MODELING AND ANALYSIS OF WINDOWED SYNCHRONOUS ALGORITHMS

Joachim Keinert

Fraunhofer
Institute for Integrated Circuits IIS
Erlangen, Germany
ket@iis.fraunhofer.de

Christian Haubelt, Jürgen Teich

Hardware-Software-Co-Design
Department of Computer Science 12,
University of Erlangen-Nuremberg, Germany
{haubelt,teich}@cs.fau.de

ABSTRACT

Sliding window algorithms are fundamental parts of each image processing system. Especially those belonging to the class of static algorithms offer various possibilities for analysis and optimization. Only if this potential is exploited, a high level synthesis of such algorithms will lead to efficient implementations.

Such an analysis relies on an efficient representation by a well-defined model of computation. It must abstract important properties of sliding windows as for instance the relation between input and output data as well as the required buffer space. In this paper, a corresponding static model of computation for sliding window algorithms is elaborated, called *Windowed Synchronous Data Flow (WSDF)*. Its main focus lies on applications with two or more dimensions. Furthermore, the WSDF balance equation is derived allowing to verify bounded token accumulation during execution.

1. INTRODUCTION

Analysis and optimization of sliding window algorithms for efficient synthesis requires an expressive representation by a formal model of computation. However, currently well known data flow models, described in Section 2, can satisfy this requirement only with difficulties as will be shown in Section 3. Thus, in Section 4 of this paper, we will propose a new model called *WSDF* that allows for intuitive descriptions of sliding window algorithms by abstraction to common important properties. Supporting an arbitrary number of dimensions, it is particularly adapted for multi-dimensional image and signal processing systems. Section 5 establishes the WSDF balance equation in order to verify bounded token accumulation on a single edge. Due to space restrictions, proofs must be omitted and can be found in [1].

2. RELATED WORK

In the past, many different static data flow models of computation have been proposed, as for instance *Synchronous Data Flow (SDF)* [2], *Cyclo Static Data Flow (CSDF)* [3], *Multidimensional Synchronous Data Flow (MDSDF)* [4], *Fractional Rate Data Flow (FRDF)* [5] and *CV-SDF* [6].

Their graphs G consist of a set V of vertices $v_i \in V$ representing actors. The latter ones are connected by edges $e \in E \subseteq V \times V$. To each edge, a buffer is assigned storing arriving tokens. If an actor is executed or *fired*, a certain amount of tokens is removed from each input edge as well as appended to each output edge. An actor *can be fired*, if on each input edge enough tokens are available to be consumed.

In SDF, the number of tokens produced and consumed by an actor v_i is the same for all invocations whereas they can vary cyclically with a fixed period in CSDF graphs. In MDSDF, token consumption and production is constant, however tokens form a rectangular multi-dimensional structure consisting of *data elements*. An actor can only be fired, if in each dimension there are enough data elements for one consumed array. FRDF introduces the concept of *fractional tokens* that are composed of several data elements. A single actor invocation can produce or consume only parts of them.

Temporal execution of a data flow graph G is controlled by a *schedule* being a sequence $S = [v_1, v_2, \ldots]$ whose elements represent invocations of the graph actors $v_i \in V$. Each repetition of the schedule is called a *schedule period*. A finite schedule which invokes each actor of the graph G at least one time and which does not cause a net graph state change is a so-called *periodic schedule*. The exact meaning of the graph net state depends on the underlying model of computation. In SDF, the state corresponds to the number of tokens stored in the edge buffers, whereas for CSDF additionally the actor's position in the firing sequence is taken into account. The number of actor invocations during a single period of a periodic schedule can be calculated by a so-called *balance equation*. Its exact form depends on the underlying data flow model and can be found in [2, 3, 4, 5].

3. MODELING OF SLIDING WINDOW ALGORITHMS

Image processing algorithms can be classified among others by their spatial locality. In this context, we distinguish *point*, *global* and *local* algorithms. The latter ones are important parts of almost every image processing system and are based on *sliding* windows which might overlap or not.

