The Systematic Derivation of Control Signals for Systolic Arrays

by

Jingling Xue and Christian Lengauer

LFCS Report Series ECS-LFCS-91-152

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

May 1991

The Systematic Derivation of Control Signals for Systolic Arrays

JINGLING XUE[†, ‡] CHRISTIAN LENGAUER[‡]
xj@lfcs.ed.ac.uk lengauer@lfcs.ed.ac.uk

LABORATORY FOR FOUNDATIONS OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF EDINBURGH
EDINBURGH EH9 3JZ
SCOTLAND

ECS–LFCS–91–152
MAY 1991

Abstract

Starting with a system of uniform recurrence equations for a systolic design, we provide a constructive method that allows a specification of control signals for systolic arrays by another system of uniform recurrence equations. An application of the standard space-time mapping technique to these two systems of equations delivers a specification of a systolic array; essentially, it describes the flow of data (i.e., operands) and control signals (i.e., instructions) in both space and time.

[‡]Supported by a University of Edinburgh Postgraduate Fellowship.
[‡]Supported by the Science and Engineering Research Council, Grant no. GR/G55457.

Contents

1 Introduction

Systems of recurrence equations map especially well to systolic arrays. Formal methods on the systematic synthesis of systolic arrays from recurrence equations have been proposed. They focus mainly on the derivation of data flow. The underlying technique is to represent the recurrence equations by a data dependence graph and transform the graph to one that describes a systolic array. This transformation completely describes the flow of data in the array. For details, we refer to [15, 20] and references therein.

A system of recurrence equations may be heterogeneous in the sense that variables of the system may not be defined for a common index space. Examples are Gauss-Jordan elimination, LU-decomposition and dynamic programming. Consequently, cells may have to perform different computations at different steps. A control mechanism is needed that specifies the appropriate computation at every step. We implement this control mechanism by the pipelining of control signals through the array.

Our method accepts uniform recurrence equations (UREs) and delivers a specification of control signals. We proceed in two steps. First, we construct a new system of uniform recurrence equations, called *uniform control recurrence equations* (UCREs), whose variables represent control signals and whose equations represent the control program for a cell. Second, we apply the standard space-time mapping technique to both the source UREs and the UCREs. This derives both data and control flow in a unified manner. Informally, the flow of data corresponds to the flow of operands, and the flow of control signals corresponds to the flow of operators (i.e., instructions). The control signals received by a cell at a particular step completely specify the action of the cell at that step.

2 The Source: A System of UREs

$\mathbb{Z}$ and $\mathbb{Q}$ denote the set of integers and rationals. Let S be $\mathbb{Z}$ or $\mathbb{Q}$. S^+ denotes the positive subset, S_0^+ the non-negative subset and S^n the n-fold Cartesian product of S.

We assume that the reader is familiar with the basic properties of UREs [6, 14, 20]. We use the following notation. We write a URE in the format: domain predicate $\longrightarrow$ recurrence equation [15]. The domain for which variable v is defined is denoted Φ_v. The index space is denoted Φ; it must be a polyhedron with at most one extreme ray [12, Sect. I.4].

We will refer to the dimension of an index space (or a subdomain of it). A set of points $x_1, \cdots, x_k \in \mathbb{Q}^n$ is *affinely independent* if the unique solution of $(\sum\ i : 0 < i \leqslant k : \alpha_i x_i = 0)$ and $(\sum\ i : 0 < i \leqslant k : \alpha_i = 0)$ is $(\forall\ i : 0 < i \leqslant k : \alpha_i = 0)$. A polyhedron $P \subset \mathbb{Q}^n$ is of dimension k, if the maximum number of affinely independent points in P is $k+1$ [12, Sect. I.4].

We assume that all variables are fully indexed, i.e., have the same dimension. We represent the source UREs by (Φ, D) and presume that D is of size $n \times k$, i.e., the UREs are n-dimensional and D consists of k data dependence vectors of length n. We sometimes refer to a variable appearing in the source UREs as a *data variable*.

We use the conventional concept of a data dependence graph and a data dependence matrix [20]. When we write $\vartheta_i \in D$, we mean that dependence vector ϑ_i is the i-th column of D. For notational convenience, we assume that, for each variable name, there is only one associated dependence vector, denoted ϑ_v. We say that v the variable associated with ϑ_v;

$v(I)$ $(I \in \Phi_v)$ are the elements constituting the variable.

If I is inside and $I-\vartheta_v$ is outside the index space, then I is called an *input point* of v and variable $v(I-\vartheta_v)$ is called an *input variable.* If I is inside and $I+\vartheta_v$ is outside the index space, then I is called an *output point* of v and variable $v(I)$ is called an *output variable.* The set of input (output) points of v is called the *input* (*output*) *space* of v. The input (output) space (of the source UREs) is the union of the input (output) spaces of all variables.

Definition 1 The input space $\mathsf{in}(\Phi, \vartheta_v)$ and output space $\mathsf{out}(\Phi, \vartheta_v)$ of v are given by:

$$\begin{array}{rcl} \mathsf{in}(\Phi,\vartheta_v) & = & \{I \mid I\in\Phi,\ I-\vartheta_v \notin \Phi\} \\ \mathsf{out}(\Phi,\vartheta_v) & = & \left\{ \begin{array}{ll} \{\} & \text{if } \vartheta_v \text{ is parallel to the extreme ray} \\ \{I \mid I\in\Phi,\ I+\vartheta_v \notin \Phi\} & \text{otherwise} \end{array} \right. \end{array}$$

The set of index vectors of input variables is: $\Phi_v^{\text{in}} \;=\; \{I \mid I = J-\vartheta_v,\ J \in \mathsf{in}(\Phi,\vartheta_v)\}$. □

Def. 1 implies that input variables are used and output variables are defined at the boundary of the index space. If this is not the case, they can be made so by adding pipelining equations [19]; a *pipelining equation* is of the form $v(I) = v(I-\vartheta_v)$. If ϑ_v is parallel to the extreme ray of the index space, the output variables of v are not defined by the source UREs. Thus, $\mathsf{out}(\Phi, \vartheta_v)$ is defined to be empty.

Not every variable needs to be initialized externally. Similarly, the output of a variable may not be of interest. The concept of input and output point, as defined here, is syntactic rather than semantic.

An equation whose right-hand side is an input value is an *input equation* and an equation whose left-hand side is an output value is an *output equation.* An equation that is neither an input nor an output equation is a *computation equation* (This is different from the concepts with the same name in [15]).

For convenience, we write $I \xrightarrow{v} J$ if $(\exists\ m : m \in \mathbb{Z}_0^+ : J = I + m\vartheta_v)$, i.e., if $J-I$ is a non-negative integral multiple of ϑ_v. We write $\vartheta_v \parallel \vartheta_w$ if $(\exists\ m : m \in \mathbb{Q} : \vartheta_v = m\vartheta_w)$, i.e., if two n-vectors ϑ_v and ϑ_w are parallel and $\vartheta_v \nparallel \vartheta_w$ otherwise.

Example: 1-D Convolution

Given two input sequences $(\forall\ i : 0 < i \leqslant m : x_i)$ and $(\forall\ i : 0 < i \leqslant m : w_i)$, 1-D convolution computes an output sequence $(\forall\ i : 0 < i < 2m : y_i)$ where y_i is given by:

Specification: $(\forall\ i : 0<i<2m :\ y_i = \sum\ k : 0<k\leqslant m \wedge 0 \leqslant i-k < m : x_{i-k+1} w_k)$

UREs:

Input Equations:

$$\begin{array}{rrcrcl} 0\leqslant i<m, & 0=k & \rightarrow & X(i,k) & = & x_i \\ m\leqslant i \leqslant 2m-2, & 0=k & \rightarrow & X(i,k) & = & 0 \\ 0=i, & 0<k<m & \rightarrow & X(i,k) & = & 0 \\ 0<i<2m, & 0=k & \rightarrow & Y(i,k) & = & 0 \\ 0=i, & 0<k\leqslant m & \rightarrow & W(i,k) & = & w_i \end{array}$$

Output Equation:

$$\begin{array}{rrcrcl} 0<i<2m, & m=k & \rightarrow & y_i & = & Y(i,k) \end{array}$$

Computation Equations:

$$\begin{array}{lcl} 0<i<2m,\ 0<k\leqslant m \rightarrow X(i,k) &=& X(i-1,k-1) \\ 0<i<2m,\ 0<k\leqslant m \rightarrow Y(i,k) &=& Y(i,k-1)+X(i-1,k-1)W(i-1,k) \\ 0<i<2m,\ 0<k\leqslant m \rightarrow W(i,k) &=& W(i-1,k) \end{array}$$

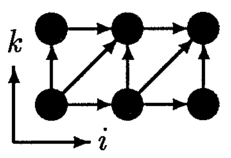
Index Space: $\Phi = \{(i,k) \mid 0<i<2m,\ 0<k\leqslant m\}$

Variables: X, Y, W

Data Dependence Matrix: $D = [\vartheta_X, \vartheta_Y, \vartheta_W] = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

$$\begin{array}{lrcl} \textit{Input Spaces:} & \mathsf{in}(\Phi,\vartheta_X) &=& \{(i,k) \mid 0<i<2m,\ 1=k\} \cup \\ & && \{(i,k) \mid 1=i,\ 1<k\leqslant m\} \\ & \mathsf{in}(\Phi,\vartheta_Y) &=& \{(i,k) \mid 0<i<2m,\ 1=k\} \\ & \mathsf{in}(\Phi,\vartheta_W) &=& \{(i,k) \mid 1=i,\ 0<k\leqslant m\} \\ \textit{Output Space:} & \mathsf{out}(\Phi,\vartheta_Y) &=& \{(i,k) \mid 0<i<2m,\ k=m\} \end{array}$$

Data Dependence Graph ($m=2$):



□

3 The Model

A systolic array model that has been the basis of many methods [2, 5, 15, 16, 19] is defined in [20].

