarizes the available algorithms for uni-processor systems; the algorithms differ by the priority assignment computation. Table 2 summarizes the available algorithms for multi-processor systems; the algorithms differ by the mapping computation but all use the same priority assignment as SP_UNI in Table 1. In order to understand how these algorithms work, an overview of the ADFG tool is presented in Section 4.1; so far unpublished contributions, as priority assignment with

[5]However, the ADFG tool also supports various EDF scheduling policies [40].

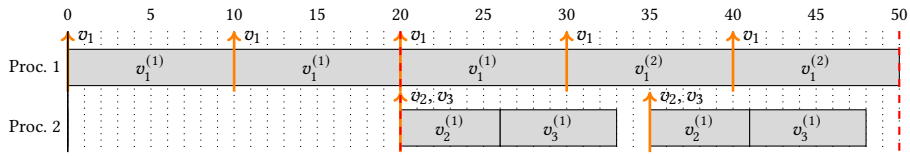


Fig. 3. Real-time schedule synthesized by ADFG tool for the input graph in Figure 1 without delays, considering two processors.

graph topological ordering, are detailed in Section 4.2. Actor mapping, or more precisely *graph partitioning*, is not addressed in this article, interested readers can refer to [40] (section 5.2 in ref.).

Name	Priority assignment
SP_UNI	Deadline Monotonic with graph topological ordering (see Section 4.2.1)
SP_UNI_LOP	Priorities assigned by ILP, to minimize total buffer size (see Section 4.2.1)
SP_UNI_UDH	Priorities assigned by combinatorial search with utilization distance heuristic (see Section 4.2.1), to minimize total buffer size

Table 1. Uni-processor FP algorithms.

Name	Mapping	Complementary information
SP_MULT_MBS	SCOTCH [68]	Offers throughput *vs* total buffer size trade-offs [40]
SP_MULT_BF_FBBFFD	Best Fit + FFD	Uses *demand bound function* [30] for the mapping
SP_MULT_BF_SRTA	Best Fit + SRTA	Used for Figure 3, SRTA is detailed in Section 4.2.2

Table 2. Multi-processor FP algorithms.

4.1 ADFG theory overview

To synthesize preemptive periodic schedules under Fixed Priority (FP) policy, the ADFG tool takes as input a weakly connected Ultimately Cyclo-Static DataFlow (UCSDF) graph with WCET of each actor, the number of available identical processors in the target system, and the solving method. The output of the ADFG tool includes the mapping (without auto-concurrency), the period and the priority of each actor, the number of delays and each buffer size, and finally, several metrics as the throughput or the processor utilization factor. This Section 4.1 recalls the workflow of the ADFG tool to compute the aforementioned schedule characteristics, with the main equations already presented in prior work and summed up in [40]. Hence, no proof is provided here.

4.1.1 Schedule abstraction. Affine DataFlow Graph (ADFG), the name of the tool, stands for the internal representation of the UCSDF, Cyclo-Static DataFlow (CSDF), and SDF graphs. In UCSDF graphs, the actors send data through buffers as in SDF and CSDF graphs, and Definition 3.1 for the repetition vector is the same. The difference between these three models lies in the production and consumption rates: static in SDF (one integer), cyclo-static in CSDF (a cyclic sequence of integers), and ultimately cyclo-static in UCSDF (two sequences of integers, the first being used only once). During its i-th firing, an actor receives and sends an amount of data equal to the i-th value of its

consumption/production rates; the cyclic sequence is iterated in a round-robin fashion. CSDF and UCSDF graphs can always be converted into SDF graphs [66] but at the cost of introducing many new nodes. In the graph depicted in Figure 4a, the production rate is of UCSDF form while the consumption rate can be seen as either CSDF or UCSDF form, the latter with an empty first part. Previous work on synchronous languages [72] made it possible to retrieve an affine function to bound the buffer sizes [19], thanks to an abstraction of the production and consumption rates. In this abstraction, exact time is not considered: the schedule abstracts events on buffers over time rather than time itself.

In the ADFG tool, each buffer is abstracted by an *affine relation* on the two *firing clocks* of the producer and the consumer actors. The firing clock is detailed in the next paragraph. In a buffer $e_{1\to 2} = (v_1, v_2, p, q)$, the production and consumption rates p and q are the concatenation of two sequences of integers: $p = p_0.p_c$ and $q = q_0.q_c$. The first *initial* sequence with 0 subscript may be empty, then symbolized by ϵ. The second non empty sequence, *cyclic* and so identified by c subscript in the equations, will be repeated indefinitely during the execution of the producer/consumer actor. The cyclic sequence is written inside parenthesis in Figure 4a and Figure 8. The two sequences of integers associated with a computed average production or consumption rate will help to define the affine relation. The average production (respectively consumption) rate $a_p \in \mathbb{Q}^+$ (respectively a_q) per firing is defined as follows in Equation (2). In an SDF graph, average rate is the only integer of the sequence; in CSDF and UCSDF graphs, the sum of the cyclic sequence is averaged by its length. Let us first denote $|s|$ the length of a sequence s, and $s(i)$ the i-th value of this sequence; if $i > |s|$, i is reassigned to $((i - |s_0|) \text{ modulo } |s_c|) + |s_0|$. The cyclic sequence sum is defined thanks to the $\oplus$ operator, that is the cumulative sum over a given length, as defined in the right part of Equation (2).

$$a_p = \frac{||p_c||}{|p_c|}, \quad \text{with} \quad ||p_c|| = \oplus p_c(|p_c|) = \sum_{1 \le i \le |p_c|} p_c(i) \tag{2}$$

While Equation (2) constitutes the linear part of the data production, the affine part comes from the following lower bound $\lambda^l \in \mathbb{Z}$ and upper bound $\lambda^u \in \mathbb{Z}$ defined in Equation (3). In an SDF graph, those two bounds λ^l_p and λ^u_p are equal to 0 because $p_0 = \epsilon$.

$$\lambda^l_p = \min_{1 \le i \le |p_0|+|p_c|} \left\{ \oplus p(i) - a_p i \right\} \quad \text{and} \quad \lambda^u_p = \max_{1 \le i \le |p_0|+|p_c|} \left\{ \oplus p(i) - a_p i \right\} \tag{3}$$

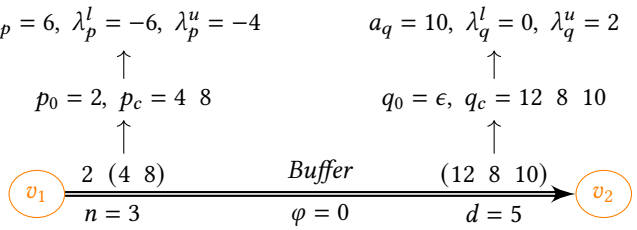
Then, the amount of data produced at the i-th firing (i without modulo reassignment) of an actor v_1 is bounded thanks to the inequalities in Equation (4).

$$\lambda^l_p + a_p i \le \oplus x(i) \le \lambda^u_p + a_p i \tag{4}$$

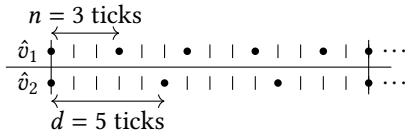
Firings themselves are abstracted over time by *firing clocks* which are related for a producer actor and its consumer: an actor firing will occur every n ticks for the producer v_1, and every d ticks for the consumer v_2. Integers n and d are related such that $\frac{n}{d}$ is the irreducible fraction of $\frac{a_p}{a_q}$. Besides, the integer $\varphi \in \mathbb{Z}$ denotes the firing offset between the actors v_1 and v_2, i.e. there are $\varphi_{1\to 2}$ ticks between the first firing of v_1 and the first firing of v_2.

Finally, an affine relation is defined by a triple of integers (n, φ, d). Contrary to n and d, φ may be negative. An example of all the aforementioned metrics is given in Figure 4 with two actors connected by a single buffer. Figure 4a details the average rate and the bounds of both actor rates; Figure 4b depicts their firing clocks.

Then the two affine Equations (5) and (6) enable to compute the buffer sizes δ_e, the delays θ_e, and all firing offsets φ thanks to an ILP solver where the objective is to minimize the sum of all buffer sizes and firing offsets. However, as this ILP formulation only concerns the firing clocks



(a) Buffer between v_1 and v_2, its affine metrics are above the buffer arrow and its derived affine relation is below it.



(b) Firing clock representation over $\text{lcm}(n, d) = 15$ ticks, the two tasks start on the same tick because $\varphi = 0$.

Fig. 4. Affine metrics and corresponding task firing clocks.

linked by affine relations, there is no concrete time and the minimization objective on buffer sizes and firing offsets has no impact on the final throughput. Moreover, Equations (5) and (6) of the ILP formulation ensure the maximal throughput according to a previously computed priority assignment and actor mapping. The ILP formulation also contains specific equations for cycles not specified here, interested readers can refer to [14] (section 2.3.2 in ref.).

$$\theta_{e_{1\to2}} + a_p \frac{\varphi_{1\to2}}{n} \geq \lambda_q^u - \lambda_p^l + a_p\, C_{\text{under}} \tag{5}$$

$$\theta_{e_{1\to2}} + a_q \frac{\varphi_{1\to2}}{d} \leq \delta_{e_{1\to2}} + \lambda_q^l - \lambda_p^u - a_q\, C_{\text{over}} \tag{6}$$

As Equations (5) and (6) contain integer and rational numbers, every term is multiplied by the least common multiplier of all denominators when generating the ILP formulation; this avoids the floating point imprecision. C_{under} and C_{over} are two coefficients depending on the scheduling policy and other characteristics as the producer and consumer priorities or the fact that v_1 and v_2 are mapped on the same processor or not [15] (end of section IV in ref.); their values are given later in Equation (15). Actor mapping is not described in this article, however, priority assignment will be discussed further, in Section 4.2.1.

The solution computed by the ILP solver closes the abstraction part of the ADFG theory, which is then followed by the synthesis part in order to retrieve the periods of each actor.

4.1.2 Schedule synthesis. The goal of the schedule synthesis part is to compute the effective periods and offsets of the actors. Offset and periods are computed in the same time unit as the WCETs inputs. First, note that the input graph is weakly connected, so defining a period on one actor induces a period for all other actors; thus only one *basis period* is considered during the synthesis. Indeed, a producer actor must produce in average the same amount of data per time unit as the consumers connected to it consume. This assumption is materialized into Equation (7) thanks to the affine relations between any pair of actors connected by a buffer $e_{1\to2}$.

$$d \cdot T_{v_1} = n \cdot T_{v_2} \tag{7}$$

Regular offsets of the actors are materialized in Equation (8), depending on the firing offsets φ.

$$O_{v_2} - O_{v_1} = \frac{\varphi_{1\to 2}}{n} T_{v_1} \tag{8}$$

Equation (7) induces a linear relation between any actor v and an arbitrary actor basis of the weakly connected graph, see Equation (9) with $\alpha_{\text{basis}} = 1$.

$$T_v = \alpha_v T_{\text{basis}}, \quad \alpha \in \mathbb{Q}^+ \tag{9}$$

This linear relation allows performing standard Response Time Analysis (RTA) [2, 43] on the actors, by setting only one period T_{basis}. Also, the processor utilization factor U can be rewritten with this single period T_{basis}, as formalized in Equation (10).

$$U = \sum \frac{C_v}{\alpha_v T_{\text{basis}}} = \frac{\sigma}{T_{\text{basis}}}, \quad \text{with} \quad \sigma = \sum \frac{C_v}{\alpha_v} \tag{10}$$

Equation (1) can be rewritten as Equation (11) and gives a minimal value of T_{basis}.

$$T_{\text{basis}} \geq \frac{\sigma}{m} \tag{11}$$

Then, thanks to similar rewriting done on the lower and upper bounds of the response time [10, 71], a bisection search is performed on T_{basis} with the RTA. The minimal period ensuring schedulability according to RTA is denoted T^l_{basis}. This point is discussed further in Section 4.2.2. As basis period and processor utilization are inversely related in Equation (10), throughput maximization corresponds to processor utilization factor maximization; so the minimal period found by the bisection search ensures the maximum throughput reachable with the given number of available processors m and according to the previously computed priority assignment and actor mapping.

