Efficient Computation of the Max-Plus Semantics of Synchronous Dataflow Graphs

H. Elahi*, M. Geilen* and T. Basten*†
*Eindhoven University of Technology, Eindhoven, The Netherlands
†ESI, TNO, Eindhoven, The Netherlands
{g.elahi, m.c.w.geilen, and a.a.basten}@tue.nl

***Abstract*—Streaming systems are naturally modeled with synchronous dataflow graphs (SDFGs). The max-plus semantics of an SDFG is a compact matrix representation of its timing behavior. The max-plus semantics enables us to analyze and control timing properties of the systems, such as the obtainable minimum guaranteed throughput and maximum latency. Deriving the max-plus semantics, and consequently, performance analysis may be computationally expensive since the state-of-the-art method simulates one iteration of an SDFG. This holds, in particular, for systems whose components operate at different levels of granularity, as this results in many executions of some components in one iteration.**

This paper aims at efficiently calculating the max-plus semantics of SDFGs. The paper proposes an optimization framework exploring decompositions of a given SDFG and finding a composition sequence whose computational effort for compositionally obtaining the max-plus semantics is minimal. Not only does our proposed technique accelerate the performance analysis of multiscale streaming systems, but it also allows us to compute the max-plus semantics of some systems for which the state-of-the-art method does not succeed because of memory limitations.

***Index Terms*—Streaming systems, Multiscale systems, Dataflow graphs, Performance analysis, Max-plus semantics**

I. Introduction

An important subclass of embedded systems runs applications that process continuous streams of data received from several sources such as cameras. These systems are called streaming systems [2]. Industries ranging from automotive to healthcare realize the need for this type of embedded systems.

Streaming systems often have strict real-time constraints on their performance properties such as their maximum latency and minimum throughput [3]. Consider, for instance, an autonomous driving system; a delay in its response time may lead to catastrophic circumstances including human fatality.

To find tight conservative bounds on performance properties of streaming systems, they may be analyzed with an appropriate model of computation. A well-known model of computation that provides support for performance analysis is synchronous dataflow (SDF) [4,5]. Dynamic behavior of streaming systems can be characterized by a collection of different scenarios, each modeled by an SDF graph (SDFG). This model is called scenario-aware dataflow (SADF) [6]. SDFGs may include binding and scheduling information [7]. In this way, it is possible to analyze multiprocessor implementations.

SDFGs can be used to reason about the temporal behavior of the system, such as minimum guaranteed throughput and maximum latency. Generally, three classes of analysis techniques can be distinguished: (1) cycle-ratio analysis on a homogeneous SDFG or a variant of it, for instance K-periodic schedule graphs [8], (2) state-space-based approaches, for instance [9], and (3) techniques based on the max-plus semantics of an SDFG [10]. Each of the techniques has advantages and disadvantages; see [8,9] for some comparisons. One important advantage of having the max-plus semantics of an SDFG is that this semantics serves as the basis for a variety of performance analysis techniques (see [10]).

We therefore focus on computing the max-plus semantics of SDFGs for timing analysis. Symbolic simulation of one iteration of the graph [10,11] is the standard method for this. Streaming systems are, however, becoming increasingly complex. Their components often operate at different granularity scales. For instance, a component might work at pixel level while another component operates at frame level. This adversely impacts symbolic simulation since it should simulate the temporal behavior of all events in one iteration of an SDFG model of the application. For multiscale applications, such an iteration may have many repetitions of component executions. Consequently, the simulation may take very long.

To overcome this hurdle, in this paper, we propose a method to decrease the computational effort for calculating the max-plus semantics of SDFGs. Given an SDFG, we propose an optimization framework to decompose the SDFG into sub-SDFGs such that compositionally obtaining the max-plus semantics of the SDFG from the max-plus semantics of the resulting sub-SDFGs according to the found composition order is computationally more efficient than symbolic simulation of the full graph. A well-chosen decomposition of an SDFG into sub-SDFGs and a well-chosen order of compositions of these sub-SDFGs reduce both the number of executions of components in symbolic simulation and the size of vectors representing timestamps of events in the symbolic simulation. After calculating the max-plus semantics of sub-SDFGs, matrix multiplications suffice to compute the max-plus semantics of the whole SDFG, which requires less time and memory.

Contribution: This work combines the compositionality concept for multi-rate max-plus-linear systems of [12] with

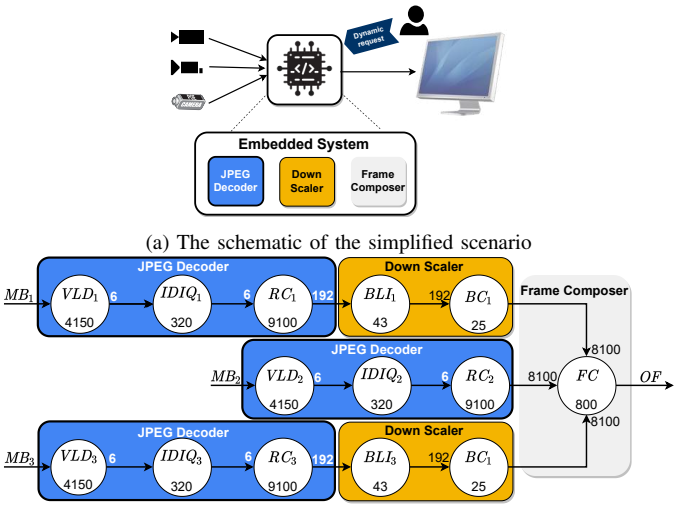
This research was supported by the Electronic Components and Systems for European Leadership (ECSEL) Joint Undertaking under grant number H2020-ECSEL-2017-2-783162 through the FitOptiVis project [1].

the concept of symbolic simulation of [10,11] to compute the max-plus semantics of an SDFG according to a composition sequence of sub-SDFGs. A composition sequence uniquely determines both a decomposition and an order of compositions. We propose a branch-and-bound optimization technique to explore the space of composition sequences for computing the max-plus semantics of an SDFG and find a composition sequence that minimizes the computation time for obtaining the semantics. To predict and bound the computation time during exploration, we analyze and quantify the computational effort for calculating the semantics of an SDFG according to a composition sequence in terms of the SDFG parameters.

We experimentally validate our method on a set of streaming systems, including a motivating example mimicking the processing of streams in an operating theater in a hospital. The results illustrate how our method can substantially accelerate performance analysis of streaming systems. For instance, for our motivating example, our technique computes the max-plus semantics in about 1.5 minutes, while the state-of-the-art method cannot compute it due to running out of memory.

Related work: The most directly related work that addresses the problem of this paper, computation of the max-plus semantics of SDFGs for timing analysis of streaming systems, is the work of [10,11]. Ref [11] obtains the max-plus semantics of SDFGs without inputs and outputs capturing external dependencies through symbolic simulation of one iteration of the SDFG. The work of [10] extends the method of [11] for SDFGs with inputs and outputs. Our work builds on to [10,11] as it computes the max-plus semantics of SDFGs compositionally, where the semantics of the sub-SDFGs is computed using symbolic simulation. Our method is an application of the compositional framework for max-plus linear systems of [12]. The framework of [12] can be applied to any streaming systems with compositional structures, but it does not consider the efficiency of the computation. Our work is a domain-specific specialization of branch-and-bound algorithms [13] to improve the efficient computation of the max-plus semantics of a given SDFG by finding an optimal graph partitioning in combination with a composition sequence for computing the max-plus semantics. Although there are many graph partitioning algorithms [14], since the order of compositions is also important, this paper develops a branch-and-bound optimization technique to efficiently find the combination of a graph partitioning and an order of composition. The work of [15] proposes a max-plus algebraic method for analyzing throughput of SDFGs that is more efficient than throughput analysis based on the standard max-plus semantics. Our work speeds up the computation of the standard max-plus semantics and as such speeds up the analysis of all timing properties that can be computed from this semantics, not only covering throughput but, for instance, also latency [10].

The paper is organized as follows. Sec. II introduces the motivating example. Sec. III provides the preliminaries on which the paper builds. Sec. IV presents the compositional computation of max-plus semantics of SDFGs. Sec. V sets up the optimization problem for finding a composition sequence



(a) The schematic of the simplified scenario

(b) The SDF model of a configuration

Fig. 1: Multi-source streaming (MSS) example

of an SDFG with minimal computation time of calculating the max-plus semantics. Sec. VI gives an algorithm to compute the semantics of an SDFG according to some composition sequence. Sec. VII presents the optimization framework to find a minimal solution. Sec. VIII proposes a cost function to estimate the computation time with the help of App. A and App. B Sec. IX presents the experimental evaluation. Sec. X presents conclusions and future work.

II. Motivating Example

We consider an operating theater from the healthcare domain as a motivating example. An important part of an operating theater is a multi-source image streaming system. The system can display different types of image streams received from different sources with different frame rates and resolutions. The hospital team may select some of those streams with desired resolutions at any time. This system must display streams with a guaranteed maximum end-to-end latency and best possible frame rate, as a delay might result in incidents with non-compensable repercussions. Therefore, analyzing the temporal behaviors of the streams is vital.

Fig. 1a depicts a simplified scenario of the motivating example. Assume that three sources stream Full HD (FHD) (1920×1080 pixels) at 30 frames per second compressed in M-JPEG (motion JPEG) format. There is an Ultra HD (UHD) display on which the user can display streams with either FHD or qHD (960×540) resolution. An embedded system receives, decodes, and if needed, down scales streams. The system then composes streams into an UHD stream to be shown on the display with the quality and performance that the users request.

Fig. 1b depicts the SDFG of a working mode of the specified system where stream 1 and 3 are decoded and down scaled, and stream 2 is only decoded. An SDFG [16] is a directed graph representing tasks within an application with *actors* (graph nodes). In Fig. 1b, VLD_1, $IDIQ_1$ and RC_1 are tasks within the JPEG decoder. These tasks represent variable-length

decoding, inverse discrete cosine transformation merged with inverse quantization, and image reconstruction [17]. Bilinear interpolation (*BLI*) and block construction (BC) are tasks within a down scaler. Actors execute their tasks repeatedly. An execution of an actor is referred to as a *firing*. Edges, so-called *channels*, denote the data dependencies between tasks within the application. For instance, the channel from VLD_1 to $IDIQ_1$ indicates that a firing of $IDIQ_1$ depends on VLD_1. To capture the external dependencies, we incorporate inputs and outputs. For instance, MB_1 is an encoded macro-block stream of the HD video and *OF* is the UHD stream to be displayed.

Once an application is mapped onto a (timing-predictable) platform, an execution-time upper bound for firing of every actor is obtained and included in to the SDFG to perform timing analysis [5]. This upper bound is referred to as the *worst-case execution time* (WCET) of the actor and annotated as a number of nanoseconds inside the nodes in Fig. 1b.

When an actor executes, it consumes and produces some entities on channels referred to as *tokens*. Every firing of an actor produces and consumes fixed numbers of tokens on channels according to so-called production and consumption *rates*. Rates are indicated by numbers on the channels. Rate 6 on the channel from VLD_1 to $IDIQ_1$ means that VLD_1 produces 6 tokens on this channel after every firing. We do not write rates equal to one. An actor executes if, and only if, it has enough tokens on all channels from which it consumes tokens. The actor is then said to be *enabled*. Channels may initially contain tokens called *initial tokens*. The actors in the SDFG of Fig. 1b all have a self-loop edge with a single initial token. These edges have been omitted from the figure for readability. These edges model the fact that actors cannot execute multiple times in parallel, because each task is assumed to be mapped onto one processor. An *iteration* of an SDFG is the smallest non-empty set of actor firings returning the token distribution on the channels to the initial state [18].

To analyze the timing behavior of the SDFG, one iteration of the graph is simulated. During simulation, the availability times of tokens are described by *symbolic timestamps*. The timestamps of re-produced initial tokens after one iteration and of outputs can in this way be characterized in terms of the timestamps of the initial tokens and inputs in the form of linear matrix equations in the max-plus algebra, the *max-plus semantics* of the SDFG. The max-plus semantics of SDFGs enables us to analyze the performance properties of systems modeled by SDFGs [11].

SDFGs like Fig. 1b with big differences in production and consumption rates introduce a large number of actor firings during an iteration. This indicates that actors operate at different levels of granularity. Since for obtaining max-plus semantics, one iteration of the SDFG is simulated, the number of actor firings strongly impacts the running time of the simulation. Performing symbolic simulation following [10,11] for one iteration of Fig. 1b, as the standard method to extract the max-plus semantics of the SDFG, takes several hours, and it cannot be completed due to running out of memory when memory usage exceeds 64 GB.

Even the simplified system in Fig. 1a has many working configurations. For every single configuration of the system, the max-plus semantics of the SDFG modeling the configuration must be calculated for timing analysis. This implies that for multiple-configuration multiscale systems, design-space exploration is computationally expensive.