Definition 2 A *systolic array* is a network of cells that are placed at the grid points of a finite lattice $\mathcal{P} \in \mathbb{Z}^{n-1}$, satisfying the following properties:

1. The array is driven by a global clock, which ticks in unit time. A cell is active in every cycle.

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

3. Postulate the existence of a directed connection from the cell at location $\ell \in \mathcal{P}$ to the cell at location $\ell+d$, for some constant vector $d \in \mathbb{Z}^{n-1}$. This postulate is either true for all $\ell \in \mathcal{P}$ or false for all $\ell \in \mathcal{P}$. A directed connection is also called a *channel*; it is an *input* channel to the cell at its destination and an *output* channel to the cell at its source.

4. If a cell receives a value (it may be the undefined value $\perp$) on an input channel at time t, then it must receive a value on the same channel and send a value on the corresponding output channel at time $t+1$. □

Stats. 1 and 2 are generally assumed implicitly. This model does not consider limitations of resources such as the size and topology of the array and its connections; it assumes that there are as many resources as required.

4 The Target: A Systolic Array

4.1 The Space-Time Mapping

The synthesis of systolic arrays amounts to finding a suitable linear index transformation, called *space-time mapping*, to the index space [15, 20]. In the transformed UREs, one index represents time and the remaining indices represent processor coordinates.

Definition 3 Assume that the evaluation of a point in the index space takes unit time. A space-time mapping consists of two components: step and place. Useful functions defined in terms of step and place are flow and pattern.

1. $\mathsf{step} : \Phi \longrightarrow \mathbb{Z}$, $\mathsf{step}(I) = \lambda I$, $\lambda \in \mathbb{Z}^n$. step specifies the temporal distribution. λ is the *scheduling vector.* $\{\lambda I \mid I \in \Phi\}$ is the *scheduling space.* I is computed at step λI.

2. $\mathsf{place} : \Phi \longrightarrow \mathbb{Z}^{n-1}$, $\mathsf{place}(I) = \sigma I$, $\sigma \in \mathbb{Z}^{(n-1)\times n}$. place specifies the spatial distribution. σ is the *allocation matrix.* $\mathcal{P} = \{\sigma I \mid I \in \Phi\}$ is the *processor space.* I is computed at cell σI.

3. $\mathsf{flow} : V \longrightarrow \mathbb{Q}^{n-1}$, $\mathsf{flow}(v) = \sigma\vartheta_v / \lambda\vartheta_v$, $\vartheta_v \in D$. V denotes the set of variable names. flow specifies the velocity at which elements of a variable travel at each step. Variable v is called *moving* if $\mathsf{flow}(v) \neq 0$ and *stationary* if $\mathsf{flow}(v) = 0$. $\sigma\vartheta_v$ represents the direction and length of a connecting channel at a cell for variable v; the number of delay buffers associated with that channel is $\lambda\vartheta_v - 1$.

4. $\mathsf{pattern} : V \longrightarrow (\Phi^{\text{in}} \longrightarrow \mathbb{Z}^{(n-1)\times n})$, $\mathsf{pattern}(v(I)) = \mathsf{place}(I) - (\mathsf{step}(I) - t_{\text{fst}})\mathsf{flow}(v)$. $\Phi^{\text{in}} = (\cup\ v : v \in V : \Phi^{\text{in}}_v)$. t_{fst} is the first step number. pattern specifies the location of variables in the processor space at the first step. $\square$

Interpreted geometrically, the processor space is obtained by a projection, namely, by eliminating one dimension from the index space along a chosen direction called the *projection vector* and denoted by $u \in \mathbb{Z}^n$ with $\sigma u = 0$ [20]; all points that are in a line parallel to the projection vector are assigned the same processor location.

In a more general form of space-time mapping, an *affine index transformation*, step is defined by $\mathsf{step}(I, v) = \lambda I + \alpha_v$, where $\alpha_v \in \mathbb{Z}$ [15, 20]. The existence of an affine index transformation guarantees the existence of a linear index transformation, by means of an appropriate translation of the index space. We use linear index transformations for notational convenience. We sometimes write Π for step and place in combination:

$$\Pi = \begin{bmatrix} \lambda \\ \sigma \end{bmatrix}$$

Not every space-time mapping describes a systolic array. A space-time mapping is *valid* if it satisfies the following two mapping requirements [15, 20]:

1. Precedence Constraint: $(\forall\ v : v \in V : \lambda\vartheta_v \geq 1)$.

2. Computation Constraint:
$$(\forall\ I, J : I, J \in \Phi\ \wedge\ I \neq J : \mathsf{place}(I) = \mathsf{place}(J) \implies \mathsf{step}(I) \neq \mathsf{step}(J)).$$

If the precedence constraint is satisfied, the step number assigned to a target variable is smaller than the step numbers assigned to its arguments. This preserves the semantics of the source UREs. If the computation constraint is satisfied, two distinct points of the index space cannot not be scheduled simultaneously at the same cell. The computation constraint is satisfied, if Π is non-singular.

Only in trivial cases, a valid space-time mapping Π may be singular, i.e., $\text{rank}(\Pi) < n$. In what follows, we shall assume that Π is non-singular, i.e., $\text{rank}(\Pi) = n$. Hence, Π is a bijection from $\mathbb{Q}^n$ to $\mathbb{Q}^n$. A bijective space-time mapping has many useful properties. For example, it preserves the semantics of the UREs and the convexity of the index space.

4.2 Input and Output Extension

There are space-time mappings that map input and output points to internal cells. This is costly because non-local connecting channels and a large number of I/O ports may be needed. To impose the restriction of border communication, the index space is extended in such a way that the resulting input and output points are all mapped to border cells.

The basic idea is to replicate a dependence vector in the data dependence graph forward and backward until the images of points so created are outside the processor space $\mathcal{P}$ [19]. This extension depends on the place function and works only for moving variables. We refer to [8] for some thoughts on tackling stationary variables.

Definition 4 The *extended index space* Ψ is defined as follows (see the solid lines of Fig. 1(a)):

$$\begin{array}{lcl}
\Psi^{\text{s}}_{v} & = & \{I \mid \sigma I \in \mathcal{P},\ I \notin \Phi,\ J \in \Phi, I \overset{v}{\rightarrow} J\} \\
\Psi^{\text{d}}_{v} & = & \{I \mid \sigma I \in \mathcal{P},\ I \notin \Phi,\ J \in \Phi, J \overset{v}{\rightarrow} I\} \\
\Psi^{\text{P}} & = & (\cup\ v : v \in V : \Psi^{\text{s}}_{v} \cup \Psi^{\text{d}}_{v}) \\
\Psi^{\perp}_{v} & = & \Psi^{\text{P}} \backslash (\Psi^{\text{s}}_{v} \cup \Psi^{\text{d}}_{v}) \\
\Psi_{v} & = & \Psi^{\text{s}}_{v} \cup \Phi \cup \Psi^{\text{d}}_{v} \\
\Psi & = & \Psi^{\text{P}} \cup \Phi
\end{array}$$

The points of Ψ^{s}_{v} are called *soaking points* of v; they are generated in the extension of v along direction $-\vartheta_v$. The points of Ψ^{d}_{v} are called *draining points* of v; they are generated in the extension of v along direction ϑ_v. Ψ^{P} contains the soaking and draining points of all variables; they are called *pipelining points.* The points of $\Psi^{\perp}_{v}$ are called *undefined points* of v; they are generated in the extension of the other variables. The points of Φ are called *computation points.* Ψ^{s}_{v}, Ψ^{d}_{v} and $\Psi^{\perp}_{v}$ are called the *soaking space*, *draining space* and *undefined space* of v, respectively. Ψ_v is the new domain for which v is defined. □

Note that the definition of soaking, draining and undefined points is tied to a particular variable. For example, a pipelining point that is a soaking point for one variable may be either a soaking point or a draining point or an undefined point for another variable. The soaking points of v serve to propagate its input values from border cells to internal cells. The draining points of v serve to propagate its output values from internal cells to border cells. A variable is undefined at its undefined points; the undefined points serve to propagate $\perp$. Since $\{\Psi^{\text{s}}_{v}, \Psi^{\text{d}}_{v}, \Psi^{\perp}_{v}\}$, for any fixed v, is a partitioning of the pipelining points Ψ^{P}, all variables are defined at pipelining points by pipelining equations.

