Fast and Efficient Dataflow Graph Generation

Citation for published version:
Bodin, B, Lesparre, Y, Delosme, J-M & Munier-Kordon, A 2014, Fast and Efficient Dataflow Graph Generation. in *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems.* ACM, New York, NY, USA, pp. 40-49. https://doi.org/10.1145/2609248.2609258

Digital Object Identifier (DOI):
10.1145/2609248.2609258

Published In:
Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems

Fast and Efficient Dataflow Graph Generation

Bruno BODIN[‡], Youen LESPARRE[⋆], Jean-Marc DELOSME[†], and Alix MUNIER-KORDON[⋆]

bbodin@inf.ed.ac.uk, youen.lesparre@lip6.fr, delosme@ibisc.univ-evry.fr, alix.munier@lip6.fr

‡ School of Informatics, University of Edinburgh, Edinburgh, United-Kingdom

⋆ Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

† IBISC, Université d'Évry-Val-D'Essonne, 91025 Évry, France

General Terms

Graph generation, Synchronous Dataflow, Cyclo-Static Dataflow, Phased Computation Graph.

Keywords

Synchronous Dataflow Graph, SDFG, Cyclo-Static Dataflow Graph, CSDFG, Phased Computation Graph, PCG, normalization, random generation, initial markings.

ABSTRACT

Dataflow modeling is a highly regarded method for the design of embedded systems. Measuring the performance of the associated analysis and compilation tools requires an efficient dataflow graph generator. This paper presents a new graph generator for Phased Computation Graphs (PCG), which augment Cyclo-Static Dataflow Graphs with both initial phases and thresholds.

A sufficient condition of liveness is first extended to the PCG model. The determination of initial conditions minimizing the total amount of initial data in the channels and ensuring liveness can then be expressed using Integer Linear Programming. This contribution and other improvements of previous works are incorporated in `Turbine`, a new dataflow graph generator. Its effectiveness is demonstrated experimentally by comparing it to two existing generators, DFTools and SDF^3.

1. INTRODUCTION

Dataflow modeling is a highly valued method for the design of embedded systems and, over the last decades, several dataflow models have been proposed [16]. Applications are usually represented by a set of actors (or tasks) that communicate data through unidirectional communication channels. The model is therefore particularly well adapted to digital signal processing (DSP) applications.

The Synchronous Dataflow Graph (SDFG) model [10] is a simple dataflow model for which the amounts of data produced or consumed at a time in each channel are fixed integers. This model is static, in the sense that the volumes of data exchanged are known before the execution of the application. Its behavior is thus predictable, with the consequence that important questions such as graph liveness or decision versions of optimization problems (in particular minimization of resources) are decidable. This characteristic allows then the development of efficient academic and industrial tools to handle these restricted dataflow graphs.

Several more expressive (still static) extensions were developed afterwards to handle larger classes of applications or particular resource constraints. The Cyclo-Static Dataflow Graph (CSDFG) model [5], in which data is produced or consumed piecemeal according to a fixed vector, is the most commonly used today. The dataflow graphs considered in this paper and generated by `Turbine` are Phased Computation Graphs (PCG) [19]. They extend the CSDFG model by including initial phases and thresholds. Phased Computation Graphs are usually considered in the context of programming languages for many-core architectures [1, 8].

As pointed before, several authors developed optimization methods with static dataflow graphs as inputs [2, 6, 20]. The first way to test the performance of their methods is to experiment using dataflow graphs obtained for specific applications, which can be obtained by analyzing directly the data processing algorithms. This is illustrated by the models of the H263 encoder [12], the MP3 playback [20] and the Reed-Solomon decoder [3], which have been developed in an academic context and do not contain more than 8 actors. A second approach consists in using a dataflow language to automatically generate the dataflow graph associated with an application [7, 18, 19]. The number of actors for real-life applications obtained by [4, 6] ranges from 38 up to 600. The increase in the size of the instances keeps pace with the evolution of embedded systems, with architectures soon comprising more than a thousand processors. It is achieved by assembling applications, corresponding to specific algorithms, to form the applications for the new systems, and by exploiting finer granularities, augmenting significantly the number of actors to achieve a higher parallelism.

The best way to validate an optimization method is to test it using randomly generated dataflow graph instances. The size of the generated graphs must be compatible to that of real instances, which can contain hundreds and even thousands of actors. Moreover, the generator must be fast enough not to slow down experimental validation of the optimization methods. General graph generation is a highly productive research area [14], whereas, to the best of our

SCOPES '14 Schloss Rheinfels, St. Goar, Germany

knowledge, only two dataflow graph generation tools have been made available to system designers. The open source dataflow library DFTools, developed in Java and mainly used by the toolchain PREESM [13], supplies an SDFG generator, while the widely used C++ library SDF^3 [17] provides both SDFG and CSDFG generation in addition to dataflow analysis algorithms [21, 15]. Our generator `Turbine` produces PCGs, hence also the more basic CSDFGs, which are commonly used to model applications in the industrial context [7]. We show experimentally in Section 6 that `Turbine` is able to produce a CSDFG with 10,000 actors within a reasonable time. Its performance is also compared, to its advantage, to that of SDF^3 and DFTools.

The critical step in dataflow graph generation is determining the initial conditions (*i.e.* computing the initial amount of data in the channels) that ensure the liveness of the model. The liveness of a static dataflow graph is decidable, but no polynomial-time algorithm exists to answer this question efficiently. The algorithm employed by SDF^3 and DFTools is based on a simple sufficient condition of liveness for an SDFG recalled in Section 3. Its extension to a CSDFG is theoretically sound, but calls for an excessive initial amount of data in the channels. Another important drawback is the time-complexity of the algorithms used by SDF^3 and DFTools, which severely limits the scalability of these two generators as shown experimentally in Section 6.

The sufficient condition exploited by `Turbine` is an extension to PCGs of the sufficient condition of liveness expressed in [11] for Weighted Event Graphs—a subclass of Petri nets strictly equivalent to SDFGs. This condition has been extended to the CSDFG model in [4]. It is extended in Section 4 to the PCG model. Our generator is based on linear programming to compute a minimum initial amount of data in the channels such that our sufficient condition of liveness is verified. The size of the linear program is a linear function of the number of channels, and is thus independent of the number of phases. We show in Section 6 that the total number of tokens (measuring the initial amount of data) generated by `Turbine` is 2.6 (*resp.* 1.6) times smaller than with SDF^3 (*resp.* DFTools).

Our paper is organized as follows. Section 2 presents the PCG model supported by `Turbine` and all the classical static dataflow models subsumed by this model. Some basic notations, followed by a brief presentation of SDF^3 and DFTools, are introduced in Section 3. Section 4 presents the sufficient condition of liveness employed by `Turbine`. Section 5 briefly describes `Turbine`. The limits of our generator and a comparison with SDF^3 and DFTools are presented in Section 6. Section 7 is our conclusion.

2. STATIC DATAFLOW MODELS

We present in this section the dataflow models supported by `Turbine`. Phased Computation Graphs are described last and encompass the models presented before.

2.1 Synchronous Dataflow Graph

Synchronous Dataflow Graphs (SDFG) were introduced by Lee and Messerschmitt [10] to model data transfers for embedded systems. An application is decomposed into actors (tasks, nodes) which communicate through FIFO buffers (channels, edges). For each buffer, two positive integers specify the number of tokens (data items) produced (*resp.* consumed) when executing its input (*resp.* output) task. These numbers are respectively noted p_a and c_a for buffer a. The number of tokens in buffer a at the start of the application is its initial marking, noted $M_0(a)$.

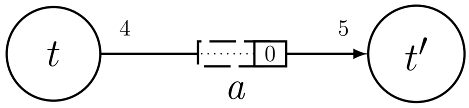

Figure 1: A buffer a between two tasks t and t' of an SDFG with production weight $p_a = 4$, consumption weight $c_a = 5$ and initial marking $M_0(a) = 0$.

In the SDFG of Figure 1, $p_a = 4$, $c_a = 5$ and $M_0(a) = 0$. At the end of an execution of task t, 4 data items are stored in buffer a. In order to be executed, task t' requires 5 data items, which will be consumed at the beginning of an execution.

