The Synthesis of Control Signals for One-Dimensional Systolic Arrays

Jingling Xue

School of Electrical and Electronic Engineering, Nanyang Technological, University, Nanyang Avenue, Singapore 2263

Christian Lengauer

Fakultät für Mathematik und Informatik, Universität Passau, Postfach 25 40, W–8390 Passau, Germany

Abstract. This paper presents a method for the synthesis of control signals for one-dimensional systolic arrays from a program expressed as a set of uniform recurrence equations (*source UREs*). The basic idea underlying the synthesis of control signals is to distinguish different types of computation prescribed by the source UREs with another set of uniform recurrence equations (*control UREs*). To obtain one-dimensional systolic arrays with a description of both data and control signals, one simply applies the standard space-time mapping technique to the source and control UREs.

Keywords. Systolic arrays, control signals, space-time mapping, uniform recurrence equations, data dependence, hyperplane, half-space.

1 Introduction

The synthesis of systolic arrays from a program specification proceeds in two successive steps:

1. The specification is refined and transformed to a systolisable source, which is usually either a set of n-dimensional uniform recurrence equations (UREs) or a set of n nested loops with constant dependences [1, 2, 3].

2. The systolisable source is mapped to a systolic array by the standard space-time mapping technique [4, 5, 6].

The space-time mapping provides a description of the velocities and distribution of data, i.e., the flow of data in space and time. This description is sufficient for the fabrication of a VLSI chip if every cell in the systolic array performs the same computation at every time step. It is incomplete, however, if some cell performs different computations at different time steps. In this case, control signals are needed that instruct the cell when to perform what computation.

To realise a systolic array in hardware, we therefore need two types of information: *data flow* ensures that the right data arrive at the right cells at the right time steps, *control flow* ensures that the right computations are computed by the right cells at the right time steps.

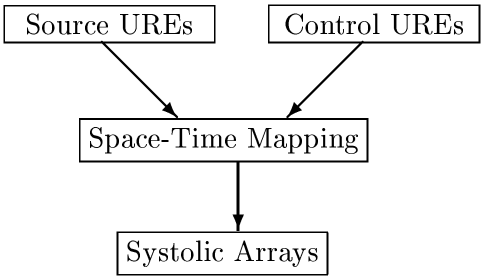

Figure 1. The synthesis of systolic arrays with a description of data and control flow.

This paper presents a method for the synthesis of control signals for one-dimensional systolic arrays. In practice, one-dimensional arrays have the following advantages [7]: 100% utilisation of non-faulty cells on a wafer, a constant I/O bandwidth that can be achieved by restricting external communication to the two border cells, and a clock rate that is independent of the size of the array. We shall present our method based on a source program that is in the form of UREs. We refer to these initial UREs as the *source UREs*. The basic idea underlying the synthesis of control signals is to distinguish different types of computation prescribed by the source UREs with another set of UREs called the *control UREs*. Then, a search for different systolic arrays with a description of both data and control flow reduces to a search for different space-time mappings. Fig. 1 depicts this design process. Our paper focuses on the construction of the control UREs.

The rest of the paper is organised as follows. Sect. 2 introduces some notation. Sect. 3 provides the technical background for the paper. We follow the left path in Fig. 1 to describe the basic techniques for the synthesis of data flow for one-dimensional arrays from n-dimensional UREs. Sect. 4 characterises the computations performed in the systolic array. Essentially, we distinguish different types of computation; each type has its own set of control signals. Sects. 5 and 6 present the construction of the control UREs based on the classification of the computations in Sect. 4. Sect. 7 concludes the paper with a discussion of the advantages of our method and the remark that the control UREs constructed in Sects. 5 and 6 are also correct for any r-dimensional systolic array $(0<r<n)$ synthesised from a superclass of the class of uniform recurrence equations.

2 Notation

$\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ denote the set of integers, rationals and reals, respectively. Let X be $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$. X^n denotes the n-fold Cartesian product of X.

Let X be a set and $P=\{P_1,\dots,P_p\}$ where each P_i is a subset of X and $X=(\bigcup\ i : 0<i\leqslant p : P_i)$. Then the set P is called a *covering* of P, and its elements are said to *cover* X. If, in addition, the elements of P are mutually disjoint, then P is called a *partition* of X, and its elements are called the P*-blocks* or *blocks* of the partition.

Let X be a set and o be an element in $\mathbb{R}^n$. The *translation* of X by o is the set $\{o+x \mid x\in X\}$.

Let $a\in\mathbb{R}^n\setminus\{0\}$, $x\in\mathbb{R}^n$, and $b\in\mathbb{R}$. The set $\{x \mid ax=b\}$ is called a *hyperplane* in $\mathbb{R}^n$. The

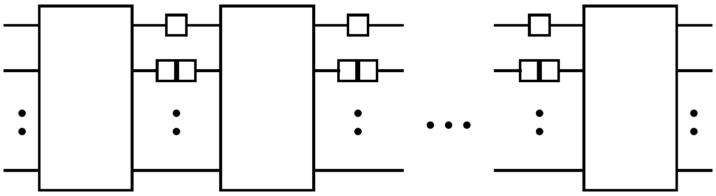

Figure 2. The one-dimensional systolic array model. The large solid boxes represent cells. The small boxes represent delay buffers. The lines represent connecting channels.

sets $\{x \mid ax \leqslant b\}$ and $\{x \mid ax \geqslant b\}$ are called the *closed half-spaces*, and the sets $\{x \mid ax < b\}$ and $\{x \mid ax < b\}$ are called the *open half-spaces* associated with the hyperplane.

Let $H = \{x \mid ax = b\}$ be a hyperplane. H is a *supporting hyperplane* to a set X if $H \cap X \neq \varnothing$ and either $(\forall\ x : x \in X : ax \leqslant b)$ or $(\forall\ x : x \in X : ax \geqslant b)$. For two non-empty disjoint sets A and B, H is a *separating hyperplane* that separates A and B if A is contained in one closed half-space and B is contained in the opposite closed half-space associated with H.

Let C be a convex polyhedron in $\mathbb{R}^n$. A subset F of C is called a *face* of C if either $F = \varnothing$ or $F = C$ or if there exists a supporting hyperplane H to C such that $F = C \cap H$. If the dimension of F is k, then F is called a k-face of C. It is customary to refer to the 0-faces of C as *vertices*, the 1-faces as *edges* and the $(n-1)$-faces as *facets*.

See [8] for further details of convex analysis and the theory of polyhedra.

The expression $f : D \rightarrowtail R$ denotes that function f is injective from domain D to range R. Let $f : X \rightarrow \{a, b\}$ be a function. If $S \subseteq X$ and $f(S) \cap f(X \setminus S) = \varnothing$, then f is called the *characteristic function* of S in X.

For $x, y \in \mathbb{Z}$, $x \mid y$ denotes that x divides y, i.e., that $y \mathbin{\textbf{mod}} x = 0$.

3 The Synthesis of Data Flow

This section is concerned with the synthesis of data flow for one-dimensional arrays. For details, we refer to [9, 10, 11].

A *one-dimensional systolic array* consists of a finite sequence of cells with the following properties (Fig. 2):

1. The execution of the array is synchronous, i.e., is governed by a global clock that ticks in unit time. Each cell is active at every time step.

2. Only the two border cells are connected to the host.

3. Only neighbouring cells are connected directly with each other.

4. A buffer retains a value for one time step. A *link* is a line of channels all of which have the same number of delay buffers. (That is, data values move along a fixed link with a constant velocity.)

A URE defines recursively the value of a variable at one point in terms of the values of others variables at other points. For details about properties of UREs, see [12]. Here, a variable V in the source UREs is of the following form:

$$V(I) \;=\; \begin{cases} I \in D_1 & \twoheadrightarrow \quad f_1(W_1(I-\vartheta_1),\ldots) \\ & \vdots \\ I \in D_p & \twoheadrightarrow \quad f_p(W_p(I-\vartheta_p),\ldots) \end{cases}$$

D_i is a convex polytope, i.e., a bounded convex polyhedron [8]; it is called the *domain of an equation*; $D_i \subset \mathbb{Z}^n$. The expression $I \in D_i$ is called a *domain predicate*. I is called an *index vector* (or *point*). Domains D_i and D_j with distinct i and j must be disjoint. This ensures that the source UREs are well-formed. V and W are variables belonging to a finite set $\mathcal{V}$. The three dots stand for an arbitrary but fixed number of similar arguments. ϑ_i is called a *data dependence vector*; $\vartheta_i \in \mathbb{Z}^n$. Φ denotes the *index space*, the union of all domains of equations. Φ must be a convex polytope.

$\mathcal{D}$ denotes the *data dependence matrix*; its columns represent the data dependence vectors [5]. When we write $\vartheta \in \mathcal{D}$, we mean that ϑ is a column of $\mathcal{D}$, i.e., a data dependence vector. For notational convenience, we assume that, for each variable name V, there is only one associated data dependence vector, denoted ϑ_V. A *data dependence graph* is a graph that has one node for each point of the index space and a directed arc from one node to another node iff a variable indexed by the former node is an argument in the equation for a variable indexed by the latter node. (Informally, the directions of arcs indicate the directions of data flow.)

Let us present a URE specification for LU-decomposition. This example will be used for illustration throughout the paper.

Example 1 (A URE specification for LU-Decomposition)

LU-decomposition is the unique decomposition of a non-singular $m \times m$ matrix A into a lower-triangular matrix L and an upper-triangular matrix U such that $A = LU$, i.e.,

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ & & \vdots & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \ell_{2,1} & 1 & & \\ & \vdots & & \\ \ell_{m,1} & \ell_{m,2} & \cdots & 1 \end{bmatrix} \times \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,m} \\ & u_{2,2} & \cdots & u_{2,m} \\ & & \vdots & \\ & & & u_{m,m} \end{bmatrix}$$

The elements of the upper triangle of L and the elements of the lower triangle of U (excluding the diagonal) are 0; the diagonal elements of L are 1.

The following UREs are adapted from the specification of LU-decomposition in the form of nested loops [4]; the corresponding data dependence graph is depicted in Fig. 3:

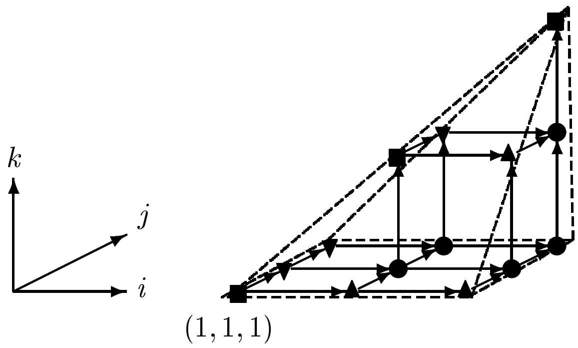

Figure 3. The data dependence graph for LU-decomposition ($m=3$).

The Source UREs:

$$L(i,j,k) \;=\; \begin{cases} k<i\leqslant m \wedge k=j \wedge 0<k\leqslant m & \twoheadrightarrow\; A(i,j,k-1)U(i-1,j,k) \quad \blacktriangle \\ k<i\leqslant m \wedge k<j\leqslant m \wedge 0<k\leqslant m & \twoheadrightarrow\; L(i,j-1,k) \quad \bullet \end{cases}$$

$$U(i,j,k) \;=\; \begin{cases} k=i \wedge k<j\leqslant m \wedge 0<k\leqslant m & \twoheadrightarrow\; A(i,j,k-1) \quad \blacktriangledown \\ k=i \wedge k=j \wedge 0<k\leqslant m & \twoheadrightarrow\; A(i,j,k-1)^{-1} \quad \blacksquare \\ k<i\leqslant m \wedge k\leqslant j\leqslant m \wedge 0<k\leqslant m & \twoheadrightarrow\; U(i-1,j,k) \quad \bullet\ \blacktriangle \end{cases}$$

$$A(i,j,k) \;=\; \begin{cases} k\leqslant i\leqslant m \wedge k\leqslant j\leqslant m \wedge 0<k\leqslant m & \twoheadrightarrow\; A(i,j,k-1) \quad \blacksquare\ \bullet\ \blacktriangle\ \blacktriangledown \\ & \; -L(i,j-1,k)U(i-1,j,k) \end{cases}$$

When we label an equation by a number of graphical symbols (e.g., $\bullet$), we mean that the equation is only defined for the points (i.e., nodes) of the corresponding data dependence graph depicted by those symbols. A variable may not be defined at every point of the index space. For example, variable L is not defined at the points depicted by $\blacksquare$ and $\blacktriangledown$. (This corresponds to the fact that the diagonal elements of L are 1 and need not be computed.) We give L the undefined value $\perp$ at these points. This amounts to adding the following equation to the source UREs:

$$L(i,j,k) \;=\; \{\; k=i \wedge k\leqslant j\leqslant m \wedge 0<k\leqslant m \;\twoheadrightarrow\; \perp \qquad \blacksquare\ \blacktriangledown$$

The index space is a convex polytope:

$$\Phi = \{(i,j,k) \mid k\leqslant i\leqslant m \wedge k\leqslant j\leqslant m \wedge 0<k\leqslant m\}$$

There are three data dependence vectors, ϑ_L, ϑ_U and ϑ_A; they are associated with variables L, U and A, respectively. The data dependence matrix is given by

$$\mathcal{D} \;=\; [\vartheta_L\ \vartheta_U\ \vartheta_A] \;=\; \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The elements of variables L, U and A flow along axes j, i and k, respectively.

