Liveness Evaluation of a Cyclo-Static DataFlow Graph

Mohamed Benazouz
CEA, LIST,
P.C. 172, 91191
Gif-Sur-Yvette, France.
mohamed.benazouz@cea.fr

Alix Munier-Kordon
LIP6, UPMC,
Place Jussieu, 75005
Paris, France.
alix.munier@lip6.fr

Thomas Hujsa
LIP6, UPMC,
Place Jussieu, 75005
Paris, France.
thomas.hujsa@lip6.fr

Bruno Bodin
KALRAY SA,
86 Rue de Paris, 91400
Orsay, France.
bruno.bodin@kalray.eu

ABSTRACT

Cyclo-Static DataFlow Graphs (CSDFG in short) is a formalism commonly used to model parallel applications composed by actors communicating through buffers. The liveness of a CSDFG ensures that all actors can be executed infinitely often. This property is clearly fundamental for the design of embedded applications.

This paper aims to present first an original sufficient condition of liveness for a CSDFG. Two algorithms of polynomial-time for checking the liveness are then derived and compared to a symbolic execution of the graph. An original method to compute close-to-optimal buffer capacities ensuring liveness is also presented and experimentaly tested. The performance of our methods are comparable to those existing in the literature for industrial applications. However, they are far more effective on randomly generated instances, ensuring their scalability for future more complex applications and their possible implementation in a compiler.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Algorithms, Design, Experimentations, Theory

Keywords

Cyclo-Static Dataflow Graphs, liveness, buffer sizing

1. INTRODUCTION

Synchronous Dataflow Graphs [12] (SDFG in short) have been used for many years in the field of embedded system design such as Digital Signal Processing (DSP in short). They are used to model a large amount of applications [16, 17] and many academic results were devoted to them [9, 11, 16, 18]. However, this model is inadequate in many applications for modeling the communication between actors. Bilsen *et al.* [5] introduced Cyclo-Static Dataflow Graphs which is a more accurate model to address this problem. Actors' exchanges are more detailed and this new model corresponds better to the description of applications. Analysis results are thus more pertinent. Besides, a CSDF live application may deadlock when modeled by a SDF[5].

CSDFGs were considered more recently in many areas to model data exchanges between applications processes. Those graphs are automatically extracted from a suitable description of the applications. In the field of synchronous languages, Mandel *et al.* improved the expressivity of Lustre [6] to handle processes of different rates communicating through buffers [13]. The intermediate representation of this extended language, Lucy-n, is then comparable to a CSDFG. CSDFGs are also considered to model embedded applications for its mapping on a parallel architecture. Several studies were performed in an academic context [1, 3, 19]. Another example is the dataflow compiler designed to map a CSDFG on the Massively Parallel Processor Array (MPPA in short) developed by Kalray company [10] that embeds 256 processors on a 28nm chip.

The popularity of CSDFGs comes from the fact that it is a decidable model. Its behavior is completely predictable, and its performances can be analysed. The aim of this paper is to provide an efficient method to evaluate the liveness of a CSDFG. A CSDFG is said to be live if all its actors can be executed with no deadlock. This property is clearly essential for applications. The main problem is that all algorithms developed to check it are of exponential time complexity and thus they cannot be integrated in an iterative compilation context [2, 5]. A detailed bibliography can be found in Subsection 2.2.

The main result of our study is to prove the first sufficient polynomial condition of liveness for CSDFGs. We deduce several polynomial time algorithms to ensure the liveness of a CSDFG and to compute the minimum buffer sizes. All of them were tested on industrial and academic benchmarks and compared to existing solutions.

Section 2 is dedicated to the presentation of CSDFGs and some behavioural properties. Section 3 describes the extension to CSDFGs of a polynomial transformation called normalization. It was previously introduced for SDFG by [15] and allowed to get a sufficient condition of liveness for SDFG. This sufficient condition is extended to CSDFG in Section 4 and two polynomial time algorithms for ensuring the liveness of CSDFG are deduced. Section 5 presents our experiments. Section 6 is our conclusion.

2. CYCLO-STATIC DATAFLOW GRAPHS

This section introduces Cyclo-Static Dataflow Graphs and some important basic definitions and properties. Basic notations are first introduced in Subsection 2.1. The liveness of a CSDF is introduced in Subsection 2.2 followed by a short bibliography on the methods developed to check it. Subsection 2.3 introduces the consistency of a CSDFG, that can be seen as a necessary condition on the liveness. As mentioned above, the complexity of the liveness of a CSDFG is a fundamental open problem and all the exact algorithms are of exponential time-complexity.

2.1 Notations for CSDFGs

A Cyclo-Static DataFlow Graph $\mathcal{G} = (T, A)$ is a directed graph; the set of nodes T models tasks (or actors); the set of arcs A corresponds to buffers (or channels).

2.1.1 Actors

Every actor $t \in T$ is decomposed into $\varphi(t) \in \mathbb{N} - \{0\}$ distinct phases that constitute a periodic execution sequence $t_1, \cdots, t_{\varphi(t)}$ where t_k denotes the kth phase of t for $k \in \{1, \cdots, \varphi(t)\}$. Moreover, two phases or two successive executions of an actor are supposed to not overlap.

We denote by $\langle t, n \rangle$, $n \in \mathbb{N} - \{0\}$, the n^{th} execution of t. Similarly, for every phase $k \in \{1, \cdots, \varphi(t)\}$, $\langle t_k, n \rangle$ denotes the n^{th} execution of the k^{th} phase of t.

