Liveness and Boundedness of Synchronous Data Flow Graphs *

A.H. Ghamarian, M.C.W. Geilen, T. Basten, B.D. Theelen, M.R. Mousavi and S. Stuijk
Eindhoven University of Technology, Electronic Systems Group
a.h.ghamarian@tue.nl

Abstract. *Synchronous Data Flow Graphs (SDFGs) have proven to be suitable for specifying and analyzing streaming applications that run on single- or multi-processor platforms. Streaming applications essentially continue their execution indefinitely. Therefore, one of the key properties of an SDFG is liveness, i.e., whether all parts of the SDFG can run infinitely often. Another elementary requirement is whether an implementation of an SDFG is feasible using a limited amount of memory. In this paper, we study two interpretations of this property, called boundedness and strict boundedness, that were either already introduced in the SDFG literature or studied for other models. A third and new definition is introduced, namely self-timed boundedness, which is very important to SDFGs, because self-timed execution results in the maximal throughput of an SDFG. Necessary and sufficient conditions for liveness in combination with all variants of boundedness are given, as well as algorithms for checking those conditions. As a by-product, we obtain an algorithm to compute the maximal achievable throughput of an SDFG that relaxes the requirement of strong connectedness in earlier work on throughput analysis.*

1 Introduction

Synchronous Data Flow Graphs (SDFGs, see [13]), also known as weighted Marked Graphs in Petri-net theory, are used widely in modelling and analyzing data flow applications. They are often used for modelling DSP applications [3, 19] and for designing concurrent multimedia applications implemented on multi-processor systems-on-chip [17]. The model is suitable for realizing a system with predictable performance properties as several analysis techniques like throughput analysis exist [8].

An SDFG is a graph with actors as vertices and channels as edges. Actors represent basic parts of an application which need to be executed. Channels represent data dependencies between actors. Execution of an actor is designated by an actor firing. Each actor generates a fixed number of tokens when it fires. These are stored in the channels with unlimited capacities. An execution of an SDFG is a sequence of actor firings which respects data dependencies. The exact order of actor firings is not determined. Consequently, several executions exist for an SDFG. Because of the usage of SDFGs for modelling streaming applications, only those SDFGs which have executions in which all actors are fired infinitely often are of interest. This property of SDFGs is called liveness. Furthermore, only executions that require a finite amount of storage for the channels are of interest. This paper formally studies three different interpretations of this second property, all in combination with liveness.

The paper investigates two known interpretations, namely boundedness (whether there exists a bounded execution of an SDFG) and strict boundedness (whether all executions are bounded). We prove necessary and sufficient conditions guaranteeing that an SDFG is live and (strictly) bounded. For strict boundedness, these conditions follow immediately from a similar result known for Petri nets.

The natural way of executing an SDFG in which all actors fire as soon as they can fire, is called self-timed execution. This execution is important since it leads to the maximal obtainable throughput of an SDFG [19]. Because of the importance of self-timed execution of SDFGs and its applications in the context of multi-processor systems, a new notion of boundedness, namely self-timed boundedness is introduced. This notion requires that self-timed execution of SDFGs is bounded. Necessary and sufficient conditions for the liveness and self-timed boundedness of SDFGs are proved. These conditions heavily depend on the throughput of actors (average number of firings of an actor per time unit). Existing techniques for throughput calculation only work for strongly connected SDFGs [6, 8]. We propose an algorithm that determines the liveness and self-timed boundedness of an SDFG and at the same time extends throughput analysis to arbitrary SDFGs. The concept of self-timed boundedness and the results proven for this notion are the main contribution of this paper.

The rest of this paper is organized as follows. Section 2 formally introduces SDFGs to allow studying liveness and boundedness in a rigorous way. Sections 3 and 4 present results for liveness and (strict) boundedness. Section 5 identifies conditions for self-timed boundedness of SDFGs and presents an algorithm for verifying the combination of liveness and this type of boundedness. Section 6 discusses related work, while Section 7 summarizes the conclusions. Proofs are omitted and can be found in [9].

2 Synchronous Data Flow Graphs

2.1 Basic Definitions

This section formally defines SDFGs and some of their basic properties. Let $\mathbb{N}_0 = \{0, 1, \ldots\}$ (and $\mathbb{N} = \mathbb{N}_0 \setminus$

*This work was supported by the Dutch Science Foundation NWO, project 612.064.206, PROMES, and the EU, project IST-004042, Betsy.

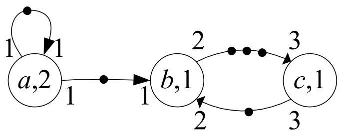

Figure 1. An example timed SDFG G_{ex}.

$\{0\}$) denote the (positive) natural numbers. The following definition captures the structure of an SDFG.