Variables that are referenced but undefined represent input data. The initialisation of variable A with the input data $a_{i,j}$ takes place at the set of points that is the translation of the bottom facet of the index space by $-\vartheta_A$:

$$(\forall\ i,j,k : 0<i\leqslant m \wedge 0<j\leqslant m \wedge 0=k : A(i,j,k) \;=\; a_{i,j})$$

Variables that represent output data must be well-defined. The output data $\ell_{i,k}$ of variable L are available at the back facet and the output data $u_{k,j}$ of variable U at the right facet:

$$\begin{array}{rcl}(\forall\ i,j,k : k<i\leqslant m \wedge m=j \wedge 0<k\leqslant m : \ell_{i,k} & = & L(i,j,k))\\ (\forall\ i,j,k : m=i \wedge k\leqslant j\leqslant m \wedge 0<k\leqslant m : u_{k,j} & = & U(i,j,k))\end{array}$$

By convention, the input data must be supplied, and the output data must be defined, at the boundary (i.e., facets) of the index space. □

To map the source UREs to a systolic array, one needs to find a space-time mapping that assigns every computation (i.e., point) in the source UREs a time step and an array cell. A space-time mapping is a linear (or affine) transformation of the index space. The advantage of linear index transformations is that they enforce easily the linearity of variables' velocities and the regularity of channel connections. A space-time mapping, π, that describes a one-dimensional array is a linear transformation from $\mathbb{Z}^n$ to $\mathbb{Z}^2$:

$$\pi = \begin{bmatrix}\lambda\\ \sigma\end{bmatrix} = \begin{bmatrix}\lambda_1 & \lambda_2 & \dots & \lambda_n\\ \sigma_1 & \sigma_2 & \dots & \sigma_n\end{bmatrix}$$

where

- λ is called the *scheduling vector*. It specifies the temporal distribution. Point $I\in\Phi$ is computed at time step λI. t_{fst} and t_{lst} are the first and last step number. $\mathcal{T}=\{t \mid t_{\mathrm{fst}}\leqslant t\leqslant t_{\mathrm{lst}}\}$ is the *scheduling space*.

- σ is called the *allocation vector*. It specifies the spatial distribution. Point $I\in\Phi$ is computed at cell σI. $p_{\min}$ and $p_{\max}$ are the coordinates of the two border cells. $\mathcal{P}=\{p \mid p_{\min}\leqslant p\leqslant p_{\max}\}$ is the *processor space*.

Each variable is allocated one link. That is, all elements of the same variable move along the same dedicated link. Useful functions defined in terms of the space-time mapping are flow, incell, outcell, intime and outtime.

- $\mathsf{flow} : \mathcal{V}\to\mathbb{Q}^r$, $\mathsf{flow}(V) = \sigma\vartheta_V/\lambda\vartheta_V$, $\vartheta_V\in\mathcal{D}$. flow specifies the velocity with which elements of a variable travel at each step. Variable V is called *moving* if $\mathsf{flow}(V)\neq 0$ and *stationary* if $\mathsf{flow}(V)=0$. By convention, variable V moves to the right if $\mathsf{flow}(V)>0$, to the left if $\mathsf{flow}(V)<0$. The number of delay buffers associated with a channel for a moving variable V is $1/\mathsf{flow}(V)-1$, which simplifies to $|\lambda\vartheta_V/\sigma\vartheta_V|-1$. (This interpretation assumes that evaluation of a point takes unit time.)

- $\mathsf{incell} : \mathcal{V}\to\Phi\to\mathbb{Z}$, $\mathsf{incell}(V(I)) = \mathbf{if}\ \mathsf{flow}(V)>0\to p_{\min}\ [\!]\ \mathsf{flow}(V)<0\to p_{\max}\ \mathbf{fi}$. incell specifies the border cells at which the input data are injected.

- $\mathsf{outcell} : \mathcal{V}\to\Phi\to\mathbb{Z}$, $\mathsf{outcell}(V(I)) = \mathbf{if}\ \mathsf{flow}(V)>0\to p_{\max}\ [\!]\ \mathsf{flow}(V)<0\to p_{\min}\ \mathbf{fi}$. outcell specifies the border cells at which the output data are ejected.

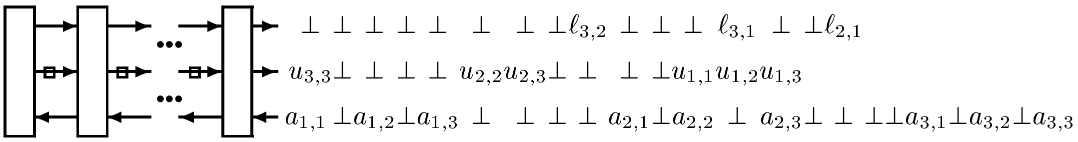

Figure 4. A one-dimensional array for LU-decomposition with the distribution of the input data of A at the first step and of the output data of L and U at the last step.

- $\mathsf{intime} : \mathcal{V} \to \Phi \to \mathbb{Z}$, $\mathsf{intime}(V(I)) = \lambda I - (\sigma I - \mathsf{incell}(V(I)))/\mathsf{flow}(V)$. intime specifies the steps at which input data are injected into the array.
- $\mathsf{outtime} : \mathcal{V} \to \Phi \to \mathbb{Z}$, $\mathsf{outtime}(V(I)) = \lambda I - (\sigma I - \mathsf{outcell}(V(I)))/\mathsf{flow}(V)$. $\mathsf{outtime}$ specifies the steps at which output data are ejected from the array.

A space-time mapping determines four parameters t_{fst}, t_{lst}, $p_{\min}$ and $p_{\max}$:

$$\begin{aligned}
t_{\text{fst}} &= (\min\ V, I : V \in \mathcal{V} \wedge I \in \Phi : \mathsf{intime}(V(I))) \\
t_{\text{lst}} &= (\max\ V, I : V \in \mathcal{V} \wedge I \in \Phi : \mathsf{outtime}(V(I))) \\
p_{\min} &= (\min\ I : I \in \Phi : \sigma I) \\
p_{\max} &= (\max\ I : I \in \Phi : \sigma I)
\end{aligned}$$

Example 2 (A one-dimensional array for LU-decomposition)

The following space-time mapping describes the array depicted in Fig. 4:

$$\pi_{\text{even}} = \begin{bmatrix} 2m-2 & 1 & m/2 \\ m-1 & 1 & -m/2 \end{bmatrix} \qquad \pi_{\text{odd}} = \begin{bmatrix} 2m & 1 & (m+1)/2 \\ m & 1 & -(m+1)/2 \end{bmatrix} \tag{1}$$

The data flows are 1 for L, 2 for U and -1 for A. The channels for L and A have no buffers, the channels for U have one buffer each. The input data of A and the output data of L and U are through the border cell on the right.

A simple calculation yields $t_{\text{fst}} = 1$, $t_{\text{lst}} = 35$, $p_{\min} = 2$ and $p_{\max} = 10$. Therefore, the array consists of 9 cells and runs in 35 time steps. $\square$

Not every space-time mapping describes a one-dimensional array. A space-time mapping is *valid* for the source UREs in the one-dimensional array model iff the following four mapping constraints are satisfied:

Precedence Constraint: The step at which a target variable is computed is later than the steps at which its arguments are computed. (This ensures that the dependence relations prescribed in the source UREs are preserved.)

Computation Constraint: Concurrent computations are mapped to different cells.

Communication Constraint: At most one value per step is injected at each link.

Delay Constraint: The number of buffers associated with a channel is a non-negative integer.

In order to generate valid space-time mappings for the source UREs systematically, we must find conditions that ensure the validity of a space-time mapping. This calls for the following notation. Pick any data dependence vector, say, ϑ_V. Consider all lines that are parallel to ϑ_V and that intersect the index space Φ. We call the segments of these lines contained in the index space the *ϑ_V-paths* (or *paths*). Each ϑ_V-path intersects the index space at one or two points. In the former case, the point is called both a *first* and a *last point* of V. In the latter case, there is a dependence in terms of ϑ_V between the two points. The dependent point of the two is called a *last point* of V, and the point on which it depends is called a *first point* of V. We denote the set of first points of V by $\text{first}(\Phi, \vartheta_V)$ and the set of last points of V by $\text{last}(\Phi, \vartheta_V)$:

$$\begin{aligned} \text{first}(\Phi, \vartheta_V) &= \{I \mid I \in \Phi \wedge I - \vartheta_V \notin \Phi\} \\ \text{last}(\Phi, \vartheta_V) &= \{I \mid I \in \Phi \wedge I + \vartheta_V \notin \Phi\} \end{aligned}$$

Example 3 (An illustration of first and last points for LU-decomposition)

Consider Fig. 3. The elements of L flow in the direction of the j axis: $\text{first}(\Phi, \vartheta_L)$ is the front facet and $\text{last}(\Phi, \vartheta_L)$ is the back facet. The elements of U flow in the direction of the i axis: $\text{first}(\Phi, \vartheta_U)$ is the left facet and $\text{last}(\Phi, \vartheta_U)$ is the right facet. The elements of A flow in the direction of the k axis: $\text{first}(\Phi, \vartheta_A)$ is the bottom facet and $\text{last}(\Phi, \vartheta_A)$ is the union of the left facet and front facet.

The point on the top, depicted by ■, is both a first and last point of L and U. The points depicted by ■, ▲ and ▼ in the bottom facet are both first and last points of A.

As mentioned previously in this section, the input data must be supplied, and the output data must be defined, at the boundary of the index space. Phrased in terms of the notion of first and last points, the input data must be supplied at the first points and the output data must be defined at the last points. □

Theorem 1 *A space-time mapping, π, is valid iff*

1. $(\forall\, V : V \in \mathcal{V} : \lambda\vartheta_V \geqslant 1)$ (*Precedence Constraint*)
2. $\pi : \Phi \rightarrowtail \mathbb{Z}^2$ (*Computation Constraint*)
3. $(\forall\, V : V \in \mathcal{V} : (\mathsf{intime} : \text{first}(\Phi, \vartheta_V) \rightarrowtail \mathbb{Z}))$ (*Communication Constraint*)
4. $(\forall\, V : V \in \mathcal{V} : \mathsf{flow}(V) \neq 0 \implies \sigma\vartheta_V | \lambda\vartheta_V)$ (*Delay Constraint*)

Proof. We consider the four constraints in turn. (1) If $\lambda\vartheta_V \geqslant 1$, then the step at which target variable $V(I)$ is computed is by at least one larger than the steps at which its arguments $W(I - \vartheta_V)$ are computed. (This interpretation assumes that evaluation of a point takes unit time.) Hence, the precedence constraint is satisfied. (2) If $\pi : \Phi \rightarrowtail \mathbb{Z}^2$, then π is injective over Φ. The points that are mapped to the same step must be mapped to different cells. Hence, the computation constraint is satisfied. (3) If $\mathsf{intime} : \text{first}(\Phi, \vartheta_V) \rightarrowtail \mathbb{Z}$, then intime is injective over $\text{first}(\Phi, \vartheta_V)$. $\text{first}(\Phi, \vartheta_V)$ is the set of index vectors of the variables that represent the input data of V. This means that all the input data of V are injected at different time steps. Hence, the communication constraint is satisfied. (4) $|\lambda\vartheta_V/\sigma\vartheta_V|-1$ represents the number of delay buffers associated with a channel for a moving variable V. It is a non-negative integer if $\sigma\vartheta_V|\lambda\vartheta_V$. Hence, the delay constraint is satisfied. $\square$

4 The Characterisation of Computations

It is convenient to depict a one-dimensional array by a two-dimensional space-time diagram, where the temporal dimension is horizontal and the spatial dimension vertical. A *space-time diagram*, Υ, is the product of the scheduling space $\mathcal{T}$ and the processor space $\mathcal{P}$:

$$\Upsilon \;=\; \mathcal{T}\times\mathcal{P} \;=\; \{(t,p) \mid t_{\text{fst}}\leqslant t\leqslant t_{\text{lst}} \;\wedge\; p_{\min}\leqslant p\leqslant p_{\max}\}$$

The points in the space-time diagram are called the *space-time points*. The space-time points on the top and bottom row represent the two border cells. The data and control signals that are input and output at a point on the top (bottom) row are depicted directly above (below) that point. The space-time points that are the images of the points in the index space are depicted by the corresponding graphic symbols. The rest space-time points are depicted by small dots.