For every couple $(k, n) \in \{1, \cdots, \varphi(t)\} \times \mathbb{N} - \{0\}$, $Pred\langle t_k, n \rangle$ is the preceding execution phase of $\langle t_k, n \rangle$. More formally,

$$Pred\langle t_k, n \rangle = \begin{cases} \langle t_{k-1}, n \rangle & if\ k > 1 \\ \langle t_{\varphi(t)}, n-1 \rangle & if\ k = 1. \end{cases}$$

The execution $\langle t_{\varphi(t)}, 0 \rangle$ is fictitious and is only introduced to simplify the definition of $Pred$.

2.1.2 Buffers

Every arc $a = (t, t') \in A$ represents a buffer $b(a)$ from the actor t to the actor t'. $\forall k \in \{1, \cdots, \varphi(t)\}$, $w_a(k) \geq 0$ data are produced in $b(a)$ at the end of an execution of t_k. To enable the execution of the phase $t'_{k'}$, $\forall k' \in \{1, \cdots, \varphi(t')\}$, $v_a(k') \geq 0$ data are needed to be available in $b(a)$. They are consumed before $t'_{k'}$ starts its execution. Moreover, a buffer associated with an arc a contains initially $M_0(a)$ data (or tokens).

The cumulative number of data produced on $b(a)$ by one execution of the actor t equals $w_a \cdot \mathbb{1} = \sum_{k=1}^{\varphi(t)} w_a(k)$. Similarly, the cumulative number of data consumed from $b(a)$ by one execution of the actor t' is $v_a \cdot \mathbb{1} = \sum_{k=1}^{\varphi(t')} v_a(k)$. While it is allowed that a phase does not produce (resp. consume) data, the cumulative number of data produced (resp. consumed) must be not null, *i.e.* $w_a \cdot \mathbb{1} > 0$ (resp. $v_a \cdot \mathbb{1} > 0$).

For any value $n \in \mathbb{N} - \{0\}$ and $k \in \{1, \cdots, \varphi(t)\}$, let $D_a^+\langle t_k, n \rangle$ denote the total number of tokens produced on the arc a by executions of t until the end of $\langle t_k, n \rangle$. Similarly, for any value $k' \in \{1, \cdots, \varphi(t')\}$, let $D_a^-\langle t'_{k'}, n \rangle$ denote the total number of tokens consumed from the arc a from the beginning for the execution of $\langle t'_{k'}, n \rangle$.

The total number of tokens contained in a buffer must remain non negative, that is to say any execution $\langle t'_{k'}, n' \rangle$ can be done at the completion of $\langle t_k, n \rangle$ if $M_0(a) + D_a^+\langle t_k, n \rangle - D_a^-\langle t'_{k'}, n' \rangle \geq 0$.

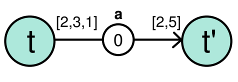

Figure 1: A buffer $b(a)$ represented by an arc $a = (t, t')$. The arc is labeled by two vectors $w_a = [2, 3, 1]$, $v_a = [2, 5]$ and by the initial number of data $M_0(a) = 0$.

Figure 1 shows a buffer $b(a)$ from t to t' that is modeled by an arc $a = (t, t')$. t (resp. t') has three (resp. two) phases *i.e.*, $\varphi(t) = 3$ (resp. $\varphi(t') = 2$). The arc is labeled by vectors of production/consumption rates $w_a = [2, 3, 1]$ and $v_a = [2, 5]$. The total number of data produced in a until the completion of $\langle t_2, 2 \rangle$ is $D_a^+\langle t_2, 2 \rangle = 6 + 5 = 11$. Similarly, the total number of tokens consumed for execution $\langle t'_1, 2 \rangle$ equals $D_a^-\langle t'_1, 2 \rangle = 7 + 2 = 9$. As $M_0(a) + D_a^+\langle t_2, 2 \rangle - D_a^-\langle t'_1, 2 \rangle = 0 + 11 - 9 \geq 0$, $\langle t'_1, 2 \rangle$ can be executed at the completion of $\langle t_2, 2 \rangle$.

An arc $a = (t, t')$ models data-dependencies introduced by a buffer $b(a)$ between the actors t and t'. However, in most real-life embedded systems, the overall amount of memory is bounded, which implies that the capacity of the buffers cannot be considered as infinite. The bounded capacity of a buffer $b(a)$ is simply modeled by adding a feedback arc $a' = (t', t)$ with $\forall k \in \{1, \cdots, \varphi(t)\}$, $v_{a'}(k) = w_a(k)$ and $\forall k' \in \{1, \cdots, \varphi(t')\}$, $w_{a'}(k') = v_a(k')$. The initial marking $M_0(a')$ corresponds to the number of empty containers initially in $b(a)$. The capacity of the buffer $b(a)$ equals the sum $M_0(a) + M_0(a')$ (see Figure 2).

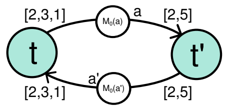

Figure 2: A bounded buffer $b(a)$ modeled by a couple of arcs $a = (t, t')$ and $a' = (t', t)$. The capacity of the buffer $b(a)$ equals $M_0(a) + M_0(a')$.

2.1.3 Particular classes of CSDFG

Synchronous DataFlow Graphs (SDFGs in short) [12] are a particular class of CSDFGs where each actor has only one phase, *i.e.* $\forall t \in T, \varphi(t) = 1$. Also note that any CSDFG $\mathcal{G} = (T, A)$ can be associated with an SDFG denoted by $SDFG(\mathcal{G}) = (T, A')$ where each arc $a = (t, t') \in A$ is associated with an arc $a' = (t, t')$ in A' with $w'_a = w_a \cdot \mathbb{1}$, $v'_a = v_a \cdot \mathbb{1}$ and $M_0(a') = M_0(a)$.