Figure 1 illustrates the principle of sliding window algorithms: Beginning at the upper left border (1), the window moves by $\Delta c_{1,1}$ pixels to the right until arriving at the border (2). Then, the window jumps back to the left image border and moves down by $\Delta c_{2,2}$ pixels. This process continues until reaching the lower right corner of the image (3). For each window position, one or more pixels of the output image are generated. In some cases, the algorithms require a border extension as shown in Figure 1 for correct results.

In the following, we suppose a given stream-based image processing system. The image to process is generated by a source actor and passed to one or more succeeding actors performing sliding window operations.

3.1. Modeling by SDF

Sliding window operations as shown in Figure 1 are cyclo-static by their nature. Supposing $\Delta c_{1,1} = \Delta c_{2,2} = 1$, a sink actor cannot

start firing until several input pixels are available. Once however started, an output value can be generated with each new input pixel. Hence, an SDF model on pixel level is not possible, but an actor has to produce or consume complete images. This, however, leads to very inefficient buffer allocation which especially in hardware implementations cannot be tolerated.

CV-SDF extends the SDF-model by splitting images into slices representing lines or blocks for more buffer efficient implementations. In order to support sliding window algorithms, it allows for multiple lecture of slices and frames. As however neither the size of the frames nor of the slices is specified, this reduces the possibilities for analysis. Furthermore, CV-SDF restricts to a one-dimensional representation of sliding window algorithms, and in general slices are coarser than pixels.

3.2. Modeling by CSDF

Figure 2 shows a CSDF model for a sliding window algorithm. It consists of three phases. During the *preparation* phase γ, no output pixel can be generated, because there are not enough input pixels available. If the *action* phase α is reached, a resulting output value can be calculated for each incoming pixel. The *termination* phase β finally assures, that the input and the output images have the same size, by generating output values without needing any further input.

In comparison with SDF, the CSDF solution is much more economic concerning memory resources, because it is not necessary to store complete images. However, the CSDF model in Figure 2 does not allow to derive the necessary amount of buffer memory for storage of intermediate lines. Furthermore, it does not show all degrees of parallelism as dependencies between the input and output pixels cannot be extracted due to hidden buffer memory. Moreover, CSDF models are not very general. In Figure 2 for instance we suppose, that the input image is generated line by line. Other production patterns would require other models.

3.3. Modeling by MDSDF

Many of the problems identified in Section 3.2 are due to the representation of a multi-dimensional problem by a uni-dimensional model of computation. Thus, MDSDF would significantly improve the situation, however it is not capable to represent overlapping, sliding windows.

4. WINDOWED SYNCHRONOUS DATA FLOW (WSDF)

In the following, a novel model of computation for sliding window algorithms called *Windowed Synchronous Data Flow (WSDF)* is elaborated. Due to limited space, proofs are omitted and can be found in [1], where extensions for models with windows of more than two dimensions are explained as well.

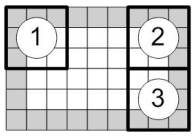


Fig. 1. Illustration of a sliding window. Each square corresponds to an image pixel. The extended border of the image frame is drawn gray hatched. 3×3 pixel windows are marked by a thick frame each.

Definition 4.1 *WSDF graph*
A WSDF graph *is a tuple*

$$G = \left(V, E, \overrightarrow{p}, \overrightarrow{v}, \overrightarrow{c}, \Delta c, \overrightarrow{u}, \overrightarrow{d}, \overrightarrow{b^s}, \overrightarrow{b^t}\right)$$

V *is the set of* vertices *representing* actors. $E \subseteq V \times V$ *is the set of* edges *connecting the actors and transporting data elements in form of tokens with dimension* n. *The source of an edge is denoted by* $src\,(e)$, *the sink by* $snk\,(e)$.

The token production in a WSDF graph is specified by the functions $\overrightarrow{p}\,(e)$ *and* $\overrightarrow{v}\,(e)$ *(see Section 4.2), the token consumption is described by the functions* $\overrightarrow{c}\,(e)$, $\Delta c\,(e)$ *and* $\overrightarrow{u}\,(e)$ *(see Section 4.3). The function* $\overrightarrow{d}\,(e)$ *defines initial tokens on an edge and is described in Section 4.4.* $\overrightarrow{b^s}\,(e)$ *and* $\overrightarrow{b^t}\,(e)$ *model border processing and are explained in Section 4.5.*

Figure 3 illustrates the WSDF notation by help of a graph with two actors *A1* and *A2* and a single edge.