Definition 1 *[Synchronous Data Flow Graph (SDFG)] An SDFG is a pair (A, C), where A denotes the set of actors and $C \subseteq A^2 \times \mathbb{N}^2$ the set of channels. Each $(s, d, p, c) \in C$ denotes that actor d depends on actor s, where p and c are the production and consumption rates of tokens of s and d, respectively. The predecessors of a in $\mathrm{Pred}(a) = \{s \in A \mid (s, a, p, c) \in C\}$ are those actors on which a depends. The channels between a and its predecessors are referred to as the input channels of a, denoted by* $\mathrm{IC}(a)$. *Similarly, the successors of a in $\mathrm{Succ}(a) = \{d \in A \mid (a, d, p, c) \in C\}$ are those actors that depend on a. The output channels (channels between a and its successors) of a are denoted by* $\mathrm{OC}(a)$. *We call a channel from an actor a to itself a* self-loop channel. *We denote the set of self-loop channels of an actor a by $\mathrm{SLC}(a) = \mathrm{IC}(a) \cap \mathrm{OC}(a)$. An SDFG in which all production and consumption rates are one is called a* Homogeneous *SDFG (HSDFG).*

Figure 1 shows a simple example of an SDFG. Actors are labeled with their names and execution times (introduced later). Channels are labeled with production and consumption rates. The black dots are tokens. To capture the execution of an SDFG, we define the channel state of an SDFG as the distribution of tokens over its channels.

Definition 2 *[Channel State] A* channel state *of an SDFG (A, C) is a function $S : C \to \mathbb{N}_0$ that returns the number of tokens stored in each channel. Each SDFG has an initial channel state S_0 denoting the number of tokens that are initially stored in the channels.*

An execution of an SDFG is defined based on the firings of its actors, which may lead to changes in the channel state.

Definition 3 *[Firing] Let $a \in A$ be an actor of an SDFG (A, C). Actor a is said to be enabled in channel state S in case $S(e) \geq c$ for all input channels $e = (s, a, p, c)$ in $\mathrm{IC}(a)$. If a is enabled in S_i and it fires, the resulting channel state S_{i+1} is defined by $S_{i+1}(e) = S_i(e) - c$ for each input channel $e = (s, a, p, c)$ in $\mathrm{IC}(a) \backslash \mathrm{SLC}(a)$, $S_{i+1}(e) = S_i(e) + p$ for each output channel $e = (a, d, p, c)$ in $\mathrm{OC}(a) \backslash \mathrm{SLC}(a)$, $S_{i+1}(e) = S_i(e) + p - c$ for each self-loop channel $e = (a, a, p, c) \in \mathrm{SLC}(a)$, and $S_{i+1}(e) = S_i(e)$ for all channels $e \notin \mathrm{IC}(a) \cup \mathrm{OC}(a)$.*

Definition 4 *[Execution and Maximal Execution] Let S_0 denote the initial channel state of an SDFG (A, C). An* execution *σ of (A, C) is a (finite or infinite) sequence of channel states $S_0, S_1 \ldots$ such that S_{i+1} is the result of firing an enabled actor in S_i for all $i \geq 0$. An execution is maximal if and only if it is finite with no actors enabled in the final channel state, or if it is infinite.*

Not all SDFGs are considered to be useful in practice. One normally seeks a system that is deadlock-free or live.

Definition 5 *[Deadlock and Liveness] An SDFG has a* deadlock *if and only if it has a maximal execution of finite length. An SDFG is* live *if and only if it has an execution in which all actors fire infinitely often.*

It is known [11] that the execution of an SDFG is determinate, which means that the order of execution does not affect the states that can eventually be reached. Thus, if one execution of an SDFG deadlocks, then all executions deadlock. The example SDFG G_{ex} is live.

2.2 Timed SDFGs

For performance analysis of streaming applications, an SDFG is often extended with time.

Definition 6 *[Execution Time] An* execution time *models the execution duration of actors for SDFGs. In an SDFG (A, C), the execution time is a function $E : A \to \mathbb{Q}_0^+ \cup \{\infty\}$ that assigns to each actor the amount of time it takes to fire, where $\mathbb{Q}_0^+ \cup \{\infty\}$ is the set of positive rational numbers plus 0 and ∞. For $a \in A$, $E(a)$ is referred to as the execution time of a.*

Definition 7 *[Timed SDFG] A* timed SDFG *is a triple (A, C, E) denoting an SDFG (A, C) with execution time E.*

The infinite execution times are used lateron to model deadlocks. Normally, SDFGs do not have infinite actor execution times.

Notice that actor firings in a timed SDFG are not atomic. Firing an actor now takes time. To define the state of a timed SDFG, we assume that all changes in the number of tokens on all channels of an actor happen at the end of its firing.

Definition 8 *[Timed State] A state of a timed SDFG (A, C, E) is a pair (S, τ), where S is a channel state and $\tau \in \mathbb{Q}_0^+$ is the accumulated time. The initial state of (A, C, E) is given by the initial channel state S_0 and the start time of the system $\tau_0 = 0$.*

Definition 9 *[Timed Execution] An execution of a timed SDFG (A, C, E) is a sequence of timed states $(S_0, \tau_0), (S_1, \tau_1), \ldots$, where $\tau_{i+1} \geq \tau_i$. Each two consecutive states (S_{i+1}, τ_{i+1}) and (S_i, τ_i) are the same except that an actor a which started its firing at $\tau_{i+1} - E(a)$ finishes its firing at τ_{i+1}. S_{i+1} is related to S_i in precisely the same way as defined in Definition 3.*

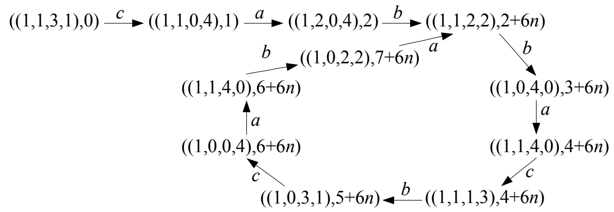

Figure 2. Self-timed execution of G_{ex}.