Homogeneous Synchronous DataFlow Graphs (HSDFGs in short) [17] are a particular class of SDFGs where $w_a = v_a = 1$ for any arc $a \in A$.

These two classes of graphs were intensively studied in the literature. Main results concerning their liveness are reviewed subsequently.

2.2 Related work on the liveness of a CSDFG

A CSDFG is said to be live if each actor can be fired infinitely often. The liveness of a CSDFG is an important basic property: finding efficient algorithms to check the liveness of such system is of great importance in an industrial context.

The complexity of the liveness of an SDFG or a CSDFG is an open problem. Up to now, all the exact algorithms for checking the liveness are of exponential time. Their main defect is that they cannot be used within a reasonable time for complex applications.

They can be grouped into two main classes: the first one is transforming an original SDFG or CSDFG into an equivalent HSDFG by replicating the actors a certain (non-polynomial) number of times [5, 17]. The liveness is then checked directly on the HSDFG, which is possible using a polynomial-time algorithm (recall that a HSDFG is live iff every circuit has at least one token). The problem is that the size of the HSDFG may be of exponential size [15], which drastically limits the efficiency of this class of methods.

Another way consists in constructing if possible a static schedule for an SDFG [8, 11] or a CSDFG [2]. If such a schedule exists, the graph is live; otherwise a deadlock is highlighted. The main drawback of this method is that the size of the static schedule may be quite large and so it cannot be computed in polynomial time.

A well-known necessary condition for the liveness of a CSDFG (or an SDFG) with bounded buffers is the consistency, as described in [12, 5]. This condition is recalled in the next subsection, and is supposed to be fulfilled by the SDFG and CSDFG considered here. A simple polynomial sufficient condition was found by [14] for *normalized* SDFG. This condition is proved to be not necessary, but allows to ensure quickly that the system is live. This paper aims to prove first that any consistent CSDFG may be normalized, and that several sufficient conditions of liveness may be obtained by expressing deadlock conditions.

2.3 Consistency of a CSDFG

Consistency is a necessary (non sufficient) condition for the existence of a valid schedule within bounded memory that was established first for SDFGs [12]. Bilsen *et al.* [5] extended this condition to CSDFG by considering the cumulative number of tokens produced/consumed by one execution of its actors. It is none other than the one proposed by Lee [12] applied to the associated SDFG of a CSDFG. This point is motivated by the fact that a CSDFG is simply obtained by refining the modeling of the data exchanges between actors of the underlying SDFG.

Let us consider the pre-post $|A| \times |T|$ matrix Γ associated with a CSDFG $\mathcal{G}$ defined by

$$\Gamma_{at} = \begin{cases} w_a \cdot \mathbb{1} & \text{if } a = (t, t'),\ t' \in T \\ -v_a \cdot \mathbb{1} & \text{if } a = (t', t),\ t' \in T \\ 0 & \text{Otherwise.} \end{cases}$$

The CSDFG is said to be *consistent* if the rank of Γ is $|T| - 1$.

In the following, we restrict our study to consistent strongly connected CSDFGs as not consistent graphs will either deadlock or need unbounded buffers.

3. NORMALIZATION OF A CSDFG

The *normalization* of a consistent SDFG is an operation introduced in [14] which simplifies the arc values without any influence on the data dependencies. This transformation can be simply extended to any CSDFG as long as its underlying SDFG is consistent. The normalization of an SDFG and a CSDFG are introduced and illustrated using a simple example in this section.

For any actor $t \in T$, let us denote by $\mathcal{A}^+(t) = \{a = (t, t') \in A, t' \in T\}$ the set of output arcs of t and $\mathcal{A}^-(t) = \{a = (t', t) \in A, t' \in T\}$ the set of input arcs.

An actor t is said to be *normalized* if there exists $Z_t \in \mathbb{N} - \{0\}$ such that $\forall a \in \mathcal{A}^+(t), w_a \cdot \mathbb{1} = Z_t$, and $\forall a \in \mathcal{A}^-(t), v_a \cdot \mathbb{1} = Z_t$.

A CSDFG is *normalized* if all of its actors are normalized.

The *normalization* of a CSDFG consists in building an equivalent CSDFG such that all actors are normalized. The idea here is to find two vectors $Z = (Z_1, \cdots, Z_{|T|})$ and $\Delta = (\delta_1, \cdots, \delta_{|A|})$ of positive values such that

$$\text{and} \quad \begin{array}{l} \forall t \in T, \forall a \in \mathcal{A}^+(t), \delta_a \times (w_a \cdot \mathbb{1}) = Z_t \\ \forall t \in T, \forall a \in \mathcal{A}^-(t), \delta_a \times (v_a \cdot \mathbb{1}) = Z_t. \end{array}$$

It has been proved in [14] that every consistent SDFG can be normalized (*i.e.* the previous system has a solution). Now, if a CSDFG $\mathcal{G}$ is consistent, then so is $SDFG(\mathcal{G})$ which is normalizable; let $Z^\star$ and $\Delta^\star$ be a solution of the corresponding previous system. Vectors associated with any arc $a \in A$ and the initial markings are then replaced respectively by $\delta^\star_a \times w_a$, $\delta^\star_a \times v_a$ and $\delta^\star_a \times M_0(a)$ in order to get an equivalent normalized CSDFG. Next theorem follows:

THEOREM 1. *Let G be a strongly connected CSDFG. G is normalizable iff G is consistent.*

Let us consider as example the CSDFG pictured in left side of Figure 3. The corresponding system is:

$$\begin{array}{ll} Z_1 = \delta_3 \times 5 = \delta_4 \times 5 = \delta_1 \times 2 & Z_3 = \delta_2 \times 2 = \delta_3 \times 3 \\ Z_2 = \delta_1 \times 3 = \delta_5 \times 5 = \delta_2 \times 5 & Z_4 = \delta_4 \times 6 = \delta_5 \times 4 \end{array}$$

A minimum solution is given by $\Delta^\star = (5, 3, 2, 2, 3)$ and $Z^\star = (10, 15, 6, 12)$. Initial markings obtained are $M_0(a_1) = 15$, $M_0(a_2) = 3$, $M_0(a_3) = 4$, $M_0(a_4) = 0$, $M_0(a_5) = 21$. The associated equivalent normalized CSDFG is pictured in Figure 3.