2.2 Computation Graph

Following Karp and Miller [9], the Computation Graph (CG) model extends the SDFG model with the specification of thresholds. A threshold indicates how many data items must be present in a buffer before the associated consumer task can be executed. The threshold of buffer a is denoted by θ_a; it satisfies $\theta_a \geq c_a$ by definition.

Figure 2 represents a CG with $c_a = 5$ and $\theta_a = 7$. A threshold value appears—separated by a colon—after the associated consumption weight. Task t' requires 7 data items to be executed and consumes 5 items at each execution.

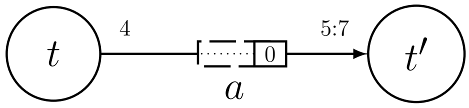

Figure 2: A buffer a between two tasks t and t' of a CG. The consumption weight and the threshold are respectively $c_a = 5$ and $\theta_a = 7$.

2.3 Cyclo-Static Dataflow Graph

The Cyclo-Static Dataflow Graph (CSDFG) model is another extension of the SDFG model, in which the data items are consumed or produced piecemeal [5]. Data production and consumption are represented by vectors of non-negative weights. Each execution of a task t is decomposed into a fixed number of phases, $\phi(t)$. For each task t, two non-negative integers, $p_a(k)$ and $c_a(k)$, denote respectively the number of items produced into or consumed from a buffer a during phase k, $1 \leq k \leq \phi(t)$. The total of the productions (*resp.* consumptions) over all the phases is denoted p_a (*resp.* c_a), *i.e.* $p_a = \sum_{k=1}^{\phi(t)} p_a(k)$ and $c_a = \sum_{k=1}^{\phi(t)} c_a(k)$.

In the CSDFG of Figure 3, the vector of numbers of items produced during the $\phi(t) = 2$ phases of an execution of task t is $[p_a(1), p_a(2)] = [3, 1]$ with $p_a = 4$, and the vector of numbers of items consumed during the $\phi(t') = 3$ phases of an execution of task t' is $[c_a(1), c_a(2), c_a(3)] = [2, 1, 2]$ with $c_a = 5$.

2.4 Cyclo-Static Dataflow Graph with initial phases

Initial phases allow to express the way a task in a dataflow graph is initialized. The initialization of a task t is decom-

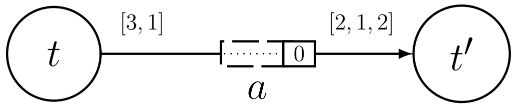

Figure 3: A buffer a between two tasks t and t' of a CSDFG. The numbers of phases are $\phi(t)=2$ and $\phi(t')=3$. The two vectors associated to buffer a are $[p_a(1),p_a(2)]=[3,1]$ on the production side and $[c_a(1),c_a(2),c_a(3)]=[2,1,2]$ on the consumption side.

posed into $\sigma(t)$ initial phases, which are executed once. The phases of t are numbered from $1-\sigma(t)$ to $\phi(t)$; numbers k belonging to $\{1-\sigma(t),\cdots,0\}$ (*resp.* $\{1,\cdots,\phi(t)\}$) refer to initial (*resp.* cyclic) phases.

In Figure 4, the vector of the numbers of items produced (*resp.* consumed) in the initial phases appears, between parentheses, just before the vector of numbers of items produced (*resp.* consumed) in the normal phases. For task t, $\sigma(t)=2$, $p_a(-1)=1$, $p_a(0)=2$, and, for task t', $\sigma(t')=2$, $c_a(-1)=2$, $c_a(0)=3$. Buffer a contains 3 data items after the two phases of initialization of task t, which executes cyclically its normal phases afterwards.

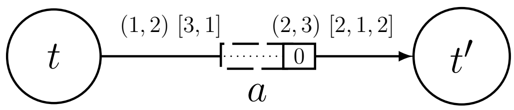

Figure 4: A buffer a between two tasks t and t' of a CSDFG with initial phases. The numbers of initial phases are $\sigma(t)=2$ and $\sigma(t')=2$. The two vectors associated to buffer a are $(p_a(-1),p_a(0))=(1,2)$ on the production side and $(c_a(-1),c_a(0))=(2,3)$ on the consumption side.

2.5 Phased Computation Graph

The Phased Computation Graph (PCG) model [19] extends the CSDFG model by using both thresholds and initial phases. On the consumer side of a buffer, all the initial phases have thresholds. The PCG notation extends that of the CSDFG with initial phases by using $\theta_a(k)$ to denote—for a task t consuming data from a buffer a—the threshold of phase k, with $1-\sigma(t)\le k\le\phi(t)$. For each phase k, $\theta_a(k)\ge c_a(k)$ by definition of a threshold; $\theta_a(k)$ is indicated only when it differs from $c_a(k)$. As in the CSDFG model, p_a (*resp.* c_a) represents the sum of the numbers of items produced into (*resp.* consumed from) buffer a by a task t during its $\phi(t)$ normal phases.

In Figure 5 the thresholds for the phases of task t' are $\theta_a(-1)=5$, $\theta_a(0)=3$, $\theta_a(1)=3$, $\theta_a(2)=1$ and $\theta_a(3)=2$.

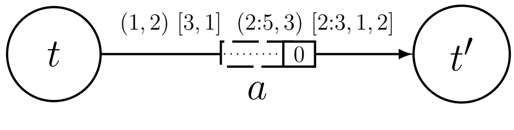

Figure 5: A buffer a between two tasks t and t' of a PCG. Thresholds are $\theta_a(-1)=5$ and $\theta_a(0)=3$ for the $\sigma(t')=2$ initial phases, and $\theta_a(1)=3$, $\theta_a(2)=1$ and $\theta_a(3)=2$ for the $\phi(t')=3$ normal phases.

3. THREE-STEP GENERATORS OF STATIC DATAFLOW GRAPHS

The two static dataflow graph generators SDF^3 [17], targetting SDFGs and CSDFGs, and DFTools [13], for SDFGs, are surveyed in this section. Both illustrate a three-step procedure to generate dataflow graphs: build a connected oriented graph, compute arc weights ensuring consistency, and find an initial live marking. As this procedure is very general, this survey also provides an introduction to `Turbine`.

The first subsection recalls the minimal properties required of an SDFG and of a PCG; these properties are characteristic of real-life applications executed on embedded systems. The last three subsections describe the three-step procedure used to generate dataflow graphs and its implementation in SDF^3 and DFTools.

3.1 Consistency and liveness of a PCG

A generator should produce instances that are both consistent and live. Indeed, in most cases, a PCG model of a real-life application should possess these properties, which are briefly recalled below.

3.1.1 *Consistency of a PCG*

A PCG is consistent if there exists an initial marking such that its actors may all be executed infinitely often with bounded buffer sizes. This property, first introduced for the SDFG model in [10], carries over to the refinements of that model such as the CSDFG and PCG models [5].

Consider a PCG $\mathcal{G}=(\mathcal{T},\mathcal{A})$. The elements of its topology matrix Γ, of size $|\mathcal{A}|\times|\mathcal{T}|$, are:

$$\Gamma_{a,t}=\begin{cases} p_a & =\sum_{k=1}^{\phi(t)} p_a(k) & \text{if } a=(t,t') \\ -c_a & =-\sum_{k=1}^{\phi(t)} c_a(k) & \text{if } a=(t',t) \\ 0 & & \text{otherwise} \end{cases}.$$

When (and only when) Γ has rank $|\mathcal{T}|-1$, the graph $\mathcal{G}$ is consistent. Then, the vectors X satisfying the matrix equation $\Gamma.X^T=0$ are proportional to a vector with strictly positive integer entries. The smallest such vector, whose entries are coprime, is known as the repetition vector [10], here noted R.

The SDFG and PCG pictured respectively in Figure 6 and Figure 7 have the same topology matrix:

$$\Gamma=\left[\begin{array}{ccc} 7 & -6 & 0 \\ 0 & 2 & -7 \\ -1 & 0 & 3 \end{array}\right].$$

The rank of Γ is 2, hence both graphs are consistent. Their repetition vector is $R=[6,7,2]$.