4.1. WSDF Tokens

In WSDF, *tokens* are n-dimensional arrays of *data elements*. The model distinguishes between *effective* and *virtual tokens* as explained in the following paragraph. Their size is specified by a vector with n dimensions. For instance, having two-dimensional tokens, the first dimension of the vector specifies the number of columns, the second the number of rows.

4.2. WSDF Token Production

Definition 4.2 *WSDF Token Production*
Each time, a source actor $src\,(e)$ *is fired, it produces an* effective token *of constant size. The latter one is given by the function* $\overrightarrow{p} : E \rightarrow \mathbb{N}^n$. *Its abbreviated form is given by* $\overrightarrow{p_e} := \overrightarrow{p}\,(e)$.

Effective tokens are combined to so-called virtual tokens *of size* $\overrightarrow{v}\,(e) =: \overrightarrow{v_e}$, *with* $\overrightarrow{v} : E \rightarrow \mathbb{N}^n$.

Figure 4 shows an example of WSDF token production: Each time, the source actor is fired, it produces an array of data elements with two columns and one row. These tokens are combined to a virtual token with five columns and two lines.

Virtual tokens form a unit of data elements belonging together and model for instance images or blocks composed of several pixels. In principle, they could also be used to specify the pixels on which the sliding window operation shall take place. However, as some algorithms as for instance difference image calculation require combination of several input images or blocks, we additionally introduce the concept of virtual token unions as explained in Section 4.3 in order to simplify modeling.

The decomposition of virtual tokens into effective tokens corresponds to the principles introduced in FRDF [5] and leads to efficient buffer allocations.

Fig. 2. CSDF model of a sliding window operator. It is supposed, that both the input and the output image are generated pixel by pixel. The translational displacement of the window in both horizontal and vertical direction amounts one. α, β and γ depend on the window and input image extensions. Further details can be found in [1].

4.3. WSDF Token Consumption

In contrast to the models of computation discussed above, WSDF allows multiple reads of individual data elements by introducing sliding windows. They are typical for many local image processing transforms. Such algorithms are defined by the size of the window and the sampling pattern as well as by the set of data elements on which the sliding window operation takes place. In WSDF, this set is represented by the concept of virtual token unions.

Definition 4.3 *Virtual Token Union*
Virtual token unions *are defined on each edge and represent the set of data elements on which the sliding window operation takes place. They consist of one or more virtual tokens whose number for edge e is defined by the function $\overrightarrow{u} : E \to \mathbb{N}^n$, with $\overrightarrow{u_e} := \overrightarrow{u}(e)$.*

$\langle \overrightarrow{u}, \overrightarrow{e_i} \rangle$, the dot product of $\overrightarrow{u}$ and $\overrightarrow{e_i}$, represents the quantity of virtual tokens in dimension i composing the virtual token union. Hence, their overall number is given by $\prod_{i=1}^{n} \langle \overrightarrow{u}, \overrightarrow{e_i} \rangle$. Furthermore, a virtual token union may contain an extended border due to border processing as explained in Section 4.5.

Example 4.1 *Figure 4 illustrates a virtual token union consisting of $2 \cdot 1$ virtual tokens. Extended borders are not included, but will be explained in Section 4.5.*

Definition 4.4 *WSDF Token Consumption*
Each time, a sink actor $snk(e)$ is executed, it reads a multidimensional token of constant size. The size of the token corresponds to that of the window and is defined by the function $\overrightarrow{c} : E \to \mathbb{N}^n$, with $\overrightarrow{c_e} := \overrightarrow{c}(e)$.

An actor v is executable, if on each incoming edge enough data elements are stored, so that to each position of the window $\overrightarrow{c}(e)$ a corresponding data element can be assigned.

