2012 38th Euromicro Conference on Software Engineering and Advanced Applications

Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs

Robert de Groote, Jan Kuper, Hajo Broersma, Gerard J.M. Smit
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
Email: {e.degroote, j.kuper, h.j.broersma, g.j.m.smit}@utwente.nl

***Abstract*—In this paper we present a novel approach to throughput analysis of synchronous dataflow (SDF) graphs. Our approach is based on describing the evolution of actor firing times as a linear time-invariant system in max-plus algebra. Experimental results indicate that our approach is faster than state-of-the-art approaches to throughput analysis of SDF graphs. The efficiency of our approach is due to the exploitation of the regular structure of the max-plus system's graphical representation, the properties of which we thoroughly prove.**

***Index Terms*—dataflow; streaming applications; timing analysis; max-plus algebra**

I. Introduction

Synchronous dataflow (SDF) graphs [1] are well-known models of computation that are widely used to model real-time embedded streaming applications. Timing analysis of SDF graphs aims at finding performance characteristics such as throughput and latency, which is crucial information when exploring the design-space of real-time critical systems.

There are two main approaches to the timing analysis of SDF graphs. The first approach is based on the transformation of an SDF graph into an equivalent homogeneous SDF (HSDF) graph, which is then analysed for its critical cycle. A disadvantage of this approach is that the HSDF graph may become quite large: in the worst case, its size is exponential in the size of the corresponding SDF graph. The second, state-of-the-art approach to timing analysis of SDF graphs is by exploring the state-space of a simulated self-timed execution until a periodic phase is found. Such a simulation-based method avoids the transformation from SDF into HSDF.

In this paper, we present an alternative, analytical approach to timing analysis of SDF graphs. Our approach consists of a novel way of constructing a max-plus algebraic description of the evolution of actor firing times in a self-timed execution of an SDF graph. As a result, we obtain HSDF-like graphs that contain significantly fewer edges than the HSDF graph obtained by the commonly followed transformation from SDF into HSDF. Furthermore, the graphs obtained by our transformation may be efficiently analysed for its maximum cycle ratio. This is due to the regular structure of these graphs, the properties of which are formally proven.

The main contribution of our work is a sound and new basis for the formal analysis of SDF graphs using max-plus algebra, which allows for an efficient method to calculate the throughput of an SDF graph. We confirm the efficiency of our method by comparing it with the state-of-the art simulation-based approach on testsets used in an earlier study [2].

The remainder of this paper is outlined as follows: in section III, we give a brief introduction to SDF graphs, equivalent HSDF graphs, max-plus algebra and the graphical representation of max-plus systems. In sections IV - V we describe how a linear, time-invariant max-plus system may be derived from an SDF graph and graphically represented. Section VI formally proves properties of the structure of these linear max-plus systems and Section VII describes the experimental comparison between our approach and the state-of-the-art simulation-based approach to throughput analysis. Finally, Section VIII concludes the paper and gives directions for future work.

II. Related work

In timing analysis of SDF graphs, the transformation of the graph into an equivalent HSDF graph is a common step that is described by various authors, e.g. [1], [3] or [4]. In these papers, the potentially huge size of the HSDF graph is often given as a main reason to resort to simulation-based methods [2]. In fact, in [2] a comparison between a simulation-based approach in which the state-space of a self-timed execution of an SDF graph is explored and methods based on analysing the equivalent HSDF graph has concluded that simulation is a few orders of magnitude faster. Our approach is based on (smaller) subgraphs of HSDF graphs, which leads to an analysis that is found to be a few orders of magnitude faster than simulation on the same benchmark as used in [2].

The potentially large size of an SDF graph's equivalent HSDF graphs has been recognised as a problem in [3], where the authors describe an approach to reduce the size of an SDF graph's equivalent HSDF graph. The main drawback of their approach is that they require the full HSDF graph to be constructed first, which is avoided in our approach.

In [5] it is described how reduced HSDF graphs are obtained from SDF graphs by representing each token in the SDF graph by a single linear max-plus expression. Although the size of the reduced HSDF graph may be small for graphs with only very few tokens, constructing the system involves simulation of the SDF graph and the symbolic manipulation of max-plus expressions, which is complicated and requires the administration of all tokens that are produced and consumed during the execution of

978-0-7695-4790-9/12 $26.00 © 2012 IEEE
DOI 10.1109/SEAA.2012.20

the SDF graph. Our approach is simpler and does not depend on the number of tokens in the graph.

III. Preliminaries

In this section we will discuss some specification formalisms and their relationships, leading to a method for timing analysis based on constraint graphs rather than on HSDF graphs.

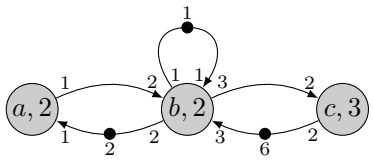
A. SDF graphs

Synchronous dataflow (SDF) graphs are often used to model streaming applications. We will assume that the reader is familiar with standard SDF terminology (such as actor, channel, firing, production/consumption rates, etc), we only define a few SDF notions that are relevant for this paper.

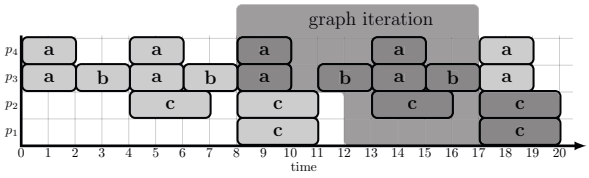
An SDF graph is *consistent* if a shortest non-empty sequence of actor firings exists, which as a whole will effectively leave the token distribution unchanged. Such a sequence of firings is called a *graph iteration*. The *repetition vector* q of a consistent SDF graph associates with each actor a the number of times q_a that actor a fires within a single graph iteration.

The time between the start and the completion of a single firing of an actor a is called the *execution time* of actor a and denoted by τ_a. The *throughput* of an SDF graph is the average number of graph iterations that are executed per unit of time, measured over a sufficientlly large amount of time. The maximum throughput is attained by a *self-timed* execution, which means that each actor fires as soon as it is enabled. As in [1] we will assume that, whenever enough tokens are available on the incoming channels of an actor, that actor may fire several times simultaneously. That is to say, we assume that enough resources are available to execute several firings of an actor in parallel.

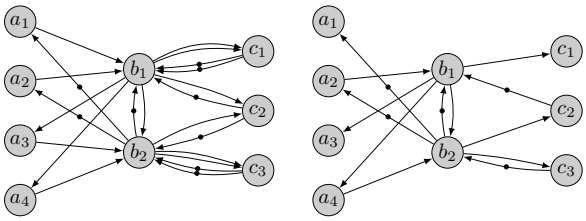
An example SDF graph is depicted in figure 1(a). The graph contains 2 initial tokens on channel ba, 1 on channel bb, and 6 on channel cb. Each actor is annotated with its execution time: 2 time units for actors a and b, 3 time units for actor c. The graph is consistent: a graph iteration consists of 4 firings of actor a, 2 firings of actor b and 3 firings of actor c. Hence, the repetition vector of the graph is $\langle 4, 2, 3 \rangle$.