Motivated by this example, we adopt the compositional model of max-plus-linear systems introduced in [12] to compositionally obtain the max-plus semantics of SDFGs. The SDFG in Fig. 1b has a natural decomposition with three JPEG-Decoders, two Down-scalers and a Frame-Composer. However, this decomposition does not provide us with a composition sequence to minimize the time to compute the max-plus semantics for the model as a whole. Even with the best composition order for this decomposition, the computation of the max-plus semantics takes longer than the computation time for obtaining the max-plus semantics using the method proposed in this paper. The proposed method is a branch-and-bound optimization algorithm [13] to decompose an SDFG into sub-SDFGs and find a composition sequence, such that the computation time for compositionally obtaining the max-plus semantics of the SDFG according to the found composition sequence is minimal. For the motivating example, the whole process including finding an optimal composition sequence and obtaining the max-plus semantics takes around 1.5 minutes. As mentioned, using the standard method, the computation does not complete within allotted memory.

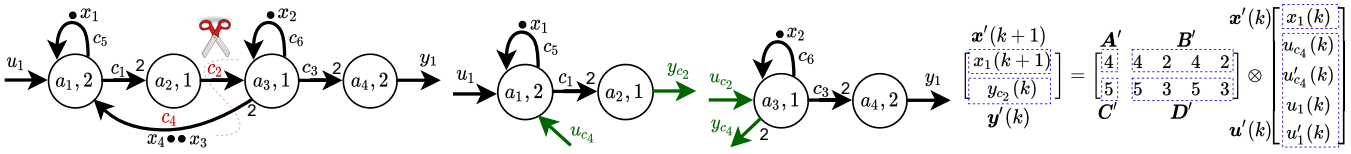
One may wonder why symbolic simulation runs out of memory. Symbolic simulation uses symbolic timestamps of tokens. It keeps the timestamp of every produced token until it is consumed. It moreover keeps the timestamps of initial tokens and inputs even after consumption so that it can return the max-plus semantics. For instance, for firing actor *FC* in Fig. 1b, 24301 tokens are consumed. The timing of each token is characterized by a symbolic vector of size 24314 (the total number of input and initial tokens consumed in an iteration; see App. A for details). Vector elements are double precision floating point numbers with a size of 8 bytes. Therefore, just before the *FC* firing, around 5 GB memory is allocated only for these tokens. By decomposing an SDFG as proposed in this paper, the number of actor firings and the vector sizes in symbolic simulations needed for computing the max-plus semantics of an SDFG can be reduced substantially.

III. Preliminaries

Throughout this paper, we denote scalars with lowercase letters, sets with uppercase letters, vectors with bold lowercase letters, matrices with bold uppercase letters, sequences with Greek upper case letters, universal sets with blackboard bold letters, and tuples with calligraphic letters. We use camel case to denote functions.

A. Synchronous Dataflow Graphs

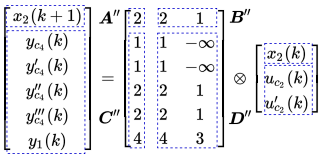
An SDFG is formally defined by a tuple $\mathcal{G} \triangleq (A, C, U, Y, srcA, dstA, srcR, dstR, iT, exT)$. Fig. 2a (ignoring the dashed line and scissors) shows an example. A, C, U



(a) SDFG $\mathcal{G}$ is a composition of $\mathcal{G}'$ and $\mathcal{G}''$ (b) SDFG $\mathcal{G}'$ (c) SDFG $\mathcal{G}''$ (d) The max-plus semantics of $\mathcal{G}'$

Firing	Symbolic start time	Symbolic completion time
1st a_1	$[0 \quad 0 \quad -\infty \quad 0 \quad -\infty]^T$	$[2 \quad 2 \quad -\infty \quad 2 \quad -\infty]^T$
2nd a_1	$[2 \quad 2 \quad 0 \quad 2 \quad 0]^T$	$[4 \quad 4 \quad 2 \quad 4 \quad 2]^T$
a_2	$[4 \quad 4 \quad 2 \quad 4 \quad 2]^T$	$[5 \quad 5 \quad 3 \quad 5 \quad 3]^T$

(e) Symbolic simulation for SDFG $\mathcal{G}'$



(f) The max-plus semantics of $\mathcal{G}''$

$oI = \{\overbrace{(y_{c_2}, u_{c_2})}^{oi}, \overbrace{(y_{c_4}, u_{c_4})}^{oi'}\}$
$ciT = \{(oi, 0), (oi', 2)\}$
(g) I/O mapping oI, and number of initial tokens on connections

$P = \{\overbrace{\{a_1, a_2\}}^{A'}, \overbrace{\{a_3, a_4\}}^{A''}\}$
(h) Partitioning $P \in \mathbb{P}(\mathcal{G})$

Fig. 2: A running example to illustrate SDFGs, the max-plus semantics of SDFGs, and their (de-)compositions

and Y are the sets of actors, channels, inputs and outputs, respectively. In the example, $A = \{a_1, a_2, a_3, a_4\}$, $C = \{c_1, c_2, \ldots, c_6\}$, $U = \{u_1\}$, and $Y = \{y_1\}$. To capture relations between actors, channels, inputs, and outputs, the following functions are defined. Function $srcA : C \cup Y \to A$ represents the source actors of channels and outputs (e.g., $srcA(y_1) = a_4$). Likewise, $dstA : C \cup U \to A$ describes the destination actors of channels and inputs ($dstA(c_4) = a_1$). The token production and consumption rates on channels, inputs and outputs are defined by $srcR : C \cup Y \to \mathbb{N}$ and $dstR : C \cup U \to \mathbb{N}$, respectively (e.g., $srcR(c_1) = 1$). $iT : C \to \mathbb{N}_0$ represents the number of initial tokens on channels (e.g., $iT(c_4) = 2$). Tokens are named (e.g., x_3 and x_4) for reference purposes. Function $exT : A \to \mathbb{R}_{\geq 0}$ specifies the execution times of actors (e.g., $exT(a_3) = 1$). We assume that graphs are consistent [18] and that $rP : A \to \mathbb{N}$ maps each actor of $\mathcal{G}$ to its repetition number in one iteration of the graph (e.g., $rP(a_1) = 4$). Note that the repetition number follows from the graph and does not need to be specified explicitly.

B. Max-plus Algebra

Max-plus algebra [19] supports two basic operations, namely maximization ($\oplus$) and addition ($\otimes$); $a \oplus b \triangleq max(a, b)$ and $a \otimes b \triangleq a + b$, for $a, b \in \mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$. $\otimes$ binds stronger than $\oplus$. As in linear algebra, the set $\mathbb{R}_{-\infty}$ with operations can be extended to vectors in $\mathbb{R}^n_{-\infty}$ and matrices in $\mathbb{R}^{n \times m}_{-\infty}$, where n and $m \in \mathbb{N}$. The entry in the i-th row and j-th column of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_{(i,j)}$. For $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times m}_{-\infty}$, $\boldsymbol{A} \oplus \boldsymbol{B}$ is defined by $[\boldsymbol{A} \oplus \boldsymbol{B}]_{(i,j)} = \boldsymbol{A}_{(i,j)} \oplus \boldsymbol{B}_{(i,j)}$, To multiply two matrices $\boldsymbol{A} \in \mathbb{R}^{n \times m}_{-\infty}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times p}_{-\infty}$, $[\boldsymbol{A} \otimes \boldsymbol{B}]_{(i,j)} \triangleq \bigoplus_{k=1}^{m} \boldsymbol{A}_{(i,k)} \otimes \boldsymbol{B}_{(k,j)}$. The b-th power of a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}_{-\infty}$ is $\boldsymbol{A}^b = \underbrace{\boldsymbol{A} \otimes \boldsymbol{A} \otimes \cdots \otimes \boldsymbol{A}}_{b}$.

C. Max-plus Semantics of Synchronous Dataflow Graphs

To capture the temporal behavior of SDFGs, synchronization and delay are two operations that are needed. Synchronization occurs when an actor waits to have sufficient tokens on all inputs before its execution. This can be captured by operation $\oplus$ in the max-plus algebra to obtain the time when the input tokens are available. In Fig. 2a, actor a_1 waits to have one token on each channel c_4, c_5, and input u_1. For example, if t_1, t_2 and t_3 are the production times of tokens on c_4, c_5, and u_1 then the firing time of a_1 is $t_1 \oplus t_2 \oplus t_3$. The production time of output tokens of an actor is its firing time increased with the duration of the firing. This can be modeled with $\otimes$ in the max-plus algebra. In Fig. 2a, if the firing time of a_1 is t, then the time of producing tokens on c_1 and c_5 after this firing is $t \otimes 2$. Similar to other discrete-event systems with only synchronization and delay [19], the timestamps of next states and outputs can be computed from the timestamps of inputs and states in canonical form called state-space equations.

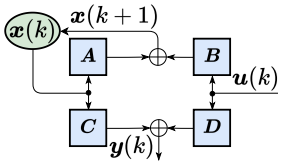


Fig. 3: State-space model

Definition 1 (Max-plus semantics). *For SDFG $\mathcal{G}$, let vector $\boldsymbol{x}(k)$ consist of the timestamps at which the initial tokens in the graph have become available (with the initial state, $\boldsymbol{x}(0)$, being the zero vector) and let vector $\boldsymbol{x}(k+1)$ contain the timestamps of re-producing the initial tokens at the end of iteration k. Vector $\boldsymbol{u}(k)$ and vector $\boldsymbol{y}(k)$ capture the timing of input and output tokens in iteration k. We refer to the state-space equations of $\mathcal{G}$ as the* max-plus semantics *of $\mathcal{G}$. The equations are as follows, visually illustrated in Fig. 3:*

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A} \otimes \boldsymbol{x}(k) \oplus \boldsymbol{B} \otimes \boldsymbol{u}(k), \\ \boldsymbol{y}(k) &= \boldsymbol{C} \otimes \boldsymbol{x}(k) \oplus \boldsymbol{D} \otimes \boldsymbol{u}(k), \end{aligned} \tag{1}$$

where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, and $\boldsymbol{D}$ are called the system matrices of $\mathcal{G}$. These matrices can be determined through a procedure called symbolic simulation of one iteration of $\mathcal{G}$ (see [10,11]).

Symbolic simulation characterizes the timestamps of tokens symbolically as an expression representing a max-plus-linear combination of the terms $\boldsymbol{x}(k)$ and $\boldsymbol{u}(k)$. Each symbolic

timestamp is represented with a specific vector $\boldsymbol{s}$, by the expression $\boldsymbol{s}^T \otimes [\boldsymbol{x}(k)^T\ \boldsymbol{u}(k)^T]^T$. As an example, for the SDFG in Fig. 2b[1] in one iteration, actor a_1 executes twice ($rP(a_1) = 2$) to produce sufficient tokens for the firing of a_2. One iteration of the SDFG consists of two firings of a_1 and one firing of a_2. a_1 consumes two input tokens from each input in one iteration (one for every firing) denoted u_{c_4}, u'_{c_4}, u_1 and u'_1. Any event in an iteration can be captured by a symbolic timestamp involving only the timestamp of the initial token, x_1, and the timestamps of the input tokens. $u_1(k)$ and $u'_1(k)$ denote the timestamps of the two input tokens consumed from u_1 in iteration k. The symbolic start and completion times of the first firing of actor a_1 are shown in row 1 of the table in Fig. 2e. The symbolic start time states that the firing happens zero time units after initial token $x_1(k)$ and inputs $u_1(k)$ and $u_{c_4}(k)$ become available, and that the first a_1 firing does not depend on the input tokens $u'_1(k)$ and $u'_{c_4}(k)$ (the $-\infty$ entries in the timestamp). As a_1 takes two time units, the symbolic timestamp vector of its completion time is obtained by adding the execution time, 2, to all entries of its symbolic start time vector. The second firing of a_1 consumes the second input token from u_1 and u_{c_4}, and the token produced on c_5 after its first firing. Its symbolic start time is represented by the maximum of the symbolic timestamp vectors of those tokens. Its execution adds two time units. Its symbolic start and completion times are shown in row 2 of the table in Fig. 2e. The two tokens produced on c_1 are consumed by a_2. The symbolic timestamp vectors of that firing are shown in row 3. This completes the iteration of $\mathcal{G}'$.

After simulating one iteration, the symbolic timestamp vectors of remaining tokens are collected as the rows of a matrix representing the four matrices of the max-plus semantics of the SDFG. In our example, the remaining tokens are on channel c_5 and output y_{c_2}. The max-plus semantics of the SDFG in Fig. 2b is shown in Fig. 2d. Similarly, the max-plus semantics of the SDFG of Fig. 2c is shown in Fig. 2f.