We shall denote the image of some x under a given space-time mapping with an overbar: $\overline{x}$. Here, x is a point, or a set of points or a data dependence vector. For a point I in $\mathbb{Z}^n$, $\overline{I}=\pi I=(\lambda I,\sigma I)=(t,p)$. For a set S in $\mathbb{Z}^n$, $\overline{S}=\pi S=\{\overline{I}\mid I\in S\}$. The image $\overline{\vartheta}_V$ of data dependence vector ϑ_V is given by

$$\overline{\vartheta}_V \;=\; \pi\vartheta_V \;=\; \begin{bmatrix}\lambda\vartheta_V\\ \sigma\vartheta_V\end{bmatrix}$$

When we speak of a $\overline{\vartheta}_V$-path, we mean a $\overline{\vartheta}_V$-path in the space-time diagram. For a fixed variable V, we refer to a $\overline{\vartheta}_V$-path that contains the images of at least one point of the index space as a *computation $\overline{\vartheta}_V$-path* (of V). We refer to a $\overline{\vartheta}_V$-path that is not a computation $\overline{\vartheta}_V$-path as a *non-computation $\overline{\vartheta}_V$-path* (of V). The elements of V are injected and ejected at computation $\overline{\vartheta}_V$-paths. The undefined value, $\perp$, is injected and ejected at non-computation $\overline{\vartheta}_V$-paths.

Example 4 (A space-time diagram for LU-decomposition)

Fig. 5 depicts the data flow for the array shown in Fig. 4 in a space-time diagram. Dotted lines depict all computation $\overline{\vartheta}_A$-paths. Small dashed lines depict all computation $\overline{\vartheta}_L$-paths excluding the three that contain the points depicted by ■; $\perp$ is ejected at the three paths. (Remember that the diagonal elements of L are 1 and their computations are not specified in the source UREs.) Long dashed lines depict all computation $\overline{\vartheta}_U$-paths. From the space-time diagram we can directly extract the following information: the latency, the number of cells required, the velocities of variables, and input and output characteristics.

The four mapping constraints of Thm. 1 can be interpreted in terms of the embedding of the data dependence graph in the space-time diagram. The precedence constraint ensures that the projection of a data dependence vector onto the time axis points in the direction of the time axis.

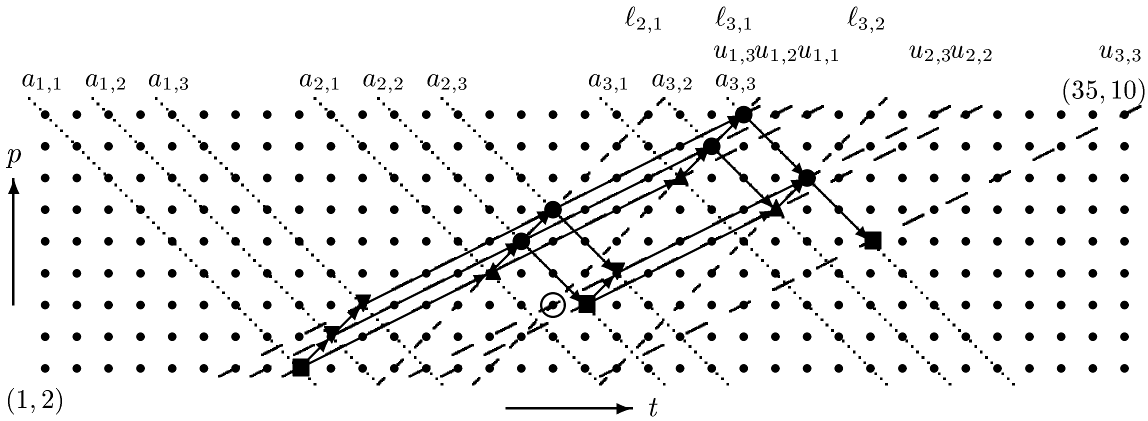

Figure 5. The data flow of the one-dimensional array of Fig. 4. The circle will be referred to later in this section.

The computation constraint ensures that no two points of the index space overlap in the space-time diagram. The communication constraint ensures that no two ϑ_V-paths of a fixed variable V overlap in the space-time diagram. The delay constraint ensures that the length of the projection of a data dependence vector onto the time axis is a positive multiple of the length of the projection of that vector onto the space axis. $\square$

For the purpose of specifying control signals for one-dimensional arrays, we classify the space-time points as follows:

- *Computation points* are the points that are the images of the points in the index space (e.g., the points depicted by ■, ●, ▲ and ▼ in Fig. 5). The computations at these points are specified by the source UREs. We further partition the computation points into different types according to the defining equations of variables at these points. A partition of the index space is called the Ⓣ*-partition* if two points of the index space are in the same Ⓣ-block iff the defining equation for every variable of the source UREs at one of the two points agrees with the defining equation of the same variable at the other point. (For a constructive definition of the Ⓣ-partition, see [13].) We say that two points are of the same *type* if they are in the same Ⓣ-block. Points that are not of the same type are said to have different types. Since a valid space-time mapping is injective over the index space (Thm. 1), we shall carry over type designations from the points in the index space to their images under the space-time mapping. This enables us to speak of the same or different type of computation points in the space-time diagram. For notational convenience, we refer to the images of the first (last) points of a variable under a given space-time mapping as the *first* (*last*) *computation points* of that variable.

  Consider Figs. 3 and 5. There are four different types of points; points of the same type are depicted by the same graphic symbol.

- *Pipelining points* are points that are not computation points (e.g., the small dots in Fig. 5).

The computations at these points follow from the meaning of the space-time mapping. The cell at a given pipelining point sends on to its output channels whatever it receives one time step earlier on its input channels. Pick an arbitrary but fixed moving variable V. We partition the pipelining points further into four categories according to the rôles that the cells of the array play in dealing with the elements of V at these points:

- The *soaking points of* V are the pipelining points in the computation $\overline{\vartheta}_V$-paths that precede the first computation points of V. The cells at these points propagate the input data of V from the respective border cell to internal cells.
- The *draining points of* V are the pipelining points in the computation $\overline{\vartheta}_V$-paths that succeed the last computation points of V. The cells at these points propagate the output data of V from internal cells to the respective border cell.
- The *relaying points of* V are the pipelining points in the computation $\overline{\vartheta}_V$-paths that succeed the first and precede the last computation points of V. The cells at these points relay the elements of V to the cells where these elements are used for computation. There are $\Gamma_V - 1$ relaying points between any two neighbouring computation points:

$$\Gamma_V \quad = \quad |\sigma\vartheta_V|$$

- The *undefined points of* V are the pipelining points in the non-computation $\overline{\vartheta}_V$-paths. The cells at these points propagate $\perp$ (for V) from one border cell to the other.

Consider variable A in Fig. 5. The undefined points of A are the small dots that are not on the dotted lines. E.g., take the computation $\overline{\vartheta}_A$-path at which element $a_{2,2}$ is injected. This path contains four soaking points, one relaying point and two draining points of A.

It should be emphasised that the pipelining points are classified with respect to a reference variable. Thus, a soaking point of one variable may be a draining point of another variable and an undefined point of a third variable. For example, the pipelining point highlighted by a circle in Fig. 5 is an undefined point of A and a soaking point of both L and U.

The specification of control signals is based on the previous classification of the space-time points. There are two types of control flow:

- *Computation control flow* distinguishes different types of computation points, (i.e., different Ⓣ-blocks). It governs the evaluation of the points in the index space.
- *Propagation control flow* distinguishes pipelining points from computation points. It governs the pipelining of input (output) data that are mapped to the internal cells of the array from (to) the border cells.

The control UREs specify the control flow for a one-dimensional array. Therefore, they can be viewed as consisting of two sets of UREs: the *computation control UREs* specify the computation control flow and the *propagation control UREs* specify the propagation control flow. In Sect. 5, we describe the specification of the computation control UREs, in Sect. 6 that of the propagation control UREs.

To conclude this section, we discuss the non-injectivity of space-time mappings for one-dimensional arrays. As will be explained in Sect. 6, this non-injectivity causes complications in the synthesis of the propagation control flow. E.g., consider the circle depicted in Fig. 5. There, the $\overline{\vartheta}_L$-path at which element $\ell_{3,1}$ is ejected intersects the $\overline{\vartheta}_U$-path at which element $u_{2,3}$ is ejected. However, the corresponding ϑ_L-path (i.e., the right edge of the bottom facet in Fig. 3) and ϑ_U-path (i.e., the middle segment of the back facet in Fig. 3) in the data dependence graph of Fig. 3 do not intersect. This problem is due to the fact that the corresponding space-time mapping is not injective from $\mathbb{Z}^n$ to $\mathbb{Z}^2$, although it is injective from Φ to $\mathbb{Z}^2$ (Thm. 1).

In the synthesis of $(n-1)$-dimensional arrays [4, 5, 6], a space-time mapping is a linear transformation from $\mathbb{Z}^n$ to $\mathbb{Z}^n$. Here, the non-injectivity problem does not arise because a valid space-time mapping is always chosen to be a non-singular $n \times n$ matrix and thus injective over $\mathbb{Z}^n$.

In [13], a method for the synthesis of the computation control flow was presented and extended for a synthesis of the propagation control flow. However, the propagation control flow specified this way can be very inefficient [14]. Sect. 6 presents a different and more efficient specification.

5 The Synthesis of Computation Control Flow

This section reviews a method for the synthesis of the computation control flow. For details, we refer to [13]. Here, we simply illustrate the method with the example of LU-decomposition.

The construction of the computation control UREs relies on the following theorem about the separation of two convex polytopes in $\mathbb{R}^n$ by hyperplanes [8].

Theorem 2 *If X and Y are non-empty disjoint convex polytopes in $\mathbb{R}^n$, then there is a hyperplane that separates them.*

To apply this theorem, each Ⓣ-block must be a convex polytope. In what follows, we assume that each Ⓣ-block is a convex polytope (if not, it must be decomposed into a set of convex polytopes [14]).

Consider the data dependence graph for LU-decomposition (Fig. 3). Φ_s, for $s \in \{\blacksquare, \bullet, \blacktriangle, \blacktriangledown\}$, denotes the set of all points of the graph that are depicted by graphic symbol s. The Ⓣ-partition of the index space is:

$$\text{Ⓣ-partition} \quad = \quad \{\Phi_{\blacksquare}, \Phi_{\bullet}, \Phi_{\blacktriangle}, \Phi_{\blacktriangledown}\}$$

All four Ⓣ-blocks are convex polytopes. For convenience, the source UREs for LU-decomposition in Sect. 3 are rewritten as follows:

The Source UREs:

$$
\begin{aligned}
L(i,j,k) &= \begin{cases} (i,j,k)\in\Phi_{\blacktriangle} \twoheadrightarrow A(i,j,k-1)U(i-1,j,k) & \blacktriangle \\ (i,j,k)\in\Phi_{\bullet} \twoheadrightarrow L(i,j-1,k) & \bullet \end{cases} \\
U(i,j,k) &= \begin{cases} (i,j,k)\in\Phi_{\blacktriangledown} \twoheadrightarrow A(i,j,k-1) & \blacktriangledown \\ (i,j,k)\in\Phi_{\blacksquare} \twoheadrightarrow A(i,j,k-1)^{-1} & \blacksquare \\ (i,j,k)\in\Phi_{\bullet} \vee (i,j,k)\in\Phi_{\blacktriangle} \twoheadrightarrow U(i-1,j,k) & \bullet\ \blacktriangle \end{cases} \\
A(i,j,k) &= \begin{cases} (i,j,k)\in\Phi_{\blacksquare} \vee (i,j,k)\in\Phi_{\bullet} \vee (i,j,k)\in\Phi_{\blacktriangle} \vee (i,j,k)\in\Phi_{\blacktriangledown} \\ \qquad \twoheadrightarrow A(i,j,k-1)-L(i,j-1,k)U(i-1,j,k) & \blacksquare\ \bullet\ \blacktriangle\ \blacktriangledown \end{cases}
\end{aligned}
$$

(By the definition of the Ⓣ-partition, the domain of every equation of the source UREs is a union of Ⓣ-blocks [14].)

The specification of computation control flow proceeds in two steps:

- Construct the computation control UREs.
- In the source UREs, replace the domain predicates in indices by predicates in computation control variables.

First, let us consider the construction of the computation control UREs. The underlying idea is to find a set of hyperplanes that mutually separate all Ⓣ-blocks. Then, we associate one distinct computation control variable with each of these separating hyperplanes. Each separating hyperplane divides the index space into two halves. The corresponding computation control variable takes on two control values: one value, say, **0** at one half and one value, say, **1** at the other half. Its associated control dependence vector is any normalised vector parallel to the separating hyperplane. The specification of a computation control variable in a URE is straightforward: the two control values are first used to initialise the variable at the first points of the variable and then pipelined across the index space along the direction of the associated control dependence vector.