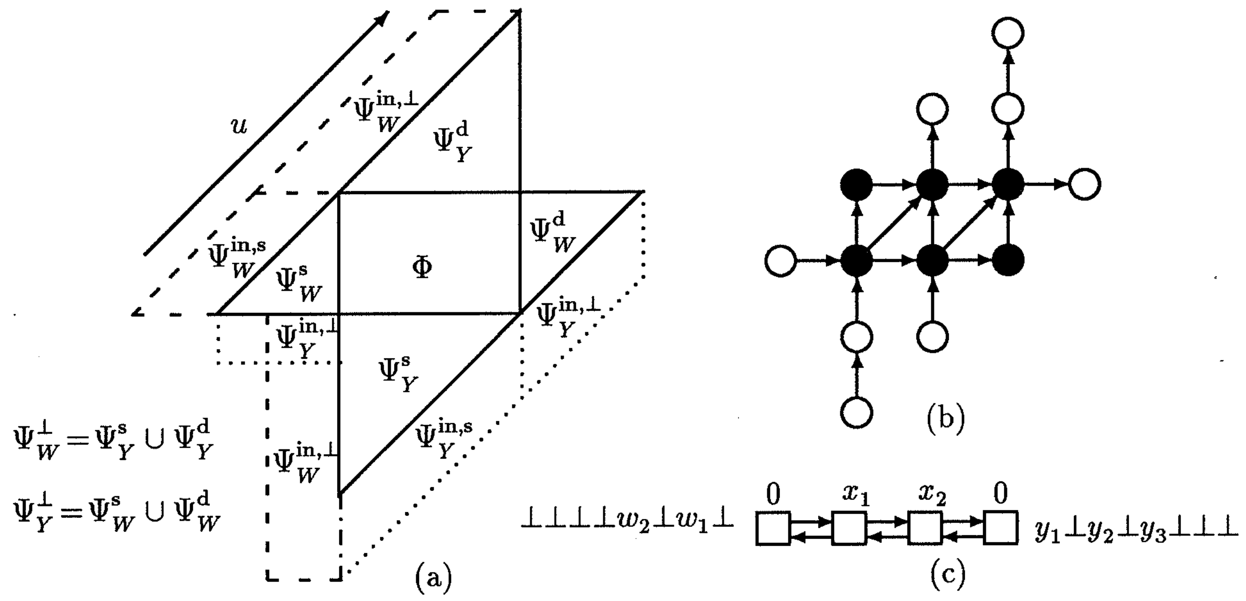

u
$\Psi_W^{\mathrm{in},\perp}$
Ψ_Y^{d}
$\Psi_W^{\mathrm{in,s}}$
Ψ_W^{s}
Φ
Ψ_W^{d}
$\Psi_Y^{\mathrm{in},\perp}$
Ψ_Y^{s}
$\Psi_Y^{\mathrm{in},\perp}$
$\Psi_W^{\mathrm{in},\perp}$
$\Psi_Y^{\mathrm{in,s}}$
$\Psi_W^{\perp} = \Psi_Y^{\mathrm{s}} \cup \Psi_Y^{\mathrm{d}}$
$\Psi_Y^{\perp} = \Psi_W^{\mathrm{s}} \cup \Psi_W^{\mathrm{d}}$
(a)
(b)
$0 \quad x_1 \quad x_2 \quad 0$
$\perp\perp\perp\perp w_2 \perp w_1 \perp$
$y_1 \perp y_2 \perp y_3 \perp\perp\perp$
(c)

Figure 1: (a) The extended index space for 1-D convolution with respect to projection vector $u=(1,1)$ is enclosed by solid lines. Some relevant subdomains of Y and W are depicted; X is stationary and is not depicted. (b) The data dependences of the extended index space for size $m=2$. Circles represent pipelining points and dots computation points. Take variable Y as an example. There are five (vertical) ϑ_Y-paths: the two near the boundaries (left and right) each contain only one, undefined point of Y, the other three paths contain computation points. (c) The systolic array for size $m=2$ and the distribution of data at the first step.

If ϑ_v is parallel to the projection vector, v is a stationary variable; the naive extension by Def. 4 does not help. In this special case, $-\vartheta_v$ is an extreme ray of Ψ_v^{s}, ϑ_v is an extreme ray of Ψ_v^{d} and the definitions of Ψ_v^{s} and Ψ_v^{d} can be used for the construction of UCREs (Sect. 7.1).

By Def. 1, $\mathsf{in}(\Psi, \vartheta_v)$ ($\mathsf{out}(\Psi, \vartheta_v)$) is the input (output) space of v in Ψ. We call it the *new* input (output) space, and its points the *new* input (output) points. Ψ_v^{in} denotes the set of index vectors of the input variables in the extended index space (the *new* input variables). It can be partitioned into two subsets (see the dashed and dotted lines of Fig. 1(a)).

- One subset, denoted $\Psi_v^{\mathrm{in,s}}$, is generated in the extension of v:
$$\Psi_v^{\mathrm{in,s}} \quad = \quad \{I \mid I \xrightarrow{v} J,\ I \in \Psi_v^{\mathrm{in}},\ J \in \Psi_v^{\mathrm{s}}\}.$$
where the I results from the extension of the J along direction $-\vartheta_v$; $v(I)$ is now initialized with the same value as $v(J)$, and

- the complement of $\Psi_v^{\mathrm{in,s}}$ in Ψ_v^{in}, denoted $\Psi_v^{\mathrm{in},\perp}$, is generated in the extension of the other variables:
$$\Psi_v^{\mathrm{in},\perp} \quad = \quad \{I \mid I \xrightarrow{v} J,\ I \in \Psi_v^{\mathrm{in}},\ J \in \Psi_v^{\perp}\}.$$
v is undefined at all points in this set: $I \in \Psi_v^{\mathrm{in},\perp} \rightarrow v(I) = \perp$.

Next, we introduce the concept of a ϑ_v-path. It reveals the structure of the extended index space or, more precisely, the distribution of pipelining points relative to computation points.

Definition 5 Consider an integral polyhedron $S \subset \mathbb{Z}^n$. A *ϑ_v-path* consists of all points $I \in S$ with the following property: the index difference between I and its direct predecessor in the path is ϑ_v. We write $p(I, v, S)$ for the (unique) ϑ_v-path that passes through point I, and $P(v, S)$ for the set of all ϑ_v-paths: $P(v, S) = \{p(I, v, S) \mid I \in S\}$. The union of all ϑ_v-paths, for any fixed v, is S, i.e., $P(v, S)$ partitions S. □

There are two types of ϑ_v-paths (Fig. 1(b)):

- A path that contains computation points. By Def. 4, this path cannot contain undefined points of v. It can be viewed as consisting of three segments:
  - The first segment, called the *soaking path* and denoted $s(I, v, \Psi)$, consists of the soaking points. The first point is a new input point in the new input space $\mathsf{in}(\Psi, \vartheta_v)$.
  - The second segment, called the *computation path* and denoted $c(I, v, \Psi)$, consists of the computation points. The first (last) point in the input (output) space $\mathsf{in}(\Phi, \vartheta_v)$ ($\mathsf{out}(\Phi, \vartheta_v)$) is called the *first* (*last*) computation point (of this path).
  - The third segment, called the *draining path* and denoted $d(I, v, \Psi)$, consists of the draining points. The last point is a new output point in the new output space $\mathsf{out}(\Psi, \vartheta_v)$.
- A path that contains no computation points. By Def. 4, this path can only contain undefined points of v.