(a) SDF Graph



(b) Self-timed schedule



(c) HSDF Graph

(d) Linear Constraint Graph

$$t_{a_1}(k) = t_{b_2}(k-1) \otimes 2 \qquad t_{c_1}(k) = t_{b_1}(k) \otimes 3$$
$$t_{a_2}(k) = t_{b_2}(k-1) \otimes 2 \qquad t_{c_2}(k) = t_{b_2}(k) \otimes 3$$
$$t_{a_3}(k) = t_{b_1}(k) \otimes 2 \qquad t_{c_3}(k) = t_{b_2}(k) \otimes 3$$
$$t_{a_4}(k) = t_{b_1}(k) \otimes 2$$
$$t_{b_1}(k) = (t_{a_2}(k) \oplus t_{b_2}(k-1) \oplus t_{c_2}(k-1)) \otimes 2$$
$$t_{b_2}(k) = (t_{a_4}(k) \oplus t_{b_1}(k) \oplus t_{c_3}(k-1)) \otimes 2$$

(e) Max-Plus equations

Fig. 1. Example SDF graph with several derived representations.

B. Self-timed schedule

The schedule of a self-timed execution of the example graph is shown in Figure 1(b). It starts with two (parallel) firings of actor a to consume the two initial tokens from channel ba. After that, one firing of actor b takes place, then two firings of a and one of c in parallel, etc. The first iteration finishes at time 11. The second iteration already starts at time 8 with two firings of actor a again, and finishes at time 20 after the two firings of actor c have completed. Of the third iteration only the initial part consisting of two firings of actor a that start at time 17 are shown. As shown in this example, iterations may overlap in time: two firings of actor a of the next iteration occur in parallel with two firings of actor c in the previous iteration.

Because of the initial token distribution, the first iteration takes 11 time units, whereas later iterations take 12 time units. However, because of the fact that iterations overlap in time, every 9 time units a single iteration is completed. Thus, the throughput achieved in a self-timed execution of the SDF graph is $\frac{1}{9}$.

Note that the borderline between iterations depends on the initial token distribution.

C. HSDF graphs

The standard approach to analyse the timing behaviour of an SDF graph is by transforming the SDF graph into an equivalent *homogeneous* SDF graph (HSDF graph), i.e., into an SDF graph in which all production and consumption rates are *one*, using the well-known procedures found in, e.g., [4] or [6]. Given an SDF graph, the equivalent HSDF graph is constructed by creating a vertex for each *firing* of an actor in the original SDF graph, and an edge for each produced/consumed *token* in the

original SDF graph. Thus (see Figure 1(c)) the first two firings a_1 and a_2 of actor a each produce one token onto channel ab, which both are consumed by the first firing b_1 of actor b. Likewise, firing b_2 produces three tokens onto channel bc, where one token of these three is consumed by firing c_2 and the other two by firing c_3.

Note that in general the equivalent HSDF graph can be a *multigraph*, i.e., multiple edges may exist between a pair of vertices, since several tokens may be produced/consumed by a single firing of an actor. Note also that the order between a_1 and a_2 is in fact arbitrary, since both firings occur simultaneaously. This arbitrary ordering does not affect the timing anlysis of an HSDF graph.

The tokens produced onto channel ba by firing b_2 are in fact consumed by firings a_5 and a_6. However, extending the HSDF graph with an actor for *each* firing would make the graph infinite. Therefore, in the HSDF graph actors a_1 and a_2 also represent firings a_5 and a_6, respectively. The dots on the edges b_2a_1 and b_2a_2 indicate that these tokens are consumed by actor a *in the next iteration* of the SDF graph. In general, a dot on an edge xy means that the corresponding token is produced in one iteration and consumed in the next iteration (see also Figure 1(b)). We call a dot in an HSDF graph a *delay*, and remark that an edge may contain zero or more delays to indicate the number of iterations later that a token is consumed.

We remark that the complexity of the equivalent HSDF graph is increased (even exponentially) in comparison with the underlying SDF graph. This increase in complexity is the primary reason that HSDF graphs are not used in most analysis methods for SDF graphs, leading to simulation based methods instead (cf. e.g. [2]). Below we will describe an alternative perspective on HSDF graphs and derive from that an analysis method which avoids this increase in complexity.

D. Linear Constraint Graphs

In the above SDF and HSDF graphs were interpreted as "models of computation", i.e., actors perform computations, and tokens denote data elements that are communicated between actors along channels in specific quantities determined by production and consumption rates (which are always one in an HSDF graph). In this section we will introduce a different perspective, which underlies the remainder of this paper: a channel in an SDF graph will be interpreted as a *constraint* for an actor to fire, thus expressing data *dependency* rather than data *communication*.

In [4] it already was observed that "parallel" edges (tokens) in an HSDF graph connecting the same pair of nodes (firings) can be combined into one edge since that does not change the dependency between the two involved firings. The resulting graph is called a *constraint graph* in [4].

Here we extend this interpretation by considering only the *last incoming token* which is required by an actor to fire, where "last" does not refer to the production time of that token, but to the chosen order for (possibly) simultaneous firings. The consequence is that several edges become redundant; we will call the remaining edges the *precedence constraints*. As a result, the indegree of each vertex is equal to the indegree of the corresponding actor in the SDF graph. Figure 1(d) gives the *linear constraint graph* (LCG) which is derived from the HSDF graph in Figure 1(c) (we will motivate the term "linear" in Section III-E).

For example, firing c_2 requires a token from firing b_1 and from firing b_2. Since the second one is the last of these two, the timing of firing c_2 only depends on firing b_2. Likewise, firing a_2 only depends on firing b_2 in the previous iteration.

Note that linear constraint graphs contain (much) fewer edges than HSDF graphs, which is illustrated in Figure 1. Fewer edges also means fewer cycles, which severely impacts the efficiency of algorithms needed for analysis of these graphs (e.g., maximum cycle ratio algorithms, see [7]).

E. Max-Plus Algebra

Timed synchronous systems may be mathematically described using *max-plus algebra* [8]–[10]. In max-plus algebra, times at which events (firings) take place are related to times at which preceding events take place by means of the operators $\oplus$ (for the *maximum* of two numbers) and $\otimes$ (for ordinary *addition*), expressing synchronisation and duration, respectively. We remark that $\otimes$ is distributive over $\oplus$.

Writing $t_x(k)$ for the moment in time that the k^{th} occurrence of firing x completes, we now can straightforwardly express the constraint graph from Figure 1(d) by means of the max-plus equations in Figure 1(e). For example, the equation

$$t_{b_1}(k) = (t_{a_2}(k) \oplus t_{b_2}(k-1) \oplus t_{c_2}(k-1)) \otimes 2 \qquad (1)$$

expresses that the completion time of the k^{th} occurrence of firing b_1 is 2 time units (the execution time of actor b) after the latest completion of firings a_2 (in the same iteration k), b_2 and c_2 (both in the previous iteration $k-1$). From the self-timed schedule in Figure 1(b) it follows that in this case c_2 is the latest firing on which b_1 depends.

Without going into details we remark that in general the behaviour of a timed synchronous system can be expressed as a linear max plus system. The max plus equations in Figure 1(e) form a linear max plus system, and the graph in Figure 1(d) is its graphical representation. This motivates our usage of the term "linear" in "linear constraint graph".