We denote the number of completed firings of an actor $a \in A$ which occurred up to time τ by $F_{a,\tau}$.

Among all timed executions there are some of special interest. A timed execution for which the firing of an actor always starts as soon as possible is called a *self-timed execution*. Self-timed executions are important in the context of performance analysis because they imply obtaining the maximal attainable throughput [19].

Definition 10 *[Self-timed Execution] A timed execution is called* self-timed *if and only if it is maximal and all actors start their firing as soon as they are enabled.*

If two or more actors complete their firing at some point in time in a self-timed execution, the order of their appearance in the execution is not determined. In other words, any permutation of such actor firings results in a self-timed execution. Thus, the number of self-timed executions is larger than one in such cases. Note that in all self-timed executions the start and end times of firings of all actors are equal. Also the channels states after completion of all actor firings that can complete at a certain point in time are the same in all self-timed executions.

Figure 2 illustrates a self-timed execution of the example SDFG G_{ex} of Figure 1. The state contains a channel component with the distribution of tokens over the channels *a*-*a*, *a*-*b*, *b*-*c*, *c*-*b*, respectively, and a time component. In the depicted cycle, the time component is denoted symbolically to emphasize that the behavior repeats itself every six time units, after some initial transient phase.

2.3 Structural Properties

The directed graph of an SDFG has some structural properties that are relevant for deciding boundedness. This paper assumes connected SDFGs for which the directed graph consists of *one* component. SDFGs consisting of multiple components can be considered as a set of single-component SDFGs, which can be analyzed separately.

A well known stronger form of connectivity is given by the following two definitions.

Definition 11 *[Path and Cycle] A directed path p is a sequence of actors $a_1, a_2 \ldots a_l$ such that $a_{i+1} \in \mathrm{Succ}(a_i)$ for all $1 \leq i < l$. Path p is simple iff $a_i \neq a_j$ for all $i \neq j$. If $a_1 = a_l$ and $l \geq 2$, then p is said to be a* cycle.

Definition 12 *[Strongly Connected SDFG] An SDFG is* strongly connected *iff there exists a directed path from any actor to any other actor. Any subgraph of an SDFG which is strongly connected is called a strongly connected component (SCC, for short). An SCC κ is maximal iff there is no SCC κ' where κ is a strict subgraph of κ'.*

Another structural property of SDFGs concerns the correspondence between production and consumption rates.

Definition 13 *[Consistency and Balance Equations] A repetition vector for an SDFG (A, C) is a function $\gamma : A \rightarrow \mathbb{N}_0$ such that for every $(s, d, p, c) \in C$, the equation $p\gamma(s) = c\gamma(d)$ holds. These equations are called* balance equations. *Repetition vector γ is called non-trivial iff $\gamma(a) > 0$ for all $a \in A$. If a non-trivial repetition vector exists, the SDFG is called* consistent. *The smallest non-trivial repetition vector of a consistent SDFG is referred to as* the *repetition vector.*

Note that the definitions in this subsection carry over to timed SDFGs in a straightforward way. Timed SDFG G_{ex} is consistent with repetition vector $(a \mapsto 3, b \mapsto 3, c \mapsto 2)$.

2.4 Throughput of Timed SDFGs

In this section the throughput of timed SDFGs is defined, and the relation between the execution of an SDFG and its throughput is explained.

Definition 14 *[Throughput] The* throughput *$Th(a)$ of an actor a for a self-timed execution of a timed SDFG (A, C, E) is defined as the average number of firings of a per time unit. Formally,*

$$Th(a) = \lim_{\tau \to \infty} \frac{F_{a,\tau}}{\tau}.$$

If $G = (A, C, E)$ is consistent, then its throughput is defined as

$$Th(G) = \min_{a \in A} \frac{Th(a)}{\gamma(a)},$$

where γ is the repetition vector of (A, C, E). That is, the throughput of G is the minimal actor throughput normalized by the repetition vector.

We define the *local* throughput of an actor as the throughput of that actor in a self-timed execution where non-self-loop input channels are removed; in other words, the throughput of an actor when it does not need to wait for data from other actors.

Definition 15 *[Local Throughput] The* local throughput *$LTh(a)$ of an actor a for a self-timed execution of a timed SDFG (A, C, E) is defined as*

$$LTh(a) = \begin{cases} 0, & \text{if there is a } ch = (a, a, p, c) \text{ in } \mathrm{SLC}(a) \\ & \quad \text{such that } p < c \text{ or } S_0(ch) < c \\ \min\limits_{ch=(a,a,r,r) \in \mathrm{SLC}(a)} \lfloor S_0(ch)/r \rfloor / E(a), & \text{otherwise.} \end{cases}$$

If an actor has a self-loop channel with a lower production rate than consumption rate or insufficient tokens for an initial firing, its local throughput is zero, i.e., it deadlocks at some point in time. Otherwise, the local throughput is determined by the self-loop channels with equal production and consumption rates. If there are no such channels, i.e., there are no self-loop channels or all self-loop channels have a higher production than consumption rate, local throughput is by definition infinite.