Example: 1-D Convolution

Choose the following space-time mapping:

$$\Pi = \begin{bmatrix} \lambda \\ \sigma \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The projection vector is $u = (1, 1)$. See Fig. 1. Without input extension, the input values $(\forall\ i : 0 < i < m : w_i)$ and $(\forall\ i : 0 < i \leqslant 2m-2 : y_i)$ are injected to internal cells. Without output extension, the output values $(\forall\ i : 2 \leqslant i < 2m : y_i)$ are ejected from internal cells. Note that variable X is stationary; its elements are loaded before step t_{fst}. □

5 Uniform Control Recurrence Equations

A valid space-time mapping ensures that the right data arrive at the right cell at the right time. What remains unspecified are control signals that instruct a cell to perform the right computation at the right time when the right data are available. We shall specify these control signals by a system of UCREs.

UCREs are UREs some of whose equations may have data-dependent predicates, i.e., may be of the form:

$$\begin{array}{rcll} v(I) & = & \mathbf{if}\, B_0(w(I-\vartheta_w),\cdots) & \rightarrow\; f_0(v(I-\vartheta_v)) \\ & & [\!]\; B_1(w(I-\vartheta_w),\cdots) & \rightarrow\; f_1(v(I-\vartheta_v)) \\ & & \quad\vdots & \\ & & [\!]\; B_{r-1}(w(I-\vartheta_w),\cdots) & \rightarrow\; f_{r-1}(v(I-\vartheta_v)) \\ & & \mathbf{fi} & \end{array}$$

v and w are called *control variables*, and ϑ_v and ϑ_w are their associated *control dependence vectors*. Each control variable is associated with a unique control dependence vector. $v(I)$ represents a control signal (i.e., control value) at point I.

$B_i(w(I-\vartheta_w),\cdots) \rightarrow f_i(I-\vartheta_v)$ is called a *guarded command* [3]. The *guard* $B_i(w(I_w-\vartheta_w),\cdots)$ is a Boolean expression; the three dots stands for an arbitrary fixed number of arguments. A guard can always be written in disjunctive normal form, where each disjunct consists of a conjunction of comparisons of an argument with a control signal. $f_i(v(I-\vartheta_v))$ is a function that recursively defines the control value of $v(I)$ in terms of argument $v(I-\vartheta_v)$.[2]

The evaluation of guards is deterministic. We require that one and only one guard evaluates to true at a point.

6 The Specification of a Control Variable

A control variable, v, is specified by two types of equations:

- One or more input equations specify the initialization of the variable by control signals.
- One computation equation defines the value of variable $v(I)$ in terms of the value of variable $v(I-\vartheta_v)$.

The output of control signals is irrelevant. The specification of a control variable, v, proceeds in three steps:

- Choose a constant non-zero n-vector as the control dependence vector ϑ_v.
- Define the input equations.
- Define the computation equation.

Several factors influence the choice of control dependence vector:

- We may introduce control dependence vectors such that no step function exists for the source UREs and UCREs. When the index space is bounded, the consistency of dependence vectors with a step function can be checked by procedures described in [19]; essentially, one looks for the absence of cycles in the dependence graph.
- The control dependence vector must satisfy certain requirements to guarantee the correctness of the control flow (Thms. 2 and 4 of Sect. 7.1.3 and Def. 13 of Sect. 7.2.2).

[2]An extension would be to include the other control variables as arguments in the defining equation of a target control variable.

- In general, we shall choose one of the data dependence vectors for ϑ_v, i.e., $\vartheta_v \in D$; we write $\ell.v$ for the corresponding data variable. This implies $\mathsf{flow}(v) = \mathsf{flow}(\ell.v)$. We say that we use control variable v to *label* data variable $\ell.v$. There are two obvious advantages. First, a step function that is valid with respect to the source UREs is also valid with respect to UCREs. Second, the latency (i.e., the total number of computation steps), of the systolic array is preserved.

- The choice of control dependence vector has an impact on the hardware cost of the systolic array. It influences, e.g., the number of channels, the number of buffers associated with the respective channels, and the bit width of a channel. At present, our method does not take limitations of these resources into account.

A variable is called a *pipelining variable* if its computation equation is a pipelining equation. Note that pipelining equations are not data-dependent.

7 The Derivation of Control Signals

We say that two computation points have the same *type* if, for every data variable of the source UREs, the defining equations at both points are the same.

The basic idea in constructing control signals is to distinguish pipelining and computation points and further distinguish different types of computation points. We use two types of control variables:

1. *separation control variables* distinguish computation from pipelining points, and

2. *computation control variables* distinguish different types of computation points.

We derive the computation control variables, for computation points only (i.e., for the original index space), with a construction that provides UCREs and that is proved correct in Sect. 7.2. The separation control variables must be for computation and pipelining points (i.e., for the extended index space). Because the generation of pipelining points depends on the space-time mapping (Def. 4), we have two options:

- We can derive the separation control variables with a construction that, consequently, also depends on the space-time mapping.

- We can define explicitly UCREs for separation control that are independent of the space-time mapping and, as it happens, also of the source UREs (Sect. 7.1.2).

We prefer the latter. To make the UCREs for separation control correct, we must choose their control dependence vectors wisely (Sect. 7.1.3). This may entail an extension of the index space (Sect. 7.1.4).

7.1 The Specification of Separation Control Variables

First, we introduce a correctness criterion for the separation control variables. We write $\varphi(I)$ for the vector of control signals at point I; each component of the vector corresponds to one separation control variable:

$$\varphi(I) \;=\; (v(I-\vartheta_v), \cdots)$$

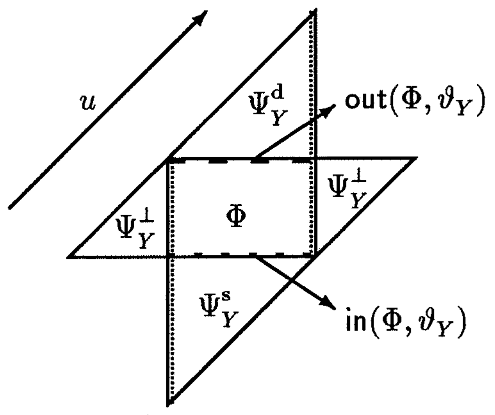


Figure 2: The partitioning $\{\Psi^{\mathrm{s}}_{Y}, \Psi^{\mathrm{d}}_{Y}, \Psi^{\perp}_{Y}, \Phi\}$ of the extended index space for 1-D convolution. There are three types of boundaries; they are highlighted by different dashed lines.

Definition 6 *The separation control variables are correct if control signals at a computation point and a pipelining point differ, i.e., if*

$$(\forall\ I, J : I \in \Psi^{\mathrm{p}} \wedge J \in \Phi : \varphi(I) \neq \varphi(J)) \qquad \square$$

7.1.1 Initialization, Termination and Evolution Control Variables

For any data variable v, $\{\Psi^{\mathrm{s}}_{v}, \Psi^{\mathrm{d}}_{v}, \Psi^{\perp}_{v}, \Phi\}$ is a partitioning of the extended index space Ψ. To distinguish computation and pipelining points, it suffices to distinguish these four partitions. To distinguish these four partitions, it suffices to identify their common boundaries (Fig. 2). This is accomplished by the pipelining of control signals. The choice of v is of no consequence to the correctness of the separation control variables.

We write $\mathcal{S}(v)$ for the set of one-bit control signals of control variable v; channel $\sigma\vartheta_v$ must have a width of $\lceil \log_2 |\mathcal{S}(v)| \rceil$ bits. There are three separation control variables F, L and E (See Fig. 2 by considering $\ell.E = Y$):

- The *initialization control variable*, F. It is a pipelining variable. Together with E, it identifies the first computation points of E, i.e., the boundary $\mathsf{in}(\Phi, \vartheta_E)$ between Ψ^{s}_{E} and Φ. $\mathcal{S}(F) = \{f, \perp\}$.
- The *termination control variable*, L. It is a pipelining variable. Together with E, it identify the last points computation of E, i.e., the boundary $\mathsf{out}(\Phi, \vartheta_E)$ between Ψ^{d}_{E} and Φ. $\mathcal{S}(L) = \{\ell, \perp\}$.
- The *evolution control variable*, E. The computation equation of E is data-dependent; it depends on F and L. Beside the rôles mentioned previously, E by itself identifies the boundary between $\Psi^{\perp}_{E}$ and the remaining three subdomains. $\mathcal{S}(E) = \{e_{\mathrm{s}}, e_{\mathrm{c}}, e_{\mathrm{d}}, \perp\}$.