Many efficient algorithms are available to analyse such a linear max-plus description [10], [11] or its graphical representation [7]. The following section describes how the firing times of actors in an SDF graph can be described by a linear max-plus system.

IV. Linear Max-Plus descriptions of SDF graphs

In this section we will use max-plus algebra to describe the evolution of *actor firing times* during the self-timed execution of an SDF graph. The moments in time that we will use in max-plus expressions are the *completion times* of firings. As explained before, these events are related through precedence constraints, which are imposed by the channels in an SDF graph: the times at which an actor may fire depends on the times

at which sufficient tokens become available on its incoming channels.

In this section we define, for each SDF channel ab, functions $\tilde{\pi}_{ab}$ and δ_{ab} that jointly map each firing b_j to a corresponding firing a_i such that edge $a_i b_j$ is the precedence constraint for firing b_j. In order to define these functions, we will write m_{ab} for the production rate of actor a on channel ab, n_{ab} for the consumption rate of actor b on channel ab and d_{ab} for the initial number of tokens on channel ab.

Per SDF channel ab, the time at which actor b may start its j^{th} firing is constrained by the time at which the *last required token* for that firing is produced onto channel ab by actor a. The completion of the j^{th} firing of actor b requires the production of *at least* $N = j \cdot n_{ab} - d_{ab}$ tokens by actor a.

To find the firing of actor a that must have completed such that the j^{th} firing of actor b may start, we must thus divide N by m_{ab} and round the result towards the nearest higher integer. Let i be this firing of a. We call i the *predecessor* of j on channel ab, denoted $i = \pi_{ab}(j)$, with $\pi_{ab}(j)$ defined as:

$$\pi_{ab}(j) = \left\lceil \frac{j \cdot n_{ab} - d_{ab}}{m_{ab}} \right\rceil. \tag{2}$$

We can use this predecessor function to relate the completion times of an actor's firings to the times at which firings of other actors complete. Let $t_b(j)$ denote the time at which actor b *completes* its j^{th} firing and $\mathcal{E}$ the set of channels in the SDF graph. The following max-plus expression then captures the precedence constraint for firings of actor b, due to the actor's incoming channels:

$$t_b(j) = \bigoplus_{ab \in \mathcal{E}} t_a\left(\pi_{ab}(j)\right) \otimes \tau_b. \tag{3}$$

These constraints may not generally be expressed as a *linear time-invariant max-plus system* [8]. In parlance of system theory, the system expressed by (3) is a so-called *linear time-variant* system, since $\pi_{ab}(j)$ may not generally be replaced by $j - k$ (for some $k \in \mathbb{N}$).

For *consistent* SDF graphs however, equation (3) is *periodically* time-variant and may be expressed by a linear time-invariant system by a change of variables: We let $b_j(k)$ denote the j^{th} firing of actor b in the $(k+1)^{\text{th}}$ iteration, thus $t_{b_j}(k) = t_b(j + k \cdot q_b)$. Note that $j \in \{1, \ldots, q_b\}$. By changing variable b into b_j, equation (3) may be rewritten as follows:

$$t_{b_j}(k) = \bigoplus_{ab \in \mathcal{E}} t_a\left(\left\lceil \frac{j \cdot n_{ab} + k \cdot q_b \cdot n_{ab} - d_{ab}}{m_{ab}} \right\rceil\right) \otimes \tau_b. \tag{4}$$

Since in a single iteration of a consistent SDF graph, with repetition vector q, the number of tokens produced onto each channel is equal to the number of tokens consumed from that channel, we have $q_b \cdot n_{ab} = q_a \cdot m_{ab}$. We use this to simplify (4) into:

$$t_{b_j}(k) = \bigoplus_{ab \in \mathcal{E}} t_a\left(\pi_{ab}(j) + k \cdot q_a\right) \otimes \tau_b. \tag{5}$$

To complete the change of variables, we must rewrite $t_a(\pi_{ab}(j) + k \cdot q_a)$ as $t_a(i + m \cdot q_a)$, which we then write as $t_{a_i}(m)$, with $i \in \{1, \ldots, q_a\}$. Terms i and m are obtained by applying basic modular arithmetic. Note that since we number an actor's firings starting with *one*, decrements and increments by one are required. Let $\tilde{\pi}_{ab}(j)$ be the *firing index* of $\pi_{ab}(j)$ within a graph iteration, defined as follows:

$$\tilde{\pi}_{ab}(j) = (\pi_{ab}(j) - 1) \bmod q_a + 1, \tag{6}$$

and $(\delta_{ab} + 1)$ the *iteration index*: the index of the iteration in which the firing takes place, given by:

$$\delta_{ab}(j) = \left\lfloor \frac{\pi_{ab}(j) - 1}{q_a} \right\rfloor. \tag{7}$$

The following expression then completes the change of variables and gives a linear time-invariant system:

$$t_{b_j}(k) = \bigoplus_{ab \in \mathcal{E}} t_{a_{\tilde{\pi}_{ab}(j)}}\left(k + \delta_{ab}(j)\right) \otimes \tau_b. \tag{8}$$

As an example, consider the SDF graph depicted in Figure 1(a). The time-variant precedence constraint for actor b is:

$$t_b(j) = \left(t_b(j-1) \oplus t_a(2j) \oplus t_c\left(\left\lceil \frac{3j-6}{2} \right\rceil\right)\right) \otimes 2. \tag{9}$$

Because actor b fires two times in a single iteration (its entry q_b in the repetition vector equals 2), we replace b by the two variables b_1 and b_2. We then have $t_{b_1}(k) = t_b(1 + k \cdot q_b)$ and $t_{b_2}(k) = t_b(2 + k \cdot q_b)$. In other words, t_{b_1} and t_{b_2} are calculated from $t_b(j)$ by substituting j with respectively $1+2k$ and $2+2k$ in (9). The resulting linear, time-invariant equations for actor b and the other actors in the SDF graph are shown in Figure 1(e).

V. Linear Constraint Graph Generation

As explained in Section III the linear time-invariant max-plus system expressed by equation (8) may be graphically represented by a linear constraint graph (LCG). In Section V-B below we present an algorithm to generate the LCG directly from a given SDF graph without first having to produce the equivalent HSDF graph.

The algorithm assumes that the SDF graph is simple, i.e., in case an SDF graph is a multigraph, we first have to remove the parallel edges that exist between the same pair of actors.

A. Reducing consistent SDF multigraphs

In an SDF multigraph, multiple channels may exist between two actors, in which case the channels are said to be *parallel*. Each of these parallel channels results in a different set of max-plus equations. However, in a consistent SDF multigraph, parallel channels may be *sorted* by the strength of the precedence constraints they imply. A set of parallel channels may then be replaced by the channel that imposes the strongest constraint.

In order to sort channels by the strength of their imposed precedence constraints, their rates first need to be equalised: Since multiplying a channel's rates and initial tokens with the same constant does not alter the channel's imposed precedence constraint (the reader may verify this using the definition of π_{ab}), we may choose suitable integers and multiply each

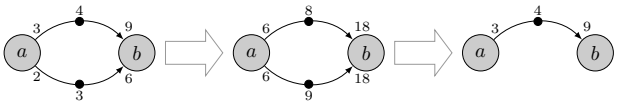


Fig. 2. An SDF multigraph may be transformed into a simple directed graph by equalising the rates of channels between two actors. Only a single channel that has the minimum number of initial tokens needs to be retained.

channel's production (or consumption) rate such that each channel has the same production (or consumption) rate. In case the SDF multigraph is consistent, each of the parallel channels will then have the same consumption (or production) rate as well (this follows directly from the fact that in a consistent graph we have $m_{ab} \cdot q_a = n_{ab} \cdot q_b$ for any channel ab).

