Modeling and Analysis of Data Flow Graphs Using the Digraph Real-Time Task Model

Morteza Mohaqeqi[(⊠)], Jakaria Abdullah, and Wang Yi

Uppsala University, Uppsala, Sweden
morteza.mohaqeqi@it.uu.se

Abstract. Data flow graphs are widely used for modeling and analysis of real-time streaming applications in which having a predictable and reliable implementation is an essential requirement. In this paper, we consider scheduling a set of data flow graphs such that liveness and boundedness properties are guaranteed, which leads to a predictable and correct behavior of the application. A formal translation method is proposed to map a given set of data flow graphs to a set of graph-based real-time tasks. Additionally, sufficient conditions are derived under which the obtained task set provides a semantically correct implementation of the given data flow graphs. It is shown that the proposed approach provides a higher level of design flexibility compared to the existing methods which use a simpler, i.e. periodic, task model.

Keywords: Data flow graphs · Real-time task models · Buffer boundedness · Schedulability analysis

1 Introduction

During the past decades, data flow graphs [1,2] have been extensively used for modeling and analysis of real-time streaming and signal processing applications. A number of prominent measures of these applications, including throughput, timeliness, liveness, and processing latency have been analyzed based on this formalism. Such analyses help the designers to have a predictable and reliable implementation of the mentioned applications.

Recently, increasing attention has been paid to study data flow graphs from a real-time scheduling point of view [3–8]. A popular approach is mapping each actor in a given data flow graph to an independent real-time task. Then, the problem is to specify the real-time tasks parameters such that the timing behavior of the data flow is correctly reflected by the task set. The advantage of this approach is that it makes it possible to *reuse* the existing analysis frameworks developed for real-time systems in the scheduling of a set of data flow graphs. For instance, using this approach, the interfering effect of different data flow applications on each other can be analyzed based on the existing theory of real-time task models.

In spite of the relatively extensive studies in this context, only a limited number of real-time task models have been explored by the researchers. In particular,

the work has been mainly focused on the periodic task model. Nonetheless, more expressive models can provide more flexibility to the designers which can lead to better solutions.

In this paper, we propose to use one of the most expressive yet efficiently analyzable real-time task models, namely the Digraph Real-Time (DRT) model [9], to specify data flow graphs. We present a translation method and discuss the potential benefits and the restrictions of this approach. The proposed method guarantees both boundedness and liveness properties of a data flow graph.

The rest of this paper is organized as follows: Sect. 2 describes the system model by presenting a brief review on the syntax and semantics of a data flow graph. The Digraph Real-Time task model is reviewed in Sect. 3. We present our translation method in Sect. 4. The proposed method is evaluated through the model of an MP3 playback application in Sect. 5. The work related to the current study is reviewed in Sect. 6. The paper is concluded in Sect. 7.

2   System and Application Model

In this paper, we consider a uniprocessor system which runs a number of applications modeled as a set of static data flow graphs. Formally, a static data flow is a directed graph (V, E), where V and E represent the set of vertices and edges, respectively. Each vertex represents an *actor*. Each edge denotes a FIFO channel (also called a buffer), connecting the input port and the output port of two (not necessarily different) actors. A channel c may contain an initial number of tokens, denoted by $\bar{c}$, at the system start time. Further, each channel c has a maximum capacity of $\tilde{c}$. This means that the number of tokens existing in c should never exceed $\tilde{c}$.

Any release of one instance (job) of an actor is called a firing. An actor can be fired only when the required number of tokens are available on its input ports. During its execution, an actor consumes the required tokens from the input ports, and generates some tokens to its output ports. The number of tokens which are produced (consumed) at each firing of an actor is called the production (consumption) rate. Static data flows are classified according to the variability of an actor behavior and its production/consumption rate in different firings. In the following, three major classes, namely synchronous, homogeneous, and cyclo-static data flows [2], are reviewed.

– Synchronous Data Flow (SDF): In an SDF, the execution time as well as the production/consumption rate of each actor is fixed.
– Homogeneous Synchronous Data Flow (HSDF): An SDF is homogeneous if all production/consumption rates are equal to one.
– Cyclo-Static Data Flow (CSDF): The cyclo-static data flow (CSDF) model is a generalization of SDF, in which each actor a has a sequence of different behaviors, affecting its execution time and the production/consumption rates, which repeats cyclically [3]. Let n_a be the length of this sequence.