3.1.2 *Liveness of a PCG*

A PCG with a fixed initial marking is live if each actor may be fired infinitely often. Since it is independent of the initial marking, consistency is not sufficient to ensure the liveness of a PCG.

Consider first the simpler case of an SDFG. A firing sequence is a sequence ν of consecutive executions (firings) of the actors $t\in\mathcal{T}$ where the symbol representing a firing is simply the name t of the actor fired. A simple way to test the liveness of a consistently marked SDFG is to construct, if possible, a firing sequence ν_R in which each actor $t\in\mathcal{T}$ is executed R_t times. Since such a sequence ν_R brings the marking back to the initial marking, an infinite sequence is obtained by infinitely repeating ν_R thus ensuring that each actor is fired infinitely often.

The SDFG of Figure 6 is live as its actors may be executed following the sequence $\nu_R = t^1t^2t^2t^3t^1t^1t^1t^2t^2t^2t^3t^1t^1t^2t^2$ with 6 executions of t^1, 7 of t^2 and 2 of t^3.

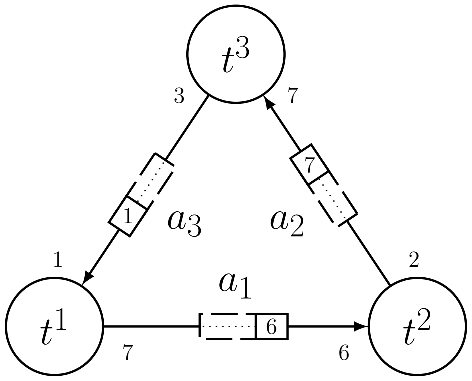

Figure 6: The SDFG is consistent; its repetition vector is $R = [6, 7, 2]$. It is live since the firing sequence $t^1t^2t^2t^3t^1t^1t^1t^2t^2t^2t^3t^1t^1t^2t^2$ may be repeated infinitely often.

The sequence ν_R has length the 1-norm of R, $\|R\| = \sum_{t\in\mathcal{T}} R_t$, which unfortunately can be very large in practice. In fact, as of today, there is no polynomial-time algorithm for testing the liveness of an SDFG.

Now consider a PCG. The initial phases must all be executed before a repetitive sequence ν_R can be reached. The length of the initial sequence is not easily bounded. If a cyclic sequence ν_R exists, since it is independent of the thresholds, its length is the same as that of the underlying CSDFG and is equal to the phase-weighted norm $\|R\|_\phi = \sum_{t\in\mathcal{T}} \phi(t)\cdot R_t$.

The PCG of Figure 7 is live. Its initial phases are executed by the sequence $t^1t^3t^3 = t^1(t^3)^2$, which leads to the marking $M(a_1) = 6$, $M(a_2) = 1$, $M(a_3) = 4$. The actors may then repeat the sequence $\nu_R = (t^1)^4(t^2)^{11}(t^3)^4(t^1)^2(t^2)^3(t^3)^2$ of length $\|R\|_\phi = 1\cdot 6 + 2\cdot 7 + 3\cdot 2 = 26$.

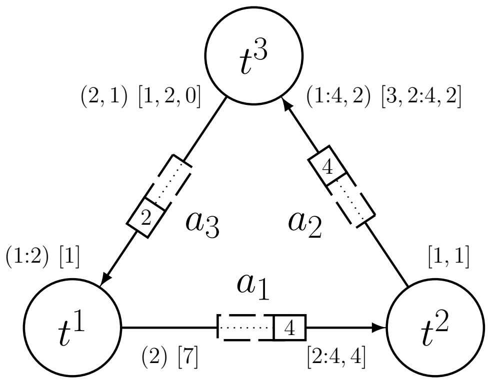

Figure 7: The PCG is consistent, with repetition vector $R = [6, 7, 2]$. The initial sequence $t^1(t^3)^2$ may be followed by the infinitely repeated sequence $(t^1)^4(t^2)^{11}(t^3)^4(t^1)^2(t^2)^3(t^3)^2$, hence the PCG is live.

A generator must build consistent and live dataflow graphs. As liveness cannot be checked polynomially, a sufficient condition must be applied to construct live instances.

3.2 Generation of a connected graph

In SDF^3 and DFTools, and also in `Turbine`, an oriented graph with a given number of vertices (tasks) and edges (buffers) is built in two phases: first all actors are connected by edges forming a tree (acyclic connected graph), then remaining edges are added (and oriented) randomly.

3.3 Weight Computation

This step consists in computing production/consumption weights such that the resulting dataflow graph be consistent. Various parameters may be used, such as setting minimum and maximum values of the weights or fixing the norm $\|R\|$ of the repetition vector.

SDF^3 considers two cases for computing the weights. If $\|R\|$ is set, the components of R are computed randomly according to this constraint. Weights are then derived to get a consistent graph by solving a linear system. If $\|R\|$ is not fixed, weights are set randomly on edges and a depth-first search algorithm is used to modify some of them so that the graph be consistent.

DFTools generates randomly a repetition vector R with components comprised between two fixed values $R_{\min}$ and $R_{\max}$. Production/consumption weights are then easily computed.

Our algorithm to compute weights is very similar and is described in Section 5. The main difference is that it produces a PCG (and not just an SDFG or a CSDFG) and is based on the normalization of a consistent SDFG.

3.4 Marking computation

The third step is the computation of an initial marking for the consistent graph constructed by the first two steps.

The algorithm presented in [17] for use in SDF^3 was clearly abandoned because of its time complexity. The algorithms implemented in SDF^3 and DFTools are based on the following simple sufficient condition of liveness:

THEOREM 1 (SCA). *Let $\mathcal{G}$ be a consistent SDFG with initial marking M_0. $\mathcal{G}$ is live if there is in every cycle $\mu = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$ at least one arc a_i, $i \in \{1, \cdots, m\}$ with $M_0(a_i) = R_{t^i}\cdot p_{a_i}$.*

Since it fulfills the condition of Theorem 1, the SDFG of Figure 6 with the marking $M_0(a_1) = M_0(a_2) = 0$, $M_0(a_3) = 6$ is live.

The computation of an initial marking following that sufficient condition consists in selecting a subset of edges $\mathcal{A}' \subseteq \mathcal{A}$ such that the subgraph $\mathcal{G}' = (\mathcal{T}, \mathcal{A} - \mathcal{A}')$ has no cycle and, then, setting the initial marking to

$$M_0(a) = \begin{cases} R_t\cdot p_a & \text{if } a = (t, t') \in \mathcal{A}' \\ 0 & \text{otherwise} \end{cases}.$$

SDF^3 computes an initial marking by selecting the edges in $\mathcal{A}'$ using a depth-first algorithm. DFTools first selects a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ such that the subgraph $\mathcal{G}'' = (\mathcal{T} - \mathcal{T}', \mathcal{A})$ is acyclic, then it sets $\mathcal{A}' = \{a = (t, t') \in \mathcal{A}, t \in \mathcal{T}'\}$. In both cases, the solutions obtained do not minimize the total amount of data.

Theorem 1 extends easily to CSDFGs. However, because of the presence of both thresholds and initial phases, its extension to PCGs is quite tricky. Instead, `Turbine` computes an initial marking for a PCG based on another sufficient condition of liveness, presented in the next section.

4. A SUFFICIENT CONDITION OF LIVENESS FOR A PCG

The aim of this section is to prove a sufficient condition for the liveness of a PCG. This condition is fundamental to ensure that the initial marking associated with a generated PCG is live.

Subsection 4.1 is dedicated to the extension to a PCG of the notion of normalization known for SDFGs and CSDFGs. Normalization is a simple polynomial transformation of the graph which does not modify the feasible firing sequences. This transformation is used to obtain lower bounds on the amount of data on the graph cycles to ensure liveness. Subsection 4.2 presents additional useful notations. Subsection 4.3 recalls the definition of a feasible sequence and shows that the markings can be restricted to a particular set of values. Our sufficient condition of liveness is presented and proved in Subsection 4.4. Subsection 4.5 presents a polynomial algorithm to check the sufficient condition on a PCG. Subsection 4.6 develops a possible application of this condition to compute the minimum buffer size required to guarantee liveness.