If parallel channels have equal production rates and equal consumption rates, the strongest precedence constraints are imposed by the channel with the fewest tokens. Hence, for a pair of parallel channels, we may remove the SDF channel that, after equalising the channels' rates, has the most initial tokens (see Figure 2). Note that this is a straightforward generalisation of the transformation of an HSDF multigraph to a simple graph found in [4].

B. Algorithm to generate the LCG

In this section we present an algorithm to generate the LCG from a consistent SDF graph (Algorithm 1). Based on the repetition vector, the algorithm first creates a node for every firing, and then for every SDF channel ab functions $\tilde{\pi}_{ab}$ and δ_{ab} calculate the corresponding firing i of actor a that produces the last token needed by each firing j of actor b. Then only the edges $a_i b_j$ are added to the LCG.

Note that the LCG is generated directly, i.e., withoug going through the HSDF expansion. As an example, the LCG as shown in Figure 1(d) may be generated by applying Algorithm 1 to the SDF graph of Figure 1(a).

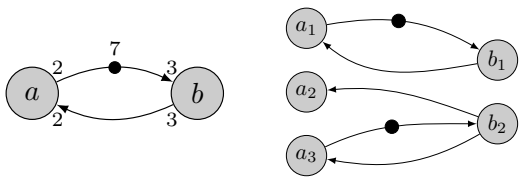
Algorithm 1 Transforms a consistent SDF graph into an LCG

```
Let 𝒢 be a simple, consistent SDF graph
Let q be the repetition vector of 𝒢
Let ℋ be an empty LCG

for each actor a in 𝒢 do
    Add vertices a_1 ... a_{q_a} to ℋ
end for

for each channel ab in 𝒢 do
    for j = 1 ... q_b do
        i ← π̃_ab(j)
        add edge a_i b_j with −δ_ab(j) delays to ℋ
    end for
end for
```

As we will demonstrate in the following section, the structure of an LCG may be exploited to allow for a much more efficient analysis, in which only a *subgraph* of the LCG is explored.



(a) SDF graph (b) Constraint graph

Fig. 3. An SDF graph and its corresponding constraint graph. The constraint graph contains two cycles, each of which has the same cycle ratio.

VI. Throughput analysis of SDF graphs

The throughput of an SDF graph is the average number of iterations that are completed per unit of time. Since the LCG of an SDF graph has exactly one vertex for each firing of a single graph iteration, the SDF graph's throughput is equal to the minimum of the average number of firings per unit of time over all vertices in the constraint graph. It is well known (see for example [7] or [12]) that this minimum average firing time is determined by the *maximum cycle ratio* of the LCG, which is the maximum of the cycle ratios of all simple cycles in the graph, where the cycle ratio λ of a cycle C is defined as:

$$\lambda(C) = \frac{\sum_{a_i b_j \in C} \tau_b}{\sum_{a_i b_j \in C} -\delta_{ab}(j)}. \tag{10}$$

A cycle that has the maximum cycle ratio is said to be a *critical cycle*. Note that a constraint graph may contain multiple critical cycles, see for example the constraint graph shown in Figure 3, which contains 2 critical cycles.

Since the Linear Constraint Graph of an SDF graph may be quite large, we shall first investigate its structure for regularity and redundancy that may be exploited. This structure becomes especially apparent when constraint graphs are depicted in the column-wise representation of Figure 4(b): we group vertices that represent firings of the same SDF actor into columns, and (vertically) order the vertices by the index of the firing they represent. The following sections describe the structural properties of Linear Constraint Graphs, starting with the simplest graph (the LCG that represents a single SDF channel), followed by more complex graphs that represent SDF paths, cycles and, finally, full SDF graphs. For the sake of readability, formal proofs for the statements that are made in these sections have been moved to the appendix.

A. Defining the structure of the Linear Constraint Graph: parallel and crossing edges

The structure of the Linear Constraint Graph that represents a single SDF channel emerges from the in-order token consumption (tokens are consumed in the same order they are produced) and the SDF graph's balance equations.

The graph's balance equations state that on SDF channel ab, tokens produced by q_a firings of actor a are consumed by q_b firings of actor b. Due to the presence of initial tokens on the channel, these q_b firings may span at most *two* consecutive graph iterations (this is the case when the number of initial tokens on the channel is not a multiple of the channel's

consumption rate). In other words, the number of delays on any two edges in the LCG of an SDF channel can not differ by more than one.

The in-order token consumption orders the number of delays on edges leaving vertices that represent consecutive firings of actor a: if a_i and a_j are two vertices in the LCG with $j > i$, then the number of delays on any edge leaving a_j can not be lower than the number of delays on any edge leaving a_i.

A direct result of these two basic rules is that for disjoint edges (two edges are disjoint if they share neither source nor sink) that represent the same SDF channel, the number of delays may be inferred simply by looking at the firing indices of the edges' sources and sinks. We introduce the following terminology to formalise the structure of a linear constraint graph of a single SDF channel:

Definition 1 (parallel and crossing edges)**.** *Let $e_1 = a_{i_1}b_{j_1}$ and $e_2 = a_{i_2}b_{j_2}$ be two edges (with $i_1 = \tilde{\pi}_{ab}(j_1)$ and $i_2 = \tilde{\pi}_{ab}(j_2)$) in the linear constraint graph that represents* SDF *channel ab. The relations* parallel *and* crossing *are defined as follows:*

- *e_1 is crossing with e_2, denoted $e_1 \nparallel e_2$, if:*

$$(i_2 > i_1 \wedge j_2 < j_1) \vee (i_2 < i_1 \wedge j_2 > j_1).$$

- *e_1 is parallel with e_2, denoted $e_1 \parallel e_2$, if:*

$$(i_1 > i_2 \wedge j_1 > j_2) \vee (i_1 < i_2 \wedge j_1 < j_2).$$

Note that our definition of *parallel* edges in an LCG should not be confused with parallel edges found in a multigraph (where they refer to multiple edges having the same source and sink vertex).