Finally, it is possible to compute all periods and offsets thanks to a second ILP formulation with the goal of minimizing T_{basis}. The ILP solver takes as inputs Equations (8), (9) and (12).

$$\max\left(\frac{\sigma}{m}, T^l_{\text{basis}}\right) \leq T_{\text{basis}} \tag{12}$$

4.1.3 Workflow of the ADFG tool. Multiple algorithms can be used to compute the final schedule and mapping, for both uni-processor or multi-processor. All implemented periodic scheduling synthesis algorithms respect the following steps:

(1) decompose the graph: compute all firing relations (with undefined φ);
(2) assign the priorities: see Section 4.2.1;
(3) partition tasks across available processors: this step may call internally step 5;
(4) compute all φ: thanks to an ILP formulation;
(5) perform symbolic synthesis: thanks to RTA, see Section 4.2.2;
(6) compute all task periods and offsets: thanks to an ILP formulation;
(7) refine buffer delay and buffer size computation: see Section 4.2.3.

Figure 5 details which schedule characteristics are computed by each of those steps, along with the corresponding objective. Steps (2) and (3) offer trade-offs between the throughput maximization and the total buffer size minimization objectives, according to the selected algorithm in Tables 1 and 2 and the input parameters. For steps (4) and (6), respectively, when the ILP problem is not feasible or when the basis period is too large, the algorithms automatically state that the system is not schedulable. Additionally, the ADFG tool computes a few metrics such as the throughput Θ, this point is discussed in Section 4.2.4.

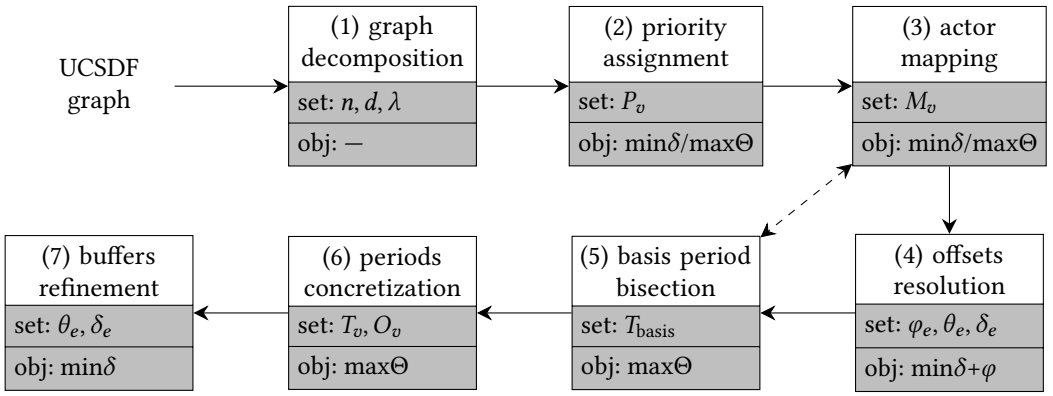


Fig. 5. Main steps of the ADFG tool: from input graph to concrete periods and buffer sizes, with intermediate results and objectives.

The next Section 4.2 details a few aspects of the computation, including new contributions.

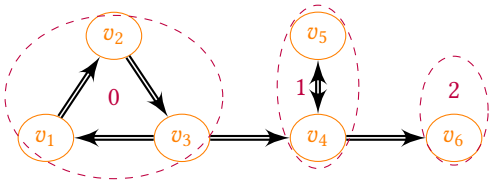
4.2 ADFG in details

Multiple improvements are presented in this section, they impact on different key points of the ADFG theory: the computation of priorities, of periods, of delays, and of metrics. All those improvements increase the accuracy or the usability of the ADFG tool and were unpublished so far.

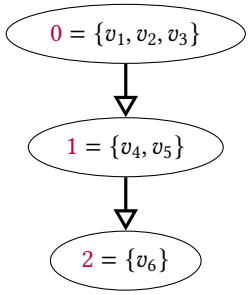
4.2.1 Priority assignment, step (2). Regarding priorities, the first improvement is a heuristic to break the tie when two actors have the same deadline in the SP_UNI algorithm (see Table 1) and in all multi-processor schedule synthesis algorithms (see Table 2). The second improvement enables to find trade-offs between total buffer sizes and throughput in the SP_UNI_UDH algorithm. We also briefly recall how the SP_UNI_LOP algorithm assigns priorities, as a pretext to introduce the buffer size approximation derived from Equations (5) and (6), which have been slightly modified. While the first improvement only aims at decreasing the actor offsets and delays on buffers, the two other priority assignment algorithms proposed by the ADFG tool try to minimize the buffer sizes.

Breaking the tie in Deadline Monotonic. The heuristic to break the tie with the Deadline or Rate Monotonic priority assignment is based on a graph topological ordering. When two actors have the same relative deadline or period, respectively, we assign different priorities for each of them, according to their appearance order in the topological ordering. With this heuristic, a producer actor has a higher priority than all its consumers having the same deadline or period. Thus, this heuristic avoids the need for actor offsets or delays since the producer is always executed before its consumers thanks to its higher priority. This phenomenon has been reproduced experimentally: all test applications having only actors with equal periods have neither delays nor offset when using this improvement on uni-processor, whereas some appear when not using the improvement. However, the graph topological order is not computed on the original SDF graph, but on its reduction to a Directed Acyclic Graph (DAG) of strongly connected components, as depicted in Figure 6. This means that the heuristic does not break the tie between two actors having the same deadline when both are present in the same strongly connected component (i.e. directed cycle): at least one actor has to start the cycle execution, whatever is its priority.

In the context of implicit deadlines, the SP_UNI algorithm assigns increasing priorities to the actors with increasing periods and the heuristic is used only when two actors have the same period.



(a) Original directed graph with three strongly connected components circled in dashed. The left one has no incoming edges so it has the smallest topological ordering number 0, the one having no outgoing edges has the greatest number, here 2.



(b) Induced directed acyclic graph (DAG), each node corresponding to a strongly connected component of the graph on the left.

Fig. 6. Graph topological order for priority assignment.

Note that the priority assignment is performed before the period synthesis: this is possible since the affine relations between the actors already hold the information of the relative firing clock speed of the actors, see Figure 4. The real periods do not matter for the priority assignment, only the natural ordering of the periods is important.

Configurable Utilization Distance Heuristic. The second improvement of the ADFG tool regarding priority assignment is based on an algorithm of variable complexity, configurable by the designer, called *Utilization Distance Heuristic*. A first version of this algorithm has been presented in the original thesis on ADFG theory [14] (pages 96-97 in ref.) but was not configurable; it is used by the SP_UNI_UDH algorithm. In this article, we explain how this algorithm can be configured by the designer. The SP_UNI_UDH algorithm relies on a first priority assignment (Deadline Monotonic) and period synthesis (see Section 4.2.2), and then its heuristic is to perform permutations on the priorities of actors in order to select the priority assignment minimizing the total buffer size. However, these permutations may increase the periods of actors since the response time of some actors will inevitably be increased (see the following Section 4.2.2).

The new implementation allows the designer to configure an acceptable trade-off between the buffer size decrease and the period increase. We redefine here the utilization distance in terms of basis period and relative to the best one, T^{DM}_{basis}, obtained with Deadline Monotonic priority assignment. It is called *utilization* distance since the throughput (inversely proportional to the basis period, see Section 4.2.4) is linear to the processor utilization factor, see Equation (10). Then the utilization distance $dist(p) \in [0, \infty[$ gives a measure of throughput loss for $T_{basis}(p)$ obtained with the permutation p, see Equation (13). $T_{basis}(p)$ is over-estimated for each permutation with a linear complexity formula thanks to the actor response times (detailed in [14] pages 96-97). Now a 100% throughput loss, i.e. $dist(p) = 1$, means a throughput division by 2.

$$dist(p) = \frac{T_{basis}(p)}{T^{DM}_{basis}} - 1 \tag{13}$$

In the ADFG tool, it is now possible to specify a maximum percentage loss in throughput (i.e. period increase) and a minimum percentage gain in total buffer size minimization: a priority permutation is selected only if the throughput loss is less than the maximum and the buffer size gain greater than the minimum specified. To reduce the run time of the algorithm, it is also possible to set the permutation length ρ; indeed the total number of permutations explodes with their length: it is $\rho!$. If the permutation length is less than the number of actors, SP_UNI_UDH will

try all priority permutations of the specified length in a sliding window iterating over the actors ordered by increasing Deadline Monotonic priorities. The factorial complexity of this algorithm may limit its use; however note that the optimal priority assignment of real-time tasks with offsets on homogeneous multi-processor also has a restrictive complexity: it is NP-hard [51].

A new approximation of buffer sizes. The SP_UNI_LOP algorithm also selects the best priority assignment minimizing the total buffer size, but not taking into account the throughput loss. Its complexity is theoretically less than SP_UNI_UDH but not configurable, since it relies on the minimization by an ILP solver of a tournament matrix weighted by approximated buffer sizes [18] (section 5.1 in ref.). Both SP_UNI_LOP and SP_UNI_UDH algorithms use the same approximation of buffer sizes formalized in Equation (14)[6]. This approximation is an underestimation, obtained from Equations (5) and (6).

$$\delta_{e_{1\to2}} \geq \lambda + a_p\, C_{\text{under}} + a_q\, C_{\text{over}}, \quad \text{with} \quad \lambda = (\lambda_q^u - \lambda_p^l) + (\lambda_p^u - \lambda_q^l) \tag{14}$$

The values of the coefficients C_{over} and C_{under} are formalized in Equation (15); the optional +1 depends on the priorities and mapping of the producer actor v_1 and the consumer actor v_2 of the buffer $\delta_{e_{1\to2}}$, see [15] (section IV in ref.) for more details.

$$C_{\text{under}} = \frac{d-1}{n} \quad (+1) \qquad \text{and} \qquad C_{\text{over}} = \frac{n-1}{d} \quad (+1) \tag{15}$$

Although SP_UNI_LOP and SP_UNI_UDH algorithms are dedicated to the uni-processor case, those coefficients are also used in the ILP formulation computing all firing offsets φ, which is generic for both uni- and multi-processor cases.

Equation (14) is approximated by removing the -1 term in the numerator of the fractions, and by considering the inequality as an equality, see Equation (16). Moreover, $\frac{n}{d}$ is the irreducible fraction of $\frac{a_p}{a_q}$ per definition. So this approximation limits integer explosion in the resulting coefficient rational number.

$$\delta_{e_{1\to2}} \approx \lambda + a_p + a_q \quad (+a_p) \quad (+a_q) \tag{16}$$

Finally, the approximation of the buffer size used by the SP_UNI_LOP and SP_UNI_UDH priority assignment algorithms is summarized in Table 3[7].

	$M_{v_1} = M_{v_2}$	$M_{v_1} \neq M_{v_2}$
$P_{v_1} < P_{v_2}$	$\lambda + a_p + 2a_q$	$\lambda + 2a_p + 2a_q$
$P_{v_1} > P_{v_2}$	$\lambda + 2a_p + a_q$	$\lambda + 2a_p + 2a_q$

Table 3. Buffer size approximation for FP scheduling. M_v is the processor number of an actor v and P_v is its priority.