In a self-timed execution of a timed SDFG, there is always a time τ_p after which only a repetitive pattern of actor firings occurs (when ignoring the order among actor firing completions occurring at the same moment in time) [8, 1]. The self-timed execution from the beginning up to time τ_p is called the transient phase, and thereafter is addressed as the periodic phase. Figure 2 illustrates this fact. Thus, the throughput of an arbitrary actor a in the self-timed execution can be calculated by counting the number of occurrences of firings of a in one period divided by the amount of time that the period takes. The firings of a in one period can be spread over the period, but the number of firings of one actor in one period is always fixed.

Consider again SDFG G_{ex} of Figure 1. The local throughput of actor a is $\frac{1}{2}$, whereas it is ∞ for b and c. The throughput of the three actors equals $\frac{3}{6} = \frac{1}{2}$, $\frac{3}{6} = \frac{1}{2}$, and $\frac{2}{6} = \frac{1}{3}$, respectively. The graph throughput $Th(G_{ex})$ is determined by actor a (with repetition-vector entry 3) and is equal to $(\frac{3}{6})/3 = \frac{1}{6}$. This illustrates that the periodic behavior of the graph as a whole needs 6 time units per period.

2.5 Boundedness Definitions

Different useful notions of boundedness can be defined for SDFGs. To enable identifying these forms, we first define boundedness for a given execution.

Definition 16 *[Bounded Channel and Bounded Execution] Let $\sigma = S_0, S_1, \ldots$ be an execution of an SDFG (A, C). We call a channel ch bounded under σ iff there exists some $B \in I\!N$ such that $S_i(ch) \leq B$ for all $i \geq 0$. If all channels of the SDFG are bounded under σ then σ is bounded.*

Definition 16 carries over to timed executions in a straightforward way. Now, we give a definition for the boundedness of an SDFG which intuitively means that it can be implemented using a finite amount of memory.

Definition 17 *[Bounded SDFG] A (timed) SDFG is called* bounded *iff there exists a bounded maximal execution. It is unbounded otherwise.*

A stronger form of boundedness is *strict boundedness*.

Definition 18 *[Strictly Bounded Channel and Strictly Bounded SDFG] A channel is* strictly bounded *iff it is bounded under all executions. A (timed) SDFG is called strictly bounded iff all of its channels are strictly bounded.*

Note that any *strictly bounded* SDFG is also bounded. We finally define another form of boundedness, which only considers self-timed executions of timed SDFGs.

Definition 19 *[Self-timed Bounded SDFG] A timed SDFG is* self-timed bounded *iff all self-timed executions are bounded. A channel in a timed SDFG is self-timed bounded iff it is bounded in all self-timed executions.*

All self-timed bounded SDFGs are bounded but not necessarily strictly bounded. Running example G_{ex} is not strictly bounded because a can be fired indefinitely without firing b and c. However, it is self-timed bounded, as Figure 2 illustrates. It is not difficult to construct bounded SDFGs that are not self-timed bounded. If the execution times of actors b and c in G_{ex} are changed to 3, for example, then the SDFG remains bounded but it is no longer self-timed bounded. These examples show that the notion of self-timed boundedness does not coincide with other notions of boundedness. Given the importance of self-timed execution, it is worth investigating this notion in some detail.

3 Boundedness

In this section, we study necessary and sufficient conditions under which an SDFG is live and bounded.

Theorem 20 *A live SDFG $G = (A, C)$ is bounded iff it is consistent.*

Theorem 20 states the consistency of an SDFG as a necessary and sufficient condition for boundedness of live SDFGs. If a subgraph of an SDFG deadlocks (which means that the SDFG is not live) then the consistency of an SDFG is not sufficient for boundedness. For example, consider G_{ex} of Figure 1 without the initial token in the c-b channel. Execution times may be ignored. The resulting SDFG is consistent but not bounded. The SCC of the graph that consists of actors b and c deadlocks after the first firing of both actors. However, actor a can continue its firing, which leads to an unbounded channel between a and b.

Proposition 21 *[20] A strongly connected SDFG is live iff it is deadlock-free.*

The definition of liveness states that a live SDFG has an execution in which all actors fire infinitely often. If a live SDFG is strongly connected, then all actors fire infinitely often in *all maximal* executions.

Lemma 22 *If one SCC in an SDFG G deadlocks then either G deadlocks or it is unbounded.*

This lemma implies that a deadlock-free and bounded SDFG is live.

Corollary 23 *An SDFG is live and bounded iff it is deadlock-free and bounded.*

The following theorem follows from Theorem 20, Proposition 21, Lemma 22, and Corollary 23.

Theorem 24 *An SDFG is live and bounded iff it is consistent and all its SCCs are deadlock-free.*

The example SDFG G_{ex} is live and bounded because it is consistent and all its SCCs are deadlock-free.