Two crossing edges can not have the same number of delays, since in that case tokens would be consumed out of order (tokens produced by a firing are consumed before tokens produced by an earlier firing are consumed). Therefore, one edge must have precisely one delay more than the other. This is formalised in the following proposition:

Proposition 1 (different delays on crossing edges)**.** *Let $a_{i_1}b_{j_1}$ and $a_{i_2}b_{j_2}$ be two* crossing *edges in the constraint graph that represents* SDF *channel ab, with delays k_1 and k_2, respectively, and with $i_2 > i_1$ (and thus $j_2 < j_1$). Then $k_2 = k_1 + 1$.*

Following a similar reasoning we may infer that two parallel edges in the constraint graph of SDF channel ab carry the same number of delays:

Proposition 2 (same delays on parallel edges)**.** *Let $a_{i_1}b_{j_1}$ and $a_{i_2}b_{j_2}$ be two* parallel *edges in the constraint graph that represents* SDF *channel ab, with delays k_1 and k_2, respectively, and with $i_2 > i_1$ (and thus $j_2 > j_1$). Then $k_2 = k_1$.*

B. Properties of paths and cycles in linear constraint graphs

The relationship between parallel (and crossing) edges and their delays can be extended to disjoint paths (two paths are disjoint if no vertex is shared between the paths) in an LCG. Instead of two actors a and b and one channel ab, we now consider the situation in which we have n actors $a_1, a_2, \ldots, a_n$, and (at least the) SDF channels a_1a_2, a_2a_3, …, $a_{n-1}a_n$. In the LCG such a sequence of channels is represented by several paths, the indices of which are now denoted as superscripts, so $(a_1^{i_1}a_2^{i_2}\ldots a_{n-1}^{i_{n-1}}a_n^{i_n})$ denotes such a path in which channel a_ka_{k+1} is represented by the edge $a_k^{i_k}a_{k+1}^{i_{k+1}}$. We refer to a path by the sequence of its vertices and denote the delay of a path P (i.e., the sum of the delays of its edges) by $|P|_d$. Paths are assumed to be *simple*, i.e., no vertex is repeated in a path. Similar to the definitions for edges in an LCG, we introduce the following terminology:

Definition 2 (parallel and crossing paths)**.** *Let $\mathcal{G}$ be the* LCG *representing a path $P = (a_1a_2\ldots a_{n-1}a_n)$ in a consistent SDF graph. Furthermore, let $P_i = (a_1^{i_1}a_2^{i_2}\ldots a_{n-1}^{i_{n-1}}a_n^{i_n})$ and $P_j = (a_1^{j_1}a_2^{j_2}\ldots a_{n-1}^{j_{n-1}}a_n^{j_n})$ be two disjoint paths in $\mathcal{G}$. Then P_i and P_j are:*

- parallel*, denoted $P_i \parallel_p P_j$, if $(j_1 > i_1 \wedge j_n > i_n) \vee (j_1 < i_1 \wedge j_n < i_n)$.*
- crossing*, denoted $P_i \nparallel_p P_j$, if $(j_1 > i_1 \wedge j_n < i_n) \vee (j_1 < i_1 \wedge j_n > i_n)$.*

Analogous to the case of disjoint edges, the relative delays on parallel and crossing paths depend only on the first and last vertices of the paths. This property can be derived from Propositions 1 and 2 in a straightforward way, using induction on the number of actors represented by the paths, and is formally stated in the following lemma.

Lemma 1 (relative delays on disjoint paths)**.** *Let $P_i = (a_1^{i_1}\ldots a_n^{i_n})$ and $P_j = (a_1^{j_1}\ldots a_n^{j_n})$ be two paths representing the same n actors and $n-1$ channels, with $i_1 > j_1$. Then:*

(1) *$|P_i|_d = |P_j|_d$ if P_i and P_j are parallel;*
(2) *$|P_i|_d = |P_j|_d + 1$ if P_i and P_j are crossing.*

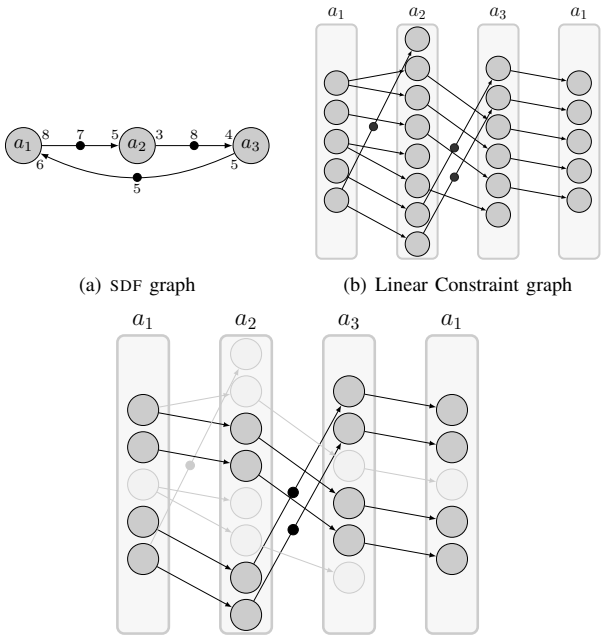
An important consequence of the above lemma is the following: If we have three pairwise disjoint paths and each crosses at least one of the other two paths, then two of these three paths must be parallel. Furthermore, if a path crosses two other, disjoint paths, these two paths must be parallel. These two implications are captured in the following corollary, which follows directly from the above lemma.

Corollary 1 (restrictions on disjoint paths)**.** *Let $P_i = (a_1^{i_1}\ldots a_n^{i_n})$, $P_j = (a_1^{j_1}\ldots a_n^{j_n})$ and $P_k = (a_1^{k_1}\ldots a_n^{k_n})$ be three* disjoint *paths, with $k_1 > j_1 > i_1$. Furthermore let P_i cross P_j. Then:*

(1) *if P_j and P_k cross, then P_i and P_k do not cross;*
(2) *if P_j and P_k are parallel, then P_i and P_k cross.*

Note that the results of the above lemma and its corollary also hold if we consider walks instead of paths (so nodes may be repeated) in the SDF graph, as long as the sequence of actors of the two walks is the same and between every pair of successive actors there is a channel represented by an edge in the walks. In particular, the result also holds for *closed walks*, i.e. walks for which the first and last actor are identical. We show below how this can help us to analyse the behaviour of an SDF graph that consists of a single (simple) directed cycle.

Let the consistent SDF graph with repetition vector q be a



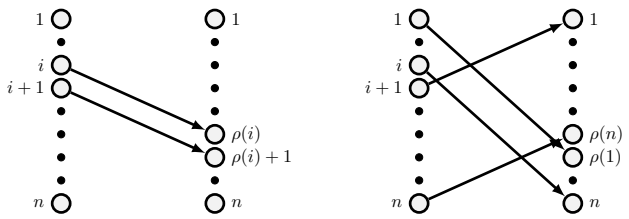
(a) SDF graph (b) Linear Constraint graph

(c) Cycle-induced subgraph

Fig. 4. An SDF graph and its corresponding LCG in a column representation, with actor a_1 duplicated. The rightmost figure depicts the cycle-induced subgraph of the LCG, which is obtained by retaining all nodes and edges that lie on a cycle. There are two cycles of length 6 in the LCG, and both cycles have a delay of one.

directed cycle consisting of actors $a_1, a_2, \ldots, a_n$ and channels $a_1a_2, a_2a_3, \ldots, a_{n-1}a_n, a_na_1$. Then the LCG has a sequence of $q_k = q_{a_k}$ nodes $a_k^1, a_k^2, \ldots, a_k^{q_k}$ for every actor a_k, and edges $a_k^i a_{k+1}^j$ (and $a_n^i a_1^j$) representing the firings, as defined before. For convenience, we repeat the sequence of q_1 nodes for actor a_1 at the end and think of the LCG as an array of $n+1$ columns, where the sequences of nodes representing the actors are ordered from left to right, and where the leftmost and rightmost sequences are identical, representing the actor $a = a_1$ (see Figure 4(b)). To distinguish these two sequences, we denote the leftmost sequence by $L = L(a)$ and the rightmost by $R = R(a)$.