Now, we can explicitly define the set of control signals $\varphi(I)$ at point I:

$$\varphi(I) = (F(I - \vartheta_F), L(I - \vartheta_L), E(I - \vartheta_E))$$

Separation control variable $v \in \{F, L, E\}$ can always be interpreted to label a data variable:

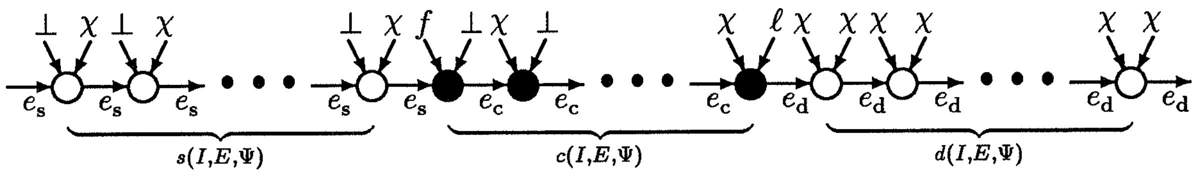

⊥ χ ⊥ χ
⊥ χ f ⊥ χ ⊥
χ ℓ χ χ χ χ
χ χ
e_s e_s e_s e_s e_s e_c e_c e_c e_d e_d e_d e_d e_d
$s(I,E,\Psi)$
$c(I,E,\Psi)$
$d(I,E,\Psi)$

Figure 3: Illustration of the construction of the separation control variables. Fat dots are computation points and circles are pipelining points. The relevant control signals at a point are shown. Arrows pointing southeast are for ϑ_F, arrows pointing southwest for ϑ_L and arrows pointing east for ϑ_E.

- The domain of the input equations is Ψ_v^{in}, and
- The domain of the computation equation is Ψ.

If it is not already, the following augmentation of the source UREs makes $\ell.v$ a data variable:

$$\begin{array}{rcl} I \in \Phi_{\ell.v}^{\text{in}} & \rightarrow & \ell.v(I) = \perp \\ I \in \Phi & \rightarrow & \ell.v(I) = \ell.v(I - \vartheta_v) \end{array}$$

The augmented UREs may incur more hardware cost and a higher latency (Sect. 5).

7.1.2 The Construction of Input and Computation Equations

In this section, we construct the UCREs for the separation control variables for an arbitrary choice of control dependence vectors. Sect. 7.1.3 describes how to choose the control dependence vectors properly. We need the following notations.

Definition 7

$$\begin{array}{rcl} \Lambda_F^{\text{in}} & = & \{I \mid I \xrightarrow{F} J,\ I \in \Psi_F^{\text{in}},\ J \in \mathsf{in}(\Phi, \vartheta_E)\}. \\ \Lambda_L^{\text{in}} & = & \{I \mid I \xrightarrow{L} J,\ I \in \Psi_L^{\text{in}},\ J \in \mathsf{out}(\Phi, \vartheta_E)\}. \\ \Lambda_E^{\text{in}} & = & \{I \mid I \xrightarrow{E} J,\ I \in \Psi_E^{\text{in}},\ J \in \mathsf{in}(\Phi, \vartheta_E)\}. \end{array}$$

Λ_F^{in} (Λ_L^{in}) is the intersection of Ψ_F^{in} (Ψ_L^{in}) and the ϑ_F-paths (ϑ_L-paths) that pass through the points in $\mathsf{in}(\Phi, \vartheta_E)$ ($\mathsf{out}(\Phi, \vartheta_E)$). $\Lambda_E^{\text{in}} = \Psi_E^{\text{in,s}}$ (Sect. 4.2). □

First, we consider the specification of evolution control variable E. Its specification imposes restrictions on and consequently determines the specification of F and L. We distinguish two cases of ϑ_E-paths in the extended index space:

Case 1. Path $p(I, E, \Psi)$ contains computation point (Fig. 3). We initialize E at the first point of $s(I, v, \Psi)$ with e_s, and adopt this value for all soaking points of E. We are going to construct F such that the first computation point of $c(I, E, \Psi)$ is distinguished by $F(I - \vartheta_F) = f$; there, e_s is converted to e_c, the value for all computation points (but the first). We are going to construct L such that the last computation point of $c(I, E, \Psi)$ is distinguished by $L(I - \vartheta_L) = \ell$; there e_c is converted to e_d, the value for all draining points. If path $p(I, E, \Psi)$ contains only one computation point, which is, therefore, both a first and last computation point, e_s changes to e_d at that point.

Case 2. Path $p(I,E,\Psi)$ contains no computation points; it contains only undefined points of E. We initialize at the first point of $p(I,E,\Psi)$ with $\perp$ and propagate it along the entire path. F and L are of no concern here.

In summary: we initialize E at Λ_E^{in} with e_s and at $\Psi_E^{\text{in}} \backslash \Lambda_E^{\text{in}}$ with $\perp$. e_s becomes e_c at the points in $\mathsf{in}(\Phi,\vartheta_E)$ and changes further to e_d at the points in $\mathsf{out}(\Phi,\vartheta_E)$. Hence, the computation equation of E must consist of three guarded commands. It can be viewed as a piecewise pipelining variable. This completes the specification of evolution control variable E.

Now, we consider the specification of initialization control variable F and termination control variable L. To establish $F(I-\vartheta_F)=f$ at the first computation points of E but not at the soaking points of E, it is necessary (but not sufficient) to initialize F at Λ_F^{in} with f and at $\Psi_F^{\text{in}} \backslash \Lambda_F^{\text{in}}$ with $\perp$. Applying the previous specification with this initialization identifies a subset of a set $\mathsf{sig.in}$(sig for "signal"):

$$\mathsf{sig.in} = \{K \mid I \xrightarrow{F} K,\ J \xrightarrow{E} K,\ I \in \Lambda_F^{\text{in}},\ J \in \Lambda_E^{\text{in}},\ K \in \Psi\}$$

$\mathsf{sig.in}$ is also a superset of the set of first computation points of E.

Similarly, to establish $L(I-\vartheta_L)=\ell$ at the last computation points I of E but not at the remaining computation points, we must initialize L at Λ_L^{in} with ℓ and at $\Psi_L^{\text{in}} \backslash \Lambda_L^{\text{in}}$ with $\perp$. This identifies a subset of a set $\mathsf{sig.out}$:

$$\mathsf{sig.out} = \{K \mid I \xrightarrow{L} K,\ J \xrightarrow{E} K,\ I \in \Lambda_L^{\text{in}},\ J \in \Lambda_E^{\text{in}},\ K \in \Psi\}$$

$\mathsf{sig.out}$ is a superset of the set of last computation points of E.

Lemma 1

1. $\mathsf{in}(\Phi,\vartheta_E) \subseteq \mathsf{sig.in} \wedge \mathsf{out}(\Phi,\vartheta_E) \subseteq \mathsf{sig.out}$.

2. $(\forall\, I : I \in \Psi : \varphi(I) = (f,\chi,e_s) \implies I \in \mathsf{sig.in})$.

3. $(\forall\, I : I \in \Psi : \varphi(I) = (f,\ell,e_s) \implies I \in \mathsf{sig.out} \cap \mathsf{sig.in}) \wedge$
$(\forall\, I : I \in \Psi : \varphi(I) = (\chi,\ell,e_c) \implies I \in \mathsf{sig.out} \backslash \mathsf{sig.in})$

Proof. Definitions of $\mathsf{sig.in}$ and $\mathsf{sig.out}$, Def. 1 and the previous construction (Def. 8). □

For sufficiency, we must choose the control dependence vectors ϑ_F, ϑ_L and ϑ_E appropriately. This does not necessarily enforce the identity of $\mathsf{sig.in}$($\mathsf{sig.out}$) with the set of first (last) computation points of E, but our construction will (Sect. 7.1.3).

Because both F and L are pipelining variables, no computation equations are required. This completes the specification of F and L.

Next, we formulate the specification of the separation control variables as UCREs. We use an auxiliary control variable Z, called the *action control variable*, with no associated control dependence vector. Z distinguishes computation and pipelining *points*. It is defined for the extended index space Ψ: $\mathcal{S}(Z)=\{z_c, z_p\}$. If I is a computation point, then $Z(I)=z_c$; if it is a pipelining point, then $Z(I)=z_p$.