Next, we give an algorithm to check liveness and boundedness of an SDFG.

Algorithm *isLive&Bounded*(G)
Input: A connected (timed) SDFG G
Output: "live and bounded" or "either deadlock or unbounded"

```
if G is inconsistent
   then return "either deadlock or unbounded"
for each maximal SCC S in G
     do if S deadlocks
          then return "either deadlock or unbounded"
return "live and bounded"
```

Consistency of SDFGs can be verified efficiently as explained in [3]. Maximal SCCs of a graph can also be computed efficiently [5]. Algorithms for detecting deadlock for consistent strongly connected SDFGs that are efficient in practice are given in [12, 8].

4 Strict Boundedness

This section identifies sufficient and necessary conditions for the liveness and strict boundedness of an SDFG.

Theorem 25 *[20, Theorem 4.11] A live (timed) SDFG is strictly bounded iff it is consistent and strongly connected.*

This theorem in combination with Proposition 21 implies the following theorem.

Theorem 26 *An SDFG is live and strictly bounded iff it is deadlock-free, consistent and strongly connected.*

So the algorithm for checking liveness and strict boundedness first checks whether the SDFG is strongly connected and consistent, and then whether it is deadlock-free using the algorithms from [5, 3, 8, 12]. The example of Figure 1 is not strictly bounded because it is not strongly connected.

5 Self-timed Boundedness

In this section, we investigate the liveness and self-timed boundedness of timed SDFGs. A self-timed execution of a live and self-timed bounded SDFG uses a finite amount of memory and all actors fire infinitely often in such an execution. Necessary and sufficient conditions for liveness and self-timed boundedness are given, and an algorithm for checking these conditions.

5.1 Some Basic Properties

Self-timed boundedness has a strong relationship with the throughput of an SDFG. In this subsection, some properties for the throughput as well as the relation between boundedness and throughput of timed SDFGs are given.

The throughput of an actor is only determined by the throughput of its predecessors and its local throughput.

Lemma 27 *The throughput of an actor $b \in A$ of a timed SDFG $G = (A, C, E)$ satisfies the equation*

$$Th(b) = \min\{\min_{(a,b,p,c)\in \mathrm{IC}(b)\setminus \mathrm{SLC}(b)} \frac{p}{c} Th(a), LTh(b)\}. \tag{1}$$

The throughput of actor b of G_{ex}, for example, is $\frac{1}{2}$, because its predecessor a has that throughput, the rates of channel a-b are 1 and its local throughput is ∞.

Corollary 28 *If actors $a, b \in A$ of an SDFG G are connected by a channel (a, b, p, c) then $Th(b) \leq (p/c)\, Th(a)$.*

After having illustrated the factors that are involved in calculating the throughput of an actor, we now show that the only case that a channel is not self-timed bounded, is when the production of tokens into one channel is larger than the consumption of tokens out of that channel.

Lemma 29 *SDFG (A, C, E) is self-timed bounded iff $Th(b) \geq (p/c)\, Th(a)$ for every channel $(a, b, p, c) \in C$.*

The next proposition gives necessary and sufficient conditions for self-timed boundedness of a live strongly connected SDFG.

Proposition 30 *A live and strongly connected SDFG G is self-timed bounded iff it is consistent.*

Lemmas 31 and 32 and Proposition 33 prove some useful properties about the relation between the throughput of various actors. Lemma 31, which follows immediately from Corollary 28 and Lemma 29, shows the relation between producer and consumer actors of an arbitrary self-timed bounded channel. Lemma 32 shows the relation between the actor throughputs for any two actors in an SCC of an SDFG. Proposition 33 gives the relation between the throughput of two arbitrary actors in consistent self-timed bounded or strongly connected SDFGs.

Lemma 31 *If a channel (a, b, p, c) connecting a and b is self-timed bounded then $Th(b) = (p/c)\, Th(a)$.*

Lemma 32 *If a and b are two actors of an SCC of a consistent SDFG with repetition vector γ, then $Th(a)/\gamma(a) = Th(b)/\gamma(b)$.*

Proposition 33 *If a and b are two actors of a consistent self-timed bounded or strongly connected SDFG G with repetition vector γ then $Th(a)/\gamma(a) = Th(b)/\gamma(b)$.*

This proposition shows that for consistent self-timed bounded or strongly connected SDFGs the throughput as defined in Definition 14 can be calculated via an arbitrary actor without explicitly computing the minimum.

5.2 Reduction to an HSDFG

In this section, we propose a method for reducing a consistent SDFG G to an HSDFG G_H which preserves (non-)liveness and self-timed (un)boundedness of G. In G_H, every actor has a self-loop channel with one initial token, rates of all channels are one (i.e., it is an HSDFG), and, ignoring self-loops, it is acyclic. Because of these simple properties, we use the reduced graph for verifying the liveness and self-timed boundedness of the original SDFG. The reduction also preserves throughput which means our algorithm also provides the throughput of the original SDFG G.

The reduction uses the notion of local throughput of an SCC of an SDFG, and it is illustrated in Figure 3 which provides the reduced graph for the running example.