Now consider the Linear Constraint Graph's *cycle-induced* subgraph. This graph is obtained by removing all edges and vertices from the LCG graph that do not lie on a cycle (see Figure 4(c)) and consists of a number of disjoint paths (since each vertex has an indegree of one), each of which starts in L and ends in R. Furthermore, if there are n paths in the subgraph, each column contains precisely n vertices. Because each path has the same length and the (relative) delay of a path is (by Lemma 1), fully determined by its start and end vertices, we choose to compactly represent the cycle-induced subgraph by a *permutation* $\rho : \{1, \ldots, n\} \to \{1, \ldots, n\}$. This permutation maps the index of a vertex in L to the index of a vertex in R, where the index of a vertex is based on the natural ordering of vertices representing the same actor (i.e., $a_k^i < a_k^j$ if $i < j$). In the remainder of this section, we shall refer to vertices (in L and R) by their index; paths in the cycle-induced subgraph then start in a node $i \in \{1, \ldots, n\}$ and terminate in $\rho(i) \in \{1, \ldots, n\}$.



(a) parallel paths (b) crossing paths

Fig. 5. Structure of the permutation ρ

Representing the cycle-induced subgraph as a permutation on a set of integers reveals a clear structure in Linear Constraint Graphs that represent SDF cycles. Consider the case where two parallel paths start in subsequent vertices, indexed i and $i+1$. Using the lemma stated above and its corollary we may derive that these paths also terminate in subsequent vertices, or $\rho(i+1) = \rho(i) + 1$ (see Figure 5(a)), which leads to the following proposition:

Proposition 3. *Let P_i and P_{i+k} be two parallel paths that start in vertices indexed i and $i+k$, respectively, with $k > 0$. Then $\rho(i+k) = \rho(i) + k$.*

For crossing paths, a similar relation in terms of ρ exists. This is illustrated in Figure 5(b) and may be understood by considering two crossing paths P_i and P_{i+1} that start in subsequent vertices i and $i+1$, respectively. We may divide the set of paths in two subsets: the first subset contains all paths starting in vertices $1, 2, \ldots, i$, and the second contains all paths starting in vertices $i+2, \ldots, n$. By Corollary 1, both subsets contain pairwise parallel paths. Furthermore, each path in one subset crosses all other paths in the other subset. As a consequence, we must have $\rho(i) = n$ and $\rho(i+1) = 1$. The following proposition formally generalises this conclusion:

Proposition 4. *Let P_i and P_{i+1} be two crossing paths that start in subsequent vertices indexed i and $i+1$, respectively. Then $\rho(i+k) = k$ for $k > 0$ and $\rho(i-k) = n-k$ for $k \geq 0$.*

We are now ready to move from paths in the LCG to cycles. A simple cycle in the LCG may be constructed by repeatedly applying the permutation ρ until the start vertex is reached again. For this, let $\rho^{k+1}(i) = \rho(\rho^k(i))$ and $\rho^1(i) = \rho(i)$. Due to the structure of the LCG, any two (simple) cycles in the graph must have the same cycle ratio. This is formally stated in the following theorem and proven (in the appendix) by exploiting the definition of ρ. Note that the theorem is not restricted to *simple* SDF cycles, but applies to any closed walk in the SDF graph.

Theorem 1 (Two cycles have the same length and delay)**.** *Let $\mathcal{G}$ be the cycle-induced subgraph of the Linear Constraint Graph corresponding to an* SDF *cycle, and let $C_i = \{i, \rho(i), \ldots, \rho^{n_i}(i) = i\}$ and $C_j = \{j, \rho(j), \ldots, \rho^{n_j}(j) = j\}$*

be two disjoint simple cycles in $\mathcal{G}$. Then $n_i = n_j$ and $|C_i|_d = |C_j|_d$.

C. Throughput analysis of arbitrary SDF graphs

Theorem 1 provides an efficient approach to throughput analysis of SDF cycles. Rather than constructing the full LCG of an SDF cycle, it suffices to pick a random vertex and follow edges in reverse direction until a cycle is found. By Theorem 1, this cycle must then be (one of) the graph's critical cycle(s).

A straightforward question is whether the same approach works for arbitrary SDF graphs: Can we choose a random vertex and restrict the search for a critical cycle to the subgraph reachable (by following edges in reverse direction) from the initial vertex? In this section we show that this is indeed the case for strongly connected SDF graphs (note that the throughput of an SDF graph that is not strongly connected may be calculated from the throughputs of its strongly connected components, as is described in [13]).

An important property in understanding why this approach works, concerns the reachability of vertices in an LCG. This is formalised in Proposition 5, which immediately follows from the fact that in a strongly connected SDF graph each actor has at least one incoming channel. Hence, in the LCG each vertex has a nonzero indegree and thus is reachable.

Proposition 5 (reachability)**.** *Let a and b be actors in a consistent and strongly connected* SDF *graph with repetition vector q. Then for each vertex b_j that represents the j^{th} firing of actor b there exists an i such that the* LCG *contains a path from a_i to b_j.*

In words, Proposition 5 states that if, by following edges in reverse, actor a is reachable from actor b, then from any vertex that represents a firing of actor b we may reach a vertex that represents a firing of actor a. We use this fact together with Theorem 1 to derive the important result that only a subgraph of the LCG needs to be explored for its critical cycle:

Theorem 2 (Subgraph analysis)**.** *Let $\mathcal{G}$ be the* LCG *that represents (consistent)* SDF *graph $\mathcal{G}_{sdf}$, and s an arbitrary vertex in $\mathcal{G}$. Furthermore, let $\mathcal{H}$ be the induced subgraph of $\mathcal{G}$ that consists of those vertices from which a path to s exists. The maximum cycle ratio of $\mathcal{H}$ is the maximum cycle ratio of $\mathcal{G}$.*

Theorem 2 implies that it is not necessary to explore the entire LCG for its critical cycle. More specifically, it does not matter whether the LCG is strongly connected or not. We may thus, in a similar way to the approach for SDF cycles proposed in the previous section, choose an arbitrary root vertex in the LCG and search the induced subgraph that consists of vertices from which the root vertex is reachable, for its critical cycle.

VII. Results

Theorem 2 yields the basis for an algorithm to analyse an SDF graph for its throughput: We start by constructing the LCG , using a random vertex as a starting point. After constructing the LCG, we use the algorithm described in [12] to find the graph's maximum cycle ratio. Note that in [2], the same algorithm was applied to equivalent HSDF graphs.

To evaluate the effectiveness of our approach to throughput analysis, we have compared the performance (in runtime) of our LCG-based method with the state-of-the-art state-space exploration approach described in [2]. To evaluate state-space exploration, we have used the publicly available SDF^3 toolkit [14], which contains several algorithms for SDF graphs. The comparison we make in this paper is similar to the comparison included in [2], where state-space exploration was compared with methods based on constructing equivalent HSDF graphs.