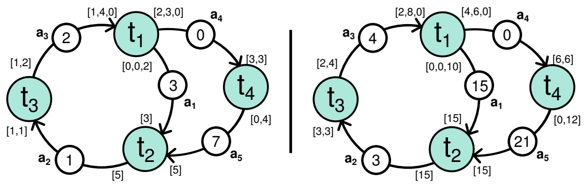

Figure 3: (Left) A CSDFG of four actors and five buffers. Initial tokens are represented by surrounded values. (Right) Its normalized version.

4. LIVENESS CHECKING ALGORITHMS

This section aims to present two original algorithms for checking a sufficient condition for the liveness of a CSDFG. The first subsection recalls that relevant values of initial markings may be limited. A first sufficient condition of liveness is then expressed, followed by a polynomial-time algorithm whose complexity depends on the total number of phases. A second condition is expressed obtained by merging the phases of the same actor in order to reduce the complexity of the corresponding algorithm. The equivalence of the two sufficient conditions of liveness is then proved as well as the fact that these conditions are not necessary. Finally, the complexity of these algorithms is expressed.

4.1 Useful tokens

It is observed that the initial markings of any buffer of an SDFG may be reduced to functions depending on the values of the arcs [14]. This result was extended to CSDFG [4, 18] as follows: let us denote for every arc $a = (t, t') \in A$,

$$step_a = \gcd(w_a(1), \cdots, w_a(\varphi(t)), v_a(1), \cdots, v_a(\varphi(t'))),$$

where gcd is the greatest common divisor of a given list of non negative integers. For every integer $\alpha \in \mathbb{Z}$, we also set

$$\lfloor \alpha \rfloor^{step_a} = \left\lfloor \frac{\alpha}{step_a} \right\rfloor \cdot step_a.$$

LEMMA 1 ([4]). *The initial marking $M_0(a)$ of any arc $a = (t, t')$ can be replaced by $\lfloor M_0(a) \rfloor^{step_a}$ without any influence on the data dependencies induced by a between the successive executions of actors t and t'.*

In the rest of this paper, it is assumed that the initial marking of any arc a is a multiple of $step_a$.

4.2 A first sufficient condition SC1

The following theorem expresses a first sufficient condition of liveness for a normalized CSDFG:

THEOREM 2. *Let $\mathcal{G}$ be a normalized CSDFG. If $\mathcal{G}$ is not live, then there exists a circuit $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ and the values $k^i \in \{1, \cdots, \varphi(t^i)\}$ for $i \in \{1, \cdots, m\}$ such that*

$$\sum_{i=1}^{m} M_0(a_i) \leq \sum_{i=1}^{m} \left[D^-_{a_{i-1}}\langle t^i_{k^i},1\rangle - D^+_{a_i} Pred\langle t^i_{k^i},1\rangle\right] - \sum_{i=1}^{m} step_{a_i}$$
with $a_0 = a_m$.

The following corollary is an immediate consequence of Theorem 2:

COROLLARY 1 (SC1). *Let $\mathcal{G}$ be a normalized CSDFG. $\mathcal{G}$ is live if for any circuit $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$ and any values $k^i \in \{1, \cdots, \varphi(t^i)\}$ for $i \in \{1, \cdots, m\}$,*
$$\sum_{i=1}^{m} M_0(a_i) > \sum_{i=1}^{m} \left[D^-_{a_{i-1}}\langle t^i_{k^i},1\rangle - D^+_{a_i} Pred\langle t^i_{k^i},1\rangle\right] - \sum_{i=1}^{m} step_{a_i}$$
with $a_0 = a_m$.

A polynomial-time algorithm is expressed subsequently to check this first necessary sufficient condition denoted by SC1.

4.3 Checking the liveness using condition SC1

The aim of this section is to express a polynomial time-algorithm for checking the sufficient condition of liveness SC1 on a CSDFG $\mathcal{G} = (T, A)$. For that purpose, let us consider the valued oriented graph $H_1 = (N_1, E_1)$ defined as follows:

- N_1 is the set of the phases of actors from T, *i.e.* $N_1 = \left\{t^i_k, i \in \{1, \cdots, |T|\}, k \in \{1, \cdots, \varphi(t^i)\}\right\}$.
- Any arc $a = (t^i, t^j) \in A$ corresponding to a buffer is associated with $\varphi(t^i) \times \varphi(t^j)$ arcs $u = (t^i_{k^i}, t^j_{k^j}) \in E_1$ for $k^i \in \{1, \cdots, \varphi(t^i)\}$ and $k^j \in \{1, \cdots, \varphi(t^j)\}$ valued by $W_1(u) = D^-_a\langle t^j_{k^j}, 1\rangle - D^+_a Pred\langle t^i_{k^i}, 1\rangle - step_a - M_0(a)$.

The cycle-mean of a circuit $c = (t^1_{k^1}, u_1, t^2_{k^2}, \cdots, t^m_{k^m}, u_m, t^1_{k^1})$ equals $W_1(c) = \frac{\sum_{i=1}^{m} W_1(u_i)}{m}$. Now, let $W_1^\star$ be the maximum cycle-mean value of a circuit from H_1. SC1 is equivalent to $W_1^\star < 0$. Several polynomial-time algorithms for computing $W_1^\star$ are detailed and compared experimentally in [7].