Then, $[f_a(1), f_a(2), \ldots, f_a(n_a)]$ represents the execution sequence of an actor $a \in V$. This means that in its ith firing, the actor execution time is given by

$$f_a\left(((i-1) \ mod \ n_a)+1\right). \tag{1}$$

Similarly, the production and consumption rates are specified by sequences of length n_a. More specifically, for an actor a, and considering a specific buffer,

- $[g_a(1), g_a(2), \ldots, g_a(n_a)]$ denotes the sequence of production rates;
- $[h_a(1), h_a(2), \ldots, h_a(n_a)]$ denotes the sequence of consumption rates.

In the current work, our focus is on the CSDF model.

Example 1. Figure 1 shows the CSDF graph of an MP3 playback application [10]. This application consists of four tasks, including MP3, Sample Rate Converter (SRC), Audio Post-Processing (APP), and Digital to Analogue Converter (DAC).
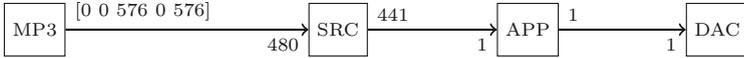
Fig. 1. A CSDF graph for the MP3 playback application [10]. Production and consumption rates are shown on the edges.

An implementation of a data flow graph is supposed to provide *liveness* and *boundedness* properties. Intuitively, liveness means that each actor will be executed infinitely many times. In contrast, boundedness necessitates the existence of a bound on the maximum size of each buffer which is never exceeded by the writing actors during the system execution.

3   Digraph Real-Time Task Model

In this section, we review the digraph real-time (DRT) task model [9]. This task model will be used in the next section for modeling CSDF graphs.

A DRT task T is specified by a directed graph $G(T) = (V(T), E(T))$, where $V(T)$ and $E(T)$ denote the graph vertices and edges, respectively. Each vertex of the graph represents a *job type*. A vertex $v \in V(T)$ is labeled by a pair $\langle e(v), d(v) \rangle$, where $e(v)$ and $d(v)$ denote the worst-case execution time (WCET) and relative deadline of the corresponding job, respectively. Further, each edge $(u, v) \in E(T)$ is labeled with a positive number, $p(u, v)$, denoting the inter-release time between the two jobs u and v[1].

Each path in the graph denotes a possible sequence of jobs which may be generated by the respective task. If the outgoing degree of each vertex in a

---

[1] In the original definition of DRT, an edge label determines the *minimum* inter-release time. Nonetheless, the DRT schedulability analyses [9,11] are valid for the modified version which we use here.

(a) A DRT task with two job types

(b) A periodic task with an initial phase of 3 modeled as a DRT task

Fig. 2. Two sample DRT tasks.

task graph is restricted by one, that is, no branching is allowed, then the model reduces to the Generalized Multiframe task model [12].

The focus of this paper is on the *constrained* deadline DRT tasks. Hence, given a DRT task T, it is assumed that for each $u \in V(T)$, we have $d(u) \leq p(u, v)$ for all $(u, v) \in E(T)$.

Example 2. A sample DRT task with two job types with inter-release times of 5 and 4 is shown in Fig. 2a. Further, Fig. 2b depicts a DRT task which models a periodic task with an initial phase.

The inherent capability of the DRT model to represent non-fixed and non-periodic behavior of a component makes it suitable for modeling CSDF graphs. In the next section, we present our method for representing CSDF graphs using set of DRT tasks.

4   Translation Method

In this section, we describe our translation method for transforming a given data flow graph to a set of DRT tasks. The method maps each actor to a real-time task. In the following, the details of the translation method and the criteria for determining the real-time task set parameters are provided.

Consider two actors a and b in a given CSDF graph. In addition, let c be a FIFO channel between them with an initial number of tokens of $\bar{c}$ and a maximum capacity of $\tilde{c}$. Let n_a be the size of the sequence which specifies the cyclically variable behavior of the actor a (as defined in the previous section). We associate a DRT task with $n_a + 1$ vertices, $v_0, \ldots, v_{n_a}$, to a. The starting vertex denotes a job type with the WCET of zero, which is used to enforce a phase (an initial phase before the release time of the first job) in the task. Additionally, for each i, $0 \leq i < n_a$, an edge is added from v_i to v_{i+1}. Also, we consider an edge from v_{n_a} to v_1. This set of edges enforce the cyclically repeating pattern of the given actor's behavior. The WCET associated with each vertex v_i, $0 < i \leq n_a$, is set to be the WCET of the ith firing of the actor, which is specified by $f_a(i)$.