For the sake of comparison, we have used the same three sets of SDF graphs that were used in [2] as a benchmark to compare the performance of state-space exploration with HSDF-based approaches. Each of these testsets contains 100 graphs that are randomly generated with different parameters set to generate different classes of graphs:

Mimic DSP: This set contains 100 random graphs in which production rates, consumption rates and execution times are all small. These settings make the graphs representative for DSP applications.

Large HSDFG: This set contains graphs in which the rates are such that the equivalent HSDF graph is very large. In [2], this set was found to be particularly difficult to be processed efficiently for methods based on constructing an equivalent HSDF graph.

Long Transient: This set contains graphs in which the self-timed schedule starts with a rather long transient phase before settling in a periodic phase. These graphs thus represent the most difficult input for the state-space exploration method.

For each graph, the runtime of our method (generating the LCG and finding its maximum cycle ratio) and of state-space exploration (using the `sdf3analysis-sdf` tool from SDF^3) was measured. Measurements were repeated 50 times. Results were obtained on an Intel Xeon CPU core running at 2.40GHz within a 24-core machine with 64GB of RAM. Table I shows the average and variance of the measured runtimes for the two approaches on the three different test sets. The results clearly indicate that our approach based on the analysis of an LCG outperforms the simulation-based approach by several orders of magnitude.

Improvement over state-space exploration was highest for the 'Long Transient' set, which is to be expected since these graphs have a longer transient phase, which affects state-space exploration.

The 'Large HSDFG' set was, in terms of runtime, the most difficult set for our method to analyse. This result is mostly due to the size of the LCGs, which although smaller than the equivalent HSDF graphs usually obtained, may still be quite large. We remark that in [2] it was found that for some of the SDF graphs in this set, the HSDF-based analysis could not be completed within 30 minutes, due to the expensive transformation from SDF to HSDF. From this perspective, the observed runtimes on the 'Large HSDFG' set clearly demonstrate the effectiveness of the LCG-based approach.

TABLE I
EXPERIMENTAL RESULTS OF THE TWO METHODS ON THE THREE TEST SETS.

	Mimic DSP	Large HSDFG	Long Transient
State-space exploration			
avg $[s]$	1.1×10^{-3}	6.6×10^{-2}	4.2×10^{-1}
var $[s^2]$	2.1×10^{-5}	2.3×10^{-1}	1.7×10^{-2}
max $[s]$	6.5×10^{-2}	5.0	1.1
Linear Constraint Graph			
avg $[s]$	5.6×10^{-5}	1.1×10^{-3}	1.7×10^{-4}
var $[s^2]$	5.3×10^{-8}	2.7×10^{-5}	1.4×10^{-7}
max $[s]$	1.0×10^{-3}	5.3×10^{-2}	1.0×10^{-3}

TABLE II
AVERAGE GRAPH SIZES

	Mimic DSP	Large HSDFG	Long Transient
SDF Graphs			
vertices	20.0	13.4	284
edges	24.4	21.7	359
Equivalent HSDF Graphs			
vertices	1008	8166	284
edges	3151	95321	359
Linear Constraint Graphs			
vertices	119 (11.8%)	754 (9.2%)	284 (100%)
edges	151 (4.8%)	1202 (1.3%)	359 (100%)

For the state space exploration method (see table I) the high variance in runtime of $0.23\ s^2$, in comparison to the average execution time of $0.066\ s$, shows that for this method too some of the graphs require a large amount of time. This is also illustrated by the rather high recorded maximum runtime (5.0 seconds) required by state-space exploration. The runtime of our method on the 'Large HSDFG' set, however, was never slower than 53 ms, shows a much lower variance and a 60 times lower average.

The measured variance for state space exploration shows a remarkable difference with the results obtained earlier in [2], where the variance reported for the 'Large HSDFG' set was lower than the variance measured on the 'Long Transient' set, whereas in our results it was higher. We have no clear explanation for this inconsistency.

For each SDF graph in the three testsets, the number of vertices and edges in the LCG as well as the number of actors and channels in the equivalent HSDF graphs was recorded. Table II shows the average reduction achieved by an LCG when compared to an SDF graph's equivalent HSDF graph. Note that the 'Long Transient' testset contains SDF graphs which are in fact HSDF graphs and thus can not be represented more compactly by an LCG. The percentages included in the last two rows of the table indicate the amount of vertices and edges relative to the equivalent HSDF graph. An important observation from Table II is the apparent redundancy found in equivalent HSDF graphs: not only do the LCGs contain much fewer edges than HSDF graphs, but large parts of the HSDF graph are simply not needed by the analysis.

VIII. CONCLUSION AND FUTURE WORK

In this paper we have presented a novel approach to timing analysis of SDF graphs. At the basis of this approach is a max-plus algebraic representation of a consistent SDF graph as a linear, periodically time-variant system, followed by a transformation into a linear time-invariant system by a change of variables. The Linear Constraint Graphs we derive are smaller (i.e., have fewer vertices and edges) than equivalent HSDF graph usually derived from an SDF graph. This is achieved by exploiting the regular structure of an LCG, the properties of which we thoroughly prove. Results convincingly show that throughput analysis based on finding the maximum cycle ratio in an SDF graph's LCG is faster than the state-of-the-art state-space exploration method.

The regular structure of the constraint graph provides a basis for new and efficient timing analysis techniques for SDF graphs. In our current and future work we aim at further improvement of our method by pruning the SDF graph before its LCG is constructed, and running an extensive comparison between simulation-based methods and our method on more general classes of graphs, such as Cyclo-Static Dataflow [15] and Weighted T-Systems [16].

ACKNOWLEDGEMENT

The authors would like to thank Marco Gerards and Jochem Rutgers for their fruitful discussions and the anonymous reviewers for their constructive feedback, which helped considerably in improving the quality of this paper.

APPENDIX

This appendix contains the proofs for the statements in Section VI, in order of appearance.

Proof of Proposition 1: First of all, since $i_1 = \tilde{\pi}_{ab}(j_1)$ and $i_2 = \tilde{\pi}_{ab}(j_2)$, we have $\tilde{\pi}_{ab}(j_2) > \tilde{\pi}_{ab}(j_1)$. Furthermore, since $j_2 < j_1$, we have $\pi_{ab}(j_2) < \pi_{ab}(j_1)$. It then follows that $\delta_{ab}(j_2) < \delta_{ab}(j_1)$. Edge $a_{i_2}b_{j_2}$ thus has more delays (recall that the number of delays on an edge is $-\delta_{ab}(j)$) than edge $a_{i_1}b_{j_1}$. The fact that the number of delays on the two edges can not differ by more than one completes the proof. ■

Proof of Proposition 2: $j_2 > j_1$ gives $\pi_{ab}(j_2) > \pi_{ab}(j_1)$ and $i_2 > i_1$ gives $\tilde{\pi}_{ab}(j_2) > \tilde{\pi}_{ab}(j_1)$. It then follows that $\delta_{ab}(j_2) = \delta_{ab}(j_1)$. ■