Figure 4 pictures the graph H_1 associated with the normalized CSDFG presented in Figure 3. The maximum cycle-mean value is $W_1^\star = -\frac{1}{4}$ and is reached for the circuit passing through t^1_2, t^4_2, t^2_1, t^3_2,t^1_2. As $W_1^\star < 0$, SC1 holds for any circuit of H_1, $\mathcal{G}$ is thus live.

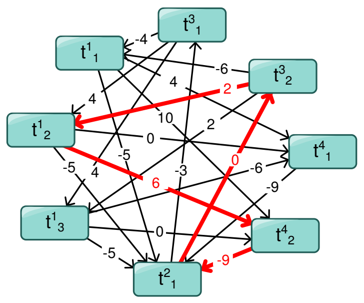

Figure 4: Graph H_1 associated with the normalized CSDFG pictured in Figure 3. The circuit passing through t^1_2, t^4_2, t^2_1, t^3_2 and t^1_2 has a maximum cycle-mean equal to $-\frac{1}{4}$.

The size of H_1 may grow quickly with the number of phases. Another sufficient condition may be expressed to significantly reduce the size of the underlying graph.

4.4 A second sufficient condition SC2

The number of phases for each actor may be too large to get an efficient algorithm for checking the liveness using SC1. The idea here is to express another condition SC2 and to prove that SC1 and SC2 are equivalent.

Let us consider that a CSDFG $\mathcal{G}$ verifies condition SC2 if, for any cycle $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$,
$$\sum_{i=1}^{m} M_0(a_i) > -\sum_{i=1}^{m} step_{a_i}$$
$$+\sum_{i=1}^{m} \max_{k^i \in \{1,\cdots,\varphi(t^i)\}} \left[D^-_{a_{i-1}}\langle t^i_{k^i},1\rangle - D^+_{a_i} Pred\langle t^i_{k^i},1\rangle\right].$$

THEOREM 3. *Let $\mathcal{G}$ be a normalized CSDFG. Conditions SC1 and SC2 are equivalent.*

A sufficient condition of liveness follows from Corollary 1 and Theorem 3:

COROLLARY 2 (SC2). *Let $\mathcal{G}$ be a normalized CSDFG. $\mathcal{G}$ is live if for any cycle $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$,*
$$\sum_{i=1}^{m} M_0(a_i) > -\sum_{i=1}^{m} step_{a_i}$$
$$+\sum_{i=1}^{m} max_{k^i \in \{1,\cdots,\varphi(t^i)\}} \left[D^-_{a_{i-1}}\langle t^i_{k^i},1\rangle - D^+_{a_i} Pred\langle t^i_{k^i},1\rangle\right].$$

Let $H_2 = (N_2, E_2)$ be a valued graph built as follows for checking SC2 on a CSDFG $\mathcal{G} = (T, A)$:

- N_2 is the set of buffers, *i.e.* $N_2 = A$.
- Each arc $e = (a, a') \in E_2$ is associated with a actor t such that $a \in \mathcal{A}^-(t)$, $a' \in \mathcal{A}^+(t)$ and is valued by
$$W_2(e) = -step_a - M_0(a)$$
$$+ \max_{k \in \{1,\cdots,\varphi(t)\}} \left[D^-_a\langle t_k, 1\rangle - D^+_{a'} Pred\langle t_k, 1\rangle\right].$$

As seen before, satisfying the condition SC2 is equivalent to checking that the maximum cycle-mean $W_2^\star$ of H_2 is strictly negative. The graph H_2 corresponding to the normalized CSDFG $\mathcal{G}$ pictured in Figure 3 is presented in Figure 5.

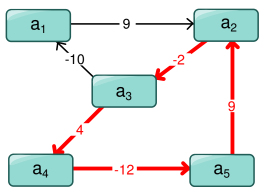

Figure 5: Graph H_2 associated with the normalized CSDFG $\mathcal{G}$ pictured in Figure 3. A circuit of maximum mean-cycle value is highlighted.

4.5 SC1 and SC2 are not necessary

These two conditions can be seen as a generalization of a result proved by Marchetti *et al.* [14] for SDFG expressed as follows:

THEOREM 4 ([14]). *Let $\mathcal{G}$ be a normalized SDFG. Then, $\mathcal{G}$ is live if for any cycle $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$*
$$\sum_{i=1}^{m} M_0(a_i) > \sum_{i=1}^{m} (Z_i - step_{a_i}).$$

Marchetti *et al.* in [14] proved that the condition of Theorem 4 is not necessary for the liveness of an SDFG. The consequence is that conditions SC1 and SC2 are also not necessary.

4.6 Worst-case complexity of these algorithms

For any actor $t \in T$, we denote by $deg^{+}(t) = |\mathcal{A}^{+}(t)|$ the output degree of t and $deg^{-}(t) = |\mathcal{A}^{-}(t)|$ the input degree of t. We also denote by $R = (R_1, \cdots, R_n)$ the repetition vector of $\mathcal{G}$: it can be defined as the smallest positive integer vector such that, for any couple of actors $(t, t') \in T^2$, $Z_t R_t = Z_{t'} R_{t'}$ [15].

For each considered algorithm, we then define the complexity as follows.

Algorithm SE : This algorithm, developed by [2] consists of a symbolic execution. Its worst-case case complexity bounded by $\mathcal{O}(|A| \times \sum_{t \in T} R_t)$ is proved to be exponential [15].