Consider the data dependence graph for LU-decomposition (Fig. 3). The following two hyperplanes mutually separate the four Ⓣ-blocks:

$$
\begin{aligned}
H_1 &= \{(i,j,k) \mid i-k=0\} \\
H_2 &= \{(i,j,k) \mid j-k=0\}
\end{aligned}
$$

H_1 (H_2) is the hyperplane containing the left (front) facet of the index space. We write C_1 for the variable associated with H_1 and C_2 for the variable associated with H_2. We let C_1 be **0** at the points of the index space contained in the closed half-space $\{(i,j,k) \mid i-k \leqslant 0\}$, i.e., at the points of $\Phi_{\blacksquare} \cup \Phi_{\blacktriangledown}$, and **1** at the points of the index space contained in the open half-space $\{(i,j,k) \mid i-k > 0\}$, i.e., at the points of $\Phi_{\blacktriangle} \cup \Phi_{\bullet}$. We let C_2 be **0** at the points of the index space contained in the closed half-space $\{(i,j,k) \mid j-k \leqslant 0\}$, i.e., at the points of $\Phi_{\blacksquare} \cup \Phi_{\blacktriangle}$, and **1** at the points of the index space contained in the open half-space $\{(i,j,k) \mid j-k > 0\}$, i.e., at the points of $\Phi_{\blacktriangledown} \cup \Phi_{\bullet}$. The control dependence vector ϑ_{C_1} is any solution of $(1,0,-1)\vartheta_{C_1}=0$. The control dependence vector ϑ_{C_2} is any solution of $(0,1,-1)\vartheta_{C_2}=0$. For the purpose of illustration, we choose

$$
\begin{aligned}
\vartheta_{C_1} &= \vartheta_L = (0,1,0) \\
\vartheta_{C_2} &= \vartheta_U = (1,0,0)
\end{aligned}
$$

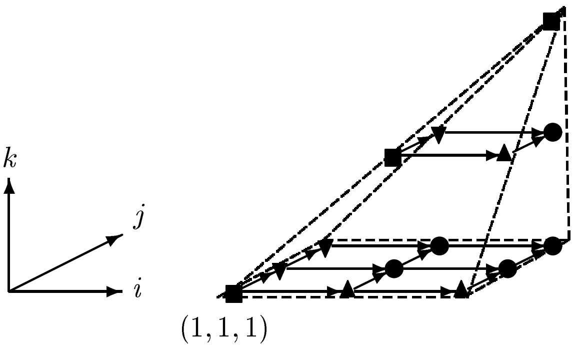

Figure 6. The control dependence graph of the computation control UREs for LU-decomposition ($m = 3$). ϑ_{C_1} points in the direction of the j axis. C_1 is initialised with $\mathbf{0}$ at $\Phi_{\blacksquare}$ and with $\mathbf{1}$ at $\Phi_{\blacktriangle}$. ϑ_{C_2} points in the direction of the i axis. C_2 is initialised with $\mathbf{0}$ at $\Phi_{\blacksquare}$ and with $\mathbf{1}$ at $\Phi_{\blacktriangledown}$.

The computation control UREs are omitted. They can also be represented by the control dependence graph depicted in Fig. 6.

For a discussion of the choice of control dependence vectors, see [13]. One interesting result is that, if there is a valid space-time mapping for the source UREs, there is always a choice of control dependence vectors that guarantees the existence of a valid space-time mapping for both the source and control UREs. Back to LU-decomposition. Because of the choice of the control dependence vectors for C_1 and C_2, the space-time mapping (1) is also valid for the control UREs (Fig. 7).

Next, we replace in the source UREs the domain predicates in indices by predicates in computation control variables. Let T be a Ⓣ-block. We write $\mathcal{B}(I \in T)$ for the predicate in computation control variable that replaces $I \in T$. By definition, the set of separating hyperplanes constructed in the first step contains a subset that separates T from the remaining Ⓣ-blocks. $\mathcal{B}(I \in T)$ is just a conjunction of equalities; each equality corresponds to one separating hyperplane in the subset and is of the form $C(I) = c$, where C is the computation control variable associated with the separating hyperplane and c is the control value that C defines at T.

Consider the data dependence graph for LU-decomposition (Fig. 3). Each Ⓣ-block is separated from the other three by the two hyperplanes H_1 and H_2. A further application of the previous construction of the computation control UREs yields (Fig. 7):

$$\begin{array}{rcl} \mathcal{B}((i,j,k) \in \Phi_{\blacksquare}) & = & C_1(i,j,k) = \mathbf{0} \wedge C_2(i,j,k) = \mathbf{0} \\ \mathcal{B}((i,j,k) \in \Phi_{\bullet}) & = & C_1(i,j,k) = \mathbf{1} \wedge C_2(i,j,k) = \mathbf{1} \\ \mathcal{B}((i,j,k) \in \Phi_{\blacktriangle}) & = & C_1(i,j,k) = \mathbf{1} \wedge C_2(i,j,k) = \mathbf{0} \\ \mathcal{B}((i,j,k) \in \Phi_{\blacktriangledown}) & = & C_1(i,j,k) = \mathbf{0} \wedge C_2(i,j,k) = \mathbf{1} \end{array} \qquad (2)$$

Replacing the domain predicates of the source UREs by predicates in computation control variables with simplification yields:

The Source UREs:

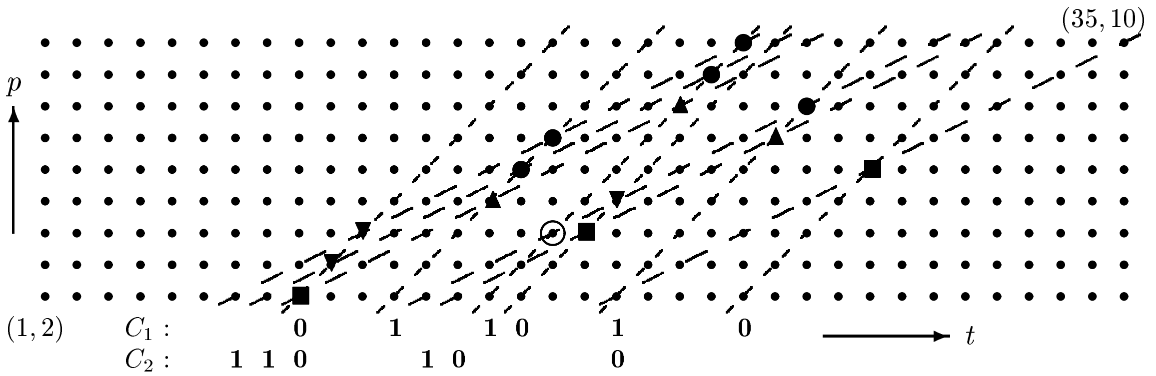

Figure 7. The computation control flow for the one-dimensional array of Fig. 4. The circle will be referred to in Sect. 6.

$$
\begin{array}{rcl}
L(i,j,k) & = & \begin{cases} C_1(i,j,k)=\mathbf{1} \wedge C_2(i,j,k)=\mathbf{0} \twoheadrightarrow A(i,j,k-1)U(i-1,j,k) & \blacktriangle \\ C_1(i,j,k)=\mathbf{1} \wedge C_2(i,j,k)=\mathbf{1} \twoheadrightarrow L(i,j-1,k) & \bullet \end{cases} \\
U(i,j,k) & = & \begin{cases} C_1(i,j,k)=\mathbf{0} \wedge C_2(i,j,k)=\mathbf{1} \twoheadrightarrow A(i,j,k-1) & \blacktriangledown \\ C_1(i,j,k)=\mathbf{0} \wedge C_2(i,j,k)=\mathbf{0} \twoheadrightarrow A(i,j,k-1)^{-1} & \blacksquare \\ C_1(i,j,k)=\mathbf{1} \twoheadrightarrow U(i-1,j,k) & \bullet\ \blacktriangle \end{cases} \\
A(i,j,k) & = & \{\ \mathsf{true} \twoheadrightarrow A(i,j,k-1)-L(i,j-1,k)U(i-1,j,k) \quad \blacksquare\ \bullet\ \blacktriangle\ \blacktriangledown
\end{array}
$$

6 The Synthesis of Propagation Control Flow

The propagation control flow serves to distinguish pipelining points from computation points. In other words, it tells which space-time points are the images of the points of the index space under the space-time mapping. Let us explain the necessity of propagation control flow for one-dimensional arrays. Recall that, for a Ⓣ-block T, $\mathcal{B}(I \in T)$ denotes the predicate in computation control variables that replaces $I \in T$. The image of $\mathcal{B}(I \in T)$, denoted $\mathcal{B}(\overline{I} \in \overline{T})$, is the predicate in which every occurrence of index vector I in $\mathcal{B}(I \in T)$ is replaced by $\overline{I}$. Thus, the images of the four predicates in (2) under the space-time mapping are:

$$
\begin{aligned}
\mathcal{B}((t,p) \in \overline{\Phi}_{\blacksquare}) &= C_1(t,p)=\mathbf{0} \wedge C_2(t,p)=\mathbf{0} \\
\mathcal{B}((t,p) \in \overline{\Phi}_{\bullet}) &= C_1(t,p)=\mathbf{1} \wedge C_2(t,p)=\mathbf{1} \\
\mathcal{B}((t,p) \in \overline{\Phi}_{\blacktriangle}) &= C_1(t,p)=\mathbf{1} \wedge C_2(t,p)=\mathbf{0} \\
\mathcal{B}((t,p) \in \overline{\Phi}_{\blacktriangledown}) &= C_1(t,p)=\mathbf{0} \wedge C_2(t,p)=\mathbf{1}
\end{aligned}
$$

By construction, $\mathcal{B}(I \in T)$ holds at $I \in \Phi$ iff I is a point in T. However, the assertion that $\mathcal{B}(\overline{I} \in \overline{T})$ holds at $\overline{I} \in \Upsilon$ iff $\overline{I}$ is a point in $\overline{T}$ is not true: the "if" part is but the "only if" part is not. Following from the injectivity of a valid space-time mapping over Φ, this assertion can be weakened as follows:

Fact 1 If $\overline{I}$ is in $\overline{T}$, then $\mathcal{B}(\overline{I} \in \overline{T})$ holds at $\overline{I}$. If $\mathcal{B}(\overline{I} \in \overline{T})$ holds at $\overline{I} \in \Upsilon$, then $\overline{I}$ is not in $\overline{\Phi} \backslash \overline{T}$.

This fact clearly indicates that the rôle that the computation control flow plays is to distinguish different types of computation points, and that the computation control flow by itself cannot

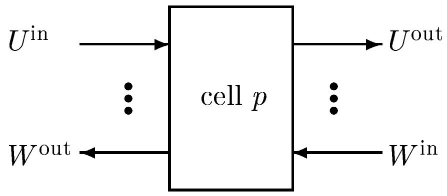

Figure 8. The notation for specifying the cell program.

distinguish pipelining from computation points when, for some T, $\mathcal{B}(\overline{I} \in \overline{T})$ holds at pipelining points.

Consider Fig. 7. $\mathcal{B}((t,p) \in \overline{\Phi}_\bullet)$ holds not only at the points depicted by $\bullet$, as intended, but also at the pipelining point highlighted by a circle. As has been explained in Sect. 4, this is caused by the non-injectivity of the corresponding space-time mapping over $\mathbb{Z}^n$. Therefore, the computation control flow alone cannot tell which space-time points are pipelining or computation points. Propagation control flow is required to distinguish pipelining points and computation points further.

From now on, we shall restrict ourselves to the specification of the propagation control flow for one-dimensional arrays. To avoid unduly complex notation, we present the specification of propagation control flow in the form of programs (rather than in the form of UREs). There are two programs: (1) the program for the host, called the *host program*, injects propagation control signals into the array, and (2) the program for the cells of the array, called the *cell program*, specifies the propagation control signals of the output channels of a cell based on those of the input channels of the cell. The cell program is identical for all cells and is executed by every cell at every time step.

How can UREs be expressed as programs? Since all cells are given the same cell program, we can abstract from the index vectors of propagation control variables. This results in one program that will be executed by every cell of the array.

Let us consider the situation where cell p is computing point $\overline{I}=(t,p)$ at step t (Fig. 8). The cell receives its input signal $V(\overline{I}-\overline{\vartheta}_V)$ of propagation control variable V at step $t-1$ and sends its output signal $V(\overline{I})$ at step t. We abstract from the index vectors of V by writing $V^{\rm in}$ for $V(\overline{I}-\overline{\vartheta}_V)$ and $V^{\rm out}$ for $V(\overline{I})$. That is, $V^{\rm in}$ ($V^{\rm out}$) denotes the input (output) channel for V at this cell. We write value(V) for the set of control values associated with V.

All propagation control variables are moving variables; each is specified by one equation. They are divided into two classes:

- *Pipelining Control Variables V.* Every cell directly sends the control signal received at the input channel of V to its corresponding output channel. That is, the defining equation of the variable is given by

$$V^{\rm out} \quad = \quad V^{\rm in}$$

  If the cell is a border cell, $V^{\rm in}$ represents one of the values in value(V) to be injected by the host program.

- *Evolution Control Variables* V. Every cell may change the control signal received at the input channel of V before sending the (changed) control signal to the corresponding output channel. The control signal sent to the output channel of V depends on all propagation control signals present at input channels of the cell. That is, the defining equation of the variable is given by

$$V^{\text{out}} \;=\; \begin{cases} \textbf{if}\;\; B_1(W_1^{\text{in}},\cdots) & \rightarrow \quad f_1(V^{\text{in}}) \\ & \quad\;\; \vdots \\ \;[\!]\;\; B_b(W_b^{\text{in}},\cdots) & \rightarrow \quad f_b(V^{\text{in}}) \\ \textbf{fi} & \end{cases}$$

  The guard $B_i(W_i^{\text{in}},\cdots)$ can always be written in disjunctive normal form, where each disjunct consists of a conjunction of tests of an argument (i.e., a propagation control variable) for a control value. $f_i(V^{\text{in}})$ is a function that recursively defines the control value of V^{out} in terms of argument V^{in}.