As described, the DRT task corresponding to an actor a contains $n_a + 1$ vertices and $n_a + 1$ edges. Edge (v_0, v_1), which represents the phase of the task, is labeled by ϕ_a. Further, let the label of the other edges (v_i, v_{i+1}), $1 \leq i < n_a$,
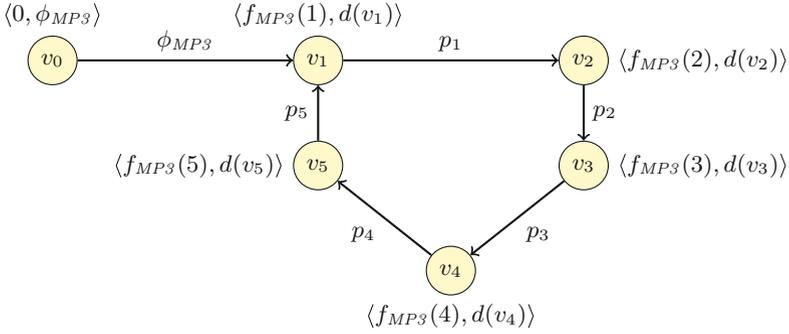
Fig. 3. DRT task for the actor MP3 of the CSDF graph presented in Fig. 1.

be denoted by $p_i{}^2$ (also, the edge (v_{n_a}, v_1) is labeled by p_{n_a}). The edge labels are parameters which should be determined such that the liveness and boundedness properties are achieved.

Example 3. Consider the data flow graph of the MP3 player application shown in Fig. 1. According to the specified translation method, the first actor (MP3) will be modeled as a DRT task with six vertices, as shown in Fig. 3. Given a sequence of execution times $[f_{MP3}(1), \ldots, f_{MP3}(5)]$ for this actor, we can assign the WCET of the job types of the DRT task as $e(v_i) = f_{MP3}(i)$, for $1 \leq i \leq 5$. Additionally, the other actors will be represented by a DRT task expressing a periodic behavior, in a similar way as shown in Fig. 2b.

For a complete translation, we need to determine the timing parameters of the DRT tasks. These parameters include the relative deadline of each job type, and the edge labels which represent the inter-release time between the jobs. The timing parameters should be assigned in such a way that the correctness conditions of the implementation are satisfied.

We use the correctness criteria in terms of the Kahn semantics [13] for a Kahn process network. As shown in [10], these criteria imply a *live* and *bounded* behavior of the system specified by a data flow graph. For this purpose, it must be guaranteed that the system never leads to a buffer overflow or buffer underflow. An overflow happens when writing to a buffer exceeds its maximum capacity. In turn, an underflow occurs whenever an actor tries to read from an empty buffer. In the following, we formalize these correctness requirements. To this end, we first need to determine the number of released and finished instances of each job (actor) up to each time instant.

---

[2] Please notice that, p_i is an actor-specific parameter. However, for brevity reasons, it is not explicitly indicated in the notation.

4.1   Number of Released/Finished Jobs

For an actor a, define $Rel_a(v,t)$ and $Fin_a(v,t)$ as follows:

$$Rel_a(v,t) \equiv \text{the number of instances of job type } v \text{ released up to and}$$
$$\text{including time } t,$$
$$Fin_a(v,t) \equiv \text{the number of instances of job type } v \text{ finished up to and}$$
$$\text{including time } t.$$

According to the translation model, the release and completion of a job corresponding to an actor firing are governed by the associated DRT tasks. As a result, $Rel_a(v,t)$ and $Fin_a(v,t)$ depend on the timing parameters of the derived DRT tasks. Here, we formally specify the relation between function $Rel_a(v,t)$ (also $Fin_a(v,t)$) and these parameters. First, it is noted that, for $t < \phi$, we have $Rel_a(v,t) = 0$ and $Fin_a(v,t) = 0$. Thus, in the following, we assume that $t \geq \phi$.

According to the defined notations, we can specify $Rel_a(v,t)$ as

$$Rel_a(v_i,t) = 1 + \left\lfloor \frac{t - \phi_a - \sum_{j=1}^{i-1} p_j}{\pi_a} \right\rfloor. \tag{2}$$

where $\pi_a = \sum_{j=1}^{n_a} p_j$ is the *super-period* of the DRT task. In words, π_a denotes the amount of time that it takes for the DRT to have a complete cycle, through which, each job (except the first job which represents the initial phase) is released exactly once. Also, a lower bound for $Fin_a(v,t)$ can be obtained by

$$Fin_a(v_i,t) \geq 1 + \left\lfloor \frac{t - \phi_a - \sum_{j=1}^{i-1} p_j - d(v_i)}{\pi_a} \right\rfloor. \tag{3}$$

It is worth noting that the equality does not necessarily hold. This is because that, depending on the scheduling approach, a job may be completed before its deadline, leading to a possibly higher number of finished jobs up to time t compared to the case in which the job finishes exactly at its deadline.