IV. Max-plus Semantics of Compositions

Discrete-event systems that follow the state-space equation of Eq. (1) are called *max-plus-linear systems* (MPLSs) [19]. To compute the max-plus semantics of a composite system from the max-plus semantics of its constituents, [12] introduced a compositional model in the form of an algebraic method to calculate the max-plus semantics of any arbitrary composition of MPLSs. This section illustrates how to use this method for SDFGs using the running example shown in Fig. 2.

Given are two SDFGs $\mathcal{G}'$ and $\mathcal{G}''$ (Fig. 2b and 2c), a partial I/O mapping *oI* from outputs of $\mathcal{G}'$, $\mathcal{G}''$ to inputs of $\mathcal{G}''$, $\mathcal{G}'$, and *ciT* representing the number of initial tokens on the new I/O connections in *oI* (Fig. 2g). The composition of $\mathcal{G}'$ and $\mathcal{G}''$ according to *oI* and *ciT* creates SDFG $\mathcal{G}$ of Fig. 2a. Ref. [12] introduces two operations that allow us to find the max-plus semantics of $\mathcal{G}$ from the max-plus semantics of $\mathcal{G}'$ and $\mathcal{G}''$ shown in Fig. 2d and 2f.

[1]SDFG $\mathcal{G}'$ is a sub-SDFG of SDFG $\mathcal{G}$ in Fig. 2a; thus, we choose the names in $\mathcal{G}'$ in accordance with the names in $\mathcal{G}$.

A. Rate Synchronization

In Fig. 2c, since $rP''(a_3) = 2$ (a_3 executes twice in a single iteration) and $srcR''(y_{c_4}) = 2$ (a_3 produces two tokens on y_{c_4} during every firing), every iteration produces four tokens on y_{c_4}. However, in Fig. 2b, as $rP'(a_1) = 2$ and $dstR'(u_{c_4}) = 1$, only two tokens are consumed in one iteration from u_{c_4}. This means that tokens produced on y_{c_4} in one iteration of $\mathcal{G}''$ can be consumed in two iterations of $\mathcal{G}'$ from u_{c_4}. Similarly, tokens produced in two iterations of $\mathcal{G}'$ on y_{c_2} are sufficient for one iteration of $\mathcal{G}''$. Therefore, the first step of the composition is to synchronize the production and consumption rates on the new I/O connections in *oI*, called *rate synchronization*.

In our example, to synchronize the rates on I/O connections, the max-plus semantics of $\mathcal{G}'$ for two iterations is calculated. We refer to such max-plus semantics, in general, as *r-fold iteration*. We let $\boldsymbol{A}^{*r}, \boldsymbol{B}^{*r}, \boldsymbol{C}^{*r}$, and $\boldsymbol{D}^{*r}$ denote the matrices of the r-fold iteration. For the example, it can be determined by considering the state-space equations of Eq. (1) for $\boldsymbol{x}(k+1)$, $\boldsymbol{y}(k)$ and $\boldsymbol{x}(k+2)$, $\boldsymbol{y}(k+1)$ together, and substituting the former equation for $\boldsymbol{x}(k+1)$ in the latter equation to determine $\boldsymbol{x}(k+2)$, $\boldsymbol{y}(k)$ and $\boldsymbol{y}(k+1)$ as functions of $\boldsymbol{x}(k)$, $\boldsymbol{u}(k)$ and $\boldsymbol{u}(k+1)$ (for details we refer to [12]). The matrices for the 2-fold iteration for $\mathcal{G}'$ are of the following form.

$$\boldsymbol{G'}^{*2} = \left[\begin{array}{c|c} \boldsymbol{A'}^{*2} & \boldsymbol{B'}^{*2} \\ \hline \boldsymbol{C'}^{*2} & \boldsymbol{D'}^{*2} \end{array}\right] = \left[\begin{array}{c|cc} \boldsymbol{A'}^{2} & \boldsymbol{A'} \otimes \boldsymbol{B'} & \boldsymbol{B'} \\ \hline \boldsymbol{C'} & \boldsymbol{D'} & -\infty \\ \boldsymbol{C'} \otimes \boldsymbol{A'} & \boldsymbol{C'} \otimes \boldsymbol{B'} & \boldsymbol{D'} \end{array}\right].$$

For the general form (r-fold iteration), see Eq. (11) in App. B.

After calculating 2-fold iteration for $\mathcal{G}'$, as the next step, rate synchronization aggregates the r-fold iterations for $\mathcal{G}'$ and $\mathcal{G}''$. Note that $\mathcal{G}''$ does not need to be iterated in the example. Aggregation essentially means that the two SDFGs are combined into one SDFG without connecting any channels between them. The final result of the rate synchronization for our running example is as follows:

$$\left[\begin{array}{cc|ccc} \boldsymbol{A'}^{2} & -\infty & \boldsymbol{A'} \otimes \boldsymbol{B'} & \boldsymbol{B'} & -\infty \\ -\infty & \boldsymbol{A''} & -\infty & -\infty & \boldsymbol{B''} \\ \hline \boldsymbol{C'} & -\infty & \boldsymbol{D'} & -\infty & -\infty \\ \boldsymbol{C'} \otimes \boldsymbol{A'} & -\infty & \boldsymbol{C'} \otimes \boldsymbol{B'} & \boldsymbol{D'} & -\infty \\ -\infty & \boldsymbol{C''} & -\infty & -\infty & \boldsymbol{D''} \end{array}\right]. \quad (2)$$

B. I/O composition

After rate synchronization, the remaining step to compose the SDFGs is to connect some of their outputs and inputs with channels, and possibly add initial tokens to the new channels. This is done one channel at a time, and the operation starts from the max-plus semantics with synchronized production and consumption rates for the outputs and inputs to be connected. After adding a connection, say (y, u), tokens produced on output y of one SDFG are consumed from input u of the other SDFG. Availability of initial tokens on (y, u) introduces a delay between production and consumption of tokens as u first consumes the $ciT((y, u))$ initial tokens from (y, u), and then the tokens produced on y. u then consumes the first part of the tokens produced on y within an iteration, namely

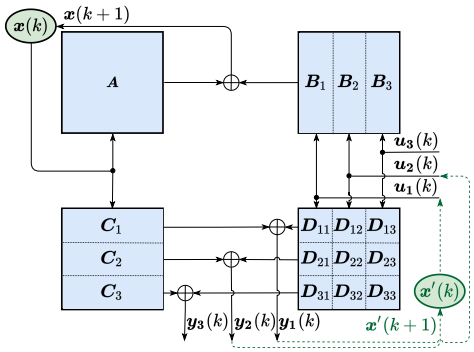


Fig. 4: State-space model for I/O composition

$rP'(dstA'(u)) \cdot dstR'(u) \cdot r - ciT((y,u))$ tokens, with r the repetition count in the rate synchronization of the consuming graph (assumed $\mathcal{G}'$). The other tokens produced on y are consumed from u in following iterations. Therefore, these tokens become part of the state vector, $\boldsymbol{x}$, of the composite SDFG. In our example, two initial tokens are available on (y_{c_4}, u_{c_4}) (see Fig. 2g). Thus, u_{c_4} first consumes these two initial tokens, and it then consumes the first two tokens produced on y_{c_4}. The next two tokens produced on y_{c_4} are consumed from u_{c_4} in the next iteration. (Note that it may happen that the number of initial tokens is greater than the total number of tokens produced and consumed on connection (y,u) in one iteration, i.e., when $ciT((y,u)) > rP(dstA(u)) \cdot dstR(u)$. For this case, we refer to [12].)

A new connection from y to u creates a new channel. For instance, connection (y_{c_2}, u_{c_2}) introduces new channel c_2 in Fig. 2a, and u_{c_2} and y_{c_2} are no longer an input and output of SDFG $\mathcal{G}$ in Fig. 2a. Since after adding connection (y,u), y and u no longer are an output respectively input, the corresponding input and output variables must be eliminated from the max-plus semantics. Deriving the final equations is straightforward, but somewhat tedious. To formulate I/O composition, first, the input and output token vectors in Eq. (1) are divided into three parts, illustrated in Fig. 4:

- $\boldsymbol{y}_1$ and $\boldsymbol{u}_2$ are the vectors of timestamps of tokens on (y,u) produced and immediately consumed in the iteration, as explained above.
- $\boldsymbol{y}_2$ and $\boldsymbol{u}_1$ are the vectors of timestamps of tokens on (y,u) produced and consumed in different iterations. Their size is $ciT((y,u))$.
- $\boldsymbol{u}_3$ and $\boldsymbol{y}_3$ are the vectors of the timestamps of input and output tokens on inputs and outputs not connected by (y,u) that remain after connecting (y,u).

The state-space equations can be written as follows.

$$\begin{bmatrix} \boldsymbol{x}(k+1) \\ \boldsymbol{y}_1(k) \\ \boldsymbol{y}_2(k) \\ \boldsymbol{y}_3(k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}_1 & \boldsymbol{B}_2 & \boldsymbol{B}_3 \\ \boldsymbol{C}_1 & \boldsymbol{D}_{11} & \boldsymbol{D}_{12} & \boldsymbol{D}_{13} \\ \boldsymbol{C}_2 & \boldsymbol{D}_{21} & \boldsymbol{D}_{22} & \boldsymbol{D}_{23} \\ \boldsymbol{C}_3 & \boldsymbol{D}_{31} & \boldsymbol{D}_{32} & \boldsymbol{D}_{33} \end{bmatrix} \otimes \begin{bmatrix} \boldsymbol{x}(k) \\ \boldsymbol{u}_1(k) \\ \boldsymbol{u}_2(k) \\ \boldsymbol{u}_3(k) \end{bmatrix}. \quad (3)$$

Fig. 4 without dashed lines in the bottom right-hand corner shows the above relation. As discussed, because of initial tokens on a connection, (y,u), in every iteration, the first part of the tokens produced on y is consumed from u after consuming available tokens on (y,u). Thus, these variables must be eliminated from the equations, which is achieved by substitution using the equation $\boldsymbol{u}_2(k) = \boldsymbol{y}_1(k)$ (for details, see [12]). To give insight, consider Fig. 4. Before adding dashed lines, the dependency of $\boldsymbol{x}(k+1)$ on $\boldsymbol{x}(k)$ is described by $\boldsymbol{A}$ as there is only one path from $\boldsymbol{x}(k)$ to $\boldsymbol{x}(k+1)$ (in the top left-hand corner of Fig. 4). Adding a dashed line from $\boldsymbol{y}_1(k)$ to $\boldsymbol{u}_2(k)$ introduces a new direct path from $\boldsymbol{x}(k)$ to $\boldsymbol{x}(k+1)$. $\boldsymbol{C}_1$ and $\boldsymbol{B}_2$ respectively describe the dependency from $\boldsymbol{y}_1(k)$ on $\boldsymbol{x}(k)$ and $\boldsymbol{x}(k+1)$ on $\boldsymbol{u}_2(k)$. Thus, $\boldsymbol{B}_2 \otimes \boldsymbol{C}_1$ is the dependency of $\boldsymbol{x}(k+1)$ on $\boldsymbol{x}(k)$ through this new path after eliminating $\boldsymbol{y}_1$ and $\boldsymbol{u}_2$. The max operation (at the top of Fig. 4) between the two paths yields $\boldsymbol{x}(k+1) = (\boldsymbol{A} \oplus \boldsymbol{B}_2 \otimes \boldsymbol{C}_1) \otimes \boldsymbol{x}(k)$.

To capture the timestamps of the newly added initial tokens on connection (y,u) and the second part of tokens produced in every iteration, a vector of new state variables $\boldsymbol{x}'(k)$ with the same size as the number of initial tokens on (y,u) is defined. Let $\boldsymbol{x}'(k+1)$ be the timestamps of the remaining tokens on (y,u) after iteration k. As discussed above, $\boldsymbol{y}_2(k)$ contains the timing of these tokens. Thus, we substitute $\boldsymbol{x}'(k+1)$ for $\boldsymbol{y}_2(k)$. On the right-hand side of Eq. (3), $\boldsymbol{x}'(k)$ is substituted for $\boldsymbol{u}_1(k)$ as, in iteration k, u first consumes remaining tokens from iteration $k-1$. After these substitutions the impact of (y,u) is completely captured. The dependencies of $\boldsymbol{x}(k+1)$, $\boldsymbol{x}'(k+1)$ and $\boldsymbol{y}_3(k)$ on each of $\boldsymbol{x}(k)$, $\boldsymbol{x}'(k)$ and $\boldsymbol{u}_3(k)$ can be seen in Fig. 4, and are also given in column 3 of Tab. II in App. B.

The IO-composition operation is used as many times as the size of IO mapping oI. It initially takes the output of the rate-synchronization operation, a connection $oi \in oI$, and $ciT(oi)$, and returns the matrices of the max-plus semantics of the SDFG including the new channel. This process is repeated for all connections in oI. In our example, IO composition is applied twice, first to connect y_{c_4} to u_{c_4} forming channel c_4, and then to connect y_{c_2} and u_{c_2}. The final results are the matrices of the max-plus semantics of $\mathcal{G}$ in Fig. 2a. Note that the order of choosing connections does not impact the final max-plus semantics. However, the computational efforts are different (see App. B).