  Similarly, if the cell is a border cell, V^{in} represents one of the values in value(V) to be injected by the host program.

The propagation control flow is *correct* if there exists a construction of a characteristic function $\chi_{\overline{\Phi}}$ in the propagation control variables of $\overline{\Phi}$ in Υ:

$$\chi_{\overline{\Phi}} : \Upsilon \rightarrow \{\chi_{\text{c}}, \chi_{\text{p}}\}, \qquad \chi_{\overline{\Phi}}(\overline{I}) = \textbf{if } \overline{I} \in \overline{\Phi} \;\rightarrow\; \chi_{\text{c}} \;[\!]\; \textbf{else} \;\rightarrow\; \chi_{\text{p}} \;\textbf{fi}$$

That is, $\overline{I}$ is a computation point if $\chi_{\overline{\Phi}}(\overline{I}) = \chi_{\text{c}}$ and a pipelining point otherwise.

The propagation control flow serves to tell which space-time points are the images of the points of the index space. Its specification depends only on the shape and the dimension of the index space. For ease of presentation, we distinguish four cases: Sect. 6.1 considers three-dimensional cubic index spaces. Sect. 6.2 considers three-dimensional triangular prismatic index spaces. Sect. 6.3 considers index spaces that are arbitrary three-dimensional convex polytopes. Sect. 6.4 considers index spaces that are arbitrary n-dimensional convex polytopes. Sect. 6.5 discusses hardware requirements.

6.1 Three-Dimensional Cubes

Fig. 9(a) depicts the three-dimensional cubic index space. One example that has such an index space is the square matrix product: $C = A \times B$, say, for matrices of size $m \times m$ (from now on simply called "matrix product"). A popular URE specification of matrix product has the data dependence graph depicted in Fig. 9(b) [1, 5, 11]; all points are specified by the inner-product step: $c_{i,j} = c_{i,j} + a_{i,k} b_{k,j}$. Since Ⓣ-partition $= \{\Phi\}$, computation control is not required.

The space-time mapping (1) for LU-decomposition is also valid for matrix product. The corresponding space-time diagram is depicted in Fig. 10. Propagation control is required. E.g., consider the pipelining point highlighted by a circle in Fig. 10, where elements $a_{3,1}$ and $c_{1,1}$ are handled. The cell at this point should propagate rather than compute $c_{1,1}$.

Let b_1, b_2 and b_3 be the standard basis, i.e., (1,0,0), (0,1,0) and (0,0,1), of a three-dimensional vector space. Let ε_1, ε_2 and ε_3 each be 1 or -1. The propagation control UREs consist of four control variables; their domains of equations are Φ:

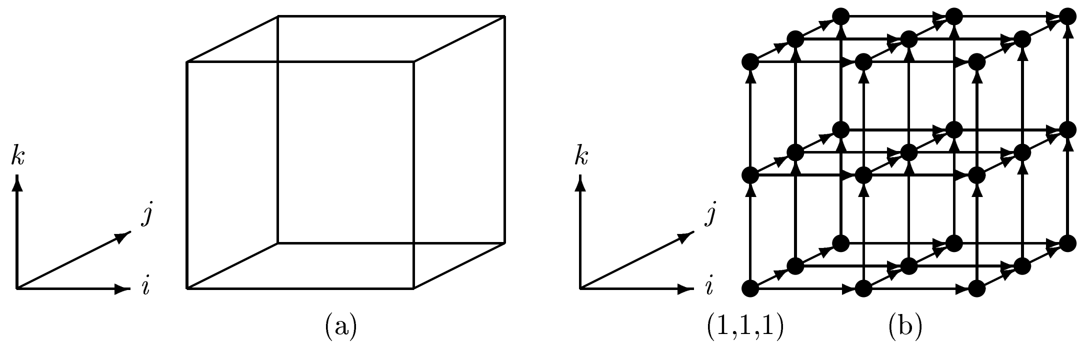

Figure 9. (a) The three-dimensional cubic index space. (b) The data dependence graph of a URE specification for matrix product ($m=3$). The data dependence vectors associated with variables A, B and C are $\vartheta_A=(0,1,0)$, $\vartheta_B=(1,0,0)$, and $\vartheta_C=(0,0,1)$.

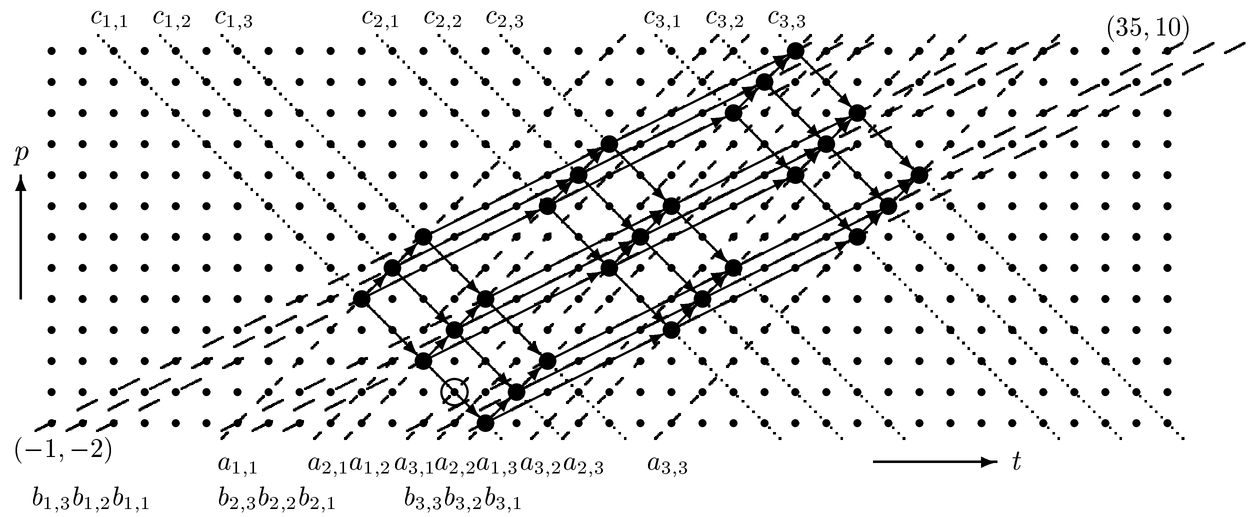

Figure 10. The data flow for matrix product.

- *Evolution control variable E.* $\vartheta_E=\varepsilon_1 b_1$. value$(E)=\{e_{\rm s}, e_1, \cdots, e_{\Gamma_E}, e_{\rm d}, \perp\}$. Both first$(\Phi,\vartheta_E)$ and last(Φ,ϑ_E) are facets of the index space Φ.

- *Initialisation control variables F_1 and F_2.* $\vartheta_{F_1}=\varepsilon_2 b_2$ and $\vartheta_{F_2}=\varepsilon_3 b_3$ (or $\vartheta_{F_1}=\varepsilon_3 b_3$ and $\vartheta_{F_2}=\varepsilon_2 b_2$). $(\forall\ i : 0<i\leqslant 2 : \text{value}(F_i)=\{f_i,\perp\})$. F_1 and F_2 are pipelining control variables; they serve to identify the first points of E, i.e., the points of first(Φ,ϑ_E).

- *Termination control variable L.* $\vartheta_L=\varepsilon_2 b_2$ (or $\vartheta_L=\varepsilon_3 b_3$). value$(L)=\{\ell,\perp\}$. L is a pipelining control variable; it serves to identify the last points of E, i.e., the points of last(Φ,ϑ_E).

Let us comment on the choice of the three parameters ε_1, ε_2 and ε_3. If there exists a valid space-time mapping for the source UREs, all data dependence vectors must generate a pointed cone [8] and vice versa. It is always possible to choose ε_1, ε_2 and ε_3 such that this cone and the

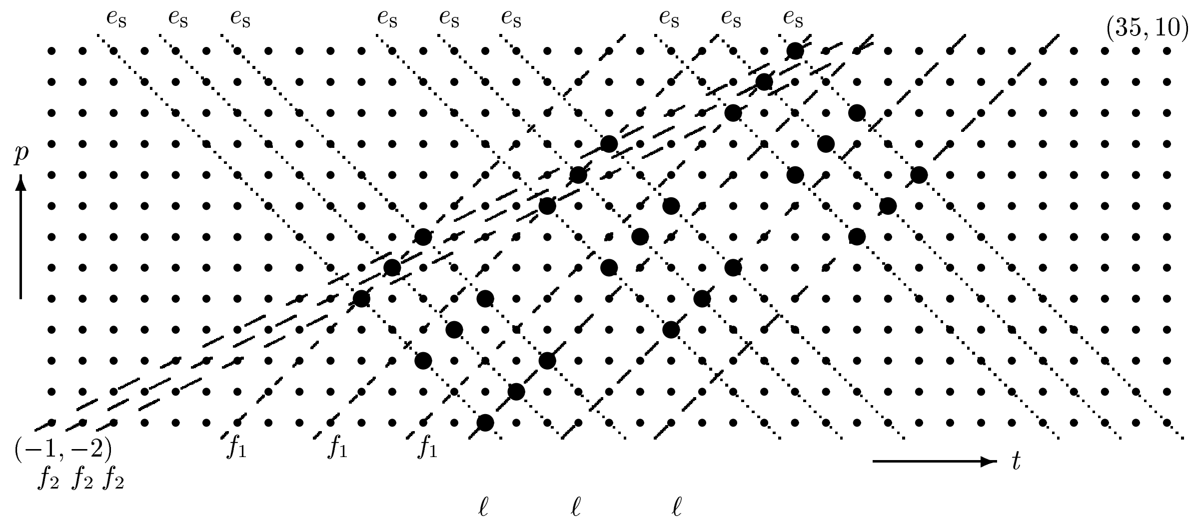

Figure 11. The propagation control flow for the space-time diagram of Fig. 10. $\perp$ is injected at the paths that are not depicted with control values.

control dependence vectors $\varepsilon_1 b_1$, $\varepsilon_2 b_2$ and $\varepsilon_3 b_3$ generate another pointed cone. Hence, the existence of a valid space-time mapping for both the source UREs and the propagation control UREs is guaranteed.

Consider the data dependence graph for matrix product (Fig. 9(b)). We choose arbitrarily $\vartheta_E = \vartheta_C$, $\vartheta_{F_1} = \vartheta_L = \vartheta_A$ and $\vartheta_{F_1} = \vartheta_B$. F_1 and F_2 serve to identify first(Φ, ϑ_E), the bottom facet of the index space. L serves to identify last(Φ, ϑ_E), the top facet of the index space.

The specification of the propagation control UREs is as follows (Fig. 11):

- The initialisation control variables F_1 and F_2. We inject f_i at the computation $\overline{\vartheta}_{F_i}$-paths that contain the first computation points of E and $\perp$ at all the remaining $\overline{\vartheta}_{F_i}$-paths. We define

$$\mathcal{B}(F) \quad = \quad F_1^{\rm in} = f_1 \wedge F_2^{\rm in} = f_2 \tag{3}$$

- The termination control variable L. We inject ℓ at the computation $\overline{\vartheta}_L$-paths that contain the last computation points of E and $\perp$ at all the remaining $\overline{\vartheta}_L$-paths. We define

$$\mathcal{B}(L) \quad = \quad E^{\rm in} = e_{\Gamma_E} \wedge L^{\rm in} = \ell$$

  (For $E^{\rm in} = e_{\Gamma_E}$, see the specification of E presented next.)

- The evolution control variable E. We distinguish two types of $\overline{\vartheta}_E$-paths:
  - Non-computation $\overline{\vartheta}_E$-paths. We inject $\perp$ at these paths and adopt this value for all the points of the paths.
  - Computation $\overline{\vartheta}_E$-paths. Each such path must contain one first and one last computation point. We inject $e_{\rm s}$ at the first point of the path and adopt this value for all soaking

```
PROGRAM: HostProg
for t from t_fst to t_lst do
    for i from 1 to 2 do
        if t ∈ {intime(F_i(I)) | I ∈ first(Φ, ϑ_E)}  →  inject(f_i, F_i(I))
        [] else  →  inject(⊥, F_i(I))
        fi
    if t ∈ {intime(L(I)) | I ∈ last(Φ, ϑ_E)}  →  inject(ℓ, L(I))
    [] else  →  inject(⊥, L(I))
    fi
    if t ∈ {intime(E(I)) | I ∈ first(Φ, ϑ_E)}  →  inject(e_s, E(I))
    [] else  →  inject(⊥, E(I))
    fi
```

```
PROGRAM: CellProg          (Let (∀ i : 0 < i ⩽ Γ_E : e_i = i).)
F_1^out  =  F_1^in
F_2^out  =  F_2^in
L^out    =  L^in
E^out    =  if  B(F)  →  e_1
            []  B(L)  →  e_d
            []  (E^in = e_s ∧ ¬B(F)) ∨ E^in = e_d ∨ E^in = ⊥  →  E^in
            []  else  →  (E^in mod Γ_E) + 1
            fi
```

Figure 12. The host and cell program for three-dimensional cubic index spaces.

points of E. $\mathcal{B}(F)$ holds at the first computation point of the path. There, e_s is converted to e_1. Then, if a point receives element e_k of E, it sends $e_{(k \,\mathbf{mod}\, \Gamma_E)+1}$; k is the distance of a relaying point from its preceding computation point. Thus, the considered element of E periodically adopts the values $e_1, \cdots, e_{\Gamma_E}$. $\mathcal{B}(L)$ holds at the last computation point of the path. There, e_{Γ_E} is converted to e_d, the value for all draining points of E.