4.1 Normalization of a PCG

Normalization is a reversible transformation which greatly simplifies the weights of a consistent SDFG or CSDFG without modifying the constraints induced by the buffers on the actors' executions. It was introduced by Marchetti and Munier in [11] for Weighted Event Graphs, a subclass of Petri Nets strictly equivalent to SDFGs.

Setting $K = lcm_{a=(t,t')\in\mathcal{A}}(R_t \cdot p_a)$, where lcm denotes the least common multiple, SDFG normalization replaces by the value $Z_t = K/R_t$ all production/consumption weights adjacent to each actor $t \in \mathcal{T}$. This transformation amounts to multiplying by the integer $\delta_a = \frac{Z_t}{p_a} = \frac{K}{R_t \cdot p_a} = \frac{K}{R_{t'} \cdot c_a} = \frac{Z_{t'}}{c_a}$ the parameters of each buffer $a = (t, t') \in \mathcal{A}$, marking included.

For the SDFG of Figure 6, $K = lcm(42, 14, 6) = 42$ so that $Z_{t^1} = 7$, $Z_{t^2} = 6$ and $Z_{t^3} = 21$. As $\delta_{a_1} = 1$, $\delta_{a_2} = 3$ and $\delta_{a_3} = 7$, the initial marking of the normalized graph is $M_0(a_1) = 1 \cdot 6 = 6$, $M_0(a_2) = 3 \cdot 7 = 21$, $M_0(a_3) = 7 \cdot 1 = 7$. The equivalent normalized SDFG is shown in Figure 8.

Normalization has been extended to the CSDFG model by Benazouz *et al.* [4]. The extension to the PCG model is immediate. Indeed, for each buffer $a = (t, t') \in \mathcal{A}$, all the values $p_a(k)$, $1 - \sigma(t) \le k \le \phi(t)$, and $\theta_a(k')$ and $c_a(k')$, $1 - \sigma(t') \le k' \le \phi(t')$, can be normalized by multiplying them by the scaling factor δ_a, since it is a positive integer.

The normalization of the PCG of Figure 7 yields the equivalent PCG shown in Figure 9.

4.2 Additional notations

Consider a PCG $\mathcal{G} = (\mathcal{T}, \mathcal{A})$. A task $t \in \mathcal{T}$ has $\sigma(t)$ initial phases and $\phi(t)$ cyclic phases.

The n^{th} execution of the k^{th} phase of task t is denoted by (t_k, n), where n is a positive integer. If $n = 1$, k belongs to $s(t) = \{1 - \sigma(t), \cdots, 0, 1, \cdots, \phi(t)\}$ and the non-positive values of k are related to the initial phases of t. Otherwise, $n > 1$ and $k \in \{1, \cdots, \phi(t)\}$ relates to t's normal phases.

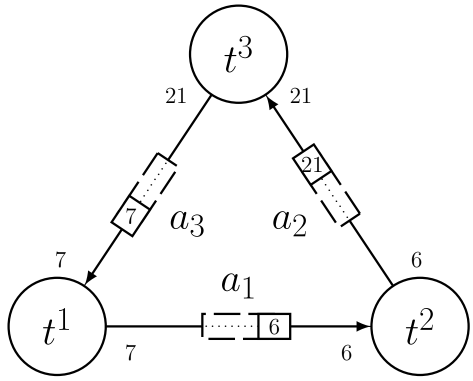

Figure 8: SDFG resulting from the normalization of the SDFG of Figure 6. The normalization factors applied to the buffer parameters are $\delta_{a_1} = 1$, $\delta_{a_2} = 3$ and $\delta_{a_3} = 7$.

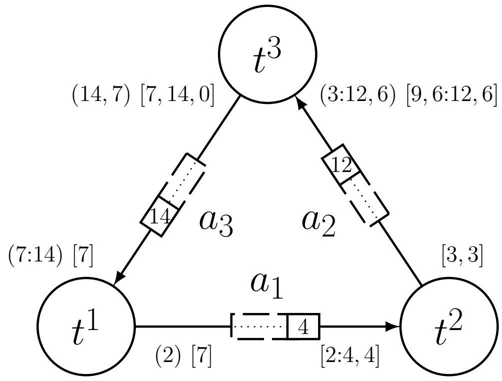

Figure 9: PCG resulting from the normalization of the PCG of Figure 7. The normalization factors applied to the buffer parameters are $\delta_{a_1} = 1$, $\delta_{a_2} = 3$ and $\delta_{a_3} = 7$.

$Pr(t_k, n)$ is the phase execution preceding (t_k, n). It is formally defined as:

$$Pr(t_k, n) = \begin{cases} (t_{k-1}, n) & \text{if } k > 1 \text{ and } n > 1 \\ (t_{\phi(t)}, n-1) & \text{if } k = 1 \text{ and } n > 1 \\ (t_{k-1}, 1) & \text{if } k \ge 1 - \sigma(t) \text{ and } n = 1 \end{cases}$$

Execution $(t_{-\sigma(t)}, 1)$ is fictitious and precedes $(t_{1-\sigma(t)}, 1)$.

Consider an execution (t_k, n) of task t and denote by $P_a(t_k, n)$ the total amount of data produced by t in a from its first phase to the end of (t_k, n). The cumulated production $P_a(t_k, n)$ satisfies the recurrence equation

$$P_a(t_k, n) = P_a Pr(t_k, n) + p_a(k) \qquad (1)$$

with $P_a(t_{-\sigma(t)}, 1) = 0$.

Now if, for any execution (t_k, n) with $k \in \{1, \cdots, \phi(t)\}$, $P_a^{\phi}(t_k, n)$ denotes the cumulated production without considering the initial phases—as if $\mathcal{G}$ was a CSDFG—hence starting with $P_a^{\phi}(t_{\phi(t)}, 0) = 0$, then, for any execution (t_k, n) with $k \in s(t)$, the cumulated production $P_a(t_k, n)$ may be expressed as

$$P_a(t_k, n) = P_a(t_k, 1) + P_a^{\phi}(t_{\phi(t)}, n-1) \quad . \qquad (2)$$

Consider for instance buffer $a = (t, t')$ in Figure 5: $\phi(t) =$

2, $P_a(t_{-1},1)=1$, $P_a(t_0,1)=3$, $P_a(t_1,1)=6$, $P_a(t_2,1)=7$. For any positive integer n, $P_a^{\phi}(t_2,n-1)=4(n-1)$. Thus, for any execution (t_k,n) of t, $P_a(t_k,n)=P_a(t_k,1)+4(n-1)$.

Now consider an execution (t'_k,n) of task t' and denote by $C_a(t'_k,n)$ the total amount of data consumed by t' in buffer a until the end of (t'_k,n). The cumulated consumption $C_a(t'_k,n)$ satisfies the recurrence equation

$$C_a(t'_k,n)=C_aPr(t'_k,n)+c_a(k). \quad (3)$$

with $C_a(t_{-\sigma(t')},1)=0$.

If, for any execution (t'_k,n) with $1\leq k\leq\phi(t')$, $C_a^{\phi}(t'_k,n)$ denotes the cumulated consumption without considering the initial phases, hence starting with $C_a^{\phi}(t'_{\phi(t')},0)=0$, then, for any execution (t'_k,n) with $k\in s(t')$, the cumulated consumption $C_a(t'_k,n)$ may be expressed as

$$C_a(t'_k,n)=C_a(t'_k,1)+C_a^{\phi}(t'_{\phi(t')},n-1). \quad (4)$$

Pursuing with Figure 5's example, $\phi(t')=3$, $C_a(t'_{-1},1)=2$, $C_a(t'_0,1)=5$, $C_a(t'_1,1)=7$, $C_a(t'_2,1)=8$, $C_a(t'_3,1)=10$. For any positive integer n, $C_a^{\phi}(t'_3,n-1)=5(n-1)$ and thus, for any execution (t'_k,n) of t', $C_a(t'_k,n)=C_a(t'_k,1)+5(n-1)$.