Algorithm SC1 : It consists of checking the condition SC1 using the determination of a maximum cycle-mean in H_1. Its complexity is $\Theta(|N_1| \times |E_1|)$ corresponding to $\Theta(\sum_{t \in T} \varphi(t) \times \sum_{(t_i, t_j) \in A} \varphi(t_i) \times \varphi(t_j))$.

Algorithm SC2 : It checks SC2 using the determination of a maximum cycle-mean in H_2. Its complexity is bounded by $\Theta(|A| \times \sum_{t \in T} (deg^{+}(t) \times deg^{-}(t)))$.

These complexities can be evaluated before executing the corresponding algorithms. A simple heuristic can then choose between these three algorithms by minimizing the theoretical complexity.

5. EXPERIMENTS

This part is focused on the actual applications of our methods and their practical interest. Benchmarks are presented, followed by the experimentations of the liveness algorithms. The last subsection is dedicated to the presentation and the experimentations of an efficient algorithm to compute the minimum buffer capacity ensuring liveness.

5.1 Benchmarks

Two different benchmarks reported in Table 1 were considered to experimentally evaluate our method. The former will focus on real-life industrial applications (JPEG2000, H264, ...). The latter is generated using different generation tools: the size and the complexity of the CSDFG are higher, and can be seen as possible future instances. The first column of Table 1 reports the name of the benchmarks. The second and third ones correspond respectively to the number of actors and buffers. The last one reports the size of a schedule issued from a symbolic execution, which is exactly equal to $\sum_{t \in T} (R_t \times \varphi(t))$.

Application	Actors	Buffers	Sched. size
BlackScholes	41	40	2379
JPEG2000	240	703	29595
Echo	38	82	42003
Pdetect	58	76	4045
H264	665	3128	1471
autogen1	90	617	250992
autogen2	70	473	41331062
autogen3	154	671	308818852
autogen4	2426	2900	51301
autogen5	2767	4894	312485

Table 1: Benchmarks

In summary, the BlackScholes application is a financial tool, JPEG2000, Echo and H264 are three multimedia applications and Pdetect is an application specialized in the detection of people.

For applications that deadlock, the computation time of algorithms SC1 and SC2 will not vary, but SE will probably be faster. However, SE reaches its maximum complexity when the instances that it processes are live, that is the reason all the benchmarks we considered are live.

5.2 Experimentations on the liveness

Table 2 reports the computation times of the three algorithms (namely SE, SC1 and SC2) on our benchmark. The tests were carried out on a standard workstation based on an Intel CORE i3 processor. Framed results are those selected by our heuristic, which choose the best algorithm following the evaluation of their theorical complexity.

In most cases, the framed results correspond to the lowest computation time. We first note the complementarity between algorithms SC1 and SC2. SC2 is often faster than SC1. However, if the actors degrees are high, such as in H264, SC1 is a better choice.

In the industrial benchmarks, the computation time of the symbolic method SE remains competitive versus the SC1 and SC2 algorithms. This is not longer true for the generated one's, for which the repetition vectors are higher. This kind of instance will be of importance with the arrival of new programming tools. In this case, the computation times of SE are clearly too long to be used in an industrial context.

Application	SC1	SC2	SE[2]
BlackScholes	2.10^5/14ms	$\boxed{1.10^3/0\text{ms}}$	2.10^5/1ms
JPEG2000	8.10^6/114ms	$\boxed{2.10^6/18\text{ms}}$	4.10^7/113ms
Echo	$\boxed{6.10^3/1\text{ms}}$	1.10^4/0ms	6.10^6/95ms
Pdetect	1.10^9/2500ms	$\boxed{8.10^3/4\text{ms}}$	6.10^5/5ms
H264	3.10^7/504ms	5.10^7/936ms	$\boxed{9.10^6/114\text{ms}}$
autogen1	$\boxed{1.10^5/13\text{ms}}$	2.10^6/55ms	3.10^8/1544ms
autogen2	$\boxed{7.10^5/41\text{ms}}$	1.10^6/37ms	3.10^{10}/4min
autogen3	$\boxed{1.10^6/55\text{ms}}$	1.10^6/55ms	4.10^{11}/21min
autogen4	7.10^7/217ms	$\boxed{1.10^7/71\text{ms}}$	2.10^8/132ms
autogen5	5.10^7/787ms	$\boxed{4.10^7/708\text{ms}}$	3.10^9/1442ms

Table 2: Complexity of SC1, SC2 and SE as defined in Subsection 4.6 with their actual computation time. Surrounded results correspond to the method selected by our heuristic.

5.3 Computation of minimum buffer capacities ensuring liveness

Condition SC1 may be considered to evaluate minimum buffer capacities of a fixed CSDFG. Indeed, let us suppose that each buffer $b(a)$ associated with an arc $a = (t, t')$ is bounded. This can be modeled using a feedback arc $a' = (t', t)$ as shown in Subsection 2.1. Now, let $\mathcal{G}' = (T, A')$ be the graph obtained by adding these feedback arcs and $H'_1 = (N'_1, E_1)$ its corresponding oriented valued graph associated to SC1. W'_1 denotes the valuation of H'_1.

The capacity of the buffer $b(a)$ equals $M_0(a) + M_0(a')$. The overall capacity of buffers of $\mathcal{G}$ is thus $\sum_{a \in A'} M_0(a)$.

Now, initial values $M_0(a)$, $a \in A'$ must be computed such that condition SC1 is fulfilled. Integer values γ_a, $a \in N'_1$ must be computed such that, for any arc $e = (u, u') \in E_1$, we get $\gamma_u - \gamma_{u'} > W'_1(e)$.