See Fig. 12 for the host and cell program. The notation $\mathsf{inject}(s, V(I))$ stands for the injection of control value $s \in \mathrm{value}(V)$ at border cell $\mathsf{incell}(V(I))$. The characteristic function $\chi_{\overline{\Phi}}$ of $\overline{\Phi}$ in Υ is given by

$$\chi_{\overline{\Phi}}(\overline{I}) \quad = \quad \textbf{if } \mathcal{B}(F) \vee E^{\mathrm{in}} = e_{\Gamma_E} \;\rightarrow\; \chi_\mathrm{c} \;[]\; \textbf{else} \;\rightarrow\; \chi_\mathrm{p} \textbf{ fi}$$

Let us motivate the cell program. The first guard of E^{out} selects first computation points of E. The second guard of E^{out} establishes whether a point is a last computation point of E. The third guard handles soaking points, draining points and undefined points of E. $E^{\mathrm{in}} = e_\mathrm{s}$ holds at the soaking points and the first computation points of E. Since $\mathcal{B}(F)$ also holds at the first computation points of E, $E^{\mathrm{in}} = e_\mathrm{s} \wedge \neg\mathcal{B}(F)$ holds at the soaking points of E. $E^{\mathrm{in}} = e_\mathrm{d}$ holds at the draining points of E. $E^{\mathrm{in}} = \perp$ holds at the undefined points of E. The **else** guard of E^{out} captures points that are relaying points or computation points that are neither first nor last computation points of E.

Remark. Because of the non-injectivity problem, the specification of the initialisation control is more complex than the specification of the termination control [14]. Intuitively, the initialisation control must by itself identify correctly the first computation points of E. Once this is done, we have partial information of the distribution of different types of pipelining points. The soaking points of E are the points at which $E^{\text{in}} = e_{\text{s}} \wedge \neg\mathcal{B}(F)$ holds and the undefined points of E are the points at which $E^{\text{in}} = \perp$ holds. To identify the last computation points of E, we exploit the fact that a point is a last computation point of E only if $E^{\text{in}} = e_{\Gamma_E}$ holds at the point. Hence, the non-injectivity problem does not cause any complication in the specification of the termination control. □

Lemma 1 *$\mathcal{B}(F)$ is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ in Υ.*

Sketch of Proof ([14, Thm. 5.3]). Let X (Y) be the set of all ϑ_{F_1}-paths (ϑ_{F_2}-paths) that contain points in $\text{first}(\Phi, \vartheta_E)$. $\mathcal{B}(F)$ holds at a space-time point iff there exists an x in X and a y in Y such that the image of x intersects the image of y at the space-time point. The intersection may occur even if x and y do not intersect, due to the non-injectivity problem. Since $\text{first}(\Phi, \vartheta_E)$ is a rectangle, every ϑ_{F_1}-path in X must intersect all ϑ_{F_2}-paths in Y and vice versa. Hence, $\mathcal{B}(F)$ holds at the points of $\overline{\text{first}(\Phi, \vartheta_E)}$ only. (Complications caused by the non-injectivity problem are avoided by construction.)

Lemma 2 *$\mathcal{B}(L)$ is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}$ in Υ.*

Proof. Follows from the fact that $E^{\text{in}} = e_{\Gamma_E}$ holds at the last computation points of E but not at undefined points of E and the fact that $L^{\text{in}} = \ell$ holds only at the last computation points of E and some undefined points of E. □

Theorem 3 *If a space-time mapping is valid for the propagation control UREs, then the propagation control flow is correct.*

Proof. The propagation control flow is correct iff $\mathcal{B}(F)$ is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ in Υ and $\mathcal{B}(L)$ is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}$ in Υ. A further application of Lemmata 1 and 2 completes the proof. □

The specification of the propagation control flow as presented previously can be extended to a larger class of index spaces called FPLF-polytopes. A convex polytope is an *FPLF-polytope* (*F*irst points: *P*arallelogram, *L*ast points: *F*acet) if there exists a choice of ϑ_E such that $\text{first}(\Phi, \vartheta_E)$ is a parallelogram and $\text{last}(\Phi, \vartheta_E)$ is a facet. Then we choose ϑ_{F_1} and ϑ_{F_2} such that two edges of $\text{first}(\Phi, \vartheta_E)$ are parallel to ϑ_{F_1} and the other two to ϑ_{F_2}, and ϑ_L as any vector that is parallel to the facet $\text{last}(\Phi, \vartheta_E)$.

The index space depicted in Fig. 13(a) is an FPLF-polytope because of the existence of the choice $\vartheta_E = (0, 1, 0)$ or $\vartheta_E = (0, -1, 0)$. This index space is special in that $\text{first}(\Phi, \vartheta_E)$ and $\text{last}(\Phi, \vartheta_E)$ are not disjoint and intersect at the vertical edge on the right. Hence, the points of the edge are both first and last computation points of E. The correctness of the propagation control UREs for the

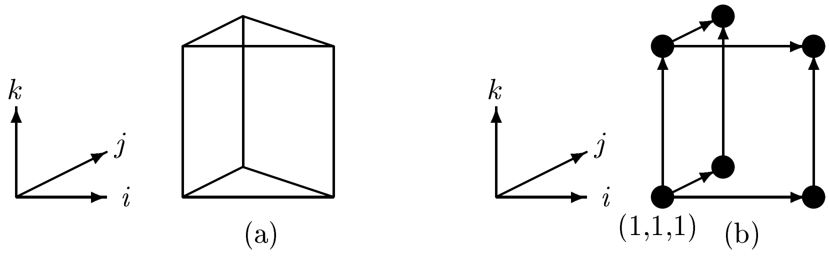

Figure 13. (a) The three-dimensional triangular prismatic index space. (b) The data dependence graph for a set of artificial UREs.

cubic index space relies implicitly on the disjointness of $\text{first}(\Phi, \vartheta_E)$ and $\text{last}(\Phi, \vartheta_E)$. If $\text{first}(\Phi, \vartheta_E)$ and $\text{last}(\Phi, \vartheta_E)$ are not disjoint, the cell program must be modified as follows:

- The first guard of E^{in} is changed to $\mathcal{B}(F) \wedge \neg\mathcal{B}(L)$.
- $\mathcal{B}(L)$ is redefined to

$$\mathcal{B}(L) \quad = (\mathcal{B}(F) \vee E^{\text{in}} = e_{\Gamma_E}) \wedge E^{\text{in}} = \ell$$

This modification is based on the fact that $\mathcal{B}(F) \wedge \neg\mathcal{B}(L)$ is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E) \setminus \text{last}(\Phi, \vartheta_E)}$ in Υ. For a computation $\overline{\vartheta}_E$-path that contains only one computation point, which is both a first and last computation point of E, the new first guard of E^{in} is false and the new second guard of E^{in} is true at the computation point. This ensures that control value e_{s} injected at the path is converted to e_{d} (rather than e_1 under the original programs) at the only computation point of the path.

Finally, we remark that the communication constraint for the initialisation control variables can be disregarded. It is always guaranteed by a valid space-time mapping because of the choice of the corresponding control dependence vectors [14].

6.2 Three-Dimensional Triangular Prisms

Fig. 13(a) depicts the three-dimensional triangular prismatic index space. Fig. 13(b) depicts the data dependence graph for a set of artificial UREs whose index space is a triangular prism. This graph is a subgraph of the data dependence graph for matrix product for the size parameter $m = 2$.

Let us examine the complications that arise when we directly apply the propagation control UREs for the cubic index space to the triangular prismatic index space. We choose $\vartheta_E = (0, 0, 1)$, $\vartheta_{F_1} = (0, 1, 0)$, $\vartheta_{F_2} = (1, 0, 0)$ and $\vartheta_L = (0, 1, 0)$. $\text{first}(\Phi, \vartheta_E)$ and $\text{last}(\Phi, \vartheta_E)$ are facets of the index space. But $\text{first}(\Phi, \vartheta_E)$ is a triangle rather than a parallelogram. The following space-time mapping is valid for the propagation control UREs with respect to the index space of Fig. 13(b):

$$\pi \quad = \quad \begin{bmatrix} 3 & 2 & 2 \\ 1 & 2 & -2 \end{bmatrix}$$

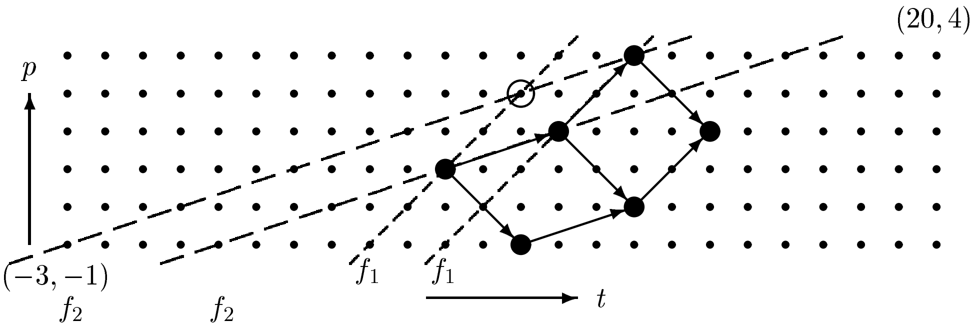

Figure 14. The initialisation control flow for the source UREs of Fig. 13(b).

The initialisation control flow is depicted in Fig. 14. $\mathcal{B}(F)$ holds not only at the three first computation points of E, as intended, but also at the soaking point of E highlighted by a circle. Hence, $\mathcal{B}(F)$ is not a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$, and consequently, the propagation control flow is not correct.

To identify and eliminate the cause of this problem, we introduce the following notations. For $x, \delta \in \mathbb{Z}^n$, we write line$(x, \delta)$ for the set of all points y whose distance to x is an integral multiple of δ:

$$\text{line}(x, \delta) \quad = \quad \{y \mid y = x + m\delta \wedge m \in \mathbb{Z}\}$$

When we speak of the line containing x that is parallel to δ, we mean the set line(x, δ). Let $\mathcal{F}$ be the intersection of the lines containing the first points of E that are parallel to ϑ_{F_1} and the lines containing the first points of E that are parallel to ϑ_{F_2}:

$$\mathcal{F} \quad = \quad (\bigcup I : I \in \text{first}(\Phi, \vartheta_E) : \text{line}(I, \vartheta_{F_1})) \cap (\bigcup I : I \in \text{first}(\Phi, \vartheta_E) : \text{line}(I, \vartheta_{F_2}))$$

$\mathcal{F}$ is a parallelogram. By the specification of the initialisation control variables, $F_1(I) = f_1 \wedge F_2(I) = f_2$ holds at the points of $\mathcal{F}$. If a space-time mapping is valid for the propagation control UREs, then $\mathcal{B}(F)$ holds at the space-time points of $\overline{\mathcal{F}}$ only.

Consider the data dependence graph of Fig. 13(b). We have

$$\begin{aligned} \text{first}(\Phi, \vartheta_E) \quad &= \quad \{(1,1,1), (2,1,1), (1,2,1)\} \\ \mathcal{F} \quad &= \quad \{(1,1,1), (2,1,1), (1,2,1), (2,2,1)\} \end{aligned}$$

$\mathcal{F}$ is a superset of first(Φ, ϑ_E). In fact, this is generally true by the definitions of first(Φ, ϑ_E) and $\mathcal{F}$. Here, $\mathcal{F} \backslash \text{first}(\Phi, \vartheta_E) = \{(2,2,1)\}$. $\mathcal{B}(F)$ is not a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ because it holds at the image of point $(2,2,1)$, which is a soaking point of E.

What is the key to ensuring that $\mathcal{B}(F)$ is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ and, consequently, the correctness of the propagation control UREs? The key lies in where the space-time points of $\overline{\mathcal{F}} \backslash \overline{\text{first}(\Phi, \vartheta_E)}$ are located in the space-time diagram. If a valid space-time mapping maps the points in $\mathcal{F} \backslash \text{first}(\Phi, \vartheta_E)$ to pipelining points that are not soaking points of E, we can ensure the correctness of the propagation control flow by redefining $\mathcal{B}(F)$ as follows:

$$\mathcal{B}(F) \quad = \quad E^{\text{in}} = e_{\text{s}} \wedge F_1^{\text{in}} = f_1 \wedge F_2^{\text{in}} = f_2 \tag{4}$$

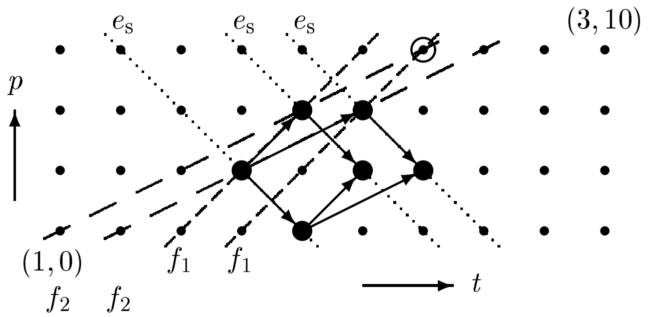

Figure 15. The evolution and initialisation control flow for the source UREs of Fig. 13(b).