V. Problem Formulation

This section presents the formulation of the optimization problem with the aid of the running example. We need to find both a decomposition of an SDFG and a corresponding composition order that determines the compositional computation of the max-plus semantics of the SDFG, such that the computation time is minimal. First, we introduce graph decomposition. To capture the order of compositions, composition sequences are defined. We conclude this section with formulating the main objective of the paper.

A decomposition of an SDFG $\mathcal{G}$ is captured by a partitioning of its actor set A. A partitioning P of A is a set of non-empty disjoint subsets of A such that the union of those subsets is A. We require such a partitioning P to partition $\mathcal{G}$ into connected SDFGs, i.e., for all $A' \in P$ and for any two actors a and

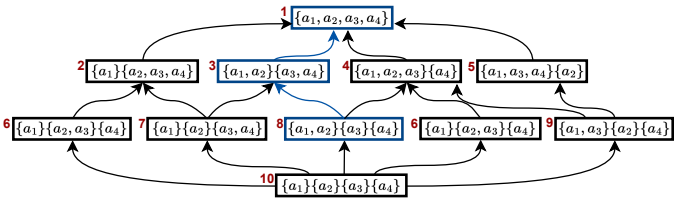


Fig. 5: Partitionings and composition sequences of the example

a' in A', there exists a path of actors connected by channels from a to a' in A'. Partitioning P in Fig. 2h partitions the SDFG of Fig. 2a in two connected SDFGs. Let $\mathbb{P}(\mathcal{G})$ denote the set of partitionings that partition $\mathcal{G}$ into connected SDFGs. Note that sub-SDFGs in a partitioning must be connected, because only if all actors in a sub-SDFG are connected, the relative firing rates of the actors are determined and it has a well-defined max-plus semantics. If, for instance, SDFG $\mathcal{G}$ of Fig. 2a is decomposed according to $\{a_1, a_4\}$ and $\{a_2, a_3\}$, the information on the relative rates of 4 to 1 for actors a_1 and a_4 in $\mathcal{G}$ is missing when considering the (non-connected) sub-SDFG consisting of a_1 and a_4. Setting the rates to be equal would lead to an incorrect max-plus semantics when following the compositional computation of the semantics explained in the previous section.

A partitioning $P \in \mathbb{P}(\mathcal{G})$ determines a cut-set $cS(P) \subseteq C$ of channels that have their actor endpoints in different partitions:

$$cS(P) \triangleq \{c \in C \mid srcA(c) \in A', dstA(c) \notin A', A' \in P\}.$$

For example, in Fig. 2a, $cS(\{\{a_1, a_2\}, \{a_3, a_4\}\}) = \{c_2, c_4\}$.

Once SDFG $\mathcal{G}$ is decomposed according to P, channels in $cS(P)$ are no longer channels of any component graph. A cut-set determines a relation oI that is a set of pairs of fresh outputs and inputs created by splitting channels:

$$oI(P) \triangleq \{(y_c, u_c) \mid c \in cS(P)\}, \tag{4}$$

where y_c and u_c are fresh names. For instance, in Fig. 2a, $\{c_2, c_4\}$ determines $oI = \{(y_{c_2}, u_{c_2}), (y_{c_4}, u_{c_4})\}$. To capture the number of initial tokens on $c \in cS(P)$, we define $ciT : oI \to \mathbb{N}_0$ such that for all $oi = (y_c, u_c) \in oI$, $ciT(oi) = iT(c)$.

Any partition $A' \in P$ determines a sub-SDFG $sG(A') \triangleq (A', C', U', Y', srcA', dstA', srcR', dstR', iT', exT')$, defined as follows. The channels of the sub-SDFG are channels of $\mathcal{G}$ with both endpoints connected to elements in its actor set:

$$C' \triangleq \{c \in C \mid srcA(c) \in A', dstA(c) \in A'\}.$$

Each $c \in C'$ inherits other properties, namely $srcA'$, $dstA'$, $srcR'$, $dstR'$ and iT' from $srcA$, $dstA$, $srcR$, $dstR$ and iT. The inputs (outputs) of $sG(A')$ are the union of inputs (outputs) in U (Y) connected to actors in A' and the new inputs (outputs) of oI and have $dstA$ ($srcA$) in A'. Hence, $U' \triangleq$

$$\{u \in U \mid dstA(u) \in A'\} \cup \{u_c \mid dstA(c) \in A', (y_c, u_c) \in oI\},$$

and similar for Y'. $U' = \{u_1, u_{c_4}\}$ and $Y' = \{y_{c_2}\}$ in Fig. 2b. Inputs and outputs inherit the properties from their original inputs and outputs. For all $u \in U' \cap U$, $dstA'(u) \triangleq dstA(u)$ and $dstR'(u) \triangleq dstR(u)$. For all $u_c \in U' \setminus U$, $dstA'(u_c) \triangleq dstA(c)$ and $dstR'(u_c) \triangleq dstR(c)$. This is similar for outputs. Finally, $exT'(a) \triangleq exT(a)$, for all $a \in A'$.

This paper aims to compute the max-plus semantics of SDFGs more efficiently compared to the state-of-the-art, symbolic simulation. We consider a sequence of pairwise compositions of sub-SDFGs from a selected partitioning to compute the max-plus semantics compositionally. Different orders of compositions can be used to compose the same set of sub-SDFGs. As the order of composition has an important impact on the efficiency of the computation, finding an optimal order is incorporated in the optimization problem. The next two definitions enable us to formalize the composition procedure.

Definition 2 (Refinement). *For two partitionings $P, P' \in \mathbb{P}(\mathcal{G})$, if every element of P' is a subset of some element of P (i.e., if every sub-SDFG determined by some element of P' is a part of some sub-SDFG determined by an element of P), then P' is called a refinement of P. We say P' is an immediate refinement of P, and denote $P' \triangleleft P$, if exactly one element in P is partitioned into two elements in P'.*

Fig. 5 shows partitionings of SDFG $\mathcal{G}$ of Fig. 2a: branches connect nodes to their immediate refinements that correspond to partitionings into connected sub-SDFGs. To avoid cluttering, node 6 is duplicated in Fig. 5.

Definition 3 (Composition sequence). *Let Π be a sequence of partitionings in $\mathbb{P}(\mathcal{G})$. Let Π_k denote the partitioning at position $k \geq 1$, and $|\Pi|$ be the length of Π. Π is a composition sequence if $\Pi_{|\Pi|} = \{A\}$, and for $1 \leq k < |\Pi|$, $\Pi_k \triangleleft \Pi_{k+1}$. Let $\mathbb{C}(\mathcal{G})$ be the set of all composition sequences of SDFG $\mathcal{G}$.*

In Fig. 5, any path ending in node 1 is a composition sequence of $\mathcal{G}$ in Fig. 2a, for instance, sequence $8, 3, 1$.

To compute the max-plus semantics of an SDFG, we perform compositions in the order of the partitionings in a given partitioning sequence Π. Starting from the most refined partitioning, sub-SDFGs are combined until the SDFG is composed. E.g, for sequence $8, 3, 1$ in Fig. 5, first the sub-SDFGs corresponding to $\{a_3\}$ and $\{a_4\}$ are composed, followed by composition of the sub-SDFGs of $\{a_1, a_2\}$ and $\{a_3, a_4\}$.

Definition 4 (Optimization problem). *For SDFG $\mathcal{G}$ and $\Pi \in \mathbb{C}(\mathcal{G})$, let $cT(\Pi)$ be the computation time for calculating the max-plus semantics of $\mathcal{G}$ according to Π. We define the optimization problem to find Π such that $cT(\Pi)$ is minimal.*

VI. Max-plus Semantics of SDFGs According to Composition Sequences

This section presents the computation of the max-plus semantics of an SDFG according to a composition sequence. First, symbolic simulation [10,11] is used to calculate the max-plus semantics of the smallest sub-SDFGs as explained in Sec. III-C. The compositional model of MPLSs from [12] and explained in Sec. IV is then used to compute the max-plus semantics of the compositions according to the sequence.

Alg. 1 presents this procedure. Note that the output of the algorithm is independent of Π, but the computation time is

Algorithm 1: Computation of the max-plus semantics of SDFG $\mathcal{G}$ according to composition sequence Π

Input: SDFG $\mathcal{G}$, $\Pi \in \mathbb{C}(\mathcal{G})$
Output: The max-plus semantics of $\mathcal{G}$

1 **Function** `computeMaxPlusSemantics`($\mathcal{G}$, Π):
2 **foreach** $A' \in \Pi_1$ **do**
3 $mpS(A') \leftarrow$ symbolic simulation of $sG(A')$
4 $k \leftarrow 1$
5 **while** $k < |\Pi|$ **do**
6 $\{A_1, A_2\} \leftarrow \Pi_k \setminus \Pi_{k+1}$
7 Let $oI(\{A_1, A_2\})$ and ciT be determined
8 $\{A'\} \leftarrow \Pi_{k+1} \setminus \Pi_k$
9 $mpS(A') \leftarrow$ the max-plus semantics of the composition of $mpS(A_1)$ and $mpS(A_2)$ according to $oI(\{A_1, A_2\})$ and ciT
10 $k \leftarrow k + 1$
11 **return** $mpS(A')$

Algorithm 2: BnB to solve the optimization problem

Input: SDFG $\mathcal{G}$ (the actor set is A)
Output: Π with minimal $cT(\Pi)$

1 **Function** `findOptimalCompositionSeq`($\mathcal{G}$):
2 $\hat{\Pi} \leftarrow \{A\}$ // Incumbent solution
3 $Q \leftarrow \{\hat{\Pi}\}$
4 **while** $Q \neq \emptyset$ **do**
5 $\Pi \leftarrow$ pick a composition sequence from Q
6 **foreach** $\Pi' \in \mathbb{C}|_\Pi$ **do** // see Def. 5
7 $l \leftarrow cT(\Pi) - \sum_{A' \in \Pi_1 \text{ s.t. } |A'| \neq 1} cT(\{A'\}) + cT_1(\Pi')$
8 **if** $l < cT(\hat{\Pi})$ **then** // Prune otherwise
9 $Q \leftarrow Q \cup \{\Pi'\}$
10 $cT(\Pi') \leftarrow$ compute from Cor. 1
11 **if** $cT(\Pi') < cT(\hat{\Pi})$ **then**
12 $\hat{\Pi} \leftarrow \Pi'$// update incumbent solution
13 $Q \leftarrow Q \setminus \{\Pi\}$
14 **return** $\hat{\Pi}$

not. In Line 2 and 3, for $A' \in \Pi_1$ symbolic simulation of $sG(A')$ computes the max-plus semantics $mpS(A')$ as explained in Sec. III-C. For instance, for sequence $8, 3, 1$ in Fig. 5, symbolic simulation of the SDFG shown in Fig. 2b computes $mpS(\{a_1, a_2\})$ as shown in Fig. 2d.

Alg. 1 proceeds with the composition process. k is the number of the composition step initialized with one in Line 4. For $1 \leq k < |\Pi|$, composition k composes the two sub-SDFGs corresponding to two elements in $\Pi_k \setminus \Pi_{k+1}$ (A_1 and A_2). In Line 7, $oI(\{A_1, A_2\})$ and ciT are determined (see Sec. V). Thus, the max-plus semantics of composition k (the sub-SDFG of the only element A' of $\Pi_{k+1} \setminus \Pi_k$) can be computed from $mpS(A_1)$ and $mpS(A_2)$, the max-plus semantics of two sub-SDFGs $sG(A_1)$ and $sG(A_2)$, through the procedure explained in Sec. IV (Line 9 of Alg. 1). For composition sequence $8, 3, 1$ in Fig. 5, the first composition computes $mpS(\{a_3, a_4\})$ from $mpS(\{a_3\})$ and $mpS(\{a_4\})$ with $oI = \{(y_{c_3}, u_{c_3})\}$ derived from cut-set $\{c_3\}$ in Fig. 2a.

By iterating over k (Line 5), for $k = |\Pi| - 1$ (the last composition), since $\Pi_{|\Pi|} = \{A\}$ (from Def. 3), $mpS(A)$ is obtained and returned. The actor set of SDFG $\mathcal{G}$ is A. Thus, $mpS(A)$ is the max-plus semantics of $\mathcal{G}$.

VII. Optimization Framework

In this section, we propose a method to find a composition sequence that has a low computation time among the possible composition sequences. We systematically explore composition sequences, by adopting the branch-and-bound (BnB) framework [13]. After finding Π, `computeMaxPlusSemantics`$(\mathcal{G}, \Pi)$ in Alg. 1 is used to perform the corresponding calculation.