Many other priority assignment algorithms exist [24], as robust priority assignment [23] or assignment with threshold to limit the number of preemptions [82]. Unfortunately these two algorithms are not suitable for the ADFG tool since they necessitate the actor periods which are not known yet (only linear relation between the periods are known during this step).

To conclude this subsection, note that there are as many priority levels as actors, which may be a constraint for some implementations. In the Real-Time Executive for Multiprocessor Systems

[6] A bug in the implementation of the approximation was affecting both SP_UNI_LOP and SP_UNI_UDH in a previous release of the ADFG tool: consumer and producer actors were exchanged in a function call. The bug has been fixed for the evaluation of the present article.

[7] Because of the new approximation in Equation (16), this table is slightly different than the original one in [14] (table 2.1): all $\frac{-2}{d}$ terms are not present anymore.

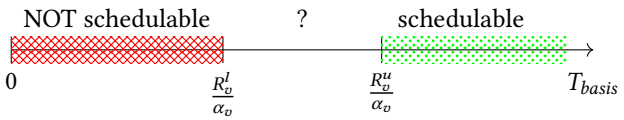


Fig. 7. Initial scheduling bounds on the basis period T_{basis} based on the response time of actor v. Considering all actors, lower bound is $\max_{v \in V} \frac{R_v^l}{\alpha_v}$ and upper bound is $\max_{v \in V} \frac{R_v^u}{\alpha_v}$.

(RTEMS) operating systems the number of tasks is unlimited, but the number of priority levels is limited[8] to 255. Thus, in the ADFG tool, the code generation for the RTEMS operating systems works only on SDF graphs having at most 255 actors. Besides, µC/OS-III is not anymore limiting the number of tasks and priorities but the previous version µC/OS-II was limiting[9] the number of tasks and priorities to 255.

4.2.2 Computation of periods, step (5). The computation of periods is based on RTA [2]: different values of the periods are tested between two bounds by performing a bisection, and the smallest valid period, i.e. greater than response time, is kept. As all actor periods can be expressed relatively to the basis period T_{basis}, the algorithm is called Symbolic Response Time Analysis (SRTA). Figure 7 depicts the main idea of SRTA. The SRTA algorithm is iterative: it starts with a period in the middle of the two bounds and tests if regular RTA ensures schedulability. If not, SRTA refines the bounds by taking the tested period as new lower bound. If yes, SRTA refines the bounds by taking the tested period as new upper bound. The algorithm stops when the two bounds are equal.

It has been stated in a previous article on the ADFG theory [18] (section 5.2 in ref.) that the lower bound of the response time R_v^l of a periodic actor v is :

$$R_v^l = \frac{C_v - \sum_{P_v < P_{v_i}} C_{v_i}(1 - U_{v_i})}{1 - \sum_{P_v < P_{v_i}} U_{v_i}}$$

by analogy with the *upper* bound [10]:

$$R_v^u = \frac{C_v + \sum_{P_v < P_{v_i}} C_{v_i}(1 - U_{v_i})}{1 - \sum_{P_v < P_{v_i}} U_{v_i}}$$

However this is an underestimation of the lower bound since it has also been proved that $R_v \geq \frac{C_v}{1-\sum_{P_v<P_{v_i}} U_{v_i}}$ [71], and yet $\frac{C_v}{1-\sum_{P_v<P_{v_i}} U_{v_i}} \geq \frac{C_v-\sum_{P_v<P_{v_i}} C_{v_i}(1-U_{v_i})}{1-\sum_{P_v<P_{v_i}} U_{v_i}}$.

Thus the period synthesis algorithm SRTA has been updated with the new equations resulting from the most accurate lower bounds: the aforementioned one [71] as well as $R_v \geq \sum_{P_v \leq P_{v_i}} C_{v_i}$ [43]. Indeed, schedulability is guaranteed when all periodic tasks (i.e. *actors* in the SDF case) have their response time lower than the deadline: $R_v \leq D_v = T_v$. Let us consider that the deadline is relative to the period: $D_v = \beta_v T_v$ with $\beta_v \in [0, 1]$. In the context of implicit deadlines, all β_v are equal to 1. As $T_v = \alpha_v T_{basis}$, ensuring schedulability with RTA leads to the constraint $T_{basis} \geq \frac{R_v^l}{\alpha_v \beta_v}$.

The response time lower bound $R_v^l = \max\{\sum_{P_v \leq P_{v_i}} C_{v_i}, \frac{C_v}{1-\sum_{P_v<P_{v_i}} U_{v_i}}\}$ [43, 71] can be rewritten knowing that $U_v = \frac{C_v}{T_v} = \frac{C_v}{\alpha_v T_{basis}}$. Thus the first part of the maximum of the response time lower

[8] https://docs.rtems.org/releases/rtems-5.1/c-user/task_manager.html#task-priority
[9] https://micrium.atlassian.net/wiki/spaces/osiiidoc/pages/131377/uC-OS+uC-OS-II+and+uC-OS-III+Features+Comparison

bound leads to:

$$\forall v \in V,\ T_{basis} \geq \frac{\sum_{P_v \leq P_{v_i}} C_{v_i}}{\alpha_v \beta_v}$$

and the second part leads to:

$$\forall v \in V,\ \alpha_v \beta_v T_{basis} \geq \frac{C_v}{1 - \sum_{P_v < P_{v_i}} \frac{C_{v_i}}{\alpha_{v_i} T_{basis}}} \Leftrightarrow$$

$$\alpha_v \beta_v T_{basis} - \alpha_v \beta_v \sum_{P_v < P_{v_i}} \frac{C_{v_i}}{\alpha_{v_i}} \geq C_v \Leftrightarrow T_{basis} \geq \frac{C_v}{\alpha_v \beta_v} + \sum_{P_v < P_{v_i}} \frac{C_{v_i}}{\alpha_{v_i}}$$

Finally, the new initial lower bound of the basis period[10], used in the SRTA algorithm, is:

$$T_{basis} \geq \max_{v \in V} \left\{ \frac{C_v}{\alpha_v \beta_v} + \max \left\{ \frac{\sum_{P_v < P_{v_i}} C_{v_i}}{\alpha_v \beta_v}, \sum_{P_v < P_{v_i}} \frac{C_{v_i}}{\alpha_{v_i}} \right\} \right\} \tag{17}$$

When targeting a multi-processor, SRTA is called separately on each processor and considers the only actors mapped on the current processor (the mapping has been performed previously): partitioned scheduling is considered. Then the maximum of the basis period found over all processors is kept as the new basis period. Alternatively, better response time analysis algorithms could be used, such as a faster algorithm [56] which is kept for future work, or an algorithm for multi-processor with global FP scheduling [6]. Holistic RTA [78], taking into account communication time in distributed systems and already implemented in a few analysis tools [61], is also another possible direction for future work. Finally, the SRTA algorithm does not take into account the offset of the actors, although theoretically possible [63], because offsets will be computed thanks to the phases in Equation (8).

4.2.3 Computation of delays and buffer sizes, steps (4) and (7). The ILP formulation solved in the abstraction part of the ADFG tool (see Section 4.1.1) already has the delays and buffer sizes as variables and computes them thanks to Equations (5) and (6), with the coefficients in Equation (15). However, in order to refine the solution of the ILP formulation which relies on safe affine overestimation, delays and buffer sizes are computed again at the very end of the synthesis part, after having computed the periods and offsets. Delays and buffer sizes are computed using a simulation performed independently for each buffer on their abstract clock relation. Their behavior is simulated during one hyperperiod $H_{e_{1\to2}} = \frac{1}{\text{lcm}(T_{v_1}, T_{v_2})}$ of the buffer $\delta_{e_{1\to2}}$, plus the time of the offsets and the time ensuring that the initial parts of the rates, p_0 and q_0, have been finished. Previous results [22] show that this simulation length is sufficient. Yet, this simulation does not take into account the WCETs of actors; nevertheless it may improve the result of the two affine Equations (5) and (6) for CSDF and UCSDF graphs since the simulation avoids the affine overestimation. For SDF graphs, λ terms are already equal to 0 in Equations (5) and (6) and the simulation will not help to improve the results.

During the simulation, the delay of a buffer is the maximum deficit of data observed (underflow). For example in Figure 4, $T_{v_1} < T_{v_2}$ so $P_{v_1} < P_{v_2}$, and as $\varphi = 0$, there is no offset between the actors, then v_1 is executed first, producing 2 tokens, followed by v_2 consuming 12 tokens. At this point of the simulation, the observed deficit is $12 - 2 = 10$ tokens. Once the delay has been computed, the simulation is performed a second time to compute the maximum number of tokens present at a time on the buffer, that is its size.

[10]This constraint is used in replacement of the former second order polynomial equation in [14] (page 97).

Unfortunately, this approach suffers an important drawback: the delays are a functional feature of the application and it is in general not valid to assign or rearrange delay values in arbitrary ways. For example, the ADFG tool does not respect retiming rules [50, 64] when computing the delays. Retiming rules have originally been depicted for task graph representing processor circuits: added delays must form a *feed-forward* graph cut of the graph. However, there is no generic retiming rule for any UCSDF graph.

Hence, three options have been added to the ADFG tool to improve the placement of delays. The first option keeps the delays or buffer sizes that the designer considers as fixed; the fixed values are injected in the first ILP formulation and the simulation checks if they are coherent. If the simulation finds higher value of delays or buffer sizes, it emits a warning. It is possible to keep only a part of the delays and buffer sizes, and let the ADFG tool compute the others. The second option enforces the delays to be a multiple of the lowest common multiple of the production and consumption rate of the buffer. It ensures that the injected tokens are synchronized with the execution of the two actors of the buffer. The third option enforces all delays to be zero, it is a specific case of the first option. Note that this zero delay option leads to unfeasible ILP formulations of Equations (5) and (6) if the UCSDF graph contains directed cycles, and leads to non-zero phases/offsets otherwise.

4.2.4 Computation of metrics, steps (1) and (6). Two main metrics are computed during the synthesis part of the ADFG tool: the processor utilization factor, and the throughput. However, the algorithms used to compute these two metrics where initially not consistent for CSDF and UCSDF graphs; they have been improved in the new implementation. These modifications have no impact on the SDF graphs.

First, the WCET of an actor may also be expressed in the form of a sequence of integers, as supported by the SDF3 [74] file format, which is a supported input of the ADFG tool. This representation allows the WCET to be specific to each firing of the actor and better fits the reality where firings consuming or producing numerous data most probably have a different execution time than firings of the same actor when consuming or producing only a few data. This situation occurs in the Digital Radio Mondiale receiver CSDF application [59]: the CD actor has two different WCETs, depending on its production rate. Unfortunately, the ADFG theory is not yet adapted to consider the WCET as a sequence of integers, and instead considers only the maximum value of this sequence. Thus, the processor utilization factor computed by the ADFG tool may be overestimated when considering CSDF and UCSDF graphs. In order to give better information to the designer, the processor utilization factor U is now computed again at the end of the synthesis part, once all periods are computed, with the average WCET over the sequence length. The second value is not used by the ADFG tool and is solely intended to inform the designer. For the Digital Radio Mondiale receiver, SP_UNI algorithm finds $U = 0.99$ whereas $U_{\text{avg}} = 0.76$.