4.2   Underflow Analysis

Based on the semantics of a data flow graph, an actor may produce (consume) the output (input) tokens at any time during its execution. As a result, for the underflow analysis, we employ a pessimistic approach [10], in which, we consider the minimum possible number of tokens that may be buffered at each instant. Based on this approach, it is assumed that each actor writes to the output buffer(s) as late as possible. In other words, the tokens are assumed to be written to the buffer when the actor completes its execution. On the other hand, we suppose that each actor reads from its input buffer(s) as soon as possible, namely at its release instant.
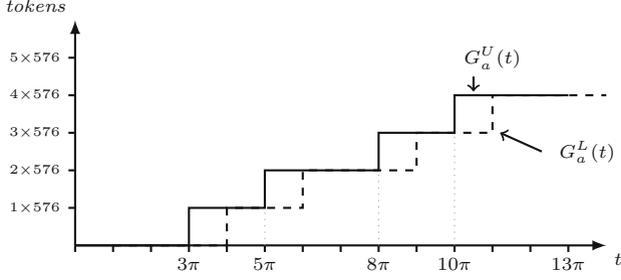
Fig. 4. Upper and lower bound on the number of produced tokens for the MP3 actor.

Regarding the abovementioned pessimistic assumptions, a lower bound for the total number of tokens written to the channel c by an actor a up to (and including) time t can be calculated as

$$G_a^L(t) = \sum_{i=1}^{n_a} g_a(i) \times Fin_a(v_i, t). \tag{4}$$

As an example, the dashed line in Fig. 4 depicts the function $G_a^L(t)$ for the MP3 actor of the CSDF graph specified in Example 1. In this example, it is assumed that any two successive firings are identically separated by a time interval of length π. Further, an implicit deadline has been considered (i.e. $d(v_i) = \pi$).

In addition, an upper bound for the total number of tokens read from a channel c by an other actor b up to and including time t is given by

$$H_b^U(t) = \sum_{i=1}^{n_b} h_b(i) \times Rel_b(v_i, t). \tag{5}$$

According to these relations, a sufficient condition for the underflow avoidance of channel c is formulated by

$$\forall t \geq 0 : \bar{c} + G_a^L(t) - H_b^U(t) \geq 0, \tag{6}$$

where $\bar{c}$ denotes the initial number of tokens of c.

4.3   Overflow Analysis

Based on an approach similar to the one presented for the underflow, we can specify a sufficient condition for overflow avoidance. In this case, the pessimistic assumptions are stated as follows:

– Each actor writes to the output buffer(s) as soon as possible (namely at its release time);
– Each actor reads from its input buffer(s) as late as possible (namely at its finish time).

Consequently, the maximum number of tokens written to a buffer c by an actor a up to time t can be specified as

$$G_a^U(t) = \sum_{i=1}^{n_a} g_a(i) \times Rel_a(v_i, t). \tag{7}$$

Figure 4 partly shows the variation of $G_a^U(t)$ for the MP3 CSDF graph, under the previously mentioned assumptions.

Additionally, the minimum number of tokens read from a buffer c by an actor b up to time t is given by

$$H_b^L(t) = \sum_{i=1}^{n_b} h_b(i) \times Fin_b(v_i, t). \tag{8}$$

Regarding the defined notations, no overflow happens if the following condition holds (recall that $\tilde{c}$ denotes the maximum buffer capacity)

$$\forall t \geq 0 : \bar{c} + G_a^U(t) - H_b^L(t) \leq \tilde{c}. \tag{9}$$

4.4   Design Space Exploration

Relations (6) and (9) provide sufficient conditions for the correctness of an implementation of a data flow graph. Then, the problem is to assign suitable values to the DRT tasks parameters, namely their inter-release times, p_i, and relative deadlines, $d(v_i)$, such that, while the mentioned conditions are satisfied, some design objective, e.g. the application throughput, is optimized. Furthermore, from a schedulability point of view, these values must be selected such that the obtained task set is schedulable, i.e., each job can complete its execution no later than its deadline. This can be checked using efficient methods proposed in [9,11] for static-priority and dynamic-priority schedulability analysis of DRT task sets.

Here, we discuss a simplifying technique for improving the efficiency of the state-space exploration. First, it is observed that the correctness criteria derived in the previous sections are independent of the worst-case execution time of the actors. In other words, they deal only with the release times and completion times. As a result, the problem of finding an appropriate value assignment to the timing parameters can be first solved irrespective of the schedulability concerns. Afterwards, we have to consider the schedulability of the system. If the system with the derived value assignments is not schedulable, one can easily scale up the timing parameters such that the obtained task set becomes schedulable.