We intend to cover the solution space ($\mathbb{C}(\mathcal{G})$) such that a search tree (Fig. 5 illustrates an example) is built iteratively using BnB. We prove a bounding condition to prune a subtree of further decompositions derived from a composition sequence.

Alg. 2 implements the BnB search, and obtains Π such that $cT(\Pi)$ is minimal. The optimal composition sequence found so far (incumbent solution) is $\hat{\Pi} \in \mathbb{C}(\mathcal{G})$, and it is initialized with $\{A\}$ (Line 2). Q is the set of composition sequences from which BnB generates the tree of new composition sequences. Q is therefore initialized with $\{\hat{\Pi}\}$ (Line 3). A composition sequence to further explore, Π, is selected from Q (Line 5). BnB explores composition sequences created by partitioning one of the partitions of Π_1 in two parts. For instance in Fig. 5, BnB explores sequences $7, 3, 1$ and $8, 3, 1$ when the current sequence is $3, 1$.

Definition 5. *For $\Pi, \Pi' \in \mathbb{C}(\mathcal{G})$ such that $\Pi'_1 \triangleleft \Pi_1$ and $\Pi_k = \Pi'_{k+1}$ for all $k \leq |\Pi|$ (see Def. 2 and Def. 3), we refer to Π' as a child of Π. We let $\mathbb{C}|_\Pi \subset \mathbb{C}(\mathcal{G})$ denote the set of children of Π. Generally, we refer to Π' as a descendant of Π, if $|\Pi'| > |\Pi|$ and for all $k \leq |\Pi|$, $\Pi_k = \Pi'_{k+|\Pi'|-|\Pi|}$.*

Alg. 2 calculates in Line 7, based on Thm. 1 below, for each $\Pi' \in \mathbb{C}|_\Pi$, a lower bound l on the computation time of any solution that can be reached from Π' (i.e., any descendants of Π'). The lower bound l is used to prune the search tree if a solution with smaller computation time has already been found, following the BnB framework. If $l \geq cT(\hat{\Pi})$, the subtree derived from Π' is pruned (Line 8), because the minimal computation time cannot be obtained by branching to Π'. Π' is added to Q if $l < cT(\hat{\Pi})$ (Line 9). $cT(\Pi')$ is then calculated according to Cor. 1 given below (Line 10). The incumbent solution is updated if $cT(\Pi') < cT(\hat{\Pi})$ (Line 12). The last updated incumbent solution is returned, once no candidates remain in Q to explore. The rest of this section provides details for finding a lower bound and pruning.

Intuitively, in the composition process (Line 5 to 10 of Alg. 1), the last $|\Pi| - 1$ compositions are the same for Π, and its descendants. Also, symbolic simulation computes $mpS(A')$

for all $A' \in \Pi_1$ such that $|A'| = 1$, for Π and its descendants.

To prove a lower bound on the computation time of descendants of Π, we need to find the relation between $cT(\Pi)$ and the computation time of its descendants. Thus, we begin with formulating $cT(\Pi)$. Let $cT_k(\Pi)$ denote the computation time for performing composition k according to Π; $cT_k(\Pi)$ is defined explicitly in the next section, in Eq. (8).

Lemma 1. $cT(\Pi) = \sum_{A' \in \Pi_1} cT(\{A'\}) + \sum_{k=1}^{|\Pi|-1} cT_k(\Pi)$.

Proof. In Alg. 1, for composition sequence $\{A'\}$, the only computation performed is $mpS(A')$ (Line 3). Thus, $cT(\{A'\})$ is the computation time for calculating $mpS(A')$.The first summation captures the computation time of symbolic simulation for $|\Pi|$ sub-SDFGs of Π_1 (Lines 2-3). The second summation captures the computation time for performing the $|\Pi| - 1$ compositions computed in the while loop (Lines 5-10). ■

The relation between a composition according to a composition sequence and its descendants is a key to find the relation between their computation times, and also the lower bound l.

Lemma 2. *For* $\Pi' \in \mathbb{C}|_\Pi$, *and* $k < |\Pi|$, $cT_{k+1}(\Pi') = cT_k(\Pi)$.

Proof. From Def. 2, only one partition of Π_1 is partitioned in two partitions in Π'_1. Thus, the procedure for the second composition onward according to Π' is the same as the $|\Pi| - 1$ compositions according to Π. This can be proven by substitution using $\Pi'_{k+1} = \Pi_k$ from Def. 5. ■

In Fig. 5, the second composition in sequence $8, 3, 1$ is the same as the first composition in $3, 1$ computing $mpS(\{a_1, a_2, a_3, a_4\})$ from $mpS(\{a_1, a_2\})$ and $mpS(\{a_3, a_4\})$.

The next lemma generalizes Lem. 2 to descendants.

Lemma 3. *For any descendant* Π' *of* Π, *and* $1 \leq k < |\Pi|$, $cT_{k+|\Pi'|-|\Pi|}(\Pi') = cT_k(\Pi)$.

Proof. Natural induction on $|\Pi'| - |\Pi|$. ■

The following theorem is the basis to prune the search space. It provides a lower bound on the computation time for any descendants of a child of a composition sequence. Intuitively, the computation times for any descendants of a child include the computation times for all compositions required for the child and the computation time for symbolic simulation of those sub-SDFGs that are not decomposed further.

Theorem 1 (Lower bound). *For any descendants* Π'' *of child* Π' *of composition sequence* Π, *the following holds:*

$$cT(\Pi'') > cT(\Pi) - \sum_{A' \in \Pi_1 \text{ s.t. } |A'| \neq 1} cT(\{A'\}) + cT_1(\Pi'). \quad (6)$$

Proof. Subtracting $cT(\Pi)$ from $cT(\Pi'')$ using Lem. 1 after expanding the second summation on the right-hand side of $cT(\Pi'')$ yields Eq. (5). In Eq. (5), from Lem. 3, the last two summations are equal; also, $cT_{|\Pi''|-|\Pi|}(\Pi'') = cT_1(\Pi')$ as $|\Pi'| = |\Pi| + 1$. Since partition A' cannot be partitioned if $|A'| = 1$, $\Pi_1 \setminus \Pi''_1 \subseteq \{A' \in \Pi_1 \mid A' \neq 1\}$ resulting in $\sum_{A' \in \Pi_1 \text{ s.t. } |A'| \neq 1} cT(\{A'\}) \geq \sum_{A' \in \Pi_1 \setminus \Pi''_1} cT(A')$. The sum of the first and third summations is positive. ■

Algorithm 3: Number of operations for Alg. 1

Input: SDFG $\mathcal{G}$, and composition sequence Π
Output: The number of operations for Alg. 1
Global: *sZ*, size vectors for all explored partitions

```
1  Function findNrOfOperations(G, Π):
2      nSymMax ← Σ_{A'∈Π_1} nSm(sG(A')) // see Eq. (9)
3      nSymPlus ← Σ_{A'∈Π_1} nSp(sG(A')) // see Eq. (10)
4      foreach A' ∈ Π_1 do
5          sZ(A') ← initialize using Def. 6
6      k ← 1
7      nMax ← 0
8      nPlus ← 0
9      while k < |Π| do
10         [m, p]^T ← calcNrOprSzComp(G, Π_k \ Π_{k+1})
11         nMax ← nMax + m
12         nPlus ← nMax + p
13         k ← k + 1
14     return [nMax, nPlus, nSymMax, nSymPlus]^T
```

Remark 1 (Tightness). As $\sum_{k=1}^{|\Pi''|-|\Pi|-1} cT_k(\Pi'') = 0$ for $\Pi'' = \Pi'$, and the summands of the first and second summations in Eq. (5) for different Π' are not the same, it is not possible to obtain a greater lower bound.

Corollary 1. *If* $\Pi'_1 \setminus \Pi_1 = \{A', A''\}$, *then* $cT(\Pi') = cT(\Pi) + cT(\{A'\}) + cT(\{A''\}) - cT(\{A' \cup A''\}) + cT_1(\Pi')$ *(see Eq. (5)).*

VIII. Estimated Computation Time

Recall our optimization problem in Def. 4 and Alg. 2. Since we cannot formulate the exact computation time for running Alg. 1 as, usually, heterogeneous and unpredictable platforms are used for analysis at design time, we propose a cost function that captures the number of operations for running our algorithm to compute the max-plus semantics of an SDFG to predict the computation time. Max and plus operations play the most significant roles in both symbolic simulation [10,11] and in the compositional computations [12] for computing the max-plus semantics. Therefore, in this section, we develop a cost function as a linear function of the number of max and plus operations in terms of the SDFG parameters. This cost function can then be used to predict the computation time for running Alg. 1 for any composition sequence, and as such forms the basis for exploring the space of all possible composition sequences in Alg. 2.

Alg. 3 calculates the number of operations to obtain the max-plus semantics of $\mathcal{G}$ according to Π. It uses a number of auxiliary functions. App. A provides equations for two functions *nSm* and *nSp* providing the number of max and plus operation in symbolic simulation of a given SDFG. For compositions, matrix multiplication and addition in the max-plus domain play the most significant roles in the computation. App. B provides `calcNrOprSzComp` in Alg. 4 to obtain the

numbers of max and plus operations for computing the max-plus semantics of a composition.

The functions use various size characteristics of an SDFG. For SDFG $\mathcal{G}$, let $iniT(\mathcal{G})$, $inpT(\mathcal{G})$ and $outT(\mathcal{G})$ denote the token, input and output vector sizes of its max-plus semantics. $iniT(\mathcal{G}) = \sum_{c \in C} iT(c)$, which is the sum of initial token counts on all channels of $\mathcal{G}$. Actors connected to inputs consume fixed numbers of input tokens per firing, being the rates of the inputs. Thus, $inpT(\mathcal{G}) = \sum_{u \in U} rP\big(dstA(u)\big) \cdot dstR(u)$; Likewise, $outT(\mathcal{G}) = \sum_{y \in Y} rP\big(srcA(y)\big) \cdot srcR(y)$. For function definitions, see Sec. III-A.

Definition 6 (**Size vector**). *For SDFG $\mathcal{G}$ and for any partition $A' \in P \in \mathbb{P}(\mathcal{G})$ (see Sec. V), we define a size vector $sZ(A') = \big(iniT(sG(A')), inpT(sG(A')), outT(sG(A')), \rho'\big)$, where $\rho' = gcd(\{rP(a) \mid a \in A'\})$ with gcd denoting the greatest common divisor.*

Alg. 3 takes an SDFG and composition sequence Π as input. The size vectors of all the analyzed partitions are globally stored for efficiency. First, the sums of the numbers of max and plus operations for symbolic simulation of $sG(A')$ for all $A' \in \Pi_1$ are calculated in Lines 2-3. Second, for the computation of the composition cost, first, for all $A' \in \Pi_1$, the size vectors $sZ(A')$ are initialized in Lines 4-5. Line 6 initializes the composition step. In Lines 7-8, the numbers of max and plus operations for compositions are initialized. The required number of max and plus operations for the compositions are calculated in Lines 9-13. In Line 10, the number of operations for calculating the max-plus semantics of composition k is calculated by `calcNrOprSzComp` from Alg. 4 in App. B. The total numbers of max and plus operations to calculate the max-plus semantics of compositions are iteratively updated according to the composition sequence in Lines 11-12. Alg. 3 finally returns the numbers of max and plus operations for symbolic simulation and the compositions.

We have implemented Alg. 1 in SDF^3 [20] and analyzed the computation time. We observed that the contributions of max and plus operations to the overall execution time are not equal. Furthermore, max/plus operations in symbolic vectors have different computational costs compared to performing them on concrete values, due to memory overhead. Therefore, our cost function for estimating the computation time of Alg. 1 has four independent variables calculated by Alg. 3.

To calibrate the cost function, we created a synthetic SDFG $\mathcal{G}$ with 11 actors, 10 channels, 3 inputs and 2 outputs. For each $\Pi \in \mathbb{C}(\mathcal{G})$ such that $|\Pi| \leq 3$, `findNrOfOperations`$(\mathcal{G}, \Pi)$ from Alg. 3 was executed, and we measured the elapsed time $cTm(\Pi)$ for `computeMaxPlusSemantics`$(\mathcal{G}, \Pi)$ in Alg. 1. The run-times were obtained on a Linux server with Intel® Core™ i9-9900K processor (16 MB Cache, up to 5 GHz) and 64 GB RAM. We then adjust parameters $\boldsymbol{b} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix}^T$ of cost function

$$cT(\Pi) = \boldsymbol{b}^T \texttt{findNrOfOperations}(\mathcal{G}, \Pi), \qquad (7)$$

such that the sum of squared relative error is minimal. We use the generalized reduced gradient (GRG) method to find the cost function parameters.

For evaluation, we compare the measured and estimated elapsed time for a set of realistic examples in columns 10 and 11 in Tab. I. For the listed applications, the average of the relative and absolute errors are 0.27 and 1.66 seconds.