Second, the computation of the average rates a_p and a_q (see Equation (2)) used in the abstraction part may lead to incorrect computation of the repetition vector of the application for CSDF and UCSDF graphs. Indeed, the ADFG tool stores all the fractions in the irreducible form and thus loses the information of the minimal amount of firings needed to iterate over all the possible production and consumption rates of CSDF and UCSDF actors. For example, the actor v_1 in Figure 8 has the average production rate of 1, as does actor v_2. Thus, the repetition vector of the graph in Figure 8 is $[1, 1]$ according to the average rates, whereas it should be $[2, 2]$ since the cyclo-static production rate of v_1 has a length of 2. Thus, the computation of the repetition vector has been extended to the CSDF and UCSDF cases in the current version of the ADFG tool, and leads to a redefinition of the throughput Θ using the repetition vector $\vec{z}$. The repetition vector is computed first using the average rates, and is multiplied by a global factor ensuring that all firings are taken into account. The global factor is the least common multiple of the first repetition factor of any actor, and of the

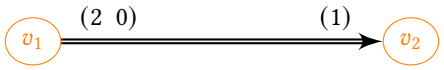


Fig. 8. CSDF graph where repetition vector needs specific computation.

length of its sequences of production or consumption rate). The previous throughput computation used the formula $\Theta = \frac{1}{\text{lcm}_{v \in V}\{T_v\}}$, and has been replaced by $\Theta = \frac{1}{\vec{z}[v] \cdot T_v}$ (equal for any $v \in V$); this new formula is used in the evaluation.

An evaluation of the ADFG tool is presented in Section 6, and includes results using these new computation of the throughput and the processor utilization factor.

5 CODE GENERATION

The ADFG tool supports automated code generation of the computed buffer sizes and other schedule characteristics for dataflow applications that are implemented in the Lightweight Dataflow Environment (LIDE) [52][11] running on the RTEMS real-time operating system[12]. The code generator inputs are the characteristics computed using the ADFG theory, including: buffer sizes, periods, offsets, priorities and mapping. The generated code is then cross-compiled and tested by using the QEMU tool to emulate an ARM platform. A detailed evaluation of the code generator has been conducted and presented in [79]. Interested readers can refer to this article for a complete presentation of our approach to integrate the ADFG tool in a framework with scheduling simulation and code generation. In this section, we summarize our approach of code generation and provide an update on the implementation of the period characteristic compared to [79].

5.1 Actors and buffers in LIDE

LIDE [52] is a flexible, lightweight design environment that enables rapid experimentations with dataflow-based implementations. The usage of LIDE allows a systematic approach to instantiate dataflow graphs with the buffer sizes computed using the ADFG theory. Actors and buffers in LIDE are designed and implemented as C-based abstract data types (ADTs). To implement a data-flow application with LIDE we need to allocate buffers, instantiate actors, and connect them together.

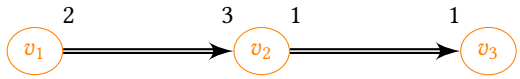
A buffer is instantiated with the function `lide_c_fifo_new` which takes two input parameters: `capacity` and `token_size`. It allows us to apply the buffer sizes computed using the ADFG theory to the code generation step.

- Capacity: the computed buffer size is given as an input parameter of the following function `lide_c_fifo_new`. It is the number of tokens of which sizes are given as the second input parameter to the function.
- Token size: this information is given in the specification of the graph and passed directly to the code generator. In case of complex types, we assume that their specifications in C are also provided.

Each actor has an associated `context`, which encapsulates pointers to the FIFO buffers that are associated with the edges incident to the actor. Four interface functions, namely `new`, `enable`, `invoke` and `terminate` are required to implement an actor. Designers can develop their own actors by appropriately specializing the prototype function templates. In this work, we assume that core functions of actors are implemented and we only generate the code to instantiate actors.

An actor is instantiated with the function `lide_c_<actor_name>_new`. The input parameters are buffers that are connected to the actor. An example of the generated code is given in Figure 9.

[11] http://dspcad.umd.edu/projects/lide/lide.htm
[12] https://www.rtems.org/




```
/* LIDE-C Buffers */
lide_c_fifo_pointer v1_v2 = lide_c_fifo_new(6, sizeof(int));
lide_c_fifo_pointer v2_v3 = lide_c_fifo_new(1, sizeof(int));
/* LIDE-C Actors */
lide_c_actor_context_type* actors[ACTOR_COUNT];
actors[0]=(lide_c_actor_context_type*) (lide_c_v1 (v1_v2));
actors[1]=(lide_c_actor_context_type*) (lide_c_v2 (v1_v2, v2_v3));
actors[2]=(lide_c_actor_context_type*) (lide_c_v3 (v2_v3));
```

Fig. 9. An example of actors and buffers declaration with LIDE.

This example is the code generated for the SDF graph presented in Figure 1 with computed buffer sizes. In the generated code, we declare the two buffers `v1_v2` and `v2_v3`. These buffers are used as input paramters of in the `*_new` functions to create the three actors `v1`, `v2`, and `v3`.

5.2 Schedule characteristics in RTEMS

RTEMS is an open source real-time operating system that supports open standard Application Programming Interfaces (APIs) such as POSIX. We consider the usage of RTEMS to generate the computed schedule characteristics. Actor invocations and fixed-priority periodic scheduling are achieved by the usage of the POSIX thread library.

Besides buffer sizes, four schedule characteristics are computed using the ADFG theory, namely: `priority`, `core mapping` (for processor mapping), `period` and `offset`. Code generation for these characteristics is done by exploiting the POSIX pthread API supported by RTEMS. One thread is generated per actor. The `priority` and `core mapping` attributes are natively supported. The `period` and `offset` are taken into account by implementing a periodic release per actor.

- Priority: POSIX `pthread_attr_setschedparam` function is used to set thread priorities.
- Mapping: POSIX provides the `cpu_set` function that allows us to choose the set of processors that a thread can execute on.
- Period: periodic release of a thread is implemented by using the `clock_nanosleep` function with the flag TIMER_ABSTIME. It allows us to set the absolute time at which the thread shall wake up again. The wake up time is computed as below:

$$wake\_up\_time := start\_time + (release\_number \times period) \tag{18}$$

  In [79], we used the `nanosleep` function to implement periodic tasks. It is because the function `clock_nanosleep` was not in the list of directives provided by the clock manager in RTEMS 5.1[13], which is the version targeted by our code generator. However, the authors in [11] indicated that it is possible to use the `clock_nanosleep` function to implement periodic tasks in RTEMS.
- Offset: is generated by adding an idle period to the first release of a thread.

An example of the generated code is given in Figure 10. We create a data structure named `rtems_actor` that encapsulates a `lide_c_actor_context` and its schedule characteristics. Then, these characteristics are passed to the attributes of a pthread accordingly. This example and its code can be duplicated systematically for any actors in the graph and their corresponding threads in order to apply schedule characteristics computed using the ADFG theory.

[13] https://docs.rtems.org/releases/rtems-5.1/posix-users/clock.html

```
rtems_actors[0].context = actors[0];
rtems_actors[0].name = "int_produce"
rtems_actors[0].priority = 1;
rtems_actors[0].period.tv_nsec = 500;
rtems_actors[0].processor = 1;

param.sched_priority=rtems_actors[0].priority; ;
pthread_attr_setschedparam(&attr,&param);
pthread_create(&id,&attr,lide_c_actor_start_routine,&rtems_actors[0]);
CPU_ZERO(&cpuset);
CPU_SET(processors[0], &cpuset);
pthread_setaffinity_np(id, sizeof(cpu_set_t), &cpuset);
```

Fig. 10. Generated code configuring the schedule characteristics in RTEMS.

6 EVALUATION

This section presents an evaluation of the schedules synthesized by the ADFG tool. Results are compared to the state-of-the-art theoretical methods to compute buffer sizes and throughput in Section 6.1; optimum is reached for both metrics. Scheduling simulations have also been performed to compute the number of preemptions and the usage of memory, see Section 6.2.

The code of the ADFG tool is open-source and can be accessed online[14]. Experiments using Cheddar tool require its own installation; on the code repository, we provide a link to a virtual machine with ADFG and Cheddar installed. In addition, all benchmarks, scripts, and instructions to reproduce the results presented in this section are included in the code repository and in the virtual machine. Old binaries are also available online[15], but they are outdated and the website hosting them is not maintained anymore.

The test applications come from StreamIt [77] and SDF3 [74] benchmarks, including pacemaker [69], h263 video decoder [62]. Eleven applications[16] have been selected for their representative characteristics detailed in Table 4: dataflow model, number of actors, and number of firings.

6.1 Scheduling synthesis evaluation

In this subsection, the synthesized schedules are evaluated directly regarding several metrics: processor utilization factor, sum of all buffer sizes, and throughput. Utilization factor and buffer sizes are studied with uni-processor algorithms provided by the ADFG tool while the throughput is studied with multi-processor algorithms and increasing number of available processors.

6.1.1 Impact of the priority in uni-processor. Figure 11 presents the utilization factor obtained for all applications with all uni-processor algorithms listed in Table 1, plus an algorithm synthesizing schedules for Earliest Deadline First (EDF) policy, given for reference. The maximum processor utilization factor, 1.0, is reached for all applications except cd2dat, pacemaker and receiver, the three CSDF applications. Despite the fact that cd2dat has 6 actors in its graph and more than a thousand firings in total, its processor utilization factor is only about 0.2 over 1. This is due to a limitation of our method: the SDF model imposes linear relation between all the actor *integer* periods (see Equation (7)), enforcing a common divisor to all. For cd2dat, the ADFG tool finds that the basis period must be a multiple of 160, which is already larger than the response time upper

[14] https://gitlab.inria.fr/ADFG/adfg
[15] http://polychrony.inria.fr/ADFG/
[16] Digital Radio Mondial receiver [59] is not included since its average processor utilization factor is lower than computed in ADFG, see Section 4.2.4.

Name	#actors	#firings
Beamformer	57	57
ChannelVocoder	55	55
DES	53	53
Filterbank	85	85
Serpent	120	120
Vocoder	114	114

(a) SDF test applications with repetition vector $\vec{z} = \vec{1}$.

Name	model	#actors	#firings
cd2dat	CSDF	6	1665
h263decoder	SDF	4	1190
mp3decoder	SDF	14	27
pacemaker	CSDF	4	169
receiver	CSDF	4	7687

(b) SDF and CSDF test applications with repetition vector $\vec{z} \neq \vec{1}$.

Table 4. Main characteristics of test applications.

bound[17]. Thus, it is better to avoid using relatively prime numbers as production and consumption rates in order to maximize the efficiency of the synthesized schedule. The processor utilization factor drop of h263decoder, mp3decoder, pacemaker and receiver while using algorithm SP_UNI_LOP is caused by another reason. The SP_UNI_LOP algorithm assigns priorities thanks to an extra ILP formulation solely minimizing an approximation of total buffer size (see Section 4.2.1). Then, the actor priorities assigned by the SP_UNI_LOP minimize the total buffer size at the cost of a lower throughput: the synthesized periods are greater than for the other algorithms.

Figure 12 compares the sum of all buffer sizes synthesized by the ADFG tool, using the same algorithms and application as in Figure 11. The last column of each cluster holds the minimum achievable total buffer size, as computed with the tool SDF3. The ADFG tool reaches the minimum total buffer size for all of its uni-processor algorithms and applications having $\vec{z} = \vec{1}$, but not always for the five other applications on the right. As for the processor utilization factor, h263decoder, mp3decoder, pacemaker and receiver applications have lower total buffer size while using algorithm SP_UNI_LOP. However while the drop factor seems constant for buffer sizes among all applications (see Figure 12), the processor utilization factor loses multiple order of magnitudes for h263decoder and receiver but remains similar for mp3decoder (see Figure 11). Besides, the complexity of the extra ILP formulation of SP_UNI_LOP is quite high (number of variables is quadratic to the number of actors), so this algorithm should be considered only at last resort by users of the ADFG tool. Instead one can use the SP_UNI_UDH algorithm described previously in Section 4.2.1: it enables to find trade-offs between the application throughput and its minimum total buffer size.