Such a space-time mapping is referred to as a *strongly* valid space-time mapping.

The space-time mapping (1) for LU-decomposition is strongly valid for the propagation control UREs as specified previously for the source UREs of Fig. 13(b). The evolution and initialisation control flow is depicted in Fig. 15. This time, point $(2,2,1)$ is mapped to an undefined point of E. Although $F_1^{\text{in}} = f_1 \wedge F_2^{\text{in}} = f_2$, i.e., the old $\mathcal{B}(F)$ of (3) holds at this point, $E^{\text{in}} = e_s$ does not. This ensures that the new $\mathcal{B}(F)$ of (4) is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$.

How do we generate strongly valid space-time mappings? We simply replace $\text{first}(\Phi, \vartheta_E)$ in Stat. 3 of Thm. 1 by $\mathcal{F}$ for the evolution control variable E. The communication constraint thus defined ensures that the points of $\mathcal{F} \backslash \text{first}(\Phi, \vartheta_E)$ are mapped to undefined points of E.

Finally, we present the specification of the propagation control flow for a class of index spaces called FFLF-polytopes that subsumes the class of FPLF-polytopes (Sect. 6.1). A convex polytope is an *FFLF-polytope* (*F*irst points: *F*acet, *L*ast points: *F*acet) if there exists a choice of ϑ_E such that $\text{first}(\Phi, \vartheta_E)$ and $\text{last}(\Phi, \vartheta_E)$ are facets of the polytope. Then we choose ϑ_{F_1} and ϑ_{F_2} as two linearly independent vectors that are parallel to $\text{first}(\Phi, \vartheta_E)$, and ϑ_L as any vector that is parallel to the facet $\text{last}(\Phi, \vartheta_E)$.

Theorem 4 *If a space-time mapping is strongly valid for the propagation control UREs with respect to an FFLF-polytope, the propagation control flow is correct.*

Proof. $\mathcal{B}(L)$ is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}$ in Υ (Lemma 2). By the proof of Thm. 3, it suffices to show that $\mathcal{B}(F)$ of (4) is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ in Υ. By a similar reasoning as in the proof of Lemma 1, we can show that $\mathcal{B}(F)$ of (3) holds only at the points of $\overline{\mathcal{F}}$. If a space-time mapping is strongly valid, the points of $\mathcal{F} \backslash \text{first}(\Phi, \vartheta_E)$ are mapped to undefined points of E and $E^{\text{in}} = e_s$ does not hold at these points. Hence, $\mathcal{B}(F)$ of (4) is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ in Υ. $\square$

6.3 Three-Dimensional Convex Polytopes

The propagation control UREs for a three-dimensional convex polytope are specified as follows (for the definitions of (5) and (6), see the specification of E presented afterwards):

- *Initialisation Control Variables* $F_{1,1}, F_{1,2}, \cdots, F_{f,1}, F_{f,2}$. $(\forall\ i,k : 0 < i \leqslant f \wedge 0 < k \leqslant 2 :$ value$(F_{i,k}) = \{f_{i,k}, \perp\})$. We shall drop index i in $F_{i,k}$ and $f_{i,k}$ when $f = 1$. Let

$$\text{Ⓕ} \quad = \quad \{\text{first}(\Phi, \vartheta_E)_i \mid 0 < i \leqslant f\}$$

be a partition of last(Φ, ϑ_E) such that dim(first$(\Phi, \vartheta_E)_i) \leqslant n-1$. One possible solution is to first decompose first(Φ, ϑ_E) into the set of facets of the index space Φ and then build a partition Ⓕ from this set. (These facets intersect at their boundaries.) We associate two initialisation control variables, $F_{i,1}$ and $F_{i,2}$, with first$(\Phi, \vartheta_E)_i$. We inject $f_{i,k}$ $(k = 1, 2)$ at the computation $\overline{\vartheta}_{F_{i,k}}$-paths that contain the points of first$(\Phi, \vartheta_E)_i$ and $\perp$ at the remaining $\overline{\vartheta}_{F_{i,k}}$-paths. We define

$$\begin{array}{rcl} \mathcal{B}(F_i) & = & E^{\text{in}} = e_{\text{s}}^i \wedge F_{i,1}^{\text{in}} = f_{i,1} \wedge F_{i,2}^{\text{in}} = f_{i,2} \\ \mathcal{B}(F) & = & (\exists\ i : 0 < i \leqslant f : \mathcal{B}(F_i)) \end{array} \tag{5}$$

- *Termination Control Variables* $L_1, \cdots, L_\ell$. $(\forall\ i : 0 < i \leqslant \ell :$ value$(L_i) = \{\ell_i, \perp\})$. Let

$$\text{Ⓛ} \quad = \quad \{\text{last}(\Phi, \vartheta_E)_i \mid 0 < i \leqslant \ell\}$$

be a covering of last(Φ, ϑ_E) such that dim(last$(\Phi, \vartheta_E)_i) \leqslant n-1$. One possible solution is to choose Ⓛ as the set of facets of the index space Φ that are contained in last(Φ, ϑ_E). We associate one termination control variable, L_i, with last$(\Phi, \vartheta_E)_i$. We inject ℓ_i at the computation $\overline{\vartheta}_{L_i}$-paths that contain the points of last$(\Phi, \vartheta_E)_i$ and $\perp$ at the remaining $\overline{\vartheta}_{L_i}$-paths. We define

$$\begin{array}{rcl} \mathcal{B}(L_i) & = & (\mathcal{B}(F) \vee E^{\text{in}} = e_{\Gamma_E}) \wedge L_i^{\text{in}} = \ell_i \\ \mathcal{B}(L) & = & (\exists\ i : 0 < i \leqslant \ell : \mathcal{B}(L_i)) \end{array} \tag{6}$$

- *Evolution control variable E*. value$(E) = \{e_{\text{s}}^1, \cdots, e_{\text{s}}^f, e_1, \cdots, e_{\Gamma_E}, e_{\text{d}}, \perp\}$. We shall drop superscript i in e_{s}^i when $f = 1$. We distinguish two types of $\overline{\vartheta}_E$-paths:

  - Non-computation $\overline{\vartheta}_E$-paths. We inject $\perp$ at these paths and adopt this value for all the points of the paths.

  - Computation $\overline{\vartheta}_E$-paths. Each must contain one and only one first computation point of E in $\overline{\text{first}(\Phi, \vartheta_E)}_i$ for some i, because Ⓕ is a partition of first(Φ, ϑ_E). We inject e_{s}^i at the first point of the path and adopt this value for all soaking points of E. (The superscript i in e_{s}^i identifies the soaking points in the $\overline{\vartheta}_E$-paths that contain first computation points of E in $\overline{\text{first}(\Phi, \vartheta_E)}_i$.) $\mathcal{B}(F)$ holds at the first computation point of the path. There, e_{s}^i is converted to e_1. Then, if a point receives element e_k of E, it sends $e_{(k \,\mathbf{mod}\, \Gamma_E)+1}$; k is the distance of a relaying point from its preceding computation point. Thus, the considered element of E periodically adopts the values $e_1, \cdots, e_{\Gamma_E}$. $\mathcal{B}(L)$ holds at the last computation point of the path. There, e_{Γ_E} is converted to e_{d}, the value for all draining points of E. If the path contains only one computation point, which is, therefore, both a first and last computation point, e_{s}^i changes to e_{d} at that point.

PROGRAM: *HostProg*
for t **from** t_{fst} **to** t_{lst} **do**
  for i **from** 1 **to** f **do**
    for k **from** 1 **to** 2 **do**
      if $t \in \{\mathsf{intime}(F_{i,k}(I)) \mid I \in \text{first}(\Phi, \vartheta_E)_i\} \;\rightarrow\; \mathsf{inject}(f_{i,k}, F_{i,k}(I))$
      $[\!]$ **else** $\rightarrow\; \mathsf{inject}(\perp, F_{i,k}(I))$
      fi
  for i **from** 1 **to** ℓ **do**
    if $t \in \{\mathsf{intime}(L_i(I)) \mid I \in \text{last}(\Phi, \vartheta_E)_i\} \;\rightarrow\; \mathsf{inject}(\ell_i, L_i(I))$
    $[\!]$ **else** $\rightarrow\; \mathsf{inject}(\perp, L_i(I)$
    fi
  if $t \notin \{\mathsf{intime}(E(I)) \mid I \in \text{first}(\Phi, \vartheta_E)\} \;\rightarrow\; \mathsf{inject}(\perp, E(I))$
  $[\!]$ **else** $\rightarrow$ **for** i **from** 1 **to** f **do**
      if $t \in \{\mathsf{intime}(E(I)) \mid I \in \text{first}(\Phi, \vartheta_E)_i\} \;\rightarrow\; \mathsf{inject}(e_{\text{s}}^i, E(I))$
      fi
  fi

PROGRAM: *CellProg* (Let $(\forall\, i : 0 < i \leqslant \Gamma_E : e_i = i)$.)
$(\forall\, i : 0 < i \leqslant f : F_{i,1}^{\text{out}} = F_{i,1}^{\text{in}})$
$(\forall\, i : 0 < i \leqslant f : F_{i,2}^{\text{out}} = F_{i,2}^{\text{in}})$
$(\forall\, i : 0 < i \leqslant \ell : L_i^{\text{out}} = L_i^{\text{in}})$
$E^{\text{out}} =$ **if** $\mathcal{B}(F) \wedge \neg\mathcal{B}(L) \rightarrow e_1$
  $[\!]\; \mathcal{B}(L) \rightarrow e_{\text{d}}$
  $[\!]\; (\exists\, i : 0 < i \leqslant f : E^{\text{in}} = e_s^i \wedge \neg\mathcal{B}(F_i)) \vee E^{\text{in}} = e_{\text{d}} \vee E^{\text{in}} = \perp \rightarrow E^{\text{in}}$
  $[\!]$ **else** $\rightarrow (E^{\text{in}}\ \mathbf{mod}\ \Gamma_E) + 1$
  fi

Figure 16. The host and cell program for three-dimensional convex polytopes.

See Fig. 16 for the host and cell program for three-dimensional UREs. The characteristic function $\chi_{\overline{\Phi}}$ of $\overline{\Phi}$ in Υ is given by

$$\chi_{\overline{\Phi}}(\overline{I}) \quad = \quad \textbf{if } \mathcal{B}(F) \vee E^{\text{in}} = e_{\Gamma_E} \;\rightarrow\; \chi_{\text{c}} \;[\!]\; \textbf{else} \;\rightarrow\; \chi_{\text{p}} \;\textbf{fi}$$

For the purpose of defining a strong valid space-time mapping, we replace $\text{first}(\Phi, \vartheta_E)$ in Stat. 3 of Thm. 1 by $\mathcal{F}$ defined as follows:

$$\begin{aligned}
\mathcal{F}_i &= (\bigcup I : I \in \text{first}(\Phi, \vartheta_E)_i : \text{line}(I, \vartheta_{F_{i,1}})) \cap (\bigcup I : I \in \text{first}(\Phi, \vartheta_E)_i : \text{line}(I, \vartheta_{F_{i,2}})) \\
\mathcal{F} &= (\bigcup i : 0 < i \leqslant f : \mathcal{F}_i)
\end{aligned}$$

Theorem 5 *If a space-time mapping is strongly valid for the propagation control UREs with respect to a convex polytope, the propagation control flow is correct.*

Proof. It suffices to prove that $\mathcal{B}(F)$ of (5) is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}$ in Υ and that $\mathcal{B}(L)$ of (6) is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}$ in Υ. $\mathcal{B}(F)$ of (5) is a characteristic

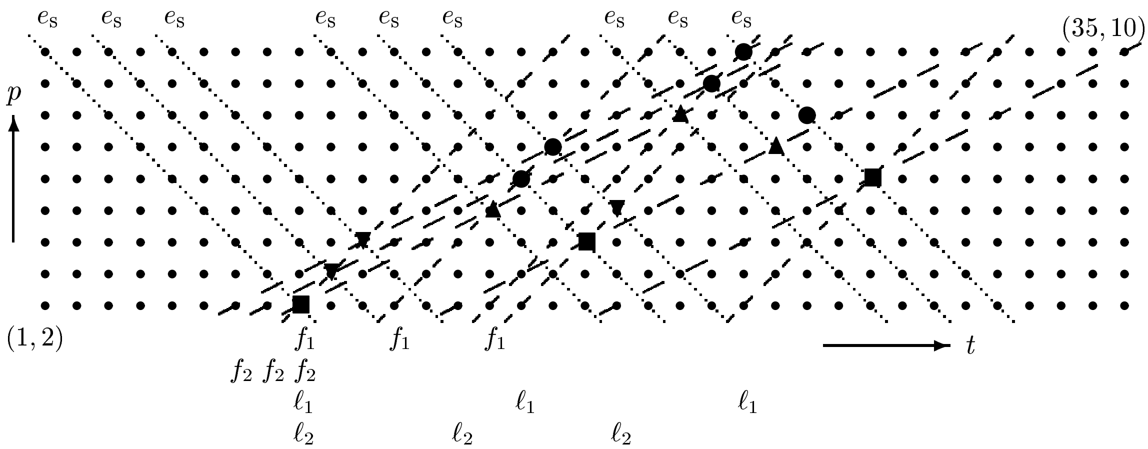

Figure 17. The propagation control flow of the one-dimensional array of Fig. 4.