The parameters obtained for Eq. (7) also provide a basis to estimate the computation time for performing a single composition:

$$cT_k(\Pi) = \begin{bmatrix} b_1 & b_2 \end{bmatrix} \texttt{calcNrOprSzComp}(\mathcal{G}, \Pi_k \setminus \Pi_{k+1}). \quad (8)$$

Alg. 2 uses Eq. (7) and Eq. (8) as cost functions to estimate the computation times.

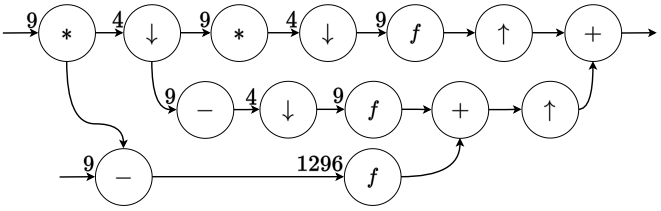


Fig. 6: A three-level pyramid filter modeled at block level

IX. Experimental Evaluation

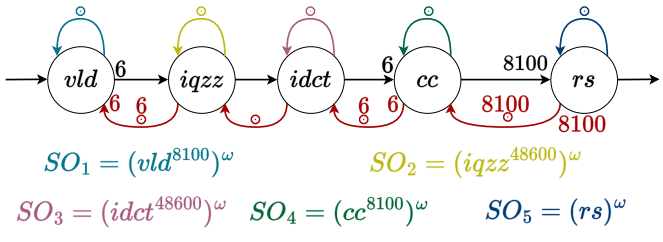
We implemented our proposed method in SDF^3 [20]. We compared our framework with the state-of-the-art approach of [10,11] that computes the max-plus semantics of an SDFG by symbolically simulating one iteration of the SDFG. Since some application models naturally suggest compositional structure, this section also evaluates composition sequences derived from the structure. For instance, the MSS example shown in Fig. 1b has a natural decomposition into six components: three JPEG-decoders, two down-scalers and a frame-composer. This section shows that our framework has usually a lower computation time than the state-of-the-art approach and a lower maximum memory allocation.

We use the following models in our evaluation: a motion-JPEG decoder (MJPEG-UC) and a down-sampler (DS-UC) for UHD frames modeled by closed-SDFGs (i.e., graphs without inputs and outputs), two different multiprocessor mappings of a Motion-JPEG decoder for FHD streams shown in Fig. 7a (MJPEG-F5P) and Fig. 7b (MJPEG-F3P), the MSS-FFF model shown in Fig. 1b, where 'FFF' indicates that all three streams

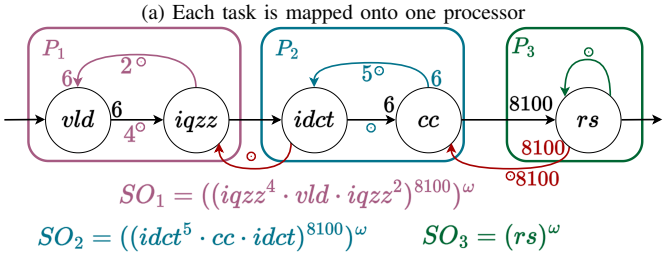
$$cT(\Pi'') - cT(\Pi) = \sum_{A' \in \Pi_1'' \setminus \Pi_1} cT(A') - \sum_{A' \in \Pi_1 \setminus \Pi_1''} cT(A') + \sum_{k=1}^{|\Pi''|-|\Pi|-1} cT_k(\Pi'') + cT_{|\Pi''|-|\Pi|}(\Pi'') + \sum_{k=|\Pi''|-|\Pi|+1}^{|\Pi''|-1} cT_k(\Pi'') - \sum_{k=1}^{|\Pi|-1} cT_k(\Pi) \qquad (5)$$

TABLE I: Experimental results (all times in seconds)

Application	Graph Parameters $\{\|A\|, \|C\|, \|U\|, \|Y\|\}$	Symbolic Simulation $cTm(\{A\})$	Symbolic Simulation $mM(\{A\})$	Natural Decomposition $\|\Pi'\|$	Natural Decomposition cTm^n_{min}	Natural Decomposition cTm^n_{max}	BnB Optimization cT_o	BnB Optimization $\|\Pi\|$	BnB Optimization $cT(\Pi)$	BnB Optimization $cTm(\Pi)$	BnB Optimization $mM(\Pi)$	BnB Optimization cTm_t
MJPEG-UC	{5, 9, 0, 0}	**<1**	**11 MB**	n/a	n/a	n/a	<1	1	<1	<1	**11 MB**	<1
MJPEG-F5P	{5, 9, 1, 1}	648	10 GB	5	11	>13	<1	2	6	10	**9.7 GB**	**10**
MJPEG-F3P	{5, 7, 1, 1}	477	9.9 GB	3	11	>13	<1	2	6	10	**9.7 GB**	**10**
DS-UC	{4, 7, 0, 0}	6	627 MB	n/a	n/a	n/a	<1	2	2	1	**370 MB**	**1**
MSS-HHH	{14, 27, 3, 1}	8627	21 GB	6	10	>19	22	4	8	8	**4.1 GB**	**30**
MSS-FFF	{14, 27, 3, 1}	>1862	>64 GB	6	50	>30	37	4	45	44	**19.5 GB**	**85**
MSS-FMH	{14, 27, 3, 1}	15465	36.4 GB	6	14	>16	35	4	12	12	**15.7 GB**	**47**
PF-B	{14, 27, 2, 1}	11	11.5 GB	n/a	n/a	n/a	<1	2	<1	1	**75 MB**	**1**
PF-32	{15, 29, 2, 1}	>34	>64 GB	n/a	n/a	n/a	<1	3	2	6	**1.6 GB**	**6**



(a) Each task is mapped onto one processor



(b) vld and $iqzz$ are mapped onto processor P_1, $idct$ and cc are mapped onto processor P_2, and rs is mapped onto processor P_3.

Fig. 7: Motion-JPEG decoder models for two different multiprocessor mappings including edges to model schedules and buffers. The static scheduling order per processor is given through ω-regular expressions.

operate at FHD resolution, variant MSS-HHH receiving compressed videos with HD resolution from three sources, MSS-FMH that receives three coded streams in FHD, 480p (640 × 480 pixels) and HD resolutions, PF-B, and PF-32 are pyramid multi-resolution filters [21] with three levels, modeled at block level as shown in Fig. 6 and frame level with input size 32×32.

Tab. I summarizes the experimental results. For each model $\mathcal{G}$, we list the graph parameters including the numbers of actors $|A|$, channels $|C|$, inputs $|U|$, and outputs $|Y|$. For symbolic simulation (the state-of-the-art method), we show the measured computation time, $cTm(\{A\})$, and the maximum allocated memory, $mM(\{A\})$. For our proposed method to compute the max-plus semantics with Alg. 1, we first consider the natural decompositions, if the models suggest any, and all possible orders of composition for that decomposition. We list the length of the composition sequence $|\Pi'|$ and the smallest and largest elapsed times cTm^n_{min} and cTm^n_{max}, since the order of compositions plays a significant role in the computational efficiency. We then proceed with evaluating composition sequence Π found by the Branch-and-Bound (BnB) optimization framework introduced in Sec. VII. We report computation time for optimization with BnB (Alg. 2) with cT_o, $|\Pi|$, which is the size of the composition sequence with the minimal cost, estimated computation time $cT(\Pi)$, determined from Eq. (7), measured computation time $cTm(\Pi)$ and maximum allocated memory $mM(\Pi)$ for computing the semantics according to Π with Alg. 1, and total computation time cTm_t for efficiently computing the semantics, which is equal to $cT_o + cTm(\Pi)$.

Tab. I reports times in seconds. Underlined bold numbers indicate that the computation time or maximum memory usage for a method is minimal among the methods. An oval indicates that the computation terminates with an error after the indicated time when memory usage exceeds 64 GB.

Our optimization framework finds composition sequence Π with a lower computation time than the computation time for the symbolic simulation in all cases, except for MJPEG-UC where BnB returns $\Pi = \{A\}$ (representing the original SDFG) as a minimal solution. Since the exploration space for MJPEG-UC is small, BnB quickly finds the optimum. The maximum memory usage for finding Π with Alg. 2 and computing the max-plus semantics with Alg. 1 is always at most equal to the memory usage for symbolic simulation.

The MSS models in our experiment have a natural decomposition, as shown in Fig. 1b. Also, the mapped motion-JPEG models have a natural decomposition, according to the binding of the actors to the different processors. In line with expectations, $cTm(\Pi)$ for all MSS applications and the two different multiprocessor mappings of motion-JPEG is at most equal to the computation time for computing the max-plus semantics according to the natural decomposition with the best possible composition order. The different orders for the natural decompositions show a lot of variations and some run out of memory within a minute. The computations run out of memory if the wrong order of compositions is chosen for the natural decomposition. For all these cases, the problem happens during composition, illustrating the importance of the order of composition.

We observe that the maximum memory for our proposed method is allocated during the symbolic simulation of sub-SDFGs, except for two strongly connected SDFGs (MJPEG-F3P and MJPEG-F5P), where maximum memory is allocated during composition. In general, we observe that memory usage

is strongly correlated with the total computation time for our method. Some possibilities for further optimization, such as exploiting sparsity of matrices, remain as future work.

Overall, we may conclude that BnB allows us to effectively compute the max-plus semantics for all models within minutes, whereas the traditional method generally takes much longer and fails to compute the max-plus semantics in several cases.

X. Conclusion

In this paper, we propose an effective technique to efficiently compute the max-plus semantics of SDFGs. This allows us to analyze performance properties for larger models than so far possible. The technique generalizes to SADF because it can be separately applied to all scenarios. Exploiting similarities in scenario SDFGs and the compositional computation of performance properties for SDFGs following the compositional computation of the max-plus semantics are interesting topics for future work.

References

[1] FitOptiVis, "The FitOptiVis ECSEL project: highly efficient distributed embedded image/video processing in cyberphysical systems," in *ACM Int'l Conf. on Computing Frontiers (CF)*. ACM, 2019, pp. 333–338.

[2] W. Thies *et al.*, "StreamIt: A language for streaming applications," in *Compiler Construction*. Springer, 2002, pp. 179–196.

[3] A. A. Jerraya and W. Wolf, "The what, why, and how of MPSoCs," in *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, 2005, pp. 1–18.

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

[5] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, 2000.

[6] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *Proc. IC-SAMOS*. IEEE, 2011, pp. 404–411.

[7] M. Bekooij *et al.*, *Dataflow Analysis for Real-Time Embedded Multiprocessor System Design*. Springer Netherlands, 2005, pp. 81–108.

[8] B. Bodin *et al.*, "Optimal and fast throughput evaluation of CSDF," in *Proc. 53rd ACM/EDAC/IEEE DAC*. ACM, 2016, pp. 1–6.

[9] A. H. Ghamarian *et al.*, "Throughput analysis of synchronous data flow graphs," in *Proc. ACSD*. IEEE, 2006, pp. 25–36.

[10] M. Geilen *et al.*, "Scenarios in dataflow modeling and analysis," in *System-Scenario-based Design Principles and Applications*. Springer, 2020, pp. 145–180.

[11] M. Geilen, "Synchronous dataflow scenarios," *ACM TECS*, vol. 10, no. 2, pp. 16:1–16:31, 2011.

[12] H. Elahi *et al.*, "A compositional model for multi-rate max-plus linear systems," *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 54–61, 2020.

[13] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[14] A. Buluç *et al.*, *Recent Advances in Graph Partitioning*. Cham: Springer International Publishing, 2016, pp. 117–158.

[15] R. de Groote *et al.*, "Max-plus algebraic throughput analysis of synchronous dataflow graphs," in *Proc. SEAA*, 2012, pp. 29–38.

[16] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, no. 1, pp. 24–35, 1987.

[17] R. F. Haines and S. L. Chuang, "The effects of video compression on acceptability of images for monitoring life sciences experiments," *NASA STI/Recon Technical Report N*, vol. 92, 1992.

[18] E. Lee, "Consistency in dataflow graphs," in *Proc. ASAP*. IEEE, 1991, pp. 355–369.

[19] F. L. Baccelli *et al.*, *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley & Sons Inc, 1993.

[20] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF for free," in *Proc. ACSD*. IEEE, 2006, pp. 276–278.