Inside a virtual token union, the window progression *is defined by the diagonal* sampling matrix *$\Delta c(e)$, $\langle \Delta c \cdot \overrightarrow{e_i}, \overrightarrow{e_i} \rangle$ defining the window translation in dimension i:*

$$\Delta c : E \to \mathbb{N}^{n \times n},\ e \mapsto \begin{bmatrix} c_{1,1}(e) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_{n,n}(e) \end{bmatrix} =: \Delta c_e$$

All window components must belong to the same virtual token union. If the sliding window leaves a virtual token union, the sampling process is restarted in the upper left corner of the next virtual token union.

The introduced token consumption and production rules correspond to those of MDSDF. However, a produced data element might be read several times. Hence, it can only be discarded from the edge buffer after the final read.

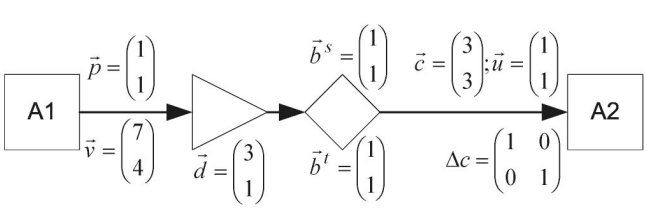


Fig. 3. WSDF graph with two actors and a single edge for illustration of the introduced notation

Example 4.2 *Figure 1 illustrates the window progression for the graph given in Figure 3. The sliding window moves as described in Section 3 corresponding to $\Delta c_{i,i} = \langle \Delta c \cdot \overrightarrow{e_i}, \overrightarrow{e_i} \rangle$, $i = 1, 2$. When the sliding window arrives at position 3, reading of a new virtual token union at position 1 is started.*

4.4. WSDF Delay Elements

The WSDF model uses the same interpretation of delay elements as the MDSDF model introduced in [4].

Definition 4.5 *WSDF Delay Element*
Each edge e of a WSDF graph G can have a positive, multidimensional delay $\overrightarrow{d} : E \to \mathbb{N}_0^n$, with $\overrightarrow{d_e} := \overrightarrow{d}(e)$. $\langle \overrightarrow{d}(e), \overrightarrow{e_i} \rangle$ defines the number of initial hyper planes orthogonal to $\overrightarrow{e_i}$.

Figure 5 illustrates the existence of initial data elements supposing $\overrightarrow{d} = \begin{pmatrix} 3 & 1 \end{pmatrix}^T$.

4.5. Modeling of Border Processing

The application of a sliding window algorithm without performing any border processing leads to output images which do not have the same size than the input images. In most cases, this is not desired. In order to remedy the situation, the input image is virtually extended by a border as shown in Figure 1.

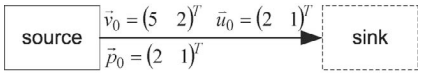
In WSDF, border processing is taken into account by enlarging the virtual token unions with an extended border. The latter one is described by two n-dimensional vectors $\overrightarrow{b^s}$ and $\overrightarrow{b^t}$.

$$\overrightarrow{b^{s,t}} : E \to \mathbb{Z}^n,\ e \mapsto \overrightarrow{b^{s,t}}(e) =: \overrightarrow{b_e^{s,t}}$$
$$\langle \overrightarrow{u}, \overrightarrow{e_i} \rangle \cdot \langle \overrightarrow{v}, \overrightarrow{e_i} \rangle + \langle \overrightarrow{b^s}, \overrightarrow{e_i} \rangle + \langle \overrightarrow{b^t}, \overrightarrow{e_i} \rangle > 0$$

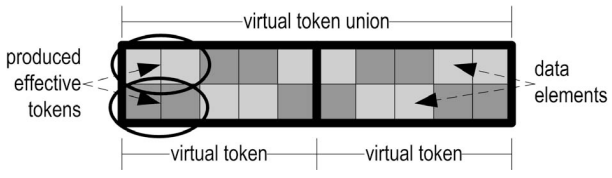
Their interpretation for positive values is shown in Figure 5. Negative values lead to suppression of data elements. Data elements belonging to the extended borders are supposed to be set to a constant value. Nevertheless, also symmetric border extension can be modeled by such an approach. Further details can be found [1].