Definition 8 (UCREs for the separation control variables over the extended index space)

Evolution control variable E:

$$\begin{array}{rcll}
I\in\Psi_E^{\text{in}}\backslash\Lambda_E^{\text{in}} & \rightarrow & E(I) = & \perp \\
I\in\Lambda_E^{\text{in}} & \rightarrow & E(I) = & e_{\text{s}} \\
I\in\Psi & \rightarrow & E(I) = & \textbf{if } E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f \wedge L(I-\vartheta_L)\neq\ell \;\rightarrow\; e_{\text{c}} \\
 & & & [\!]\ ((E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_{\text{c}}) \\
 & & & \quad \wedge L(I-\vartheta_L)=\ell \;\rightarrow\; e_{\text{d}} \\
 & & & [\!]\ \textbf{else} \;\rightarrow\; E(I-\vartheta_E) \\
 & & & \textbf{fi}
\end{array}$$

Initialization control variable F:

$$\begin{array}{rcll}
I\in\Psi_F^{\text{in}}\backslash\Lambda_F^{\text{in}} & \rightarrow & F(I) = & \perp \\
I\in\Lambda_F^{\text{in}} & \rightarrow & F(I) = & f \\
I\in\Psi & \rightarrow & F(I) = & F(I-\vartheta_F)
\end{array}$$

Termination control variable L:

$$\begin{array}{rcll}
I\in\Psi_L^{\text{in}}\backslash\Lambda_L^{\text{in}} & \rightarrow & L(I) = & \perp \\
I\in\Lambda_L^{\text{in}} & \rightarrow & L(I) = & \ell \\
I\in\Psi & \rightarrow & L(I) = & L(I-\vartheta_L)
\end{array}$$

Action control variable Z:

$$\begin{array}{rcll}
I\in\Psi & \rightarrow & Z(I) = & \textbf{if } (E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_{\text{c}} \;\rightarrow\; z_{\text{c}} \\
 & & & [\!]\ \textbf{else} \;\rightarrow\; z_{\text{p}} \\
 & & & \textbf{fi}
\end{array}$$

□

The first guard of E selects first computation points of E but excludes points that are both first and last computation points of E: $E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f$ holds at a point if the point is a first computation point of E. If it is also a last computation point of E, $L(I-\vartheta_L)=\ell$ holds. The second guard of E establishes whether a computation point is a last computation point of E: $E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f$ holds at the first computation points of E, while $E(I-\vartheta_E)=e_{\text{c}}$ holds at the remaining computation points. Hence, If the original input space and output space are disjoint, i.e., $\mathsf{in}(\Phi,\vartheta_E)\cap\mathsf{out}(\Phi,\vartheta_E) = \{\}$, the first (second) guard of E can be simplified to $E(I-\vartheta_E)=e_{\text{s}} \wedge F(I-\vartheta_F)=f$ $(E(I-\vartheta_E)=e_{\text{c}} \wedge L(I-\vartheta_L)=\ell)$.

Remark.

- $|\,\lambda u\,|$ is the *period* of the array [19]. That is, a cell evaluates either a pipelining or a computation point every $|\,\lambda u\,|$ clock ticks. If $|\,\lambda u\,|$ is not of unit value, there are cells whose operation is at some step undefined. The according input control signals also remain unspecified (Def. 8).

- In Def. 8, $\perp$ is defined at the points in $\Psi_v^{\text{in}}\backslash\Lambda_v^{\text{in}}$ $(v\in\{F,T,E\})$. If v is a data variable, $\perp$ can be ignored. But here, $\perp$ is a meaningful control signal and must be input. Remember that Ψ_v^{in} consists of two subsets $\Psi_v^{\text{in,s}}$ and $\Psi_v^{\text{in},\perp}$ (Fig. 1). The input of $\perp$ for points at $\Psi_v^{\text{in,s}}$ is to border cells. This may not be true for points at $\Psi_v^{\text{in},\perp}$. Here are two possible solutions: the input of $\perp$ can be implemented either by a system reset before step t_{fst} or by a pre-pipelining of $\perp$. The latter can be enforced by an appropriate extension of the extended index space, similar to that of input and output extension (Sect. 4.2). This time, the extension is to enfore the input of control signals at border cells. In the following presentation, we assume that $\perp$ has been appropriately initialized before step t_{fst}. □

Example: 1-D Convolution

We continue to choose the previous space-time mapping (Sect. 4.2). With $\ell.F = \ell.T = W$ and $\ell.E = Y$, we obtain the following by Defs. 1 and 7:

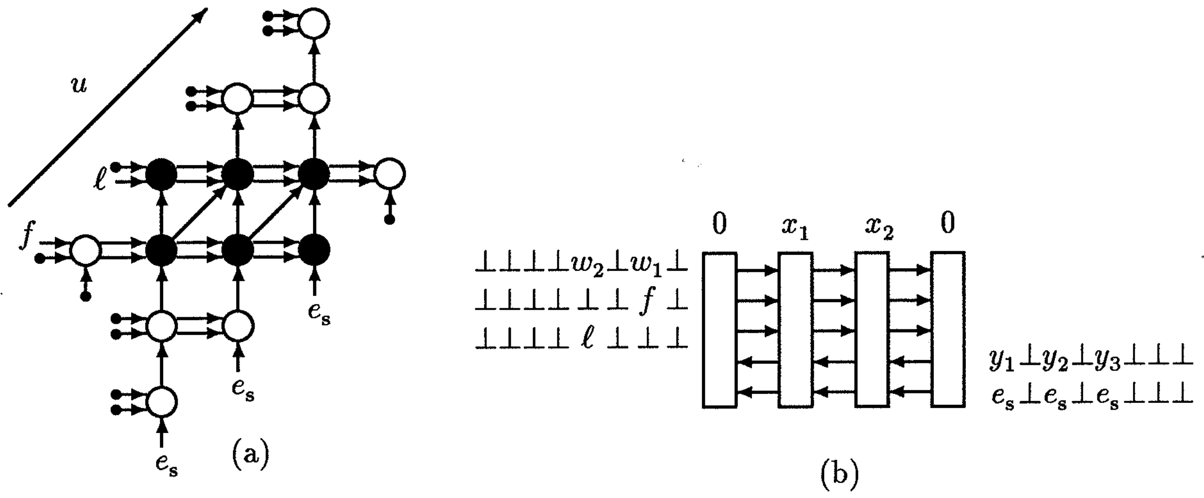
$$\begin{array}{lclll}
\Lambda_F^{\text{in}} & = & \{(i,k) \mid i=-m+1, & k=1 & \} \\
\Psi_F^{\text{in}} & = & \{(i,k) \mid i=0, & -2m+3 \leqslant k \leqslant 0 & \} \cup \\
 & & \{(i,k) \mid -m+1<i<2m-1, & k=i+m & \} \cup \Lambda_F^{\text{in}} \\
\Lambda_L^{\text{in}} & = & \{(i,k) \mid i=0, & k=m & \} \\
\Psi_L^{\text{in}} & = & \{(i,k) \mid i=0, & -2m+3 \leqslant k \leqslant 0 & \} \cup \\
 & & \{(i,k) \mid -m<i<0, & k=i+m & \} \cup \\
 & & \{(i,k) \mid 0<i<2m-1, & k=i+m & \} \cup \Lambda_L^{\text{in}} \\
\Lambda_Y^{\text{in}} & = & \{(i,k) \mid 0<i<2m, & k=i-2m+1 & \} \\
\Psi_Y^{\text{in}} & = & \{(i,k) \mid -m+1<i\leqslant 0, & k=0 & \} \cup \\
 & & \{(i,k) \mid 2m\leqslant i<3m-1, & k=i-2m+1 & \} \cup \Lambda_Y^{\text{in}} \\
\Psi & = & \{(i,k) \mid 0<i<2m, & i-2m+2\leqslant k\leqslant i+1 & \} \cup \\
 & & \{(i,k) \mid -m+1<i\leqslant 0, & 0<k\leqslant i+m-1 & \} \cup \\
 & & \{(i,k) \mid 2m\leqslant i<3m-1, & i-2m+2\leqslant k\leqslant m & \}
\end{array}$$

The UCREs for separation control variables are obtained by substituting the above definitions of Λ_F^{in}, Λ_L^{in}, Λ_E^{in}, Ψ_F^{in}, Ψ_L^{in}, Ψ_E^{in}, and Ψ into Def. 8 (Fig. 4). □

A manual input and output extension and calculation of the new input domain for each control variable is, of course, tedious and error-prone. In Sect. 7.1.5, we automate the input and output extension with a synthesis procedure that returns a specification of the separation control variables. But we stress that the concept of input and output extension is also of tutorial value. It provides insight into the construction of UCREs for separation control variables. It also leads to a formalism for the proof of their correctness.