Definition 34 *[Local Throughput of an SCC] The* local throughput $LTh(\kappa)$ *of an SCC* $\kappa = (A_\kappa, C_\kappa, E_\kappa)$ *in a consistent SDFG* $G = (A, C, E)$ *with repetition vector* γ *is defined as the actor throughput of an arbitrary actor* $a \in A_\kappa$ *when all input channels from* $A \backslash A_\kappa$ *to* A_κ *are removed, divided by* $\gamma(a)$.

Lemma 32 implies that this definition is sound.

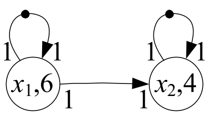

Figure 3. The reduced HSDFG for G_{ex}.

Definition 35 *[Reduced Graph] Let a consistent SDFG* $G = (A, C, E)$ *contain* n *maximal SCCs* $\kappa_1 = (A_{\kappa_1}, C_{\kappa_1}, E_{\kappa_1}), \ldots, \kappa_n = (A_{\kappa_n}, C_{\kappa_n}, E_{\kappa_n})$. *Suppose* γ *is the repetition vector of* G. *We define the reduced SDFG* $G_H = (A_H, C_H, E_H)$ *as follows:* $A_H = \{x_i | 1 \leq i \leq n\}$ *(which means one actor for each maximal SCC in G);* C_H *contains a channel* $(x_i, x_j, p\gamma(a), c\gamma(b))$ *for every channel* $(a, b, p, c) \in C$ *where* $a \in A_{\kappa_i}$, $b \in A_{\kappa_j}$, $i \neq j$; C_H *also contains self-loop channels* $(x_i, x_i, 1, 1)$ *for every actor; the execution time* $E_H(x_i)$ *equals* $1/LTh(\kappa_i)$ *if* κ_i *does not deadlock and* ∞ *if it does. According to the balance equations we know that for each channel in the original graph* (a, b, p, c), $p\gamma(a) = c\gamma(b)$. *Thus, the production and consumption rates for every channel in* C_H *are equal. Therefore, we can simplify the reduced* G *by setting all rates of all channels in* C_H *to one. Consequently, we obtain an HSDFG as the result. Finally, every self-loop channel in* G_H *contains one initial token, and all the other channels are empty.*

Since the HSDFG resulting from the reduction is acyclic when ignoring self-loops, the preservation of throughput, (non-)liveness and self-timed (un-)boundedness that we are aiming at, is independent of the number of initial tokens on the non-self-loop channels. Hence, we choose to leave those channels empty.

Consider the reduced graph shown in Figure 3. The original graph G_{ex} has two maximal strongly connected components, containing actor a, and actors b and c, respectively. These SCCs are reduced to actors x_1 and x_2. Since actor a has throughput $\frac{1}{2}$ and repetition-vector entry 3, the execution time of x_1 is set to 6, illustrating that 3 firings of a take 6 time units. Considering the other SCC in isolation, it can be verified that one period of this SCC containing 3 firings of b and 2 of c consists of 4 time units. Given the repetition vector of G_{ex} and Definition 34, this gives a local throughput of $\frac{1}{4}$ and an execution time of 4 for x_2.

The following proposition shows the relation between the throughput of actors in a maximal SCC of an SDFG and the throughput of the actor corresponding to that SCC in the reduced SDFG.

Proposition 36 *Let* G_H *be the reduced SDFG of a consistent timed SDFG* G *with repetition vector* γ. *If a maximal SCC* $\kappa = (A_\kappa, C_\kappa, E_\kappa)$ *in* G *is replaced by actor* x *in* G_H, *then for any* $a \in A_\kappa$, $Th(a) = \gamma(a)Th(x)$.

It is easy to verify that Proposition 36 holds for the running example. Consider for instance actor x_2 of the reduced graph. Its throughput in the reduced graph is fully determined by the throughput of x_1 and becomes therefore $\frac{1}{6}$. Proposition 36 states that $Th(b) = 3(\frac{1}{6}) = \frac{1}{2}$ and $Th(c) = 2(\frac{1}{6}) = \frac{1}{3}$, which corresponds to the throughput values for b and c computed at the end of Section 2.4.

The next corollary follows from the definition of throughput, the observation that all repetition-vector entries of an HSDFG are always one, and Propositions 33 and 36.

Corollary 37 *The throughput of a consistent self-timed bounded SDFG is equal to the throughput of its reduced graph.*

The reduction also preserves self-timed (un-)boundedness.

Theorem 38 *A consistent timed SDFG is self-timed bounded iff its reduced graph is self-timed bounded.*

Proposition 36 implies that non-zero throughput (i.e., (non-)liveness) is preserved.

Corollary 39 *A consistent timed SDFG is live iff its reduced graph is live.*

5.3 Verifying Self-timed Boundedness

This section introduces an algorithm that determines whether an SDFG is live and self-timed bounded. The following theorem follows from the results obtained so far.