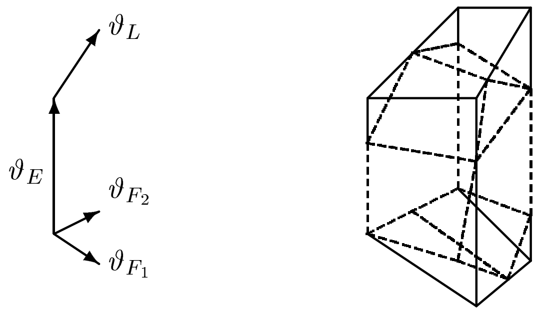

Figure 18. Adaptation of the index space for the specification of the propagation control flow.

function of $\overline{\text{first}(\Phi, \vartheta_E)}$ if $\mathcal{B}(F_i)$ of (5) is a characteristic function of $\overline{\text{first}(\Phi, \vartheta_E)}_i$. $\mathcal{B}(L)$ of (6) is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}$ if $\mathcal{B}(L_i)$ of (6) is a characteristic function of $\overline{\text{last}(\Phi, \vartheta_E)}_i$. The rest of the proof follows from Thm. 6.2. □

Example 5 (The propagation control UREs for LU-decomposition)

We choose $\vartheta_E = \vartheta_A$. first$(\Phi, \vartheta_E)$ is a facet. We choose Ⓕ $= \{\text{first}(\Phi, \vartheta_E)\}$. We need two initialisation control variables F_1 and F_2. We choose $\vartheta_{F_1} = \vartheta_L$ and $\vartheta_{F_2} = \vartheta_U$. This gives rise to $\mathcal{F} = \text{first}(\Phi, \vartheta_E)$. We choose Ⓛ $= \{\text{last}(\Phi, \vartheta_E)_1, \text{last}(\Phi, \vartheta_E)_2\}$, where last$(\Phi, \vartheta_E)_1$ and last$(\Phi, \vartheta_E)_2$ are the left and front facets of the index space, respectively (Fig. 3):

$$
\begin{aligned}
\text{last}(\Phi, \vartheta_E)_1 &= \Phi \cap \{(i,j,k) \mid i-k=0\} \\
\text{last}(\Phi, \vartheta_E)_2 &= \Phi \cap \{(i,j,k) \mid j-k=0\}
\end{aligned}
$$

We choose $\vartheta_{L_1} = \vartheta_L$ and $\vartheta_{L_2} = \vartheta_U$. The space-time mapping (1) for LU-decomposition is strongly valid for the propagation control UREs thus specified (Fig. 17). □

Let us now have a look at how an adaptation of the index space can simplify the specification of the initialisation and termination control. We describe the basic idea with respect to an artificial index space depicted by the dashed polytope in Fig. 18: first(Φ, ϑ_E) and last(Φ, ϑ_E) each contain two facets of the index space. Therefore, $|$Ⓕ$| \geqslant 2$ and $|$Ⓛ$| \geqslant 2$. Assume that $|$Ⓕ$| = |$Ⓛ$| = 2$.

Then, we need four initialisation control variables and two termination control variables. A simple adaptation of the index space can reduce the number of each type of control variable by half. Let us denote the two initialisation control variables by F_1 and F_2 and the termination control variable by L. Let their control dependence vectors be depicted as in Fig. 18. We extend the index space along ϑ_E and $-\vartheta_E$ and denote the extended index space by Φ_{E}, the solid polytope in Fig. 18, until both $\mathrm{first}(\Phi_{\mathrm{E}}, \vartheta_E)$ and $\mathrm{last}(\Phi_{\mathrm{E}}, \vartheta_E)$ are facets of the extended index space Φ_{E}. (Of course, the computation control UREs must be specified for the extended index space Φ_{E}.)

Φ_{E} is a superset of Φ. A fixed space-time mapping may yield the same latency and processor space in both cases. This happens when the extension does not change the longest dependence path and the size of the projection space. If such an adaptation of the index space degrades the performance of the systolic array too much, it had better be avoided.

6.4 N-Dimensional Convex Polytopes

The specification of the propagation control UREs for an n-dimensional convex polytope is based on the observation that our previous construction of the propagation control UREs for a three-dimensional convex polytope supports a hierarchical specification of control signals.

First, we specify an evolution control variable E and proceed exactly as we did for a three-dimensional convex polytope. Then, we specify initialisation control variables to identify the first points in $\mathrm{first}(\Phi, \vartheta_E)$ and termination control variables to identify the last points in $\mathrm{last}(\Phi, \vartheta_E)$. The specification of the termination control proceeds exactly as for a three-dimensional convex polytope. To specify the initialisation control variables, we treat $\mathrm{first}(\Phi, \vartheta_E)$ as a new index space and subject it recursively to the same construction. Since the initial index space is bounded and the new index spaces resulting from the decomposition always contain less points than the preceding one, this decomposition must eventually lead to a three-dimensional space. Then, we apply the construction of the propagation control flow for a three-dimensional convex polytope.

The index spaces for practical applications are regular. If the index space is a n-dimensional parallelepiped, we can always choose the vectors parallel to the edges of the parallelepiped in the decomposition. An three-dimensional convex polytope is obtained after $n-1$ decompositions. Details can be found in [14].

6.5 Hardware Requirements

Let us have a look at the requirements that the propagation control UREs impose on hardware. Consider Figs. 7 and 17. Some paths may start at internal cells. The corresponding control values at these paths can be implemented by a system reset or by pipelining before the first step. The construction of the propagation control UREs requires that all input channels of the propagation control variables be reset to $\perp$.

The bit width required to communicate values of V is $\lceil \log_2 |\mathrm{value}(V)| \rceil$. For example, evolution control variable E needs $\lceil \log_2(\Gamma_E + |Ⓕ| + 2) \rceil$ bits: $\Gamma_E - 1$ for relaying points, one for computation points, $|Ⓕ|$ for soaking points, one for draining points and one for undefined points. We should choose ϑ_E and the space-time mapping such that Γ_E becomes a constant. This way, the number of the control values for E is independent of the size of the problem. The modulo operation in the cell program can be implemented by a circular shift (or rotate) operation if Γ_E is a power of 2. It is superfluous if $\Gamma_E = 1$.

7 Conclusions

We have presented a method that enables a specification of control signals for systolic arrays in terms of control UREs. The control UREs consist of two sets of UREs: the computation control UREs specify the control signals to distinguish different types of computations in the index space and the propagation control UREs specify how to propagate the input data from the two border cells to internal cells and the output data from internal cells to the two border cells.

Our method has the following advantages. First, specifying control signals amounts to choosing separating hyperplanes and control dependence vectors in the case of specifying computation control UREs and control dependence vectors in the case of specifying propagation control UREs. This enables users of interactive systolic design systems to apply their geometrical intuition in the search for a good control flow. Once the control flow is specified in the form of UREs, its optimisation with respect to some prescribed design constraints can be formulated as a linear and integer programming or combinatorial optimisation problem. Second, the notion of separating hyperplanes and control dependence vectors is explicit in the specification of control signals. Therefore, we can choose space-time mappings following prescribed design constraints with respect to not only the data but also the control flow. For example, we can enforce the pipelining of control signals by simply choosing the scheduling vector such that it is not perpendicular to any of the separating hyperplanes. We can also enforce neighbouring communication for both the data and control flow, if possible, because both the data and control dependence vectors are explicit in the source and control UREs. Third, the correctness of the control UREs is independent of the space-time mapping. The control UREs constitute the specification of control signals for not just one but a class of systolic arrays that can be obtained by the choice of different space-time mappings. The source UREs resulting from domain replacement and control UREs can be viewed as a refinement of the initial source UREs. Therefore, they can be manipulated like conventional UREs by available synthesis methods. For example, they can be mapped easily to fixed-size arrays [15, 16]. Of course, such a partitioning calls for extra control signals due to the change in the specification. These extra control signals can be specified so as to preserve the behaviour of the specification.

Our method for the construction of control UREs applies for any set of recurrence equations that are defined over a domain in $\mathbb{Z}^n$; each equation is of the form:

$$V(I) \quad = \quad \begin{cases} I \in D_1 & \twoheadrightarrow \quad f_1(W_1(\rho_1(I)), \dots) \\ & \vdots \\ I \in D_p & \twoheadrightarrow \quad f_p(W_p(\rho_p(I)), \dots) \end{cases} \tag{7}$$

where ρ_i is any function from $\mathbb{Z}^n$ to $\mathbb{Z}^\ell$ $(0 \leqslant \ell \leqslant n)$. The specification of the computation control UREs is straightforward once the Ⓣ-partition is defined. The specification of the propagation control UREs depends only on the shape and dimension of the index space. Recall that only UREs map directly to systolic arrays. If we are concerned with the design of systolic arrays, we must first transform the specifying recurrence equations to UREs and then add control UREs in order to obtain the control flow.

Finally, the control UREs presented here work for systolic arrays of r dimensions ($1 < r < n$). The presentation of the propagation control UREs is based on the space-time diagram for one-dimensional arrays. The space-time diagram for an r-dimensional array is just the combination of the r-dimensional processor space $\mathcal{P}$ and the one-dimensional scheduling space $\mathcal{T}$; it is of dimension $r+1$. However, the propagation control UREs can be greatly simplified in the case of $(n-1)$-dimensional systolic arrays, since valid space-time mappings are injective over $\mathbb{Z}^n$. This has the following two consequences. First, the hierarchical decomposition of the index space, as in Sect. 6.4, in order to construct the propagation control for n-dimensional convex polytopes is not necessary. The propagation control UREs that are presented with respect to three-dimensional convex polytopes apply for n-dimensional ones. Second, the construction of the initialisation control variables can be simplified. We need to associate only one initialisation control variable with every block in Ⓕ. The construction of the initialisation control variables can be conducted in exactly the same way as that of the termination control variables. This halves the number of initialisation control variables required.

References

[1] P. Quinton and V. van Dongen. The mapping of linear recurrence equations on regular arrays. *J. VLSI Signal Processing*, 1(2):95–113, Oct. 1989.

[2] S. V. Rajopadhye and R. M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing*, 14(2):163–189, June 1990.

[3] Y. Wong and J. M. Delosme. Transformation of broadcasts to propagations in systolic algorithms. *J. Parallel and Distributed Computing*, 14(2):121–145, Feb. 1992.

[4] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, 71(1):113–120, Jan. 1983.

[5] S. K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Department of Electrical Engineering, Stanford University, Oct. 1985.

[6] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc. 11th Ann. Int. Symp. on Computer Architecture*, pages 208–214. IEEE Computer Society Press, 1984.

[7] H. T. Kung and M. S. Lam. Wafer-scale integration and two-level pipelined implementations. *J. Parallel and Distributed Computing*, 1(1):33–63, Aug. 1984.

[8] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, 1970.

[9] P. Lee and Z. Kedem. Synthesizing linear-array algorithms from nested for loop algorithms. *IEEE Trans. on Computers*, C-37(12):1578–1598, Dec. 1988.

[10] I. Ramakrishnan and P. Varman. Modular matrix multiplication on a linear array. *IEEE Trans. on Computers*, C-33(11):952–958, Nov. 1984.

[11] J. Xue and C. Lengauer. Specifying control signals for one-dimensional systolic arrays by uniform recurrence equations. In P. Quinton and Y. Robert, editors, *Algorithms and Parallel VLSI Architectures II*, pages 181–187. Elsevier (North-Holland), 1991.

[12] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14(3):563–590, July 1967.

[13] J. Xue. Specifying control signals for systolic arrays by uniform recurrence equations. *Parallel Processing Letters*, 1(2):83–93, 1992.

[14] J. Xue. *The Formal Synthesis of Control Signals for Systolic Arrays.* PhD thesis, Department of Computer Science, University of Edinburgh, July 1992.

[15] J. Bu, E. F. Deprettere, and P. Dewilde. A design methodology for fixed-size systolic arrays. In S. Y. Kung and E. E. Swartzlander, editors, *Application Specific Array Processors*, pages 591–602. IEEE Computer Society Press, 1990.

[16] A. Darte. Regular partitioning for synthesizing fixed-size systolic arrays. *Integration*, 12(3):293–304, Dec. 1991.