The linear system is thus:

Minimize $\sum_{a \in A'} M_0(a)$ with

$$\begin{cases} \gamma_u - \gamma_{u'} > W'_1(e), & \forall e = (u, u') \in E_2 \\ M_0(a) \in \mathbb{N}, & \forall a \in A' \\ \gamma_{t_k} \in \mathbb{R}, & \forall t \in T, \forall k \in \{1, \cdots, \varphi(t)\} \end{cases}$$

An equivalent linear program can be expressed based on SC2. Our algorithm will choose between SC1 and SC2, whichever has the lowest theoretical complexity. We use GLPK to solve a continuous version of the linear program. The buffer capacities are computed using a rounding method.

Our experimental results are compared with a greedy algorithm based on SE and inspired by [17]. Actors are executed so that the buffer sizes are minimized. Our experiments are reported in Table 3. The first column is the benchmark's names. Second and third columns report the computation time and the overall buffer size computed by SC1 or SC2 (following the theoretical evaluation of the complexity). Fourth and fifth one's report the computation time and the overall buffer size obtained using the greedy algorithm.

	SC1/SC2		Greedy algorithm	
Application	time	buf. size	time	buf. size
BlackScholes	**8 ms**	16 KB	9 ms	16 KB
JPEG2000	3089 ms	3807 KB	**2055 ms**	**3651 KB**
Echo	**5 ms**	**28 KB**	315 ms	52 KB
Pdetect	**26 ms**	3959 KB	61 ms	3959 KB
H264	4808 ms	1368 KB	**937 ms**	1368 KB
autogen1	**169 ms**	**1849 KB**	3043 ms	2009 KB
autogen2	**1704 ms**	**227 MB**	7 min	244 MB
autogen3	**2407 ms**	**1080 MB**	36 min	1296 MB
autogen4	**16605 ms**	47 KB	20522 ms	**34 KB**
autogen5	**2 min**	**1555 KB**	3 min	3069 KB

Table 3: Computed buffer sizes of the different algorithms for each CSDF.

Judging by the data in Table 3, the quality is comparable, but the running time of the greedy algorithm is much higher in more complex applications.

6. CONCLUSION

This paper presents significant advances in both fundamental and applicative point of views for evaluating the liveness of a CSDFG.

The normalization of a CSDFG is first introduced and should be used to effectively address other CSDFG problems such as the minimization of buffer considering throughput [18]. In addition, two sufficient equivalent conditions of liveness are proved. Efficient original polynomial-time algorithms for checking the liveness of a CSDFG and computing its minimal buffer sizes (ensuring liveness) are deduced.

These algorithms are the first polynomial ones to solve approximatively these two problems. They were successfully tested on industrial and academic benchmarks. The experiments highlighted that they are well suited for real-life applications and more robust than the existing methods for complex applications. Their low complexity ensures that these algorithms can safely be integrated in a compiler.

7. REFERENCES

[1] B. Akesson, S. Stuijk, A. Molnos, M. Koedam, R. Stefan, A. Andrew Nelson and Beyranvand Nejad, and K. Goossens. Virtual platforms for mixed time-criticality applications: The CoMPSoC architecture and SDF3 design flow. In *Quo Vadis, Virtual Platforms?(QVVP)*, pages 1–2, 2012.

[2] S. R. Anapalli, K. C. Chakilam, and T. W. O'Neil. Static Scheduling for Cyclo Static Data Flow Graphs. In *Parallel and Distributed Processing Techniques and Applications, PDPTA 2009*, pages 302–306. CSREA Press, 2009.

[3] M. Bamakhrama and J. Zhai. A methodology for automated design of hard-real-time embedded streaming systems. *Design, Automation & Test in Europe (DATE)*, 2012.

[4] M. Benazouz. *Buffer Sizing for Stream Processing Applications*. PhD thesis, Université P. et M. Curie, Paris, France, 2012.

[5] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cycle-static data flow. *IEEE Transactions on Signal Processing*, pages 3255–3258, 1995.

[6] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: a declarative language for real-time programming. In *Symposium on Principles of programming languages - POPL '87*, pages 178–188. ACM Press, 1987.

[7] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. *Design Automation Conference (DAC'99)*, pages 37–42, 1999.

[8] A. Ghamarian and M. Geilen. Liveness and boundedness of synchronous data flow graphs. *Formal Methods in Computer Aided Design (FMCAD'06)*, 2006.

[9] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, B. Theelen, M. Mousavi, A. Moonen, and M. Bekooij. Throughput Analysis of Synchronous Data Flow Graphs. In *International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 25–36, 2006.

[10] Kalray. Manycore processors for embedded computing. www.kalray.eu.

[11] S. F. Khasawneh, M. E. Ritcher, and T. W. O'Neil. Static Scheduling for synchronous data flow graphs. *Computers and Their Applications*, 1:38–43, 2007.

[12] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[13] L. Mandel, F. Plateau, and M. Pouzet. Lucy-n: a n-synchronous extension of Lustre. *Mathematics of Program Construction*, 2010.

[14] O. Marchetti and A. Munier-Kordon. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research*, 197(2):532–540, Sept. 2009.

[15] O. Marchetti and A. Munier Kordon. Cyclic Scheduling for the Synthesis of Embedded Systems. In Y. Vivien and R. Frederic, editors, *Introduction to scheduling*, chapter 6, pages 135–164. Chapman and Hall/CRC Press, 2009.

[16] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee. A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs. Technical report, University of California, Berkeley, 1995.

[17] S. Sriram and S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC, 2009.

[18] S. Stuijk, M. Geilen, and T. Basten. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. *IEEE Transactions on Computers*, 57(10):1331–1345, 2008.

[19] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. *Compiler Construction*, pages 179–196, 2002.