Theorem 40 *A timed SDFG* G *is live and self-timed bounded iff* $isLive\&SelftimedBounded(G)$ *returns "yes".*

Algorithm *isLive&SelftimedBounded*(G=(A, C, E))
Input: A connected timed SDFG G
Output: "yes, $Th(G)$" if self-timed bounded and live, "no" otherwise

1. **if** not *isLive&Bounded*(G)
2. **then return** “no”
3. $G_H = (A_H, C_H, E_H) \leftarrow$reduce($G$)
4. $AL[1..|A_H|] \leftarrow$topologicalSort(G_H)
5. **if** $|A_H| = 1$
6. **then return** “yes, $\frac{1}{E_H(AL[1])}$”
7. **for** $i \leftarrow 1$ **to** $|A_H|$
8. **do** $AL[i].Th \leftarrow \frac{1}{E_H(AL[i])}$
9. **if** $\mathrm{Pred}(AL[i]) = \{AL[i]\}$ **and** $AL[i].Th = \infty$
10. **then return** “no”
11. $maxPTh \leftarrow 0$
12. **for** each $j \in \mathrm{Pred}(AL[i]) \backslash \{AL[i]\}$
13. **do** $AL[i].Th \leftarrow \min(AL[i].Th, AL[j].Th)$
14. $maxPTh \leftarrow \max(maxPTh, AL[j].Th)$
15. **if** $maxPTh > AL[i].Th$
16. **then return** “no”
17. **return** “yes, $AL[1].Th$”

The algorithm works in two steps. The first step checks the liveness and boundedness (as defined by Definition 17) of the graph by calling algorithm *isLive&Bounded* (lines 1 and 2). If the graph is not live and bounded, it cannot be live and self-timed bounded. The second step concerns determining whether the reduced HSDFG is self-timed bounded (lines 3 to 17).

If *isLive&Bounded* returns “yes”, we know that the SDFG is consistent. Then, line 3 of the algorithm reduces the SDFG according to Definition 35 and stores the result in G_H. Note that the reduction requires throughput calculations for all SCCs. For efficiency reasons, these throughput calculations can be delayed till the algorithm really needs this information. Calculations may then be avoided if the algorithm returns “no” early. We have not made this explicit in the algorithm. Since G is at this point known to be live and consistent, by Corollary 39, also G_H is live. It remains to determine self-timed (un-)boundedness.

Ignoring self-loops, G_H is acyclic. Line 4 topologically sorts the actors of G_H, and stores them in array AL, so that the predecessors of an actor $AL[i]$ are only among the $AL[j]$ for $j \leq i$. If G_H contains only one actor, then G is strongly connected, and hence, by Proposition 30, self-timed bounded, and the algorithm terminates. Based on Corollary 37, it returns the local throughput of the only actor of G_H as the throughput of G. Note that every actor in a reduced graph has a self-loop channel with one token on it, so this value is equal to $1/E_H(AL[1])$. Also note that $E_H(AL[1])$, and $E_H(AL[i])$ in general, may be 0. In this case, we assume that $1/E_H(AL[i])$ is equal to ∞.

Each iteration of the loop of lines 7 to 16 starts by calculating the local throughput of each actor $AL[i]$, $1 \leq i \leq |A_H|$, storing the result in $AL[i].Th$. In case of detecting a source actor (an actor without any input channel except its self-loop channel) with an infinite throughput, the algorithm returns “no”, because this implies that its output channels are unbounded. The loop continues by setting $maxPTh$ to zero. This variable is a temporary variable for storing the maximum throughput of the predecessors of actor $AL[i]$ in iteration i. In the loop of lines 12 to 14, the minimum between the local throughput of actor $AL[i]$ and the minimum throughput of its predecessors is assigned to $AL[i].Th$. This value, according to Lemma 27, is the throughput of the actor $AL[i]$. Note that since the actors are topologically sorted in AL, the throughput of all predecessors has already been calculated. The maximum throughput of the predecessors of actor $AL[i]$ is assigned to $maxPTh$.

The test of line 15 checks whether the maximum throughput of predecessors of actor $AL[i]$ (excluding $AL[i]$) is greater than the throughput of actor $AL[i]$ itself. In case it is, according to Lemma 29 at least one channel connecting a predecessor of actor $AL[i]$ to $AL[i]$ is unbounded.

If the algorithm reaches line 17, then no unbounded channel has been detected, and the graph is live and self-timed bounded. According to Corollary 37 and the fact that the reduced SDFG is an HSDFG with all repetition-vector entries one, the value of $AL[i].Th$ for all actors $AL[i] \in A_H$ is equal to the throughput of G. The algorithm returns $AL[1].Th$. The emphasis of algorithm *isLive&SelftimedBounded* is on verifying liveness and self-timed boundedness of an SDFG, so it returns as soon as it detects that the graph is not live or not self-timed bounded. It can be easily adapted to compute the throughput for SDFGs which are not self-timed bounded as well.

6 Related Work

There are interesting similarities between SDFGs and Petri nets. In particular, there is a straightforward translation from SDFGs to a subclass of Petri nets, called weighted Marked Graphs and vice versa, where actors are transitions, and channels are places. Marked Graphs, also called T-Graphs are known to be the subclass of Petri nets that is most amenable to rigorous analysis. Thus, it makes sense to compare the results obtained in this paper with the corresponding results in the literature concerning Petri nets. We studied liveness in combination with three different definitions of boundedness (Definitions 17, 18 and 19) for (timed) SDFGs.