It is worth noting that scaling up all the parameters by the same amount does not affect the correctness of the system, i.e. the validity of (6) and (9). This is because that, in this situation, the numerator and denominator of the respective fractions in (2) and (3) are scaled with the same factor, and the value of the fraction remains unchanged. It should be noted that a similar approach, called abstraction-refinement, has been previously used for the periodic task model [6] to overcome the complexity of the problem. In addition to this technique,

another simplification can be made by a linear approximation, which is specified in the following.

Linear Approximation: Based on Relations (2) and (3), it is observed that the functions $Rel_a(v_i, t)$ and $Fin_a(v_i, t)$, respectively, can be over-approximated and under-approximated by some linear functions. These approximations are constructed on the basis of the inequality $x < 1 + \lfloor x \rfloor \leq 1 + x$, which holds for any real number x [14]. Using this, we specify an overapproximation for the function $Rel_a(v_i, t)$ as

$$Rel_a^{UApp}(v_i, t) = 1 + \frac{t - \phi_a - \sum_{j=1}^{i-1} p_j}{\pi_a}. \tag{10}$$

In fact, for any $t \geq 0$, we have $Rel_a(v_i, t) \leq Rel_a^{UApp}(v_i, t)$. In addition, we can obtain an under-approximation for the function $Fin_a(v_i, t)$ as

$$Fin_a^{LApp}(v_i, t) = \frac{t - \phi_a - \sum_{j=1}^{i-1} p_j - d(v_i)}{\pi_a}. \tag{11}$$

These approximate functions can be used in calculating $G_a^L(t)$, $H_b^U(t)$, $G_a^U(t)$, and $H_b^L(t)$, defined in Eqs. (4), (5), (7), and (8). Then, we can rewrite the underflow and overflow avoidance conditions, presented in (6) and (9), based on these approximations. In the following, we elaborate the underflow condition; the procedure for the overflow condition can be done in a similar manner.

Using the provided approximations, we can rewrite the underflow avoidance condition as

$$\forall t \geq 0 : \bar{c} + \sum_{i=1}^{n_a} g_a(i) \times Fin_a^{LApp}(v_i, t) - \sum_{i=1}^{n_b} h_b(i) \times Rel_b^{UApp}(v_i, t) \geq 0.$$

Moreover, by replacing $Fin_a^{LApp}(v_i, t)$ and $Rel_a^{UApp}(v_i, t)$ from (11) and (10), we will get

$$\forall t \geq 0 : \tag{12}$$
$$\bar{c} + \sum_{i=1}^{n_a} g_a(i) \left(\frac{t - \phi_a - \sum_{j=1}^{i-1} p_j - d(v_i)}{\pi_a} \right) - \sum_{i=1}^{n_b} h_b(i) \left(1 + \frac{t - \phi_b - \sum_{j=1}^{i-1} p_j}{\pi_b} \right) \geq 0$$

From [10], it is known that, as a necessary condition for overflow and underflow avoidance, the average production rate for any buffer must be equal to its average consumption rate, namely

$$\frac{\sum_{i=1}^{n_a} g_a(i)}{\pi_a} = \frac{\sum_{i=1}^{n_b} h_b(i)}{\pi_b}. \tag{13}$$

Based on this fact, we can simplify the inequality specified in (12) as

$$\forall t \geq 0 :$$
$$\bar{c} + \sum_{i=1}^{n_a} g_a(i) \left(\frac{-\phi_a - \sum_{j=1}^{i-1} p_j - d(v_i)}{\pi_a} \right) - \sum_{i=1}^{n_b} h_b(i) \left(1 + \frac{-\phi_b - \sum_{j=1}^{i-1} p_j}{\pi_b} \right) \geq 0.$$

Also, from (13), we can write π_b as a linear function of π_a, that is, $\pi_b = \gamma\pi_a$ for some constant γ. Hence, we have

$$\forall t \geq 0:$$

$$\bar{c}\pi_a + \sum_{i=1}^{n_a} g_a(i)\left(-\phi_a - \sum_{j=1}^{i-1} p_j - d(v_i)\right) - \sum_{i=1}^{n_b} h_b(i)\left(\pi_a + \frac{-\phi_b - \sum_{j=1}^{i-1} p_j}{\gamma}\right) \geq 0.$$

As seen, the obtained relation specifies a *linear* constraint on the problem parameters, which significantly reduces the complexity of the problem.