5. WSDF BALANCE EQUATION

Implementation of static models of computation requires, that they are balanced, i.e. that the number of tokens accumulated on a single edge stays finite upon periodic activation. This property can be verified by the so-called balance equation. It assures, that within one



(a) WSDF graph fragment



(b) Combination of effective tokens to virtual tokens

Fig. 4. Example for WSDF token production showing the composition of a virtual token $\overrightarrow{v_0}$ by effective tokens $\overrightarrow{p_0}$

period of a periodic schedule, the number of produced and consumed tokens is identical.

This section shows that it is possible to introduce such a balance equation also for WSDF graphs. Due to restricted space, proofs must be omitted and can be found in [1].

Corollary 5.1 *Given a sink actor $snk(e)$ of edge e. Then, the number of invocations for one single input virtual token union in dimension i is given by*

$$\langle \overrightarrow{r_{vtu}}(e), \overrightarrow{e_i} \rangle = \frac{\langle \overrightarrow{u_e}, \overrightarrow{e_i} \rangle \cdot \langle \overrightarrow{v_e}, \overrightarrow{e_i} \rangle + \langle \overrightarrow{b_e^s} + \overrightarrow{b_e^t} - \overrightarrow{c_e}, \overrightarrow{e_i} \rangle}{\langle \Delta c_e \cdot \overrightarrow{e_i}, \overrightarrow{e_i} \rangle} + 1$$

Example 5.1 *Given the graph in Figure 3 belonging to Figure 1, the number of sink actor invocations per virtual token union can be calculated to*

$$\overrightarrow{r_{vtu}} = \begin{pmatrix} \frac{1 \cdot 7 + 2 - 3}{1} + 1 \\ \frac{1 \cdot 4 + 2 - 3}{1} + 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$$

Definition 5.1 *A WSDF graph is called* valid, *if and only if* $\forall e \in E, \forall 1 \leq i \leq n : \langle \overrightarrow{r_{vtu}}(e), \overrightarrow{e_i} \rangle \in \mathbb{N}$.

Theorem 5.2 *WSDF Graph Balance Equation*
Given a valid WSDF-Graph. Then the number of actor invocations in dimension i in order to return the WSDF graph into its initial state can be calculated by

$$\forall 1 \leq i \leq n : \overrightarrow{r_i} = \begin{bmatrix} L_i(v_1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & L_i(v_{|V|}) \end{bmatrix} \cdot \overrightarrow{q_i} \qquad (1)$$

$L_i(v)$ designates the minimal actor period and can be calculated by $L_i(v) := \langle \overrightarrow{L}(v), \overrightarrow{e_i} \rangle = scm_{e \in E,\, snk(e)=v}(\{\langle \overrightarrow{r_{vtu}}(e), \overrightarrow{e_i} \rangle\})$. $scm(A)$ is the smallest common multiple of all members of the set A. If $A = \emptyset$, we define $scm(A) = 1$.

Both $\overrightarrow{r_i}$ and $\overrightarrow{q_i}$ are strictly positive integer vectors. $\langle \overrightarrow{r_i}, \overrightarrow{e_j} \rangle$ is the number of invocations of actor v_j in dimension i.
$\overrightarrow{q_i}$ can be calculated by

$$\underbrace{\begin{bmatrix} \Gamma_{1,1,i} & \cdots & \Gamma_{1,v,i} & \cdots & \Gamma_{1,|V|,i} \\ \vdots & & \vdots & & \vdots \\ \Gamma_{e,1,i} & \cdots & \Gamma_{e,v,i} & \cdots & \Gamma_{e,|V|,i} \\ \vdots & & \vdots & & \vdots \\ \Gamma_{|E|,1,i} & \cdots & \Gamma_{|E|,v,i} & \cdots & \Gamma_{|E|,|V|,i} \end{bmatrix}}_{\Gamma_i} \cdot \overrightarrow{q_i} = \overrightarrow{0} \qquad (2)$$

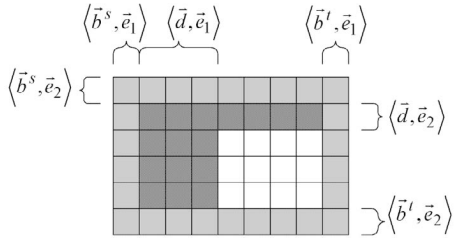


Fig. 5. Illustration of the border processing operator combined with initial data elements for the graph shown in Figure 3 and for a single virtual token union. It consists of an extended border frame hatched in light grey and one virtual token. The latter one includes one initial row and three columns.