We do not know of any related results for boundedness as defined by Definition 17. The only result we know for this type of boundedness is in [16] which only introduces it without providing necessary and sufficient conditions, as we do.

For strict boundedness in the sense of Definition 18, the problem has been studied from different viewpoints in the Petri-net literature (see for an overview [7, 15]). In particular, [20] gives necessary and sufficient conditions for strict boundedness of live weighted Marked Graphs (our Theorem 25). Strict boundedness is also the only kind of boundedness which has been investigated formally in the literature on SDFGs themselves; Karp and Miller in their seminal paper [11] introduced computation graphs, which are slightly more general than SDFGs. They proved necessary and sufficient conditions for liveness and strict boundedness in their model. Their results as well as those in [20] correspond to those presented in this paper.

Our third definition of boundedness, self-timed boundedness (see Definition 19) is defined on timed SDFGs. Therefore, we need to compare it with time-enabled Petri nets. Petri nets have been extended with quantitative time in different ways, by adding timing information to places, transitions and/or tokens (see [4] for a survey). The timed Petri net model that comes closest to timed SDFGs is the “time Petri net” model originally defined by [14]. This extension of Petri nets associates a duration (delay) and a deadline to transitions. We are not aware of any study of the self-timed boundedness problem for the subclass of time Marked Graphs. In [18], the liveness and strict boundedness problem for time Petri nets is studied but only some sufficient conditions are given. These conditions guarantee that once a time Petri net satisfies certain syntactic constraints, it is live and strictly bounded if the underlying untimed Petri net is live and strictly bounded. Unfortunately, the results of [18] cannot be applied in our setting since the syntactic constraints require the absence of either duration or deadline both of which are necessary for translation of timed SDFGs to time Petri nets. [10] proves a general undecidability result for strict boundedness of time Petri net of [14]. However, in [2], two sufficient conditions are given for strict boundedness of time Petri nets. We are not aware of any result about self-timed boundedness as defined in Definition 19. To the best of our knowledge, both the concept and the derived results are novel.

7 Conclusions

We have studied the liveness and boundedness of Synchronous Data Flow Graphs, which are also known as weighted Marked Graphs in the Petri-net literature. Liveness and boundedness is a prerequisite of any meaningful SDFG model of a streaming multi-media application. Two known notions of boundedness, namely boundedness and strict boundedness, have been studied rigorously, and in particular necessary and sufficient conditions for liveness in combination with these two types of boundedness have been given. For strict boundedness, these conditions were already known from the Petri-net literature. Furthermore, a new notion, self-timed boundedness, was introduced. Self-timed boundedness checks whether self-timed execution of an SDFG is bounded. A self-timed execution yields the maximum throughput for an SDFG. Necessary and sufficient conditions for self-timed boundedness and liveness have been proven. An algorithm for checking these conditions was presented. Besides, existing throughput analysis techniques, which are only valid for strongly connected graphs, are extended to arbitrary consistent SDFGs.

References

[1] F. Baccelli, G. Cohen, G. Olsder, and J.-P. Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. Wiley, 1992.

[2] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[3] S. Bhattacharyya, P. Murthy, and E. Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal on VLSI Signal Process. Syst.*, 21(2):151–166, 1999.

[4] F. D. Bowden. A brief survey and synthesis of the roles of time in Petri nets. *Mathematical and Computer Modelling*, 31(10):55–68, 2000.

[5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[6] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. on Design Automation of Electronic Systems*, 9(4):385–418, 2004.

[7] J. Esparza. Decidability and complexity of Petri net problems - an introduction. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.

[8] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi. Throughput analysis of synchronous data flow graphs. In *ACSD, Proc.*, pages 25–34. IEEE, 2006.

[9] A. H. Ghamarian, M. C. W. Geilen, T. Basten, B. Theelen, M. M. R., and S. Stuijk. Liveness and boundedness of synchronous data flow graphs. Tech. report ESR-2006-04, TU Eindhoven, http://www.es.ele.tue.nl/esreports/, 2006.

[10] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4(3):277–299, 1977.

[11] 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.

[12] E. Lee. *A coupled hardware and software architecture for programmable digiral signal processors*. PhD thesis, University of California, Berkeley, 1986.

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

[14] P. M. Merlin. *A Study of Recoverability of Processes*. PhD thesis, Department of Information and Computer Science, University of California at Irvine, 1975.

[15] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[16] T. M. Parks. *Bounded Scheduling for Process Networks*. PhD thesis, 1995.

[17] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES, Proc.*, pages 63–72. ACM, 2003.

[18] L. Popova-Zeugmann. On liveness and boundedness in time Petri nets. In *Proceedings of the Workshop on Concurrency, Specification and Programming (CS&P'95)*, pages 136–145, 1995.

[19] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc, New York, NY, USA, 2000.

[20] E. Teruel, P. Chrzastowski, J. M. Colom, and M. Silva. On weighted T-systems. In Jensen, K., editor, *13th Internationald Conference on Application and Theory of Petri Nets 1992, Sheffield, UK*, volume 616 of Lecture Notes in Computer Science, pages 348–367. Springer-Verlag, 1992.