[21] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

Appendix A
Number of Operations for Symbolic Simulation

To determine the max-plus semantics of SDFG $\mathcal{G}$, one iteration of $\mathcal{G}$ is symbolically simulated [11]. To determine the symbolic enabling time of actor a, called Earliest Start Time (EST), symbolic simulation computes the maximum of the symbolic timestamp vectors of tokens consumed by a, which are of the same size as $[\boldsymbol{x}(k)^T\ \boldsymbol{u}(k)^T]^T$. To implement this, first, the symbolic timestamp of the EST of a is initialized with an all-minus-infinity vector. The symbolic timestamp of the EST is iteratively updated by taking the maximum with the symbolic timestamp of each token consumed by a from each channel c and input u. In our implementation, for each c (u) such that $\mathit{dstA}(c) = a$ ($\mathit{dstA}(u) = a$), the symbolic timestamp of availability of the *dstR(c)* (*dstR(u)*) tokens is initialized with an all-minus-infinity vector. For computing the symbolic timestamp for consuming *dstR(c)* (*dstR(u)*) tokens on each c (u), *dstR(c)* (*dstR(u)*) max operations on vectors are done after which the symbolic timestamp of the EST of a is updated, adding one more vector max operation per channel/input. Let $\mathit{sTv}(\mathcal{G})$ be the size of symbolic timestamp vectors. $\mathit{sTv}(\mathcal{G}) = \mathit{iniT}(\mathcal{G}) + \mathit{inpT}(\mathcal{G})$. The number of binary scalar max operations to compute the symbolic timestamp of the EST of a is then as follows.

$$\mathit{nSm}(a) = \mathit{sTv}(\mathcal{G})\Bigg(\sum_{\substack{c\in C \text{ s.t.}\\ \mathit{dstA}(c)=a}} \big(\mathit{dstR}(c)+1\big) + \sum_{\substack{u\in U \text{ s.t.}\\ \mathit{dstA}(u)=a}} \big(\mathit{dstR}(u)+1\big)\Bigg).$$

The total number of max operations for symbolic simulation of SDFG $\mathcal{G}$ is the sum of required binary scalar max operations for all actor firings:

$$\mathit{nSm}(\mathcal{G}) = \sum_{a\in A} \mathit{rP}(a)\cdot \mathit{nSm}(a). \tag{9}$$

Since all tokens are simultaneously produced by actor a as soon as its execution is completed, and therefore have the same timestamps, the number of produced tokens does not impact the computational effort. The timestamp of the produced tokens is calculated by adding the actor execution time, a scalar, to the EST. Therefore, the number of plus operations for each actor firing is equal to $\mathit{sTv}(\mathcal{G})$. The total number of pairwise additions for symbolic simulation of $\mathcal{G}$ is then as follows.

$$\mathit{nSp}(\mathcal{G}) = \mathit{sTv}(\mathcal{G}) \sum_{a\in A} \mathit{rP}(a). \tag{10}$$

Appendix B
Number of Operations for Composition

We analyze the computational effort of the two operations introduced in [12] and explained in Sec. IV. To calculate the max-plus semantics of the composition of $\mathit{sG}(A')$ and $\mathit{sG}(A'')$ with size vectors $\mathit{sZ}(A') = (|\boldsymbol{x}'|, |\boldsymbol{u}'|, |\boldsymbol{y}'|, \rho')$ and $\mathit{sZ}(A'') = (|\boldsymbol{x}''|, |\boldsymbol{u}''|, |\boldsymbol{y}''|, \rho'')$, let $r' = \frac{\rho'}{\mathit{gcd}(\rho',\rho'')}$ and $r'' = \frac{\rho''}{\mathit{gcd}(\rho',\rho'')}$ be the repetitions of $\mathit{sG}(A')$ and $\mathit{sG}(A'')$ for the synchronization.

A. Number of Operations for Rate Synchronization

Recall the state-space equations of Eq. (1). $\boldsymbol{A}^{*r}, \boldsymbol{B}^{*r}, \boldsymbol{C}^{*r}$, and $\boldsymbol{D}^{*r}$ are the matrices of the r-fold iteration, calculated as follows [12].

$$\begin{aligned}
\boldsymbol{A}^{*r} &= \boldsymbol{A}^{r}, \\
\boldsymbol{B}^{*r} &= \left[\boldsymbol{A}^{r-1} \otimes \boldsymbol{B}\ \boldsymbol{A}^{r-2} \otimes \boldsymbol{B} \ldots \boldsymbol{A} \otimes \boldsymbol{B}\ \boldsymbol{B}\right], \\
\boldsymbol{C}^{*r} &= \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C} \otimes \boldsymbol{A} \\ \vdots \\ \boldsymbol{C} \otimes \boldsymbol{A}^{r-1} \end{bmatrix}, \\
\boldsymbol{D}^{*r} &= \begin{bmatrix} \boldsymbol{D} & -\infty & \cdots & -\infty \\ \boldsymbol{C} \otimes \boldsymbol{B} & \boldsymbol{D} & \cdots & -\infty \\ \boldsymbol{C} \otimes \boldsymbol{A} \otimes \boldsymbol{B} & \boldsymbol{C} \otimes \boldsymbol{B} & \cdots & -\infty \\ \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{C} \otimes \boldsymbol{A}^{r-2} \otimes \boldsymbol{B} & \cdots & \boldsymbol{C} \otimes \boldsymbol{B} & \boldsymbol{D} \end{bmatrix}.
\end{aligned} \tag{11}$$

$\boldsymbol{A}^{*r} = \boldsymbol{A}^{r}$ is computed through exponentiation by squaring. The worst-case number of matrix multiplications happens when $r \in \{2^n - 1 \mid n \in \mathbb{N}\}$. Hence, for computing $\boldsymbol{A}^{r}$, maximally, $2\big(-1+\log_2(r+1)\big)$ times square matrices of order $|\boldsymbol{x}|$ are squared. The number of pairwise max and plus are $2\big(-1+\log_2(r+1)\big)(|\boldsymbol{x}|-1)|\boldsymbol{x}|^2$ and $2\big(-1+\log_2(r+1)\big)|\boldsymbol{x}|^3$. Note that for $\boldsymbol{M} \in \mathbb{R}_{-\infty}^{l\times m}$ and $\boldsymbol{N} \in \mathbb{R}_{-\infty}^{m\times n}$, $\boldsymbol{M} \otimes \boldsymbol{N}$ requires $m \times l \times n$ plus and $(m-1) \times l \times n$ pairwise max operations.

To compute matrix $\boldsymbol{B}^{*r}$, first, $\boldsymbol{A} \otimes \boldsymbol{B}$ is computed. Then, matrix product $\boldsymbol{A} \otimes \boldsymbol{B}$ is multiplied by $\boldsymbol{A}$ to compute $\boldsymbol{A}^2 \otimes \boldsymbol{B}$. This process is repeated until $\boldsymbol{A}^{r-1} \otimes \boldsymbol{B}$ is determined. Thus, $r-1$ matrices with the size of matrix $\boldsymbol{B}$ are multiplied by matrix $\boldsymbol{A}$. $\boldsymbol{A} \in \mathbb{R}_{-\infty}^{|\boldsymbol{x}|\times|\boldsymbol{x}|}$ and $\boldsymbol{B} \in \mathbb{R}_{-\infty}^{|\boldsymbol{x}|\times|\boldsymbol{u}|}$. Hence, $(r-1)(|\boldsymbol{x}|-1)|\boldsymbol{x}||\boldsymbol{u}|$ pairwise max and $(r-1)|\boldsymbol{x}|^2|\boldsymbol{u}|$ pairwise plus operations are performed.

To compute $\boldsymbol{C}^{*r}$, similar to $\boldsymbol{B}^{*r}$, $r-1$ times matrix $\boldsymbol{A}$ is multiplied with matrices of the same size as $\boldsymbol{C}$. $\boldsymbol{C} \in \mathbb{R}_{-\infty}^{|\boldsymbol{y}|\times|\boldsymbol{x}|}$, then $(r-1)(|\boldsymbol{x}|-1)|\boldsymbol{y}||\boldsymbol{x}|$ and $(r-1)|\boldsymbol{y}||\boldsymbol{x}|^2$ are the numbers of pairwise max and plus operations for computing $\boldsymbol{C}^{*r}$.

To compute $\boldsymbol{C} \otimes \boldsymbol{B}$, $\boldsymbol{C} \otimes \boldsymbol{A} \otimes \boldsymbol{B}$, $\boldsymbol{C} \otimes \boldsymbol{A}^2 \otimes \boldsymbol{B}$, …, $\boldsymbol{C} \otimes \boldsymbol{A}^{r-2} \otimes \boldsymbol{B}$ of $\boldsymbol{D}^{*r}$, we use some blocks of $\boldsymbol{B}^{*r}$. To compute the $r-1$ multiplications, matrices of size $|\boldsymbol{x}|\times|\boldsymbol{u}|$ are multiplied by $\boldsymbol{C}$. Thus, $(r-1)(|\boldsymbol{x}|-1)|\boldsymbol{y}||\boldsymbol{u}|$ pairwise max and $(r-1)|\boldsymbol{x}||\boldsymbol{y}||\boldsymbol{u}|$ pairwise plus operations are required.

To compute r-fold iteration for $sG(A)$, we let $nMrF(A,r)$ and $nPrF(A,r)$ denote the number of max and plus operations. They are therefore calculated as follows.

$$\begin{aligned}
nMrF(A,r) =& (|\boldsymbol{x}|-1)\big(2\big(-1+\log_2(r+1)\big)|\boldsymbol{x}|^2 \\
&+ (r-1)(|\boldsymbol{x}||\boldsymbol{u}| + |\boldsymbol{u}||\boldsymbol{y}| + |\boldsymbol{y}||\boldsymbol{x}|)\big), \\
nPrF(A,r) =& 2\big(-1+\log_2(r+1)\big)|\boldsymbol{x}|^3 \\
&+ (r-1)(|\boldsymbol{x}|^2|\boldsymbol{u}| + |\boldsymbol{x}||\boldsymbol{u}||\boldsymbol{y}| + |\boldsymbol{y}||\boldsymbol{x}|^2).
\end{aligned} \tag{12}$$

Rate synchronization returns the aggregation of matrices of r-fold iteration for $sG(A')$ and $sG(A'')$ as follows:

$$\left[\begin{array}{cc|cc} \boldsymbol{A}'^{*r'} & -\infty & \boldsymbol{B}'^{*r'} & -\infty \\ -\infty & \boldsymbol{A}''^{*r''} & -\infty & \boldsymbol{B}''^{*r''} \\ \hline \boldsymbol{C}'^{*r'} & -\infty & \boldsymbol{D}'^{*r'} & -\infty \\ -\infty & \boldsymbol{C}''^{*r''} & -\infty & \boldsymbol{D}''^{*r''} \end{array}\right]. \tag{13}$$

Eq. (13) represents the final result of rate synchronization and is taken as input for the I/O composition. As Eq. (13) shows, aggregation does not impact the number of operations but the size of matrices. Thus, the total numbers of max and plus operations for rate synchronization are $nMrF(A',r') + nMrF(A'',r'')$, $nPrF(A',r') + nPrF(A'',r'')$.

We define function $sZrS(sZ(A'), sZ(A''))$ to denote the size vector of the result of rate synchronization for composing $sG(A')$ and $sG(A'')$, needed as input for I/O composition as well, and calculated as follows.

$$\begin{aligned}
&sZ_{rs}(sZ(A'), sZ(A'')) = \\
&(|\boldsymbol{x}'|+|\boldsymbol{x}''|, r'|\boldsymbol{u}'|+r''|\boldsymbol{u}''|, r'|\boldsymbol{y}'|+r''|\boldsymbol{y}''|, gcd(\rho', \rho''))
\end{aligned} \tag{14}$$

B. Number of Operations of I/O Composition

The I/O composition operation calculates the max-plus semantics after adding a connection (see Sec. IV-B). It is applied for all connections between the components. Let $oi_j \in oI(\{A', A''\})$ be the j^{th} connection for composing $sG(A')$ and $sG(A')$. Let $i_j = ciT(oi_j)$ be equal to $|\boldsymbol{u}_1| = |\boldsymbol{y}_2|$ in Eq. (3). Let s_j denote the rate on connection oi_j after synchronization, that is $|\boldsymbol{u}_1| + |\boldsymbol{u}_2| = |\boldsymbol{y}_1| + |\boldsymbol{y}_2|$ in Eq. (3).