The construction of UCREs following Def. 8 aims at defining the boundaries between the partitions Ψ_E^{s}, Ψ_E^{d}, $\Psi_E^{\perp}$, and Φ. The construction of evolution control variable requires that the UCREs of Def. 8 satisfy the following specification (Fig. 3):

$$\begin{array}{lll}
\text{Spec. 1.} & (\forall\, I : (I\in\Psi : \varphi(I) = (f,\chi,e_{\text{s}}) & \Longleftrightarrow \quad I\in \mathsf{in}(\Phi,\vartheta_E))) \\
\text{Spec. 2.} & (\forall\, I : (I\in\Psi : \varphi(I) = (f,\ell,e_{\text{s}}) & \Longleftrightarrow \quad I\in \mathsf{out}(\Phi,\vartheta_E)\cap\mathsf{in}(\Phi,\vartheta_E))) \;\wedge \\
 & (\forall\, I : (I\in\Psi : \varphi(I) = (\chi,\ell,e_{\text{c}}) & \Longleftrightarrow \quad I\in \mathsf{out}(\Phi,\vartheta_E)\backslash\mathsf{in}(\Phi,\vartheta_E))) \\
\text{Spec. 3.} & (\forall\, I : (I\in\Psi : \varphi(I) = (\chi,\chi,\perp) & \Longleftrightarrow \quad I\in\Psi_E^{\perp})) \\
\text{Spec. 4.} & (\forall\, I : (I\in\Psi : \varphi(I) = (\chi,\chi,e_{\text{s}}) & \Longleftrightarrow \quad I\in\Psi_E^{\text{s}})) \\
\text{Spec. 5.} & (\forall\, I : (I\in\Psi : \varphi(I) = (\chi,\chi,e_{\text{d}}) & \Longleftrightarrow \quad I\in\Psi_E^{\text{d}})) \\
\text{Spec. 6.} & (\forall\, I : (I\in\Psi : \varphi(I) = (\chi,\chi,e_{\text{c}}) & \Longleftrightarrow \quad I\in\Phi\backslash(\mathsf{in}(\Phi,\vartheta_E)\cup\mathsf{out}(\Phi,\vartheta_E))))
\end{array}$$



u
ℓ
f
e_s
e_s
e_s
(a)
0 x_1 x_2 0
$\perp\perp\perp\perp w_2 \perp w_1 \perp$
$\perp\perp\perp\perp\perp\perp f \perp$
$\perp\perp\perp\perp \ell \perp\perp\perp$
$y_1 \perp y_2 \perp y_3 \perp\perp\perp$
$e_s \perp e_s \perp e_s \perp\perp\perp$
(b)

Figure 4: (a) The control dependence graph with respect to projection vector $u = (1, 1)$ ($m=2$). The small dots at the tails of arrows stand for $\perp$. (b) The systolic array along with the distribution of both data and control signals at the first step. Compare Fig. 1.

where χ stands for "don't care". The satisfaction of this specification trivially ensures the correctness of the separation control variables (Def. 6). Note that Spec. 2 is a conjunction: one conjunct captures the points in both $\mathsf{out}(\Phi, \vartheta_E)$ and $\mathsf{in}(\Phi, \vartheta_E)$ and the other the points only in $\mathsf{out}(\Phi, \vartheta_E)$ (Fig. 3).

Now, we state a theorem on the correctness of the separation control variables. This theorem describes the essential rôle of the evolution control variable E.

Theorem 1 *(Separation Theorem) If Specs. 1–2 are satisfied, so are Specs. 3–6.*

Proof. Def. 8. □

Thm. 1 states that the correctness of the separation control variables is fully established by the satisfaction of Specs. 1–2. In the next section, we present necessary and sufficient conditions for the satisfaction of Specs. 1–2. We also present sufficient conditions that allow a construction of control dependence vectors such that Specs. 1–2 are satisfied. They impose constraints on the topological relationship between control dependence vectors ϑ_F, ϑ_L and ϑ_E and, consequently, on the shape of the index space Φ.

7.1.3 The Construction of Control Dependence Vectors

The construction of UCREs following Def. 8 aims at defining the boundaries between the partitions Ψ^{s}_E, Ψ^{d}_E, $\Psi^{\perp}_E$ and Φ. The proof of Thm. 1 implies that the boundary between $\Psi^{\perp}_E$ and the remaining partitions is correctly identified by the construction of the evolution control variable. To identify the boundary $\mathsf{in}(\Phi, \vartheta_E)$ between Ψ^{s}_E and Φ and the boundary $\mathsf{out}(\Phi, \vartheta_E)$ between Ψ^{d}_E and Φ correctly (Fig. 2), we rely on an appropriate choice of control dependence vectors ϑ_F, ϑ_L and ϑ_E.

Now, we state a necessary and sufficient condition that ensures the satisfaction of Spec. 1 and that guides the construction of control dependence vectors ϑ_F and ϑ_E.

Theorem 2 *(Initialization Theorem I)* $\Psi^{\mathrm{s}}_E \cap \mathsf{sig.in} = \{\}$ *iff Spec. 1 is satisfied.*

Proof.

$\Longleftarrow$ Trivial. Otherwise a soaking point of E would be treated as a computation point.

$\Longrightarrow$ We first prove $\Longleftarrow$ and then $\Longrightarrow$ of Spec. 1.

- $\Longleftarrow$ By Stat. 1 of Lemma 1, $\mathsf{in}(\Phi, \vartheta_E) \subseteq \mathsf{sig.in}$. The proof follows from the specification of evolution control variable E and hypothesis $\Psi^{\mathrm{s}}_E = \{\}$.
- $\Longrightarrow$ Since $\Longleftarrow$ of Spec. 1 has been established, it suffices, by Stat. 2 of Lemma 1, to prove that, at any point I that is in $\mathsf{sig.in}$ but not in $\mathsf{in}(\Phi, \vartheta_E)$, $\varphi(I) \neq (f, \chi, e_{\mathrm{s}})$ holds. By the definitions of $\mathsf{sig.in}$ and $\mathsf{in}(\Phi, \vartheta_E)$, there must exist $J \in \mathsf{in}(\Phi, \vartheta_E)$ such that $J \xrightarrow{E} I$ holds. Following the specification of evolution control variable E, either $E(J) = e_{\mathrm{c}}$ or $E(J) = e_{\mathrm{d}}$ and, consequently, either $E(I) = e_{\mathrm{c}}$ or $E(I) = e_{\mathrm{d}}$. That is, $\varphi(I) \neq (f, \chi, e_{\mathrm{s}})$. □

Thm. 2 does not provide a construction for ϑ_F and ϑ_E. Next, we present a construction for ϑ_F and ϑ_E. This requires additional terminology:

- A set $S \subset \mathbb{Q}^n$ is called an *affine set* if $(\forall\, k, x, y : k \in \mathbb{Q} \wedge x, y \in S \Longrightarrow kx + (1-k)y \in S)$. The *affine hull* of a set $S \subset \mathbb{Q}^n$, denoted $\mathsf{affine}(S)$, is the intersection of all affine supersets of S. An affine set M is said to be *parallel* to an affine set S if $(\exists\, a : a \in \mathbb{Q} : M = S + a)$, where $S + a = \{x + a \mid x \in S\}$. $\mathsf{linear}(M)$ denotes the unique linear space that is parallel to $\mathsf{affine}(M)$: $\mathsf{linear}(M) = \{x - y \mid x, y \in \mathsf{affine}(M)\}$ [21].
- $\pi x \leqslant \pi_0$ $(\pi, \pi_0 \in \mathbb{Q}^n)$ is a *valid inequality* for a polyhedron $P \subset \mathbb{Q}^n$, if it is satisfied at all points of P. If $F = \{x \mid x \in P,\ \pi x = \pi_0\}$ and $\pi x \leqslant \pi_0$ is a valid inequality, F is called a *face* of P. A face F of P is called a *facet* of P if $\dim(F) = \dim(P) - 1$ [12, Sect. I.4].

Motivated by Thm. 2, the construction of ϑ_F and ϑ_E requires that $\mathsf{sig.in} = \mathsf{in}(\Phi, \vartheta_E)$, which implies $\Psi^{\mathrm{s}}_E \cap \mathsf{in}(\Phi, \vartheta_E) = \{\}$.

Theorem 3 *(Initialization Theorem II) If* $\mathsf{in}(\Phi, \vartheta_E)$ *is a facet of index space* Φ *and* $\vartheta_F \in \mathsf{linear}(\mathsf{in}(\Phi, \vartheta_E))$*, then Spec. 1 is satisfied.*

Proof. By Thm. 2, it suffices to prove that $\mathsf{in}(\Phi, \vartheta_E) = \mathsf{sig.in}$, since $\mathsf{in}(\Phi, \vartheta_E) \cap \Psi^{\mathrm{s}}_E = \{\}$. By Stat. 1 of Lemma 1, $\mathsf{in}(\Phi, \vartheta_E) \subseteq \mathsf{sig.in}$. We only need to prove that $\mathsf{in}(\Phi, \vartheta_E) \supseteq \mathsf{sig.in}$. For any $I \in \mathsf{sig.in}$, by the definitions of $\mathsf{sig.in}$ and $\mathsf{in}(\Phi, \vartheta_E)$, there must exist $J \in \mathsf{in}(\Phi, \vartheta_E)$ such that $J \xrightarrow{F} I$ or $I \xrightarrow{F} J$. Because $\mathsf{in}(\Phi, \vartheta_E)$ is a facet of Φ and ϑ_F is the point in the linear space $\mathsf{linear}(\mathsf{in}(\Phi, \vartheta_E))$, we must have $I \in \mathsf{in}(\Phi, \vartheta_E)$. Otherwise we would obtain $\dim(\mathsf{in}(\Phi, \vartheta_E)) = \dim(\Phi)$, which contradicts the hypothesis that $\mathsf{in}(\Phi, \vartheta_E)$ is a facet of Φ. □

An analogue, for termination control variable L, of Thm. 2 is stated next.