We briefly evaluate the SP_UNI_UDH algorithm on the receiver application, because this application has the highest drop of throughput for SP_UNI_LOP: it is divided by about 300 compared with the SP_UNI algorithm. At the same time, the total buffer size is divided by about 2. Thus, we

[17]In comparison, a simplification of the CSDF production rate on the last buffer of cd2dat, from (0101011) to (1111111), makes the related affine relation become $\frac{n}{d} = \frac{1}{1}$ instead of $\frac{4}{7}$. With such simplifaction, the basis period must be a multiple of only 40 instead of 160 initially and the utilization factor increases to 0.96 instead of 0.2.

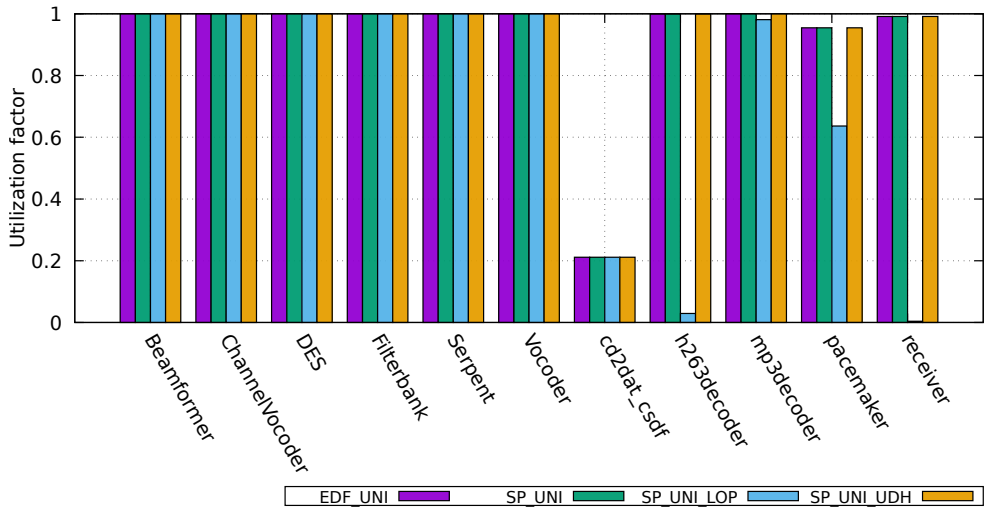


Fig. 11. Processor utilization factor of test applications.

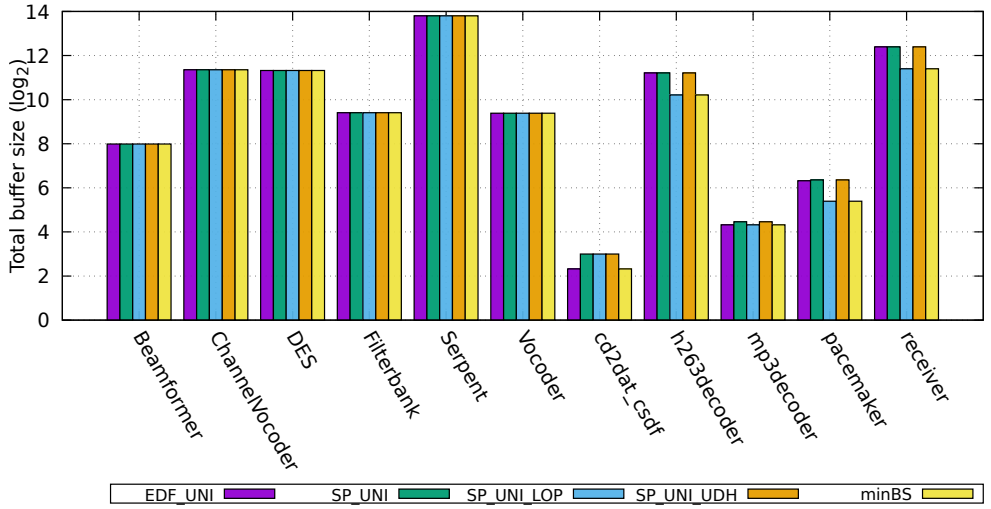


Fig. 12. Total buffer size of test applications.

expect that SP_UNI_UDH helps to find possible trade-offs for receiver between the two extrema found by SP_UNI and SP_UNI_LOP. With the default configuration of 10 percent throughput loss allowed against 10 percent total buffer size decrease with permutation of length 4, denoted `10-10-4`, SP_UNI_UDH cannot find better than SP_UNI: both throughput and total buffer size are identical on Figure 11 and Figure 12, respectively. However, the designer can specify manually the allowed loss and, for example, the SP_UNI_UDH configuration `6000-30-3` leads to a trade-off with throughput divided by about 35 and total buffer size divided by about 1.5. An automated exploration of possible trade-offs is kept for future work, but due to the approximation used to compute the periods in

SP_UNI_UDH, the throughput loss is overestimated (indeed, 6000 percent is even larger than the maximum observed ratio of 300).

Results of Figures 11 and 12 are similar with the zero delay option. Indeed, to compensate the zero delay option, the firing offset of the actors increases according to Equations (5) and (6). In turns, the zero delay option results in an increase of the regular offsets[18] of the actors according to Equation (8), whereas they are most often equal to zero without the zero delay option. Moreover all six applications with equal actor periods (Beamformer to Vocoder, with $\vec{z} = \vec{1}$) reach the same processor utilization factor and total buffer size independently to the algorithm used to assign actor priorities. In such case the priorities only impact on the actor offsets or on the delays, and their presence is completely avoided thanks to the first improvement presented in Section 4.2.1. Note that with the zero delay option enabled, all graphs with directed cycles deadlock and the ILP formulations of the ADFG theory are infeasible. However, the selected applications do not have directed cycles and so the ADFG tool finds a solution.

6.1.2 A closer look to SRTA bisection. All synthesis algorithms provided by the ADFG tool for FP scheduling use the SRTA algorithm described in Section 4.2.2, and some of them call it multiple times[19]; hence, it is important to check its efficiency. As SRTA performs a bisection between the response time lower and upper bound, it is especially important to measure the gap between these bounds and so the number of bisection steps.

Figure 13 depicts the number of bisection steps of SRTA for SP_MULT_BF_FBBFFD and SP_UNI algorithms. For the multi-processor algorithm, 4 processors were considered and the reported number of bisection is actually the sum of all the 4 SRTA calls (one per partition). The number of steps per processor is quite stable, slightly below 20, for all applications having more than 10 actors. This number of steps decreases for the applications having only a few actors, cd2dat, h263decoder, pacemaker and receiver. In only one case, h263decoder, the number of bisection steps is reduced for the multi-processor algorithm compared with the uni-processor one. Most probably, this is due to the high amplitude of actor firings and WCETs in h263decoder. However, more applications and a proper statistical analysis are needed to assert the causes of this phenomenon.

Last but not least, the bisection steps plotted in Figure 13 were actually counted separately when going up or down in the search interval. Surprisingly, the bisection is always going down towards the response time lower bound, which could mean that the lower bound is close to be optimal. The first six applications (from the left) all have equal periods, it is a plausible cause of the response time lower bound being sufficient for checking schedulability. For cd2dat, the cause is the high value of the common divisor to all integer periods. But again, a more extensive analysis is needed to assert the causes of this phenomenon, especially for the four last applications (to the right).

6.1.3 A closer look to (U)CSDF applications. ADFG tool supports two extensions of the SDF model: CSDF and UCSDF, especially thanks to an approximation of the evolution of tokens in buffers with an average rate (see Equation (2)). However, this approximation is safe regarding to processor utilization factor and buffer sizes but makes the synthesized periods and buffer sizes suboptimal. Two independent refinements performed at the end of the synthesis part of the ADFG theory partially overcome this problem. First, for the processor utilization factor, as described in Section 4.2.4: actor periods are not updated and eventually remain suboptimal, but a more accurate value of the metric is given to the designer. Second, the buffer sizes, as described in Section 4.2.3: the ADFG

[18] See examples of regular offset in Figures 3 and 16.

[19] SP_UNI_UDH calls SRTA twice. All multi-processor algorithms call it m times, that is once per partition. Additionally SP_MULT_BF_SRTA calls it $m \times |V|$ extra times, when selecting the partition of each actor.

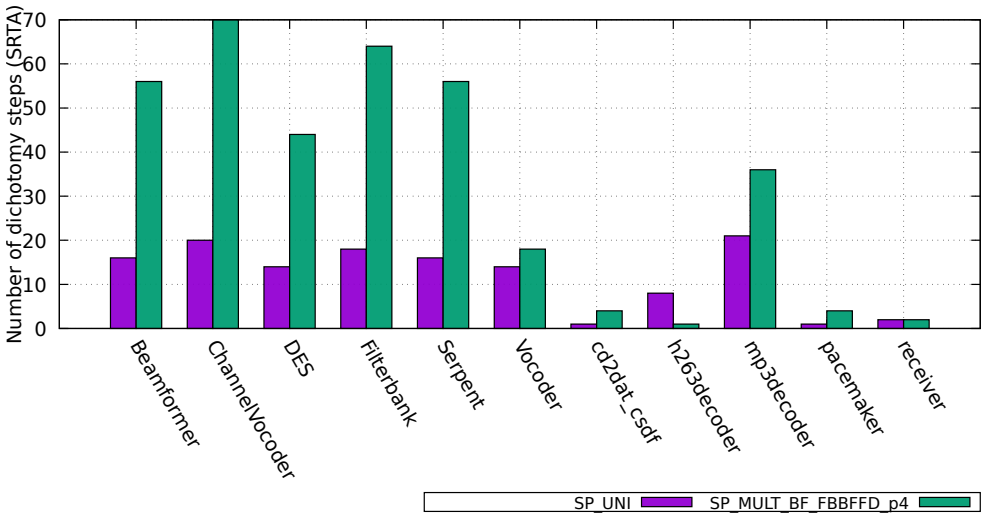


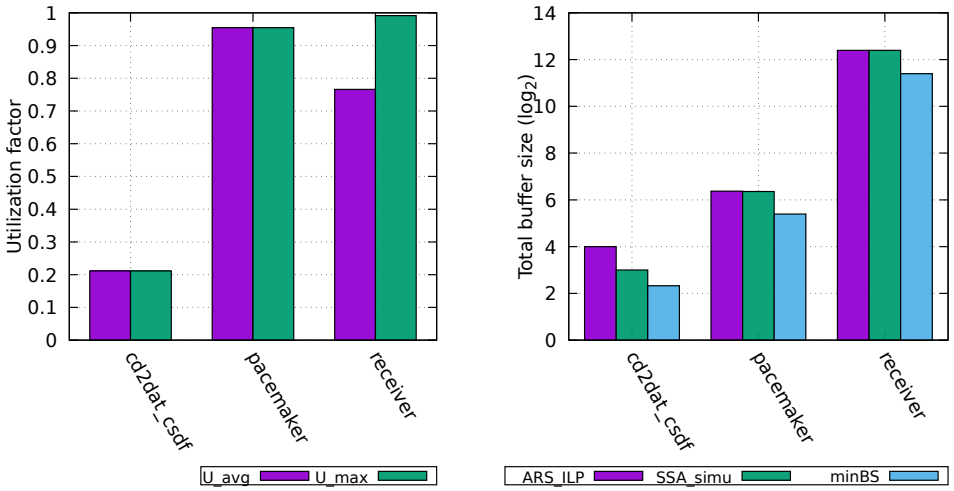
Fig. 13. Number of bisection steps of the SRTA algorithm in the ADFG tool.

tool simulates each buffer over the logical clock abstraction of its producer and consumer firings, and updates the sizes to the simulated smallest value.