5   Evaluation

In this section, we evaluate the effectiveness of the proposed approach compared to a previously proposed method which employs a periodic task model for the analysis of CSDF graphs [6,10]. We compare the two methods in terms of the throughput [15] and the buffer size requirements [16]. The throughput of a dataflow graph measures how often the application is executed in a unit of time. We assume the preemptive EDF algorithm for scheduling the obtained real-time tasks.

   For the evaluation purpose, we apply the mentioned methods to the MP3 playback application shown in Fig. 1. According to [10], the execution time of the MP3 actor is specified as the sequence $f_{MP3}(.) = [670, 2700, 720, 2700, 720]\,\mu s$. Further, the execution time of SRC, APP, and DAC are specified as $2500\,\mu s$, $22\,\mu s$, and $22\,\mu s$, respectively.

   The primary objective is to specify the timing parameters of the task set so as to minimize the total required buffer sizes, while the correctness criteria specified in (6) and (9) are respected and the task set is EDF-schedulable. As well, it is desired to increase the application throughput. In the following, we first present the obtained task sets for each approach. Then, the buffer requirement and the throughput achieved by each method are reported and discussed.

5.1   Obtained Task Sets

In this section, we first specify the periodic task set obtained in [10] for the MP3 playback application. Next, the corresponding DRT task set is described.

Periodic Task Model: According to the approach utilized in [10], a periodic task is considered for each actor. In order to have a safe analysis, one needs to consider the maximum execution time of each actor as the WCET of the corresponding periodic task. As a result, a WCET of $\max\{670, 2700, 720, 2700, 720\} = 2700\,\mu s$ is considered for the task associated to the MP3 actor. As the other actors have a fixed execution time, WCET of the respective tasks are simply set to those fixed values. The periods and phases assigned to the tasks according to this method are shown in Table 1 [10]. This parameter assignment leads to the system utilization of 99.96 %, which reveals the schedulability of the task set.

DRT Task Model: As pointed out before, the MP3 application can be modeled by four DRT tasks. When constructing the tasks, in order to decrease the number

Table 1. Task set parameters obtained for the periodic tasks [10]

	Period (μs)	Phase (μs)
MP3	13219.416	0
SRC	27540.45	66647.889
APP	62.45	121760.014
DAC	62.45	121916.139

of design parameters, we assume that the relative deadline of each job type is set to be equal to the inter-release time between that job and the next one. As noted in Example 3, for the actors SRC, APP, and DAC we can use the DRT task structure which models a periodic task with a specific phase. This is because these actors have a periodically repeating behavior. On the other hand, the MP3 actor is modeled by a DRT task with six different job types, as shown in Fig. 3. It is worth noting that, here, as opposed to the periodic task model, we can consider the actual pattern of the execution times for the MP3 actor, instead of using one conservative maximum value. The goal is to assign the relative deadline of each job such that the problem objective is optimized.

Initially, we use the same values reported in Table 1 for the DRT tasks associated to SRC, APP, and DAC. Additionally, for the DRT task related to MP3, we assume that the inter-release times, in the average, are equal to the period specified for the corresponding periodic task. As a result, the super-period of this task is $\pi_{MP3} = 5 \times 13219.416$. Now, we attempt to determine the concrete value of the inter-release times for each pair of job types of this task. In order to decrease the utilization of the task (and hence, increase the schedulability of the task set), we assign the relative deadline of each job (or equivalently, the inter-release time between that job and the next one) in proportion to its execution time. Since in the DRT task, we can consider the actual pattern of execution times instead of a fixed and pessimistic value (which is done in the periodic task model), the total utilization is lower than that of the periodic task. As a result, we can scale down the timing parameters, namely the phases and inter-release times, so as to increase the application throughput, while the task set is still schedulable. The results of this approach are shown in Table 2 and Fig. 5.

Table 2. Task set parameters for the DRT tasks (μs)

	Period	Phase
SRC	25061.809	60649.578
APP	56.829	110801.612
DAC	56.829	110943.686
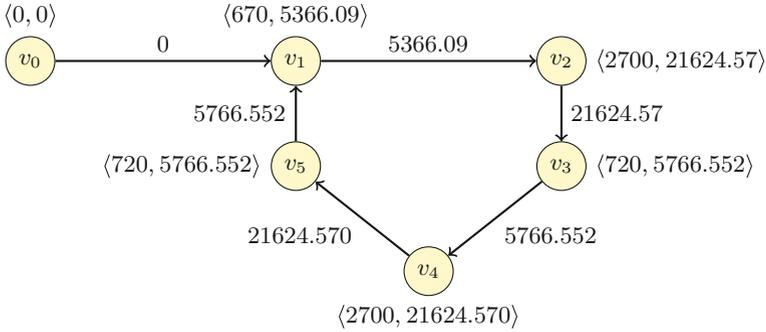
Fig. 5. Parameters for the DRT task which models the MP3 actor (μs).