APPENDIX

This appendix provides the proofs for Theorems 2 through 4.

Proof of Theorem 2

PROOF. Let us suppose that $\mathcal{G}$ is not live. Then, there exists a circuit $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$, and values $n^i \in \mathbb{N} - \{0\}$ and $k^i \in \{1, \cdots, \varphi(t^i)\}$ for $i \in \{1, \cdots, m\}$ such that:

- Executions $Pred\langle t^i_{k^i}, n^i\rangle$ with $i \in \{1, \cdots, m\}$ can be performed;
- the amount of data is not sufficient to execute any execution $\langle t^i_{k^i}, n^i\rangle$, for $i \in \{1, \cdots, m\}$.

Let us first consider the arc $a_1 = (t^1, t^2)$. Since the phase $\langle t^2_{k^2}, n^2\rangle$ cannot be executed even if $Pred\langle t^1_{k^1}, n^1\rangle$ is, the number of tokens on a_1 must verify

$$M_0(a_1) + D^+_{a_1} Pred\langle t^1_{k^1}, n^1\rangle - D^-_{a_1}\langle t^2_{k^2}, n^2\rangle < 0.$$

By definition of $D^+_{a_1}$ and $D^-_{a_1}$,

$$D^+_{a_1} Pred\langle t^1_{k^1}, n^1\rangle = D^+_{a_1}\langle t^1_{\varphi(t^1)}, n^1 - 1\rangle + D^+_{a_1} Pred\langle t^1_{k^1}, 1\rangle$$

and

$$D^-_{a_1}\langle t^2_{k^2}, n^2\rangle = D^-_{a_1}\langle t^2_{\varphi(t^2)}, n^2 - 1\rangle + D^-_{a_1}\langle t^2_{k^2}, 1\rangle.$$

The previous inequality thus becomes

$$M_0(a_1) + D^+_{a_1}\langle t^1_{\varphi(t^1)}, n^1 - 1\rangle + D^+_{a_1} Pred\langle t^1_{k^1}, 1\rangle - D^-_{a_1}\langle t^2_{\varphi(t^2)}, n^2 - 1\rangle - D^-_{a_1}\langle t^2_{k^2}, 1\rangle < 0.$$

Now, by Lemma 1, this sum is divisible by $step_{a_1}$, so we get

$$M_0(a_1) + D^+_{a_1}\langle t^1_{\varphi(t^1)}, n^1 - 1\rangle + D^+_{a_1} Pred\langle t^1_{k^1}, 1\rangle - D^-_{a_1}\langle t^2_{\varphi(t^2)}, n^2 - 1\rangle - D^-_{a_1}\langle t^2_{k^2}, 1\rangle \leq -step_{a_1}.$$

Similarly, by setting $t^{m+1} = t^1$, we get for any value $i \in \{1, \cdots, m\}$,

$$M_0(a_i) + D^+_{a_i}\langle t^i_{\varphi(t^i)}, n^i - 1\rangle + D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle - D^-_{a_i}\langle t^{i+1}_{\varphi(t^{i+1})}, n^{i+1} - 1\rangle - D^-_{a_i}\langle t^{i+1}_{k^{i+1}}, 1\rangle \leq -step_{a_i}.$$

Since $\mathcal{G}$ is normalized,
$\forall i \in \{1, \cdots, m-1\}$,

$$D^-_{a_{i-1}}\langle t^i_{\varphi(t^i)}, n^i - 1\rangle = D^+_{a_i}\langle t^i_{\varphi(t^i)}, n^i - 1\rangle = (n^i - 1)\cdot Z_{t^i}.$$

By summing all the previous inequalities, we then obtain that

$$\sum_{i=1}^{m} M_0(a_i) + \sum_{i=1}^{m}\left[D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle - D^-_{a_{i-1}}\langle t^i_{k^i}, 1\rangle\right] \leq -\sum_{i=1}^{m} step_{a_i}$$

which concludes the proof. □

Proof of Theorem 3

PROOF. Let us suppose that $\mathcal{G}$ verifies SC2 and let $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ be a circuit of $\mathcal{G}$. Then, for any values $k^i \in \{1, \cdots, \varphi(t^i)\}$, $i \in \{1, \cdots, m\}$, we get that

$$\max_{k^i\in\{1,\cdots,\varphi(t^i)\}}\left[D^-_{a_{i-1}}\langle t^i_{k^i}, 1\rangle - D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle\right] \geq D^-_{a_{i-1}}\langle t^i_{k^i}, 1\rangle - D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle.$$

Thus,

$$\sum_{i=1}^{m} M_0(a_i) > -\sum_{i=1}^{m} step_{a_i} + \sum_{i=1}^{m}\left[D^-_{a_{i-1}}\langle t^i_{k^i}, 1\rangle - D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle\right].$$

and $\mathcal{G}$ verifies SC1. Conversely, if SC1 is true, then, for any cycle $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ and any phases $k^\star_i$ of t_i maximizing $D^-_{a_{i-1}}\langle t^i_{k^i}, 1\rangle - D^+_{a_i} Pred\langle t^i_{k^i}, 1\rangle$, the inequality is true. SC2 is thus verified, which concludes the proof. □

Proof of Theorem 4

PROOF. Any normalized SDFG $\mathcal{G} = (T, A)$ is a CSDFG for which each actor has a unique phase, thus $\forall t \in T$, $\varphi(t) = 1$. Since $\mathcal{G}$ is normalized, for any arc $a = (t, t')$, $D^-_a\langle t_1, 1\rangle = Z_t$ and $D^+_a Pred\langle t_1, 1\rangle = D^+_a\langle t_1, 0\rangle = 0$. Thus, SC2 is equivalent to the condition expressed by the theorem. □