4.3 Feasible sequences and useful tokens

A sequence of firings ν is defined by a sequence of actors $\nu=t^1t^2\cdots t^p$, $t^i\in\mathcal{T}$ for $i\in\{1,\cdots,p\}$, that are executed (fired) consecutively. Furthermore, the execution of t reached at the end of the subsequence $t^1\cdots t^i$ of ν, $i\in\{1,\cdots,p\}$, is denoted by $(t_{k(i,t)},n_t^i)$. The sequence ν is then feasible if the following two conditions hold:

1. For any arc $a=(t,t')\in\mathcal{A}$ and any value $i\in\{1,\cdots,p\}$, the marking of a is non negative after the execution of the subsequence $t^1\cdots t^i$, thus $M_0(a)+P_a(t_{k(i,t)},n_t^i)-C_a(t'_{k(i,t')},n_{t'}^i)\geq 0$.
2. For any arc $a=(t,t')\in\mathcal{A}$ and any value $i\in\{1,\cdots,p\}$, a contains at least $\theta_a(k(i,t'))$ data items before the execution of $(t'_{k(i,t')},n_{t'}^i)$, thus $M_0(a)+P_a(t_{k(i,t)},n_t^i)-C_aPr(t'_{k(i,t')},n_{t'}^i)-\theta_a(k(i,t'))\geq 0$.

Note that, as $\theta_a(k(i,t'))\geq c_a(k(i,t'))$, the second condition is more restrictive. The first one can thus be omitted.

The notion of useful token is a limitation on the initial markings. It was first introduced by Marchetti and Munier in [11] and extended to CSDFGs by Benazouz *et al.* in [4]. Its generalization to PCGs is guaranted by the following lemma.

LEMMA 1 (USEFUL TOKENS). *In a PCG $\mathcal{G}=(\mathcal{T},\mathcal{A})$, replacing for each arc $a=(t,t')\in\mathcal{A}$, $M_0(a)$ by $M_0^{\star}(a)=\lfloor M_0(a)/h_a\rfloor\cdot h_a$, with $h_a=gcd(p_a(1-\sigma(t)),\cdots,p_a(\phi(t)),c_a(1-\sigma(t')),\cdots,c_a(\phi(t')),\theta_a(1-\sigma(t')),\cdots,\theta_a(\phi(t')))$, has no incidence on the feasible sequences.*

PROOF. For any sequence $\nu=t^1t^2\cdots t^p$, and for any value $i\in\{1,\cdots,p\}$, the condition of feasibility associated to the arc $a=(t,t')$ is strictly equivalent to

$$M_0^{\star}(a)+P_a(t_{k(i,t)},n_t^i)-C_aPr(t'_{k(i,t')},n_{t'}^i)-\theta_a(k(i,t'))\geq 0,$$

as h_a is a divisor of all the terms of the inequality. This proves the assertion. □

We suppose in the sequel that the initial marking fulfills the condition of Lemma 1.

4.4 A sufficient condition of liveness for a PCG

Lemma 2 is a technical result related to the total number of data items produced into a buffer.

LEMMA 2. *The cumulated production P_a into buffer $a=(t,t')\in\mathcal{A}$ satisfies, for any execution (t_k,n) of t with n a positive integer,*

$$P_aPr(t_k,n)=P_a^{\phi}(t_{\phi(t)},n-1)+P_aPr(t_k,1).$$

PROOF. Three cases must be considered:

- If $k>1$ and $n>1$, then $Pr(t_k,n)=(t_{k-1},n)$ and $Pr(t_k,1)=(t_{k-1},1)$. The equation is true following equality (2).
- If $k=1$ and $n>1$, then $Pr(t_1,n)=(t_{\phi(t)},n-1)$ and $Pr(t_1,1)=(t_0,1)$. Now, by equality (2),
$$P_a(t_{\phi(t)},n-1)=P_a^{\phi}(t_{\phi(t)},n-2)+P_a(t_{\phi(t)},1).$$
As $P_a(t_{\phi(t)},1)=P_a(t_0,1)+Z_t$ and $P_a^{\phi}(t_{\phi(t)},n-1)=P_a^{\phi}(t_{\phi(t)},n-2)+Z_t$, we get
$$P_a(t_{\phi(t)},n-1)=P_a^{\phi}(t_{\phi(t)},n-1)+P_a(t_0,1),$$
hence the equation is true.
- Lastly, if $k\geq 1-\sigma(t)$ and $n=1$, then $Pr(t_k,1)=(t_{k-1},1)$ and, since $P_a^{\phi}(t_{\phi(t)},0)=0$ by convention, the equation is true. □

Lemma 3 is similarly true for consumption weights. Its proof is therefore ommitted.

LEMMA 3. *The cumulated consumption C_a from buffer $a=(t,t')\in\mathcal{A}$ satisfies, for any execution (t'_k,n) of t' with n a positive integer,*

$$C_aPr(t'_k,n)=C_a^{\phi}(t_{\phi(t')},n-1)+C_aPr(t'_k,1).$$

The core of our proof is to characterize a deadlock. A cycle $\mu=(t^1,a_1,t^2,a_2,\cdots,t^m,a_m,t^1)$ is said to be blocked if there exists a set of executions $\mathcal{S}=\{(t^i_{k_i},n_i),t^i\in\mathcal{T}\}$ and a feasible sequence of firings reaching all the executions $Pr(t^i_{k_i},n_i)$, $t^i\in\mathcal{T}$, such that no execution from $\mathcal{S}$ can be fired because of μ. $\mathcal{G}$ initially marked by M_0 is then live if no cycle may be blocked.

Note that, since $\mathcal{G}$ is normalized, the total amount of data remains constant in every cycle. The following lemma provides an upper bound on the total amount of data in a blocked cycle.

LEMMA 4. *Let $\mathcal{G}$ be a normalized PCG. If a cycle $\mu=(t^1,a_1,t^2,a_2,\cdots,t^m,a_m,t^1)$, with $a_m=a_0$, is blocked, then there exists $k_i\in s(t^i)$, $i\in\{1,\cdots,m\}$, such that*

$$\sum_{i=1}^m M_0(a_i)+\sum_{i=1}^m w(i,k_i)\leq-\sum_{i=1}^m h_{a_i}$$

with $w(i,k_i)=P_{a_i}Pr(t^i_{k_i},1)-C_{a_{i-1}}Pr(t^i_{k_i},1)-\theta_{a_{i-1}}(k_i)$.

PROOF. Suppose that the cycle μ is blocked, then there exists a feasible sequence of firings such that each actor t^i, $i\in\{1,\cdots,m\}$, has reached execution $Pr(t^i_{k_i},n_i)$ but cannot execute $(t^i_{k_i},n_i)$.

Thus, for any arc $a_i=(t^i,t^{i+1})$ of μ, with $t^{m+1}=t^1$,

$$M_0(a_i)+P_{a_i}Pr(t^i_{k_i},n_i)-C_{a_i}Pr(t^{i+1}_{k_{i+1}},n_{i+1})-\theta_{a_i}(k_{i+1})<0.$$

Now it follows from Lemmas 2 and 3 that

$$P_{a_i}Pr(t^i_{k_i}, n_i) = P^{\phi}_{a_i}(t^i_{\phi(t^i)}, n_i - 1) + P_{a_i}Pr(t^i_{k_i}, 1)$$

$$C_{a_i}Pr(t^{i+1}_{k_{i+1}}, n_{i+1}) = C^{\phi}_{a_i}(t^{i+1}_{\phi(t^{i+1})}, n_{i+1}-1)+C_{a_i}Pr(t^{i+1}_{k_{i+1}}, 1)$$

so that the above inequality may be rewritten as
$M_0(a_i) + P^{\phi}_{a_i}(t^i_{\phi(t^i)}, n_i - 1) + P_{a_i}Pr(t^i_{k_i}, 1)$
$- C^{\phi}_{a_i}(t^{i+1}_{\phi(t^{i+1})}, n_{i+1} - 1) - C_{a_i}Pr(t^{i+1}_{k_{i+1}}, 1) - \theta_{a_i}(k_{i+1}) < 0.$