5.2   Evaluation Results

The total buffer requirement and the throughput which is achieved by the two approaches are reported in Table 3. As seen, the DRT-based method outperforms the other one in terms of both the buffer requirement and the application throughput.

Table 3. Total buffer requirement and throughput for each method

	Buffer requirement	Throughput (s^{-1})
Periodic task set	2273	16013
DRT task set	2155	17596
Improvement	5%	9.8%

As a conclusion, it is seen that the DRT-based approach provides a higher degree of flexibility in the design of data flow graphs which can lead to better solutions. Of course, this advantage is achieved at the cost of treating more parameters, which means a larger state-space which must be explored.

6   Related Work

Synchronous Data Flow (SDF) [1] and Cyclo-Static Data Flow (CSDF) [2] are two very basic data flow models. In the past, several variants of these models have been proposed to provide more expressiveness and flexibility in the design of streaming applications. For instance, the *parametric* extensions of the SDF have been developed [17–19] which allow the data flow graph properties, such as the production and consumption rates, to be changed at runtime. In particular, Boolean Parametric Data Flow (BPDF) [18] is a parametric model in which the graph topology can be changed as well as the production and consumption rates

of the actors. In this model, an edge can be labeled with a boolean expression which is modified by some actor. At runtime, according to the actual value of the boolean expression, an edge may be enabled or disabled, determining whether the edge should be considered in the firing of the actors at that moment. An assumption made in the related stuies, such as [18–20], is that each actor runs in a dedicated core. Hence, when analyzing the data flow graph, one does not need to take into account the interference of the actors (caused by resource contention) on each other. While this approach provides a high degree of predictability for a single data flow, it is not easily extendible to incorporate the impact of multiple data flows on each other when they are running on the same processing platform with possible resource contention.

Meanwhile, due to the increasing use of real-time operating systems in complex embedded systems which work in dynamic environments, using dynamic scheduling policies, such as rate-monotonic and earliest-deadline first (EDF), for SDFs has been considered recent studies. The advantage of this approach is that the already existing analyses for different scheduling algorithms can be used in this context. This provides the possibility of running multiple applications on the same processing resource, while the interfering effects is considered. One approach to utilize this facility is to use a set of independent real-time tasks to reflect the timing behavior of data flow graphs. One of the basic studies which use periodic real-time task model for data flow graphs is presented by Bmakhra and Stefanov [3]. They explore that how the execution of actors can be parallelized to achieve a maximum throughput. In the same realm, Ali et al. [8] consider the problem of assigning parameters of periodic tasks modeling an HSDF. They suppose a given set of applications each one modeled as an HSDF graph. Each application exhibits two kinds of requirements: a minimum throughput, which is the minimum output data rate (or iteration rate of the whole graph); and one or multiple *latency* constraints put on a number of pairs of actors. A latency constraint is a timing constraint between firing of two actors located on a path. While they consider more constraints compared to the model considered in this paper, their work is specific to HSDF, which is less expressive compared to the CSDF.

Moreover, Bouakaz et al. considered a more general category of data flow graphs. They extended the CSDF model by introducing ultimately periodic CSDF [10] in which the system behavior becomes repetitive after a finite interval, but it is not needed to be periodic from the beginning. They define the affine firing relation which specifies the condition under which a data flow implementation can satisfy the correctness criteria. They investigate the correctness of the implementation based on the periodic task model. In their work, the correctness conditions of an implementation, including boundedness, completeness, and soundness, are obtained based on the Kahn process network semantics [13].

The work presented in [6] can be regarded as one of the most related work to the current study. They consider a CSDF model with a set of buffer size constraints. The goal is to construct a set of periodic tasks reflecting the execution of the given SDFs. The main difference of that work compared to our approach is that we use a more expressive real-time task model, which suggests more flexibility, and thus, a higher degree of schedulability. This, in turn, allows to look for more efficient solutions.

7   Conclusion

In this paper, we proposed a formal translation method for converting a given set of data flow graphs to a graph-based real-time task model. We focused on cyclo-static data flow graphs in which an actor behavior, including its worst-case execution time, consumption rate, and production rate, is not necessarily fixed in different firings. We presented sufficient conditions for a correct translation in terms of liveness and boundedness of data flow graphs. The proposed method provides the opportunity of exploring a larger state-space for finding optimal or near optimal solutions for the design of corresponding applications. Based on the translated task model, one can easily perform analyses such as schedulability tests, while taking into account the interfering of the applications running on the same processing core.