The two aforementioned refinements have been evaluated on the three considered CSDF test applications: cd2dat, pacemaker and receiver. Figure 14a plots the average refined processor utilization factor (left column) against the overestimated one used in the schedule synthesis (right column). The refinement is useful only for CSDF applications specifying different WCET per firing, which is only the case of the receiver application. Figure 14b plots the overestimated buffer size computed in the schedule abstraction ILP (left column) against the refined one simulated after the schedule synthesis (middle column) and the minimum possible without any resource constraint (right column). Only cd2dat benefits from the buffer size refinement because two of its actors (*fir3* and *fir4*) alternatively produce one or zero token on their outgoing buffers. The buffer size reduction is 50%: from a total size of 16 to 8 for the SP_UNI algorithm.

6.1.4 Impact of multi-processor. The authors of the DARTS [3] tool proved that maximum throughput is obtained for the Beamformer application with a minimum of 29 processors for any partitioned scheduling, and 51 processors is an upper bound for the partitioned EDF policy [55] (which is close to the number of firings equal to 57). To evaluate the efficiency of the schedule characteristics computed using the ADFG theory, we synthesized the schedule of Beamformer for an increasing number of processors, as shown in Figure 15.

On Figure 15, line plots of EDF_MULT_BF_UF_ID, SP_MULT_BF_FBBFFD and SP_MULT_BF_SRTA algorithms are actually the same, above all other lines. They are close to the maximum throughput from 28 processors, and actually reach the maximum possible throughput for 44 processors, before the upper bound of 51. Only the SP_MULT_MBS-BUFFER_MIN algorithm does not reach the maximum, even with 52 processors. This is expected since this latter algorithm



(a) Processor utilization factor.

(b) Total buffer size.

Fig. 14. Refinement of processor utilization and buffer size for CSDF applications.

minimizes the total buffer size instead of maximizing the throughput [40][20]; previous results proved that both objectives cannot be optimized at the same time [75]. The execution time of the synthesis is not impacted by the number of processors: it is stable for all number of processors, around one second for all algorithms except the SP_MULT_MBS ones, which needs up to three minutes when minimizing the buffer sizes of the Serpent application (with an Intel i7-3740QM @2.70GHz processor).

Note that even if EDF scheduling policy is considered better than FP policy [20], the ADFG tool synthesizes periods with the same utilization factor for both policies in the uni-processor case, and reaches the maximum throughput at the same time as the SP_MULT_BF_FBBFFD and SP_MULT_BF_SRTA synthesis algorithms for FP policy.

Finally, we recall that the ADFG theory always synthesizes the periods so that the application is schedulable. Hence, the ADFG tool is inherently safe to the Dhall's effect [27]. Moreover, as we use partitioned scheduling and best fit mapping with SRTA[21], the ADFG tool synthesizes efficient period even when the WCET of an actor is the double of all others for example, as illustrated in Figure 16.

6.1.5 Impact of the graph. Different kinds of graph topology are present in the test applications. For example, Beamformer is symmetrical with at most 12 actors in parallel paths, while DES is long and thin with at most 3 parallel paths. Vocoder is a mix: widely parallel at the beginning and sequential at the end. Despite these differences of topology, we do not observe substantial differences in the efficiency of the synthesized schedules. Yet the efficiency can be severely decreased by the integer properties of the rates, as seen for the cd2dat application in Section 6.1.1.

[20] SP_MULT_MBS-BUFFER_MIN algorithm actually minimizes the number of memory transfer between processors when mapping the actors, but to do so it may avoid using all available processors. Indeed, in the Table 3 used by the SP_MULT_MBS-BUFFER_MIN algorithm, buffers are larger if producer and consumer actors are not on the same processor. In practice SP_MULT_MBS-BUFFER_MIN uses only half of the available processors.

[21] The mapping algorithm maps the actors, sorted by increasing priority, to the processor ensuring the highest throughput.

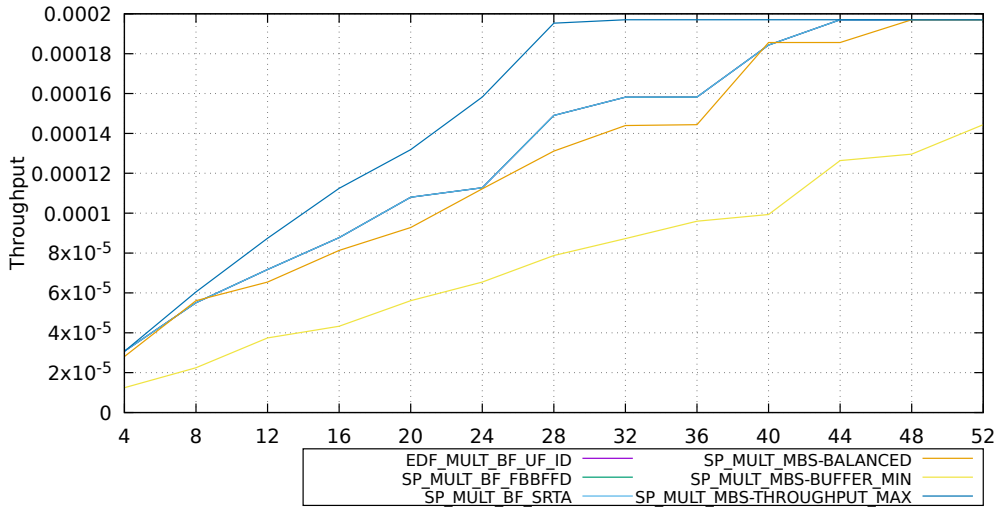
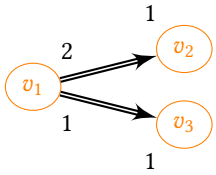
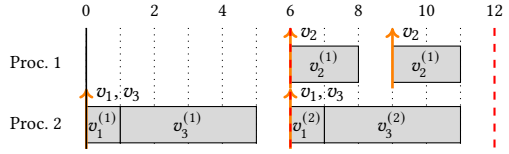


Fig. 15. Throughput of Beamformer synthesized by the ADFG tool, in function of the number of processors.



(a) SDF graph with repetition vector $\vec{z} = [1, 2, 1]$ and WCETs $C_{v_1} = 1$, $C_{v_2} = 2$, $C_{v_3} = 4$.



(b) Schedule of graph 16a synthesized by the ADFG tool on two processors. Given periods $T_{v_1} = 6$, $T_{v_2} = 3$, $T_{v_3} = 6$. Computed priorities $P_{v_1} = 2$, $P_{v_2} = 1$, $P_{v_3} = 3$ and offset $O_{v_2} = 6$.

Fig. 16. Real-time schedule synthesized by SP_MULT_BF_SRTA algorithm, $U = 1.5$.

One can think that the ADFG theory restriction to weakly connected graphs is too limiting. Yet this restriction is easily overcome by adding a dummy source actor having virtual dependencies to the unconnected actors. For example, one can consider in the graph of Figure 16a that the actor v_1 is a dummy actor (with the smallest possible WCET equal to 1), present only in order to connect the graph consisting of actors v_2 and v_3.

Another drawback of the ADFG theory could be the absence of auto-concurrency. However, it can be overcome by duplicating actors in the input graph, as it is done in the Beamformer SDF graph with the symmetrical parallel paths corresponding to the same kernel applied to different data.

6.2 Simulation with CHEDDAR

Scheduling simulation is used in order to provide a thorough evaluation of the schedules synthesized by the ADFG tool. It allows us not only to verify the correctness of the results but also to obtain additional information on the application execution. The ADFG tool can inter-operate with Cheddar [70], which is an open source scheduling analyzer. The CSDF graph model and the buffer verification (i.e. overflow and underflow) are already supported by the simulator. The additional

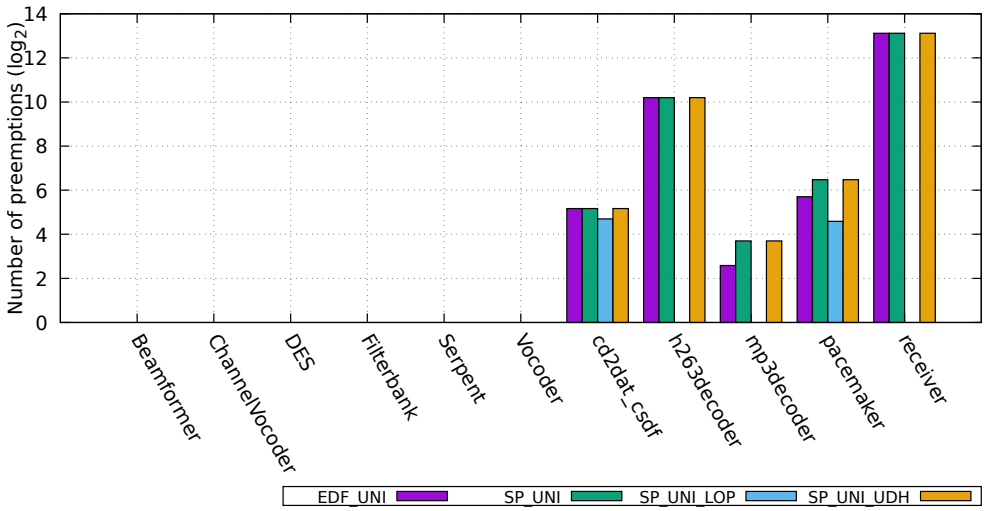


Fig. 17. Number of preemptions per single execution, in the uni-processor case.

information obtained by scheduling simulation includes the number of preemptions and the buffer utilization. The simulation is performed over one hyperperiod plus the maximum of all offsets.

Figure 17 presents the number of preemptions measured in the uni-processor case. There is no preemption for all six applications in which each actor is fired only once (see Table 4a). Indeed, these actors all have the same period, and the heuristic to break priority tie presented in Section 4.2.1 avoids offsets; all firings are released at the same time and executed completely one after the other in the order of their priorities being the graph topological order. The number of preemptions is of the same order of magnitude as the total number of firings for the five other applications (see Table 4b). Moreover, when minimizing the total buffer size with SP_UNI_LOP algorithm, it also reduces the number of preemptions especially for h263decoder, mp3decoder and receiver applications.

Buffer simulation is even more interesting. Figure 18 details the buffer utilization BT_{max}, which is computed by taking into account the maximum number of tokens that are present in all buffers during the simulation length divided by the total buffer size. Equation 19 defines BT_{max}.

$$BT_{max} = \left(\max_{\forall t, 0 \leq t \leq t_{sched}} \left(\sum_{e \in E} nb\_token_e^t \right) \right) / \sum_{e \in E} \delta_e \tag{19}$$

For each time unit t during the simulation length from 0 to t_{sched}, the number of tokens stored in every buffer e at time t, denoted $nb\_token_e^t$, is summed. Then, the maximum value over time is divided by the sum of buffer sizes. Tokens are popped from the input buffers of an actor at its start time, and pushed to its output buffers at its finish time. When two or more actors write or read at the same time from different processors, the simulation analysis reorders all the write operations before the reads. Buffers are all statically and independently allocated: they do not share memory.