By Lemma 1, the initial marking is supposed to be divisible by h_a, and so are all the terms of the last inequality. This inequality may thus be rewritten as
$M_0(a_i) + P^{\phi}_{a_i}(t^i_{\phi(t^i)}, n_i - 1) + P_{a_i}Pr(t^i_{k_i}, 1)$
$- C^{\phi}_{a_i}(t^{i+1}_{\phi(t^{i+1})}, n_{i+1} - 1) - C_{a_i}Pr(t^{i+1}_{k_{i+1}}, 1) - \theta_{a_i}(k_{i+1}) \leq -h_{a_i}.$

As $\mathcal{G}$ is normalized, $C^{\phi}_{a_i}(t^i_{\phi(t^i)}, n_i - 1) = P^{\phi}_{a_i}(t^i_{\phi(t^i)}, n_i - 1)$. Summing the previous inequalities yields
$\sum_{i=1}^m M_0(a_i) + \sum_{i=1}^m w(i, k_i) \leq -\sum_{i=1}^m h_{a_i}$,
with $w(i, k_i) = P_{a_i}Pr(t^i_{k_i}, 1) - C_{a_{i-1}}Pr(t^i_{k_i}, 1) - \theta_{a_{i-1}}(k_i)$, the result which was to be proven. □

The next theorem is based on Lemma 4 and expresses a general sufficient condition of liveness of a PCG.

THEOREM 2 (SCB). *Let $\mathcal{G}$ be a normalized PCG. $\mathcal{G}$ is live if for every cycle $\mu = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$, with $a_m = a_0$, the following inequality is true:*

$$\sum_{i=1}^{m} M_0(a_i) > \sum_{i=1}^{m} W(a_{i-1}, a_i),$$

with $W(a_{i-1}, a_i) = \max_{k \in s(t^i)}[C_{a_{i-1}}Pr(t^i_k, 1) + \theta_{a_{i-1}}(k) - P_{a_i}Pr(t^i_k, 1)] - h_{a_i}$.

PROOF. Suppose that the condition expressed by the theorem is true for any cycle $\mu = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m)$. Since $-w(i, k) - h_{a_i} \leq W(a_{i-1}, a_i)$, $\forall i \in \{1, \cdots, m\}$, $\forall k \in s(t^i)$, for any sequence of $k_i \in s(t^i)$,

$$\begin{array}{ll}\sum_{i=1}^m M_0(a_i) & > \sum_{i=1}^m W(a_{i-1}, a_i) \\ & > \sum_{i=1}^m -w(i, k_i) - \sum_{i=1}^m h_{a_i}.\end{array}$$

Since this inequality is true for any sequence of $k_i \in s(t^i)$, $i \in \mathcal{T}$, by taking the contraposition of Lemma 4, cycle μ is not blocked. The consequence is that $\mathcal{G}$ does not contain any blocked cycle and hence is live, thus proving the theorem. □

Note that, if the graph $\mathcal{G}$ is an SDFG, then for any couple of arcs $a = (t, t')$ and $a' = (t', t'')$, $s(t') = \{1\}$, $C_a Pr(t_1, 1) = C_a(t_0, 1) = 0$ and $P_{a'}Pr(t_1, 1) = P_{a'}(t_0, 1) = 0$. We get then $W(a, a') = Z_t - gcd(Z_t, Z_{t'})$, leading to the following theorem proved initially by Marchetti and Munier in [11]:

THEOREM 3 ([11]). *Let $\mathcal{G}$ be a normalized SDFG. $\mathcal{G}$ is live if for every cycle $\mu = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ of $\mathcal{G}$, with $a_m = a_0$, the following inequality is true:*

$$\sum_{i=1}^{m} M_0(a_i) > \sum_{i=1}^{m} Z_{t^i} - \sum_{i=1}^{m} gcd(Z_{t^i}, Z_{t^{i+1}}).$$

4.5 Computation of the sufficient condition

We suppose in this subsection that $\mathcal{G} = (\mathcal{T}, \mathcal{A})$ is a normalized PCG, initially marked by M_0, and show that our sufficient condition of liveness (SCB) can be tested polynomially.

Consider the valued graph $\mathcal{H} = (\mathcal{A}, \mathcal{E}, v)$ defined as follows:

1. The vertices of $\mathcal{H}$ are the buffers;
2. An arc $e = (a, a')$ in $\mathcal{E}$ valued by $v(e) = W(a, a') - M_0(a')$ is associated to any couple of buffers $a = (t, t')$ and $a' = (t', t'')$.

(SCB) is then true if and only if $\sum_{e \in c} v(e) < 0$ for every circuit c of $\mathcal{H}$.

The graph $\mathcal{H}$ corresponding to the PCG pictured in Figure 9 is shown in Figure 10. The only circuit in $\mathcal{H}$, $c = (a_1, a_2, a_3)$, verifies $\sum_{e \in c} v(e) = 0 - 2 - 6 < 0$, and thus the PCG is live.

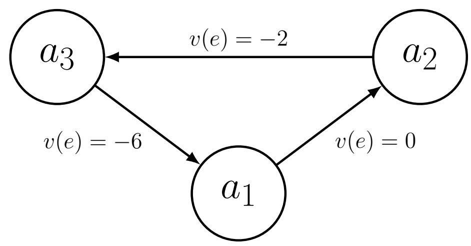

Figure 10: The valued graph $\mathcal{H} = (\mathcal{A}, \mathcal{E}, v)$ associated to the PCG of Figure 9.

The computation of $v(e)$ for a fixed arc $e = (a, a')$ is bounded by $\mathcal{O}(\max_{t \in T}(\phi(t) + \sigma(t))$. The computation of $\mathcal{H}$ has a complexity of $\mathcal{O}(|\mathcal{A}|^2 \times (\max_{t \in T}(\phi(t) + \sigma(t))$. Checking the condition can be done using the Bellmann-Ford algorithm coupled with a depth-first algorithm in time complexity $\mathcal{O}(|\mathcal{A}|^3)$ as described in [11].

4.6 Lower bound for buffer sizing

Computation of lower bounds for the buffer sizes is a simple application of the two sufficient conditions of liveness (SCA) and (SCB). Indeed, if a buffer $a = (t, t')$ has a limited size Δ, an associated backward arc $a' = (t', t)$ may be added ensuring that this size limit will be verified by any feasible sequence of firings. The parameters of a' for a PCG are then set to:

$$\begin{array}{ll}\forall k' \in s(t'), & p_{a'}(k') = c_a(k') \\ \forall k \in s(t), & c_{a'}(k) = p_a(k) \\ \forall k \in s(t), & \theta_{a'}(k) = p_a(k) \\ & M_0(a') = \Delta - M_0(a)\end{array}$$

Figure 11 represents a buffer with size limited to $\Delta = 10$.

The sufficient condition (SCA) can be applied to bound the buffer size in an SDFG or a CSDFG. The lower bound obtained is then

$$\Delta_A = R_t \cdot p_a = R_t \cdot \sum_{k=1}^{\phi(t)} p_a(k).$$

The sufficient condition (SCB) yields for an SDFG the lower bound $\Delta_B = Z_t + Z_{t'} - gcd(Z_t, Z_{t'})$. This value is proved to be optimum in [11] (*i.e.* it is the minimum buffer

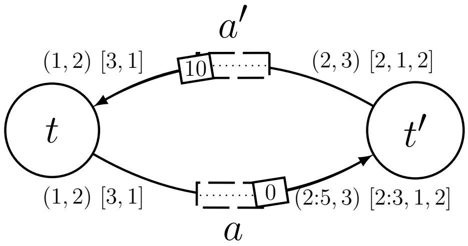

Figure 11: A buffer a with size limited to 10 between two tasks t and t' of a PCG.

size ensuring liveness), and thus $\Delta_B \leq \Delta_A$. These two values may be different. For instance, for the SDFG of Figure 1, $\Delta_A = 20$ while the minimum buffer size ensuring liveness is $\Delta_B = 4 + 5 - 1 = 8$.

The following theorem expresses the lower bound obtained using condition (SCB) for a general PCG.