The proposed approach can be extended by employing efficient optimization methods for finding task set parameters such that, while the design constraints are met, design objectives like the total buffer size or the application throughput are optimized. In addition, when the DRT tasks exhibit a restricted structure in which only a single cycle is contained, they can be modeled as a set of Generalized Multiframe (GMF) Tasks [12]. In this case, one may employ more efficient analysis methods specific to this task model for schedulability test of the translated tasks.

References

1. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proc. IEEE **75**(9), 1235–1245 (1987)
2. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cycle-static dataflow. IEEE Trans. Sig. Process. **44**(2), 397–408 (1981)
3. Bamakhrama, M., Stefanov, T.: Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In: International Conference on Embedded Software, pp. 195–204 (2011)
4. Dkhil, A., Do, X.K., Dubrulle, P., Louise, S., Rochange, C.: Self-timed periodic scheduling for a cyclo-static dataflow model. In: International Conference on Computational Science, pp. 1134–1145 (2014)
5. Lele, A., Moreira, O., Bastos, J., Almeida, R., Pedreiras, P., van Berkel, K.: Analyzing preemptive fixed priority scheduling of data flow graphs. In: 12th Symposium on Embedded Systems for Real-time Multimedia, pp. 50–59 (2014)
6. Bouakaz, A., Gautier, T.: An abstraction-refinement framework for priority-driven scheduling of static dataflow graphs. In: 12th ACM/IEEE International Conference on Formal Methods and Models for Codesign, pp. 2–11 (2014)
7. Do, X.K., Dkhil, A., Louise, S.: Self-timed periodic scheduling of data-dependent tasks in embedded streaming applications. In: Wang, G., Zomaya, A., Martinez Perez, G., Li, K. (eds.) ICA3PP 2015. LNCS, vol. 9529, pp. 458–478. Springer, Heidelberg (2015). doi:10.1007/978-3-319-27122-4_32
8. Ali, H.I., Akesson, B., Pinho, L.M.: Generalized extraction of real-time parameters for homogeneous synchronous dataow graphs. In: 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 701–710 (2015)

9. Stigge, M., Ekberg, P., Guan, N., Yi, W.: The digraph real-time task model. In: Real-Time and Embedded Technology and Applications Symposium, pp. 71–80 (2011)
10. Bouakaz, A., Talpin, J.P., Vitek, J.: Affine data-flow graphs for the synthesis of hard real-time applications. In: 12th International Conference on Application of Concurrency to System Design, pp. 183–192 (2012)
11. Stigge, M., Yi, W.: Combinatorial abstraction refinement for feasibility analysis. In: 34th IEEE Real-Time Systems Symposium, pp. 340–349 (2013)
12. Baruah, S., Chen, D., Gorinsky, S., Mok, A.: Generalized multiframe tasks. J. Real-Time Syst. **17**(1), 5–22 (1999)
13. Geilen, M., Basten, T.: Requirements on the execution of Kahn process networks. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 319–334. Springer, Heidelberg (2003)
14. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics - A Foundation for Computer Science. Addison-Wesley, Reading (1989)
15. Ghamarian, A.H., Geilen, M.C.W., Stuikj, S., Basten, T., Theelen, B.D., Mousavi, M.R., Moonen, A.J.M., Bekooij, M.J.G.: Throughput analysis of synchronous data flow graphs. In: 6th International Conference on Application of Concurrency to System Design, pp. 25–36 (2006)
16. Benazouz, M., Marchetti, O., Munier-Kordon, A., Michel, T.: A new method for minimizing buffer sizes for cyclo-static dataflow graphs. In: 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia, pp. 11–20 (2010)
17. Fradet, P., Girault, A., Poplavko, P.: SPDF: a schedulable parametric data-flow MoC. In: Design, Automation and Test in Europe Conference and Exhibition, pp. 769–774 (2012)
18. Bebelis, V., Fradet, P., Girault, A., Lavigueur, B.: BPDF: a statically analyzable dataflow model with integer and boolean parameters. In: International Conference on Embedded Software, pp. 3:1–3:10 (2013)
19. Bouakaz, A., Fradet, P., Girault, A.: Symbolic analysis of dataflow graphs (extended version). Doctoral dissertation, Inria-Research Centre Grenoble-Alpes (2016)
20. Bebelis, V., Fradet, P., Girault, A.: A framework to schedule parametric dataflow applications on many-core platforms. In: Conference on Languages, Compilers and Tools for Embedded Systems, pp. 125–134 (2014)