In average of the applications and synthesis algorithms plotted in Figure 18, the memory containing all buffers is used at most to only 56% of its capacity. This percentage is similar for all applications and algorithms, except for the cd2dat CSDF application which uses approximately 68% of memory. The SP_MULT_MBS-BUFFER_MIN algorithm reaches a slightly better buffer utilization, but at the cost of a throughput loss (see Footnote 20). The fact that buffers are never filled to their maximum size is not surprising knowing that their size is refined by the ADFG tool

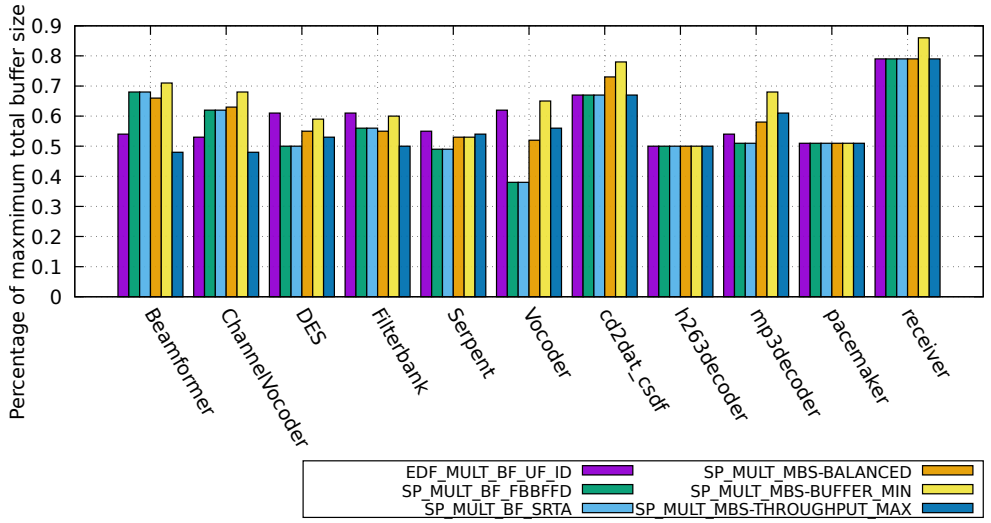


Fig. 18. Maximum number of token present at a time in all buffers, in the multi-processor case (4 processors).

(see Section 4.2.3) without taking into account the execution time. Such refinement may be useful for CSDF applications but has no impact otherwise. Moreover while one buffer is full, others may be empty.

The h263decoder application having only 4 actors is a simple enough example to explain why the buffer size is overestimated. Its source actor *vld* produces 594 token and has a WCET more than 10 times smaller than its period. *vld* is executed once while its only consumer, *iq* actor consuming 1 token, is executed 594 times. In such case the ADFG tool safely approximates the number of delays to 594 (that are all the ones consumed by *iq* between two executions of *vld*) and doubles the buffer size accordingly because *vld* has a lower priority than *iq*. However the optimal number of delays is only 44 (if fully partitioned) due to the small WCET of *vld* compared to its period: after 44 executions of *iq*, *vld* has already finished its execution and thus has produced 594 new token.

In any case, the buffer simulation done by Cheddar demonstrates that our synthesized buffer sizes and delays are correct but that an important ratio of memory is wasted (see Figure 12). The memory waste is due to the ADFG theory overestimation and to the independent buffer allocation model itself. Thus, other techniques to reduce the memory footprint used for static scheduling, such as the computation of a mutual exclusion graph of all firing dependencies and buffer merging techniques [26], would be interesting to adapt to our periodic scheduling case.

7 CONCLUSION

In this article, we have presented an overview of the ADFG tool, which uses affine relations as a model abstraction to efficiently map actors in dataflow graphs to real-time tasks. This tool synthesizes multiple schedule characteristics, and generates code for the RTEMS operating system. The ADFG tool applies an original and efficient combination of formal methods, and the correctness of the implementation has been evaluated through extensive synthesis experiments and simulations. Moreover, inter-operability between tools (scheduling synthesizer, simulator, code generator) was a key property to perform these experiments, and it greatly helps the system designers. Finally, the 12-years work done on the ADFG tool has shown that the theory it relies on is mature enough to go up

to code generation and execution[22], although some important timing metrics (as preemption delay) are still not taken into account. Future work includes connecting our framework with a WCET analysis tool, such as Heptane [38], to provide a holistic analysis. In addition, more theoretical investigations lie ahead, such as employing cyclo-static WCETs for CSDF actors with a multiframe model [58], or improving the effectiveness of the delay computation with retiming techniques [50].

REFERENCES

[1] H. Andrade, J. Correll, A. Ekbal, A. Ghosal, D. Kim, J. Kornerup, R. Limaye, A. Prasad, K. Ravindran, T. N. Tran, M. Trimborn, G. Wang, I. Wong, and G. Yang. 2012. From Streaming Models to FPGA Implementations. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. WORLDCOMP'12, 1.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. 1993. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal* 8 (1993), 284–292.

[3] M. Bamakhrama and T. Stefanov. 2011. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *International Conference on Embedded Software (EMSOFT)*.

[4] A. Benabid-Najjar, C. Hanen, O. Marchetti, and A. Munier-Kordon. 2012. Periodic Schedules for Bounded Timed Weighted Event Graphs. *IEEE Trans. Automat. Control* 57, 5 (May 2012), 1222–1232. https://doi.org/10.1109/TAC.2012.2191871

[5] M. Benazouz, O. Marchetti, A. Munier-Kordon, and T. Michel. 2010. A new method for minimizing buffer sizes for Cyclo-Static Dataflow graphs. In *8th IEEE Workshop on Embedded Systems for Real-Time Multimedia*.

[6] M. Bertogna and M. Cirinei. 2007. Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. 149–160. https://doi.org/10.1109/RTSS.2007.31

[7] Loïc Besnard, Adnan Bouakaz, Thierry Gautier, Paul Le Guernic, Yue Ma, Jean-Pierre Talpin, and Huafeng Yu. 2015. Timed behavioural modelling and affine scheduling of embedded software architectures in the AADL using Polychrony. *Sci. Comput. Program.* 106 (2015), 54–77. https://doi.org/10.1016/j.scico.2014.05.014

[8] S. S. Bhattacharyya, E. A. Lee, and P. K. Murthy. 1996. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, Norwell, MA, USA.

[9] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. 1996. Cycle-static Dataflow. *Trans. Sig. Proc.* 44, 2 (Feb. 1996), 397–408. https://doi.org/10.1109/78.485935

[10] E. Bini, T. Huyen Chau Nguyen, P. Richard, and S. K. Baruah. 2009. A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines. *IEEE Trans. Comput.* 58, 2 (Feb 2009), 279–286. https://doi.org/10.1109/TC.2008.167

[11] Gedare Bloom, Joel Sherrill, Tingting Hu, and Ivan Cibrario Bertolotti. 2020. *Real-time systems development with RTEMs and multicore processors*. CRC Press.

[12] B. Bodin, A. Munier-Kordon, and B. D. de Dinechin. 2013. Periodic schedules for Cyclo-Static Dataflow. In *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*. 105–114. https://doi.org/10.1109/ESTIMedia.2013.6704509

[13] B. Bodin, L. Nardi, P. H. J. Kelly, and M. F. P. O'Boyle. 2016. Diplomat: Mapping of Multi-kernel Applications Using a Static Dataflow Abstraction. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 241–250. https://doi.org/10.1109/MASCOTS.2016.35

[14] Adnan Bouakaz. 2013. *Real-time scheduling of dataflow graphs*. Theses. Université Rennes 1.

[15] A. Bouakaz and T. Gautier. 2014. An abstraction-refinement framework for priority-driven scheduling of static dataflow graphs. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*. 2–11. https://doi.org/10.1109/MEMCOD.2014.6961838

[16] A. Bouakaz, T. Gautier, and J. P. Talpin. 2014. Earliest-deadline first scheduling of multiple independent dataflow graphs. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. 1–6. https://doi.org/10.1109/SiPS.2014.6986102

[17] Adnan Bouakaz and Jean-Pierre Talpin. 2013. Buffer Minimization in Earliest-deadline First Scheduling of Dataflow Graphs. In *Proceedings of the 14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems* (Seattle, Washington, USA) *(LCTES '13)*. ACM, New York, NY, USA, 133–142. https://doi.org/10.1145/2465554.2465558

[18] A. Bouakaz and J.-P. Talpin. 2013. Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks. In *International Workshop on Software and Compilers for Embedded Systems*. St. Goar, Germany, 58–67. https://doi.org/10.1145/2463596.2463600

[19] A. Bouakaz, J.-P. Talpin, and J. Vitek. 2012. Affine Data-Flow Graphs for the Synthesis of Hard Real-Time Applications. In *Proceedings of the 2012 12th International Conference on Application of Concurrency to System Design*. ACM, Hamburg, Germany, 183–192. https://doi.org/10.1109/ACSD.2012.16

[20] Giorgio C Buttazzo. 2005. Rate monotonic vs. EDF: judgment day. *Real-Time Systems* 29, 1 (2005), 5–26.

[22]For research purpose only, since our implementation is not certified.

[21] J. Castrillon, R. Leupers, and G. Ascheid. 2013. MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics* 9, 1 (Feb 2013), 527–545. https://doi.org/10.1109/TII.2011.2173941

[22] L. Cucu and J. Goossens. 2006. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *IEEE Conference on emerging Technologies and Factory Automation, 2006.* 397–404.

[23] R. I. Davis and A. Burns. 2007. Robust priority assignment for fixed priority real-time systems. In *Proceedings of the 28^{th} IEEE International Real-Time Systems Symposium (RTSS).* 3–14.

[24] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. 2016. A review of priority assignment in real-time systems. *Journal of systems architecture* 65 (2016), 64–82.

[25] R. de Groote, J. Kuper, H. Broersma, and G. J. M. Smit. 2012. Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications.* 29–38. https://doi.org/10.1109/SEAA.2012.20

[26] K. Desnos, M. Pelcat, J.-F. Nezan, and S. Aridhi. 2016. On Memory Reuse Between Inputs and Outputs of Dataflow Actors. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 2 (Feb. 2016), 30. https://doi.org/10.1145/2871744

[27] Sudarshan K. Dhall and C. L. Liu. 1978. On a Real-Time Scheduling Problem. *Operations Research* 26, 1 (1978), 127–140.

[28] A. Dkhil, X. Do, P. Dubrulle, S. Louise, and C. Rochange. 2014. Self-timed Periodic Scheduling for a Cyclo-static DataFlow Model. In *Procedia Computer Science*, Vol. 29.