THEOREM 4. *The value*

$$\Delta_B = \max_{k \in s(t)} p_a(k) + \max_{k' \in s(t')} \theta_a(k') - h_a$$

is an upper bound of the minimum size for any buffer $a = (t, t')$.

PROOF. For any value $k \in s(t)$,

$$\begin{aligned} C_{a'}Pr(t_k, 1) &= C_{a'}(t_{k-1}, 1) &= \textstyle\sum_{\ell=1-\sigma(t)}^{k-1} c_{a'}(\ell) \\ &= \textstyle\sum_{\ell=1-\sigma(t)}^{k-1} p_a(\ell) \\ &= P_a(t_{k-1}, 1) &= P_a Pr(t_k, 1). \end{aligned}$$

Hence, for any $k \in s(t)$,
$\theta_{a'}(k) + C_{a'}Pr(t_k, 1) - P_a Pr(t_k, 1) = \theta_{a'}(k)$
and

$$\begin{aligned} W(a', a) &= \max_{k \in s(t)} \theta_{a'}(k) - h_a \\ &= \max_{k \in s(t)} p_a(k) - h_a \end{aligned}$$

Similarly, $C_a Pr(t'_{k'}, 1) = P_{a'} Pr(t'_{k'}, 1)$ for any $k' \in s(t')$, and thus

$$W(a, a') = \max_{k' \in s(t')} \theta_a(k') - h_a.$$

The sufficient condition of liveness of Theorem 2 is thus:

$$\Delta_B > \max_{k \in s(t)} p_a(k) + \max_{k' \in s(t')} \theta_a(k') - h_{a'} - h_a.$$

Since h_a is a divisor of $h_{a'}$, the value $\Delta_B = M_0(a) + M_0(a')$ is divisible by h_a. The lowest feasible value fulfilling the inequality is thus $\Delta_B = \max_{k \in s(t)} p_a(k) + \max_{k' \in s(t')} \theta_a(k') - h_{a'}$. This completes the proof. $\square$

For example, the minimum size for the PCG of Figure 5 is $\Delta_B = 3 + 5 - 1 = 7$.

The question of the optimality of Δ_B is an interesting challenge, even for CSDFGs.

5. DESCRIPTION OF TURBINE

This section briefly presents our generator `Turbine`. Subsection 5.1 provides the list of input parameters and some implementation details. Subsection 5.2 explains how the various weights of a PCG are determined. The last subsection is devoted to the computation of an initial live marking, which is the most important contribution of our generator.

5.1 Overview

`Turbine` generates randomly PCGs, as well as all the subclasses described in Section 2. Several parameters must be fixed. Some parameters, such as the total number of actors $|\mathcal{T}|$ and bounds of the degree of the actors, relate directly to the graph structure. The other parameters control the generation of random weight vectors; they are the average value of the repetition vector, $R_{\mathcal{T}} = \|R\| / |\mathcal{T}|$ with $\|R\| = \sum_{t \in \mathcal{T}} R_t$, the average number $\phi_{\mathcal{T}}$ of cyclic phases and the average number of initial phases of a task.

`Turbine` is implemented using Python and the graph library NetworkX. The dataflow graphs are generated using the XML format issued from SDF^3. The Linear Programming Solver used for the determination of a live marking is GLPK.

The overall structure of `Turbine` reflects a natural division in three steps. The connected graph generation step is very similar to that of SDF^3 and DFTools. The weight computation step generates randomly (following a uniform law) a normalized dataflow graph. The live marking computation step is based on sufficient condition (SCB).

5.2 Weight computation

This step generates randomly all the weight vectors associated with the buffers, ensuring that the final PCG be consistent. First the components R_t of the repetition vector are generated randomly, with average the value $R_{\mathcal{T}}$ selected by the user, and are redrawn whenever needed to enforce $gcd(R_1, R_2, \cdots, R_{|\mathcal{T}|}) = 1$. The normalized weights Z_t, $t \in \mathcal{T}$, are then deduced as detailed in Subsection 4.1.

Next, the numbers of phases $\phi(t)$ and $\sigma(t)$, $t \in \mathcal{T}$, are randomly generated with given averages. Finally, for every buffer $a = (t, t') \in \mathcal{A}$, the vectors p_a, c_a and θ_a are randomly generated such that $\sum_{k=1}^{\phi(t)} p_a(k) = Z_t$, $\sum_{k'=1}^{\phi(t')} c_a(k) = Z_{t'}$ and, for any $k' \in \{1 - \sigma(t'), \cdots \phi(t)\}$, $\theta_a(k') \geq c_a(k')$.

5.3 Marking computation

The third PCG generation step is the computation of a live initial marking. This step is critical and takes much more time than the other two.

The input is a normalized PCG $\mathcal{G} = (\mathcal{T}, \mathcal{A})$. A couple of buffers (a, a') such that $a = (t, t') \in \mathcal{A}$ and $a' = (t', t'') \in \mathcal{A}$, with t, t' and $t'' \in \mathcal{T}$, defines an edge $e = (a, a')$ in the graph $\mathcal{H} = (\mathcal{A}, \mathcal{E})$ with arc weight $W(e) = W(a, a')$ defined in Subsection 4.4.

The minimization of the total amount of initial data ensuring that the sufficient condition of liveness expressed by Theorem 2 is met can be modeled by the following Integer Linear Program:

Minimize $\sum_{a \in \mathcal{A}} M_0(a)$
subject to :

$$\begin{cases} \gamma_{a'} - \gamma_a + M_0(a) - \epsilon \geq W(a, a') & \forall (a, a') \in \mathcal{E} \\ M_0(a) = h_a \cdot m_0(a) & \forall a \in \mathcal{A} \\ M_0(a) \in \mathbb{N}, m_0(a) \in \mathbb{N} & \forall a \in \mathcal{A} \\ \gamma_a \in \mathbb{R} & \forall a \in \mathcal{A} \end{cases}$$

The value of the parameter ϵ is set to $1/|\mathcal{T}|$. By minimizing the total amount of data, we ensure that the buffers with a strictly positive number of initial items necessarily belong to cycles, *i.e.* any buffer a which is not in a cycle verifies $M_0(a) = 0$.

Now suppose that $c = (t^1, a_1, t^2, a_2, \cdots, t^m, a_m, t^1)$ is a cycle of $\mathcal{G}$. Summing the m inequalities associated with a_i, $i \in \{1, \cdots, m\}$, gives:

$$\sum_{i=1}^{m} M_0(a_i) - m.\epsilon \geq \sum_{i=1}^{m} W(a_i, a_{i+1}).$$

As any value $M_0(a)$, $a \in \mathcal{A}$, is divisible by h_a, the condition of Theorem 2 is met and the marking is live.

The exact resolution of this integer linear program is unfortunately not possible within a reasonable time. The idea is thus to solve a linear programming relaxation of that problem by replacing $M_0(a) \in \mathbb{N}$ and $m_0(a) \in \mathbb{N}$, $\forall a \in \mathcal{A}$, by $M_0(a) \in \mathbb{R}^+$ and $m_0(a) \in \mathbb{R}^+$. A feasible solution is then derived from the optimal real solution $m_0^{real}(a)$, $\forall a \in \mathcal{A}$, by setting $M_0(a) = \lceil m_0^{real}(a) \rceil \cdot h_a$.

6. EXPERIMENTAL RESULTS

This section is devoted to experiments on `Turbine` and experimental comparisons with the generators SDF^3 and DFTools. Subsection 6.1 presents the experimental conditions. Performances of `Turbine` as a function of several parameters (size of the instances, average values of the repetition vector components and of the number of phases) are discussed in Subsection 6.2. The performances and the sum of the initial markings for the three generators are compared in the last two subsections.

6.1 Experimental conditions

All our algorithms were coded using Python 2.6.6. Original source codes of SDF^3 and DFTools were used.

Most of the experiments were performed on an Intel Core2-Duo E7500@3GHz using 2GB of RAM with a 32-bit OS. A server with two Xeon E5-2637 and 128GB was used to compute large instances with DFTools since it requires a Java Virtual Machine with 32GB of RAM.