with

$$\Gamma_{e,v,i} = \begin{cases} L_i(v) \cdot \langle \overrightarrow{p_e}, \overrightarrow{e_i} \rangle & \text{if } v = src(e) \\ 0 & \text{otherwise} \end{cases} - \begin{cases} \frac{L_i(v) \cdot \left(\langle \overrightarrow{v_e}, \overrightarrow{e_i} \rangle \cdot \langle \overrightarrow{u_e}, \overrightarrow{e_i} \rangle \right)}{\langle \overrightarrow{r_{vtu}}(e), \overrightarrow{e_i} \rangle} & \text{if } v = snk(e) \\ 0 & \text{otherwise} \end{cases}$$

Γ_i is called the *topology matrix of dimension i*. It has exactly the same consistency properties as for SDF graphs. Hence, we can derive from [2] that for a connected graph there exist strictly positive integer solutions $\overrightarrow{q_i}$ and $\overrightarrow{r_i}$, if and only if $rank(\Gamma_i) = |V| - 1$.

Example 5.2 *For the graph given in Figure 3, we obtain $\overrightarrow{L}(v_1) = \begin{pmatrix} scm(\emptyset) & scm(\emptyset) \end{pmatrix}^T = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, $\overrightarrow{L}(v_2) = \begin{pmatrix} 7 & 4 \end{pmatrix}^T$. This leads to $\Gamma_1 = \begin{bmatrix} 1 & -\frac{7 \cdot 7}{7} \end{bmatrix}$, $\Gamma_2 = \begin{bmatrix} 1 & -4 \end{bmatrix}$.*
As $rank(\Gamma_1) = rank(\Gamma_2) = 1 = |V| - 1$, Equation (2) can be solved: $\overrightarrow{q_1} = \begin{pmatrix} 7 & 1 \end{pmatrix}^T \cdot \alpha$, $\overrightarrow{q_2} = \begin{pmatrix} 4 & 1 \end{pmatrix}^T \cdot \beta$. The minimal repetition vectors are hence given by $\overrightarrow{r_1} = \begin{pmatrix} 7 & 7 \end{pmatrix}^T$, $\overrightarrow{r_2} = \begin{pmatrix} 4 & 4 \end{pmatrix}^T$. In other words, both actors have to fire 7 times in horizontal and 4 times in vertical direction.

6. CONCLUSION

In this paper, we developed a model of computation for sliding window algorithms. It has been shown, that application of existing data flow models for such applications leads to difficulties and incomplete descriptions. The new model WSDF abstracts important characteristics of sliding window algorithms and thus leads to more precise representations. As a first example for analysis, the WSDF balance equation has been introduced in order to guarantee bounded token accumulation on the graph edges.

In future work, we will develop an efficient analysis of required buffer space for WSDF. First results have already been elaborated, and will be presented in a future paper.

7. REFERENCES

[1] Joachim Keinert, Christian Haubelt, and Jürgen Teich, "Windowed Synchronous Data Flow (WSDF)," Tech. Rep. 02-2005, University of Erlangen-Nuremberg, Institut for Hardware-Software-Co-Design, 2005.

[2] Edward Ashford Lee and David G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, January 1987.

[3] Greet Bilsen, Marc Engels, Rudy Lauwereins, and Jean Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.

[4] Praveen K. Murthy and Edward A. Lee, "Multidimensional synchronous dataflow," *IEEE Transactions on Signal Processing*, vol. Vol50, no. 7, pp. 2064–2079, July 2002.

[5] Hyunok Oh and Soonhoi Ha, "Fractional rate dataflow model and efficient code synthesis for multimedia applications," *LCTES-SCOPES*, 2002.

[6] Dirk Stichling and Bernd Kleinjohann, "CV-SDF - a model for real-time computer vision applications," in *WACV 2002: IEEE Workshop on Applications of Computer Vision*, Orlando, FL, USA, December 2002.