Proof of Lemma 1: We prove this by induction on the number of actors n of the paths. First of all, note that $a_k^{i_k} \neq a_k^{j_k}$ for all $k = 1, 2, \ldots, n$ since two distinct edges can not have the same sink (and by assumption the inequality holds for $k = 1$). In case $n = 2$ both paths are edges and we obtain the result from Propositions 1 and 2. Next, suppose the result holds for all paths with at most n actors, and consider two paths P_i and P_j with $n + 1 \geq 3$ actors and with final edges $e_i = a_n^{i_n} a_{n+1}^{i_{n+1}}$ and $e_j = a_n^{j_n} a_{n+1}^{j_{n+1}}$, respectively. We assume again that $i_1 > j_1$. There are four cases to consider, depending on the relative orders of i_n, j_n and i_{n+1}, j_{n+1}. If $j_n < i_n$ ($j_n > i_n$) and $j_{n+1} < i_{n+1}$, then the paths $P_i - a_{n+1}^{i_{n+1}}$ and $P_j - a_{n+1}^{j_{n+1}}$ are parallel (crossing) and $a_n^{i_n} a_{n+1}^{i_{n+1}}$ and $a_n^{j_n} a_{n+1}^{j_{n+1}}$ are parallel (crossing), and the claims follow by induction and by Propositions 1 and 2. The other two cases are similar. ■

Proof of Corollary 1: Both claims may be easily proven by contradiction using Lemma 1. As the proofs for both claims is similar, it suffices to prove claim (1). Assume paths P_i and P_k do cross. Then by Lemma 1, P_i, P_j and P_k must all have different delays, in particular $|P_i|_d \neq |P_j|_d$. But since $k_1 > j_1$ and $k_1 > i_1$, by the same lemma we have $|P_k|_d = |P_i|_d + 1$ and $|P_k|_d = |P_j|_d + 1$, which implies the contradiction $|P_i|_d = |P_j|_d$. ∎

Proof of Proposition 3: Let $k = 1$. Note that since P_i and P_{i+1} are parallel, we have $\rho(i+1) > \rho(i)$. Assume $\rho(i+1) > \rho(i)+1$. There must exist j such that $\rho(j) = \rho(i)+1$. Let P_j be the path that connects j with $\rho(i)+1$. In case $j < i$, we have $P_j \nparallel_p P_i$ and $P_j \parallel_p P_{i+1}$. By Corollary 1 however, we have $P_j \nparallel_p P_{i+1}$, which is a contradiction. The assumption that $j > i+1$ leads to a contradiction in a similar way. We thus have $\rho(i+1) = \rho(i)+1$, and by straightforward induction on k it follows that $\rho(i+k) = \rho(i)+k$. ∎

Proof of Proposition 4: Assume $\rho(i+1) > 1$. There must exist j such that $\rho(j) = 1$. Let P_j be the path that connects j with $\rho(j)$. In case $j < i$, we have $P_j \parallel_p P_{i+1}$ and $P_j \parallel_p P_i$. By Corollary 1 however, we have $P_i \nparallel_p P_j$, which is a contradiction. In case $j > i+1$, we have $P_j \nparallel_p P_i$ and $P_j \nparallel_p P_{i+1}$, which again contradicts Corollary 1, thus $\rho(i+1) = 1$. Following a similar reasoning it follows that $\rho(i) = n$. By straightforward induction on k it follows that $\rho(i+k) = k$ and $\rho(i-k) = n-k$. ∎

Proof of Theorem 1: Let $\mathcal{G}$ consist of N paths. We may define ρ as follows: $\rho(i+k) = (\rho(i)+k-1) \bmod N+1$, for some $k \in \{1,\ldots,n\}$. Using the definition of ρ, the length l of a cycle starting at vertex a may be calculated by finding the minimum positive value of l that satisfies the linear congruence: $a + l \cdot k \equiv a \pmod N$. This solution l is independent of a, which implies $n_i = n_j$. Since a cycle is a path that starts and ends in the same vertex, C_1 and C_2 are parallel paths. Then by Lemma 1, C_1 and C_2 must have the same delay. ∎

Proof of Theorem 2: Let the critical cycle in $\mathcal{G}$ be $C = (a_1^{i_1} \ldots a_n^{i_n} = a_1^{i_1})$. Cycle C is contained in the LCG that corresponds to a cycle W in the SDF graph, with $W = (a_1, a_2, \ldots, a_n = a_1)$ (Note that this cycle may be a walk in the SDF graph, i.e., vertices may be repeated).

We prove the theorem by contradiction. Let $\mathcal{H}$ not contain C. Then s obviously does not lie on C. Furthermore, there is no path from a vertex on C to s, since if this were the case, C would be in $\mathcal{H}$.

Now choose a vertex $v = a_1^{i_m}$ that is not reachable from C, but from which there is a path to s (i.e., v is in $\mathcal{H}$). By Proposition 5, such a vertex can always be found. If in the LCG that represents SDF cycle W we follow edges in reverse direction from v, then eventually a cycle C' will be found. By Theorem 1, C' has both the same length and the same delay as C. We may thus restrict our search to $\mathcal{H}$. ∎

REFERENCES

[1] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[2] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij, "Throughput analysis of synchronous data flow graphs," in *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, ser. ACSD '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 25–36.

[3] K. Ito and K. K. Parhi, "Determining the minimum iteration period of an algorithm," *Journal of VLSI Signal Processing*, vol. 11, no. 3, pp. 229–244, Dec. 1995.

[4] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. Boca Raton, FL, USA: CRC Press, Inc., 2009.

[5] M. Geilen, "Reduction techniques for synchronous dataflow graphs," *Annual ACM IEEE Design Automation Conference*, pp. 911–916, 2009.

[6] M. Nakamura and M. Silva, "Cycle time computation in deterministically timed weighted marked graphs," in *1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA '99 (Cat. No.99TH8467)*, vol. 2. IEEE, 1999, pp. 1037–1046.

[7] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 4, pp. 385–418, Oct. 2004.

[8] G. Cohen, S. Gaubert, and J.-P. Quadrat, "Max-plus algebra and system theory: Where we are and where to go now," *Annual Reviews in Control*, vol. 23, pp. 207–219, Jan. 1999.

[9] G. Cohen, G. J. Olsder, and J.-p. Quadrat, *Synchronization and linearity*. Wiley New York, 1992.

[10] B. Heidergott, G. J. Olsder, and J. van der Woude, *Max Plus at Work: modeling and analysis of synchronized systems*. Princeton University Press, 2006.

[11] J. C.-T. Guy, G. Cohen, S. Gaubert, M. Mc, and G. J. pierre Quadrat, "Numerical computation of spectral elements in max-plus algebra," in *In Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[12] N. E. Young, R. E. Tarjan, and J. B. Orlin, "Faster parametric shortest path and minimum balance algorithms," *CoRR*, vol. cs.DS/0205041, 2002.

[13] A. H. Ghamarian, M. C. W. Geilen, T. Basten, B. D. Theelen, M. R. Mousavi, and S. Stuijk, "Liveness and boundedness of synchronous data flow graphs," in *Proceedings of the Formal Methods in Computer Aided Design*, ser. FMCAD '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 68–75.

[14] S. Stuijk, M. C. W. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*. IEEE Computer Society Press, Los Alamitos, CA, USA, Jun. 2006, pp. 276–278.

[15] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[16] E. Teruel, P. Chrzastowski-Wachtel, J. M. Colom, and M. Silva, "On weighted t-systems," in *Proceedings of the 13th International Conference on Application and Theory of Petri Nets*. London, UK, UK: Springer-Verlag, 1992, pp. 348–367.