Let $|\boldsymbol{x}_{j^-}|$, $|\boldsymbol{u}_{j^-}|$ and $|\boldsymbol{y}_{j^-}|$ be the sizes of token, input and output vectors of the intermediate max-plus semantics before adding connection oi_j; they are respectively equal to $|\boldsymbol{x}|$, $|\boldsymbol{u}_1| + |\boldsymbol{u}_2| + |\boldsymbol{u}_3|$, and $|\boldsymbol{y}_1| + |\boldsymbol{y}_2| + |\boldsymbol{y}_3|$ in Eq. (3). Let $|\boldsymbol{u}_{j^+}|$ and $|\boldsymbol{y}_{j^+}|$ be the sizes of input and output vectors of the intermediate max-plus semantics after adding connection oi_j; they are $|\boldsymbol{u}_3|$ and $|\boldsymbol{y}_3|$ in Eq. (3). After connecting oi_j, input and output tokens connected by oi_j are no longer input and output tokens; thus, their sizes are decreased by s_j. Owing to the availability of initial tokens on oi_j, i_j state elements are concatenated to the state vector after connecting oi_j. Hence,

$$\begin{aligned}
|\boldsymbol{x}_{j^-}| &= |\boldsymbol{x}'| + |\boldsymbol{x}''| + \sum_{k=1}^{j-1} i_j, \\
|\boldsymbol{u}_{j^-}| &= r'|\boldsymbol{u}'| + r''|\boldsymbol{u}''| - \sum_{k=1}^{j-1} s_j, \\
|\boldsymbol{y}_{j^-}| &= r'|\boldsymbol{y}'| + r''|\boldsymbol{y}''| - \sum_{k=1}^{j-1} s_j,
\end{aligned} \tag{15}$$

Thus, in Eq. (3), the row sizes of $\boldsymbol{C}_3$, $\boldsymbol{D}_{31}$, $\boldsymbol{D}_{32}$, $\boldsymbol{D}_{33}$ are $|\boldsymbol{y}_{j^+}|$. The column sizes of $\boldsymbol{B}_3$, $\boldsymbol{D}_{13}$, $\boldsymbol{D}_{23}$, $\boldsymbol{D}_{33}$ are $|\boldsymbol{u}_{j^+}|$.

The column sizes of $\boldsymbol{B}_1$, $\boldsymbol{D}_{11}$, $\boldsymbol{D}_{21}$ and $\boldsymbol{D}_{31}$, in addition to the row sizes of $\boldsymbol{C}_2$, $\boldsymbol{D}_{21}$, $\boldsymbol{D}_{22}$ and $\boldsymbol{D}_{23}$ are i_j.

Similarly, since $|\boldsymbol{y}_1| = |\boldsymbol{u}_2| = s_j - i_j$, the column sizes of $\boldsymbol{B}_2$, $\boldsymbol{D}_{12}$, $\boldsymbol{D}_{22}$ and $\boldsymbol{D}_{32}$ are equal to the row sizes of $\boldsymbol{C}_1$, $\boldsymbol{D}_{11}$,

TABLE II: Number of operations to determine blocks for computing the max-plus semantics after adding connection oi_j

Dependency of	Dependency on	Block	Number of max/plus
$\boldsymbol{x}(k+1)$	$\boldsymbol{x}(k)$	$\boldsymbol{A} \oplus \boldsymbol{B}_2 \otimes \boldsymbol{C}_1$	$(s_j - i_j)\lvert\boldsymbol{x}_{j^-}\rvert^2$
$\boldsymbol{x}(k+1)$	$\boldsymbol{x}'(k)$	$\boldsymbol{B}_1 \oplus \boldsymbol{B}_2 \otimes \boldsymbol{D}_{11}$	$i_j\lvert\boldsymbol{x}_{j^-}\rvert(s_j - i_j)$
$\boldsymbol{x}'(k+1)$	$\boldsymbol{x}(k)$	$\boldsymbol{C}_2 \oplus \boldsymbol{D}_{22} \otimes \boldsymbol{C}_1$	$i_j\lvert\boldsymbol{x}_{j^-}\rvert(s_j - i_j)$
$\boldsymbol{x}'(k+1)$	$\boldsymbol{x}'(k)$	$\boldsymbol{D}_{21} \oplus \boldsymbol{D}_{22} \otimes \boldsymbol{D}_{11}$	$(i_j)^2(s_j - i_j)$
$\boldsymbol{x}(k+1)$	$\boldsymbol{u}_3(k)$	$\boldsymbol{B}_3 \oplus \boldsymbol{B}_2 \otimes \boldsymbol{D}_{13}$	$\lvert\boldsymbol{x}_{j^-}\rvert\lvert\boldsymbol{u}_{j^+}\rvert(s_j - i_j)$
$\boldsymbol{x}'(k+1)$	$\boldsymbol{u}_3(k)$	$\boldsymbol{D}_{23} \oplus \boldsymbol{D}_{22} \otimes \boldsymbol{D}_{13}$	$i_j\lvert\boldsymbol{u}_{j^+}\rvert(s_j - i_j)$
$\boldsymbol{y}_3(k)$	$\boldsymbol{x}(k)$	$\boldsymbol{C}_3 \oplus \boldsymbol{D}_{32} \otimes \boldsymbol{C}_1$	$\lvert\boldsymbol{x}_{j^-}\rvert\lvert\boldsymbol{y}_{j^+}\rvert(s_j - i_j)$
$\boldsymbol{y}_3(k)$	$\boldsymbol{x}'(k)$	$\boldsymbol{D}_{31} \oplus \boldsymbol{D}_{32} \otimes \boldsymbol{D}_{11}$	$i_j\lvert\boldsymbol{y}_{j^+}\rvert(s_j - i_j)$
$\boldsymbol{y}_3(k)$	$\boldsymbol{u}_3(k)$	$\boldsymbol{D}_{33} \oplus \boldsymbol{D}_{32} \otimes \boldsymbol{D}_{13}$	$\lvert\boldsymbol{u}_{j^+}\rvert\lvert\boldsymbol{y}_{j^+}\rvert(s_j - i_j)$

$n_j = (s_j - i_j)\big((\lvert\boldsymbol{x}_{j^-}\rvert + i_j)^2 + (\lvert\boldsymbol{x}_{j^-}\rvert + i_j)(\lvert\boldsymbol{u}_{j^+}\rvert + \lvert\boldsymbol{y}_{j^+}\rvert) + \lvert\boldsymbol{u}_{j^+}\rvert\lvert\boldsymbol{y}_{j^+}\rvert\big)$

$n_j^{\otimes} = \frac{n_j}{s_j - i_j} \cdot \lceil \frac{s_j - i_j}{max(srcR(c_j), dstR(c_j))} \rceil$

$n_j^{\oplus} = \lceil \frac{s_j - i_j}{max(srcR(c_j), dstR(c_j))} \rceil \big(\frac{n_j}{s_j - i_j} + 3(\lvert\boldsymbol{x}_{j^-}\rvert + i_j + \lvert\boldsymbol{u}_{j^+}\rvert) \big)$

$\boldsymbol{D}_{12}$ and $\boldsymbol{D}_{13}$, namely $s_j - i_j$. Note that this assumes that $i_j < s_j$. When $i_j \geq s_j$, matrices for the SDFG obtained by adding oi_j are calculated by rearranging the already available matrices [12]. Therefore, no max and plus operations are needed in this case and the computational effort is estimated to be zero. In the remainder, we assume that $i_j < s_j$.

To determine the matrices of the max-plus semantics including oi_j, I/O composition calculates nine blocks as shown in column 3 of Tab. II. The number of max/plus operations of each block is shown in the right column. Let n_j be the sum of the entries of the last column of Tab. II as shown in the last row of the table. Note that for $\boldsymbol{M} \in \mathbb{R}_{-\infty}^{n \times p}$, $\boldsymbol{N} \in \mathbb{R}_{-\infty}^{n \times m}$ and $\boldsymbol{P} \in \mathbb{R}_{-\infty}^{m \times p}$, to compute $\boldsymbol{M} \oplus \boldsymbol{N} \otimes \boldsymbol{P}$, mnp max and mnp plus operations are required.

Optimization for repeated entries: The dependencies of reproduced initial tokens and output tokens in an iteration on all $dstR(u)$ tokens consumed from u are the same. Therefore, all $dstR(u)$ columns of $\boldsymbol{B}_2$, $\boldsymbol{D}_{22}$ and $\boldsymbol{D}_{32}$ are identical. Similarly, all $srcR(y)$ rows of $\boldsymbol{C}_1$, $\boldsymbol{D}_{11}$ and $\boldsymbol{D}_{13}$ are the same. Considering that entries of matrices are identical, the matrix multiplications in Tab. II can be performed with fewer max and plus operations. With this optimization, which we implemented for our method, the total numbers of max and plus operations for adding oi_j are captured by $n_j^{\oplus}$ and $n_j^{\otimes}$ as given in the last row of Tab. II.

Let $A''' = A' \cup A''$. Alg. 4 calculates the number of max and plus operations to compositionally compute $mpS(A''')$ from $mpS(A')$ and $mpS(A'')$. It also computes size vector $sZ(A''')$ (globally stored for future use). The inputs of the algorithm are SDFG $\mathcal{G}$ and a set of two partitions, $\{A', A''\}$. The number of max and plus operations are initialized with the values for computing rate synchronization in Lines 2 and 3 using Eq. (12). The size vector is initialized in Line 4 using Eq. (14). Line 5 determines cut-set $cS(\{A', A''\})$. The size vector and numbers of operations are updated for each channel in the cut set in Lines 7-20. For each c_j determining connection oi_j, first, Lines 7-10 calculates the rate on oi_j after synchronization, s_j. Line 11 assigns the number of initial tokens on oi_j.

Algorithm 4: Determine the number of operations to compute the max-plus semantics of a composition

Input: SDFG $\mathcal{G}$ and $\{A', A''\}$, set of two partitions
Output: number of max and plus operations
Global: $sZ(A''')$, where $A''' = A' \cup A''$

1 **Function** `calcNrOprSzComp`$(\mathcal{G}, \{A', A''\})$**:**
2 $\quad nMax \leftarrow nMrF(A', r') + nMrF(A'', r'')$
3 $\quad nPlus \leftarrow nPrF(A', r') + nPrF(A'', r'')$
4 $\quad sZ(A''') \leftarrow sZrS(sZ(A'), sZ(A''))$ *// Eq. (14)*
5 $\quad \bar{C} \leftarrow cS(\{A', A''\})$
6 $\quad$ **foreach** $c_j \in \bar{C}$ **do**
7 $\quad\quad$ **if** $srcA(c_j) \in A'$ **then**
8 $\quad\quad\quad s_j \leftarrow srcR(c_j)\frac{rP(srcA(c_j))}{gcd(\rho', \rho'')}$
9 $\quad\quad$ **else**
10 $\quad\quad\quad s_j \leftarrow dstR(c_j)\frac{rP(dstA(c_j))}{gcd(\rho', \rho'')}$
11 $\quad\quad i_j \leftarrow iT(c_j)$
12 $\quad\quad \lvert\boldsymbol{x}_{j^-}\rvert \leftarrow sZ_1(A''')$
13 $\quad\quad \lvert\boldsymbol{u}_{j^+}\rvert \leftarrow sZ_2(A''') - s_j$
14 $\quad\quad \lvert\boldsymbol{y}_{j^+}\rvert \leftarrow sZ_3(A''') - s_j$
15 $\quad\quad n_j^{\otimes} \leftarrow \lceil \frac{s_j - i_j}{max(srcR(c_j), dstR(c_j))} \rceil \big((\lvert\boldsymbol{x}_{j^-}\rvert + i_j)^2 + (\lvert\boldsymbol{x}_{j^-}\rvert + i_j)(\lvert\boldsymbol{u}_{j^+}\rvert + \lvert\boldsymbol{y}_{j^+}\rvert) + \lvert\boldsymbol{u}_{j^+}\rvert\lvert\boldsymbol{y}_{j^+}\rvert \big)$ *// Tab. II*
16 $\quad\quad nMax \leftarrow nMax + n_j^{\otimes} + 3\lceil \frac{s_j - i_j}{max(srcR(c_j), dstR(c_j))} \rceil (\lvert\boldsymbol{x}_{j^-}\rvert + i_j + \lvert\boldsymbol{u}_{j^+}\rvert)$
17 $\quad\quad nPlus \leftarrow nPlus + n_j^{\otimes}$
18 $\quad\quad sZ_1(A''') \leftarrow \lvert\boldsymbol{x}_{j^-}\rvert + i_j$
19 $\quad\quad sZ_2(A''') \leftarrow \lvert\boldsymbol{u}_{j^+}\rvert$
20 $\quad\quad sZ_3(A''') \leftarrow \lvert\boldsymbol{y}_{j^+}\rvert$
21 $\quad$ **return** $[nMax, nPlus]^T$

To calculate the number of max/plus operations for adding connection oi_j (i.e., n_j in the last row of Tab. II), $\lvert\boldsymbol{x}_{j^-}\rvert$, $\lvert\boldsymbol{u}_{j^+}\rvert$ and $\lvert\boldsymbol{y}_{j^+}\rvert$ are updated in Lines 12-14, where sZ_1, sZ_2 and sZ_3 refer to token-vector, input-vector and output-vector sizes of the size vector. The need for this update is explained before Eq. (15). Line 15 computes the number of plus operations for adding oi_j, $n_j^{\otimes}$. In Lines 16-17, the numbers of max and plus operations for compostionally computing $mpS(A''')$ from $mpS(A')$ and $mpS(A'')$ are updated. The size vector after adding oi_j is updated in Lines 18-20. The algorithm ends if all connections have been processed.