[29] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. 2003. Taming heterogeneity - the Ptolemy approach. *Proc. IEEE* 91, 1 (2003), 127–144. https://doi.org/10.1109/JPROC.2002.805829

[30] N. Fisher, S. Baruah, and T. P. Baker. 2006. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS '06).* IEEE Computer Society, Washington, DC, USA, 118–127. https://doi.org/10.1109/ECRTS.2006.30

[31] A. Gamatié, S. Le Beux, É. Piel, R. Ben Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser. 2011. A Model-Driven Design Framework for Massively Parallel Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 10, 4, Article 39 (Nov. 2011). https://doi.org/10.1145/2043662.2043663

[32] Thierry Gautier, Clément Guy, Alexandre Honorat, Paul Le Guernic, Jean-Pierre Talpin, and Loïc Besnard. 2019. Polychronous automata and their use for formal validation of AADL models. *Frontiers of Computer Science* 13, 4 (Aug. 2019), 677–697. https://doi.org/10.1007/s11704-017-6134-5

[33] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. 2006. Throughput Analysis of Synchronous Data Flow Graphs. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD '06).* IEEE Computer Society, Washington, DC, USA, 25–36. https://doi.org/10.1109/ACSD.2006.33

[34] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen. 2007. Latency Minimization for Synchronous Data Flow Graphs. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007).* 189–196. https://doi.org/10.1109/DSD.2007.4341468

[35] R. Govindarajan, Guang R. Gao, and Palash Desai. 2002. Minimizing Buffer Requirements under Rate-Optimal Schedule in Regular Dataflow Networks. *Journal of VLSI signal processing systems for signal, image and video technology* 31, 3 (2002), 207–229. https://doi.org/10.1023/A:1015452903532

[36] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. Sangiovanni-Vincentelli, and E. A. Lee. 2014. Metronomy: A function-architecture co-simulation framework for timing verification of cyber-physical systems. In *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* 1–10. https://doi.org/10.1145/2656075.2656093

[37] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo. 2008. PeaCE: A Hardware-software Codesign Environment for Multimedia Embedded Systems. *ACM Trans. Des. Autom. Electron. Syst.* 12, 3, Article 24 (May 2008), 24:1–24:25 pages. https://doi.org/10.1145/1255456.1255461

[38] D. Hardy, B. Rouxel, and I. Puaut. 2017. The Heptane Static Worst-Case Execution Time Estimation Tool. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017) (International Workshop on Worst-Case Execution Time Analysis, Vol. 8).* Dubrovnik, Croatia. https://doi.org/10.4230/OASIcs.WCET.2017.8

[39] Joost P. H. M. Hausmans, Stefan J. Geuns, Maarten H. Wiggers, and Marco J. G. Bekooij. 2014. Temporal Analysis Flow Based on an Enabling Rate Characterization for Multi-Rate Applications Executed on Mpsocs with Non-Starvation-Free Schedulers. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems* (Sankt Goar, Germany) *(SCOPES '14).* Association for Computing Machinery, New York, NY, USA, 108–117. https://doi.org/10.1145/2609248.2609262

[40] A. Honorat, H. N. Tran, L. Besnard, T. Gautier, J.-P. Talpin, and A. Bouakaz. 2017. ADFG: a scheduling synthesis tool for dataflow graphs in real-time systems. In *International Conference on Real-Time Networks and Systems* . Grenoble, France, 1–10. https://doi.org/10.1145/3139258.3139267

[41] W. A. Horn. 1974. Some simple scheduling algorithms. *Naval Research Logistics Quarterly* 21, 1 (1974), 177–185. https://doi.org/10.1002/nav.3800210113

[42] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. 2008. From the prototype to the final embedded system using the Ocarina AADL tool suite. *ACM Transactions on Embedded Computing Systems (TECS)* 7, 4 (2008), 42.
[43] M. Joseph and P. Pandya. 1986. Finding Response Times in a Real-Time System. *Comput. J.* 29, 5 (1986), 390. https://doi.org/10.1093/comjnl/29.5.390
[44] J. Khatib, A. Munier-Kordon, E. C. Klikpo, and T.-C. Kods. 2016. Computing latency of a real-time system modeled by Synchronous Dataflow Graph. In *Real-Time Networks and Systems RTNS*. Brest, France, 87 – 96. https://doi.org/10.1145/2997465.2997479
[45] Enagnon Cédric Klikpo and Alix Munier-Kordon. 2016. Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs. In *Real-Time Networks and Systems RTNS*. Brest, France, 77 – 86. https://doi.org/10.1145/2997465.2997474
[46] Hee-Hwan Kwak, Insup Lee, Anna Philippou, Jin-Young Choi, and Oleg Sokolsky. 1998. Symbolic schedulability analysis of real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*. IEEE, 409–418.
[47] Yu-Kwong Kwok and Ishfaq Ahmad. 1999. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Comput. Surv.* 31, 4 (Dec. 1999), 406–471. https://doi.org/10.1145/344588.344618
[48] E. A. Lee and D. G. Messerschmitt. 1987. Synchronous data flow. *Proc. IEEE* 75, 9 (Sept 1987), 1235–1245. https://doi.org/10.1109/PROC.1987.13876
[49] E. A. Lee and T. M. Parks. 1995. Dataflow process networks. *Proc. IEEE* 83, 5 (May 1995), 773–801. https://doi.org/10.1109/5.381846
[50] C. E. Leiserson and J. B. Saxe. 1991. Retiming synchronous circuitry. *Algorithmica* 6, 1 (01 Jun 1991), 5–35. https://doi.org/10.1007/BF01759032
[51] Joseph Y-T Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation* 2, 4 (1982), 237–250.
[52] S. Lin, Y. Liu, K. Lee, L. Li, W. Plishker, and S. S. Bhattacharyya. 2017. The DSPCAD framework for modeling and synthesis of signal processing systems. In *Handbook of Hardware/Software Codesign*, Soonhoi Ha and Jürgen Teich (Eds.). Springer, Netherlands, Chapter 36, 1185–1219. https://doi.org/10.1007/978-94-017-7267-9_36
[53] S. Lin, J. Wu, and S. S. Bhattacharyya. 2018. Memory-constrained Vectorization and Scheduling of Dataflow Graphs for Hybrid CPU-GPU Platforms. *ACM Transactions on Embedded Computing Systems* 17, 2 (January 2018), 50:1–50:25.
[54] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.
[55] J. M. López, J. L. Díaz, and D. F. García. 2004. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems* 28, 1 (01 Oct 2004), 39–68. https://doi.org/10.1023/B:TIME.0000033378.56741.14
[56] Wan-Chen Lu, Jen-Wei Hsieh, Wei-Kuan Shih, and Tei-Wei Kuo. 2006. A faster exact schedulability analysis for fixed-priority scheduling. *Journal of Systems and Software* 79, 12 (2006), 1744 – 1753. https://doi.org/10.1016/j.jss.2006.03.023
[57] O. Marchetti and A. Munier-Kordon. 2009. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research* 197, 2 (Sept. 2009), 532–540. https://doi.org/10.1016/j.ejor.2008.07.037
[58] A. K. Mok and D. Chen. 1997. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering* 23, 10 (Oct 1997), 635–645. https://doi.org/10.1109/32.637146
[59] A. Moonen, M. Bekooij, R. van den Berg, and J. van Meerbergen. 2008. Cache Aware Mapping of Streaming Applications on a Multiprocessor System-on-Chip. In *2008 Design, Automation and Test in Europe*. 300–305. https://doi.org/10.1109/DATE.2008.4484696
[60] Orlando M. Moreira and Marco J. G. Bekooij. 2007. Self-Timed Scheduling Analysis for Real-Time Applications. *EURASIP Journal on Advances in Signal Processing* 2007, 1 (2007), 083710. https://doi.org/10.1155/2007/83710
[61] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. 2011. Implementation of Holistic Response-time Analysis in Rubus-ICE. *In proceeding of: The 32nd IEEE Real-Time Systems Symposium (RTSS), WIP Session* (2011).
[62] Hyunok Oh and Soonhoi Ha. 2002. Fractional Rate Dataflow Model and Efficient Code Synthesis for Multimedia Applications. *SIGPLAN Not.* 37, 7 (June 2002), 12–17. https://doi.org/10.1145/566225.513834
[63] José Carlos Palencia and M González Harbour. 1998. Schedulability analysis for tasks with static and dynamic offsets. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*. IEEE, 26–37.
[64] Keshab K. Parhi. 2007. *VLSI digital signal processing systems : design and implementation*. John Wiley \& Sons.
[65] K. K. Parhi and D. G. Messerschmitt. 1991. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Trans. Comput.* 40, 2 (Feb 1991), 178–195. https://doi.org/10.1109/12.73588
[66] T. M. Parks, J. L. Pino, and E. A. Lee. 1995. A comparison of synchronous and cycle-static dataflow. In *Signals, Systems and Computers, 1995. 1995 Conference Record of the Twenty-Ninth Asilomar Conference on*, Vol. 1. IEEE, 204–210.
[67] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi. 2014. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming. In *Education and Research Conference (EDERC), 2014 6th European Embedded Design in*. 36–40. https://doi.org/10.1109/EDERC.2014.6924354

[68] F. Pellegrini. 2012. Scotch and PT-Scotch Graph Partitioning Software: An Overview. In *Combinatorial Scientific Computing*, Uwe Naumann and Olaf Schenk (Eds.). CRC Press, 373–406. https://doi.org/10.1201/b11644-15

[69] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha. 2009. Handling Mixed-criticality in SoC-based Real-time Embedded Systems. In *Proceedings of the Seventh ACM International Conference on Embedded Software* (Grenoble, France) *(EMSOFT '09)*. ACM, New York, NY, USA, 235–244. https://doi.org/10.1145/1629335.1629367

[70] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. 2004. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, Vol. 24. ACM, 1–8.

[71] M. Sjödin and H. Hansson. 1998. Improved response-time analysis calculations. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. 399–408. https://doi.org/10.1109/REAL.1998.739773

[72] Irina Smarandache, Thierry Gautier, and Paul Le Guernic. 1999. Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints. In *World Congress on Formal Methods in the Development of Computing Systems (FM'99) (LNCS vol. 1709)*. Springer, Toulouse, France, 1364–1383. https://doi.org/10.1007/3-540-48118-4_22

[73] S. Stuijk, M. Geilen, and T. Basten. 2006. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *2006 43rd ACM/IEEE Design Automation Conference*. 899–904. https://doi.org/10.1145/1146909.1147138

[74] S. Stuijk, M.C.W. Geilen, and T. Basten. 2006. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference (ACSD '06)* (Turku, Finland).

[75] S. Stuijk, M. Geilen, and T. Basten. 2008. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. *IEEE Trans. Comput.* 57, 10 (Oct 2008), 1331–1345. https://doi.org/10.1109/TC.2008.58

[76] Youcheng Sun, Étienne André, and Giuseppe Lipari. 2015. Verification of two real-time systems using parametric timed automata. http://waters2015.inria.fr/program/ Waters 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems ; Conference date: 07-07-2015 Through 07-07-2015.

[77] W. Thies, M. Karczmarek, and S. P. Amarasinghe. 2002. StreamIt: A Language for Streaming Applications. In *Proceedings of the 11th International Conference on Compiler Construction (CC '02)*. Springer-Verlag, London, UK, UK, 179–196.

[78] Ken Tindell and John Clark. 1994. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming* 40, 2-3 (1994), 117–134.

[79] Hai Nam Tran, Alexandre Honorat, Shuvra S. Bhattacharyya, Jean-Pierre Talpin, Thierry Gautier, and Loïc Besnard. 2021. A Framework for Fixed Priority Periodic Scheduling Synthesis from Synchronous Data-flow Graphs. In *21th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*.

[80] W. F. J. Verhaegh, E. H. L. Aarts, P. C. N. van Gorp, and P. E. R. Lippens. 2001. A two-stage solution approach to multidimensional periodic scheduling. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20.

[81] G.Q. Wang, R. Allen, H.A. Andrade, and A. Sangiovanni-Vincentelli. 2014. Communication Storage Optimization for Static Dataflow with Access Patterns Under Periodic Scheduling and Throughput Constraint. *Comput. Electr. Eng.* 40, 6 (Aug. 2014), 1858–1873. https://doi.org/10.1016/j.compeleceng.2014.05.002

[82] Yun Wang and Manas Saksena. 1999. Scheduling fixed-priority tasks with preemption threshold. In *Sixth IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA)*. 328–335.

[83] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. 2007. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *2007 44th ACM/IEEE Design Automation Conference*. 658–663.

accepted 22 July 2023