Each experiment was launched 100 times, and the average value of interest (time or sum of initial markings) was returned. Four graph sizes have been selected: *Tiny* for 10 actors, *Small* for 100, *Medium* for 1000 and *Large* for 10,000. Both input and output degree of the actors belong to the set $\{1, 2, 3, 4, 5\}$. The average value $R_{\mathcal{T}}$ of the components of the repetition vector is set to 5. For CSDFGs and PCGs the number of phases (initial and cyclic) belongs to $\{1, 2, 3, 4, 5\}$ with an average $\phi_{\mathcal{T}} = 3$.

6.2 Performance of Turbine

Table 1 presents the average generation time of a PCG as a function of graph size. The generation time grows roughly quadratically with $|\mathcal{T}|$, thus approaching a linear growth with respect to the number of buffers.

Table 1: Average PCG generation time for `Turbine`.

$\|\mathcal{T}\|$	generation time
Tiny	0.01s
Small	0.3s
Medium	53s
Large	2h7mn15s

The influence on `Turbine`'s generation time of the average values of the repetition vector components, $R_{\mathcal{T}}$, and of the number of phases, $\phi_{\mathcal{T}}$, was measured for medium-size graphs with $R_{\mathcal{T}} \in \{5, 10, 15\}$ and $\phi_{\mathcal{T}} \in \{5, 10, 15\}$. The results do not vary significantly. Indeed, both parameters have no influence on the size of the linear program, which depends linearly on the number of buffers.

6.3 Comparison of the generation time

The comparison of the performances of SDF^3, DFTools and `Turbine` is only possible for SDFGs.

Table 2 presents the average generation time of the three generators for the four graph sizes, from tiny to large. All three generators support instances of up to medium size within a reasonable time.

Table 2: Comparison of the SDFG generation times of `Turbine`, SDF^3 and DFTools. A $\star$ indicates that the time was measured on a two-Xeon server.

$\|\mathcal{T}\|$	Turbine	SDF^3	DFTools
Tiny	0.01s	0.03s	0.2s
Small	0.1s	2s	0.5s
Medium	2.8s	1h10mn30s	11.5s
Large	10mn14s	-	1h2mn36s*

Table 3 presents a comparison between `Turbine` and SDF^3 for the generation of CSDFGs (DFTools only generates SDFGs). The comparison was only possible for up to small instances since SDF^3 takes more than a day to generate a single CSDFG of medium size.

Table 3: Comparison of the CSDFG generation times of `Turbine` and SDF^3.

$\|\mathcal{T}\|$	Turbine	SDF^3
Tiny	0.01s	0.02s
Small	0.1s	3.8s
Medium	46s	-

In all cases, `Turbine` substantially outperforms SDF^3 and DFTools. It is the only one able to generate large SDFGs and medium to large CSDFGs on a regular desktop computer.

6.4 Comparison of the initial markings

Averages of the sum of the initial markings were measured when using SDF^3, DFTools and `Turbine` for SDFGs and CSDFGs.

Table 4 shows that for SDFGs the average sums of the initial markings grow linearly with the number of actors for all three generators. As illustrated in Figure 12, the proportionality factors are in the ratios 1 : 1.6 : 2.6 for `Turbine`, DFTools and SDF^3, respectively.

Table 4: Average sums of the initial markings for SDFGs generated by SDF^3, DFTools and `Turbine`.

$\|\mathcal{T}\|$	Turbine	SDF^3	DFTools
Tiny	3.5×10^3	9.2×10^3	5.2×10^3
Small	1.2×10^5	3.1×10^5	2.1×10^5
Medium	1.2×10^6	3.1×10^6	2.1×10^6
Large	1.2×10^7	-	2.1×10^7

The comparison of initial markings between SDF^3 and `Turbine` for CSDFGs is not displayed as it is already not possible for graphs of medium size because of the excessive generation time of SDF^3.

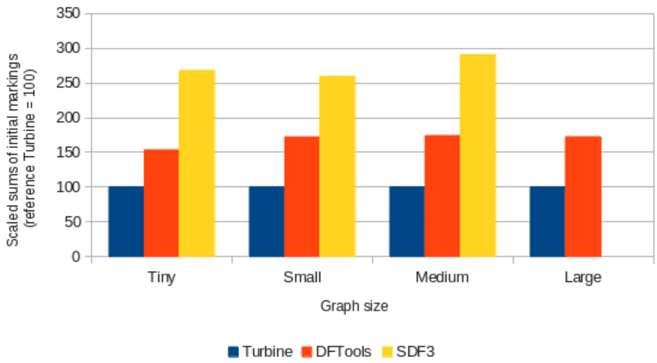

Figure 12: Comparison of the average sums of initial markings for SDFGs using `Turbine` as reference (set to 100).

7. CONCLUSION

This paper presents the first scalable generator for live Phased Computation Graphs (PCG). The PCG model is employed in an industrial context for a new generation of embedded systems. It extends the CSDFG model by including initial phases and thresholds.

Another major contribution is the extension to the PCG model of mathematical results—the normalization and a necessary sufficient condition of liveness—previously known for the SDFG and CSDFG models.

Our generator **`Turbine`** uses fruitfully these results for the computation of initial conditions minimizing the total amount of initial data in the channels. The scalability of **`Turbine`** is experimentally established for PCGs of 10,000 actors. Its performances (time, total amount of initial data) for SDFGs and CSDFGs compare favorably with those of the two existent generators SDF^3 and DFTools.

8. REFERENCES

[1] www.tilera.com.

[2] M. A. Bamakhrama, J. T. Zhai, H. Nikolov, and T. Stefanov. A methodology for automated design of hard-real-time embedded streaming systems. In *Design, Automation & Test in Europe (DATE'12)*, pages 941–946, Mar. 2012.

[3] M. Benazouz, O. Marchetti, A. Munier-Kordon, and T. Michel. A new method for minimizing buffer sizes for cyclo-static dataflow graphs. In *8th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'10)*, pages 11–20, 2010.

[4] M. Benazouz, A. Munier-Kordon, T. Hujsa, and B. Bodin. Liveness Evaluation of a Cyclo-Static DataFlow Graph. In *Design Automation Conference (DAC'13)*, pages 3–7, 2013.

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

[6] B. Bodin, A. Munier-Kordon, and B. Dupont de Dinechin. Periodic Schedules for Cyclo-Static Dataflow. In *11th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'13)*, pages 105–114, 2013.

[7] T. Goubier, R. Sirdey, S. Louise, and V. David. ΣC: A programming model and language for embedded manycores. *11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, pages 385–394, 2011.

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

[9] R. M. Karp and R. E. Miller. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal on Applied Mathematics*, 14(6):1390–1411, 1966.

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

[11] 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.

[12] H. Oh and S. Ha. Efficient code synthesis from extended dataflow graphs for multimedia applications. In *Design Automation Conference (DAC '02)*, pages 275–280, 2002.

[13] M. Pelcat, J. Piat, M. Wipliez, S. Aridhi, and J.-F. Nezan. An Open Framework for Rapid Prototyping of Signal Processing Applications. *EURASIP Journal on Embedded Systems*, pages 1–13, 2009.

[14] R. Read. A survey of graph generation techniques. *Combinatorial Mathematics VIII*, pages 77–89, 1981.

[15] F. Siyoum, M. Geilen, O. Moreira, and H. Corporaal. Worst-case throughput analysis of real-time dynamic streaming applications. *8th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'12)*, pages 463–472, 2012.

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

[17] S. Stuijk, M. Geilen, and T. Basten. SDF3: SDF For Free. In *6th International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 276–278, 2006.

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

[19] W. Thies, J. Lin, and S. Amarasinghe. Phased computation graphs in the polyhedral model. *Technical Report LCS-TM-630, MIT Laboratory for Computer Science*, 2002.

[20] M. H. Wiggers, M. J. Bekooij, P. G. Jansen, and G. J. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Real-Time Systems with Back-Pressure. *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 281–292, Apr. 2007.

[21] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal. Exploring Trade-offs between Performance and Resource Requirements for Synchronous Dataflow Graphs. In *7th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'09)*, pages 96–105, 2009.