Theorem 4 *(Termination Theorem I) Assume* $\Psi^{\mathrm{s}}_E \cap \mathsf{sig.in} = \{\}$. $\Phi \cap \mathsf{sig.out} = \mathsf{out}(\Phi, \vartheta_E)$ *iff Spec. 2 is satisfied.*

Proof. Similar to the proof of Thm. 2. □

By symmetry, there is an analogue, for termination control variable L, of Thm. 3; we require that $\mathsf{sig.out} = \mathsf{out}(\Phi, \vartheta_E)$, which implies $\Phi \cap \mathsf{sig.out} = \mathsf{out}(\Phi, \vartheta_E)$.

Theorem 5 *(Termination Theorem II) Assume* $\Psi^{s}_{E} \cap \mathsf{sig.in} = \{\}$. *If* $\mathsf{out}(\Phi, \vartheta_E))$ *is a facet of the index space* Φ *and* $\vartheta_L \in \mathsf{linear}(\mathsf{out}(\Phi, \vartheta_E))$, *then Spec. 2 is satisfied.*

Proof. Similar to the proof of Thm. 3. □

Note that the satisfaction of Spec. 2 depends on the satisfaction of Spec. 1. This is the reason for the prerequisite $\Psi^{s}_{E} \cap \mathsf{sig.in} = \{\}$ in Thms. 4 and 5.

Remark. The correctness of the separation control variables does not depend on whether any of them is moving or stationary. However, in an implementation, F and L must be moving for the following reasons. The control signals of F and L serve to change the control signals of E; they must be supplied from the outside environment such that they can arrive at the right cells at the right time. If F (L) is stationary, we have to pre-load its control signals – but we cannot, because the presence of these control signals ahead of schedule will validate some guards of E at the wrong time (unless we add a second level of control variables, this one governing the access of stationary control variables, and so on).

E may be stationary. In this case, control values e_s and $\perp$ of E can be pre-loaded before step t_{fst}. The steps at which a stationary element is accessed are completely specified by the space-time mapping. But, if a cell contains more than one stationary element, additional control circuitry is needed to direct access to the right stationary element. The systematic synthesis of control circuitry for reading and writing local memory is future work. □

The next theorem characterizes the relationship between ϑ_F and ϑ_E and between ϑ_L and ϑ_E further.

Theorem 6 *Assume that Specs. 1–2 are satisfied.*

1. $\Psi^{s}_{E} \neq \{\} \implies \vartheta_F \nparallel \vartheta_E$.

2. $\Phi \neq \mathsf{out}(\Phi, \vartheta_E) \implies \vartheta_L \nparallel \vartheta_E$.

Proof.

1. Assume $\vartheta_F \parallel \vartheta_E$. Then $\mathsf{sig.in} = \Psi_F = \Psi_E$. By Def. 4, $\Psi^{s}_{E} \subseteq \Psi_E$. By further using hypothesis $\Psi^{s}_{E} \neq \{\}$, we obtain $\Psi^{s}_{E} \cap \mathsf{sig.in} \neq \{\}$. By Thm. 2, Spec. 1 is violated.

2. Assume $\vartheta_L \parallel \vartheta_E$. Then $\mathsf{sig.out} = \Psi_L = \Psi_E$. By Def. 4 and the definition of $\mathsf{sig.out}$, $\Phi \subseteq \mathsf{sig.out}$. By further using hypothesis $\Phi \neq \mathsf{out}(\Phi, \vartheta_E)$, we have $\Phi \cap \mathsf{sig.out} \neq \mathsf{out}(\Phi, \vartheta_E)$. By Thm. 4, Spec. 2 is violated. □

In summary: except for the trivial case, ϑ_F and ϑ_L must not be parallel to ϑ_E, and F and L must be moving.

7.1.4 Adaptation of the Source

The following definitions are used in this and subsequent subsections.

Definition 9 Given $\delta \in \mathbb{Z}$, a non-zero vector $\pi \in \mathbb{Z}^n$ and a vector $x \in \mathbb{Z}^n$, the set $\{x \mid \pi x = \delta\}$ is called a *hyperplane* in $\mathbb{Z}^n$. The set $\{x \mid \pi x \leqslant \delta\}$ ($\{x \mid \pi x < \delta\}$) is called a *closed* (*open*) *half-space.* Given a convex set $C \in \mathbb{Z}^n$, a *supporting half-space* is a closed half-space that contains C and has a point of C on its boundary. A *supporting hyperplane* to C is a hyperplane that is the boundary of a supporting half-space to C [21]. □

Thms. 3 and 5 require that the index space Φ be of the certain shape:

- $\mathsf{in}(\Phi, \vartheta_E)$ must be a facet of Φ, i,e., a subset of hyperplane $\{I \mid \vartheta_E I = H_E^-\}$, where $H_E^- = \min\{\vartheta_E I \mid I \in \Phi\}$, and
- $\mathsf{out}(\Phi, \vartheta_E)$ must be a facet of Φ, i.e., a subset of hyperplane $\{I \mid \vartheta_E I = H_E^+\}$, where $H_E^+ = \max\{\vartheta_E I \mid I \in \Phi\}$.

The source can always be put into this form: simply replicate ϑ_E forward and backward until the extended points intersect the two bounding hyperplanes $\{I \mid \vartheta_E I = H_E^-\}$ and $\{I \mid \vartheta_E I = H_E^+\}$. The points added in the extension are pipelining points. Thus, the UREs defined for the extended index space can be obtained straight-forwardly from the source UREs. Then, we need to specify computation control variables for the extended UREs (Sect. 7.2).

7.1.5 A Synthesis Procedure

The following synthesis procedure returns a correct specification of the separation control variables; the procedure does not require an input and output extension.

Procedure 1 (Synthesis procedure for the separation control variables)
INPUT: UREs (Φ, D).
OUTPUT: The distribution of control signals f, ℓ, e_s and $\perp$ at step t_{fst}, and a control program for cells.

1. Choose ϑ_F, ϑ_L and ϑ_E such that Thms. 3 and 5 are satisfied. If necessary, extend the index space Φ as described in Sect. 7.1.4.
2. The distribution of input control signals – f of F, ℓ of L and e_s of E – is given by $\{\mathsf{pattern}(v(I)) \mid v \in \{F, T, E\}, I \in \Psi_v^{\mathrm{in}}\}$.
3. Fill the unoccupied locations of the processor space with $\perp$; if $\perp$ is inside a cell, the cell can be initialized by a system reset (Sect. 7.1.2).
4. We write v_{in} (v_{out}) for the control signal of control variable v at the input port (output port) of a cell. At every step, a cell executes the following program:

```
$F_{\mathrm{out}}$ = $F_{\mathrm{in}}$
$L_{\mathrm{out}}$ = $L_{\mathrm{in}}$
$E_{\mathrm{out}}$ = if $E_{\mathrm{in}} = e_s \wedge F_{\mathrm{in}} = f \wedge L_{\mathrm{in}} \neq \ell \ \rightarrow\ e_c$
                 [] $((E_{\mathrm{in}} = e_s \wedge F_{\mathrm{in}} = f) \vee E_{\mathrm{in}} = e_c) \wedge L_{\mathrm{in}} = \ell \ \rightarrow\ e_d$
                 [] else $\rightarrow\ E_{\mathrm{in}}$
                 fi
$Z_{\mathrm{out}}$ = if $(E_{\mathrm{in}} = e_s \wedge F_{\mathrm{in}} = f) \vee E_{\mathrm{in}} = e_c \ \rightarrow\ z_c$
                 [] else $\rightarrow\ z_p$
                 fi
```

□

This procedure delivers separation control variables as specified by the UCREs of Def. 8: Steps 2 and 3 construct the input equations and Step 4 constructs the computation equations.

Proc. 1 constructs UCREs only for the index space Φ. We can then obtain UCREs for the extended index space as follows:

- The defining equation of a control variable at the pipelining points is the same as that at the computation points.
- The input equations are redefined as follows: $v(I)$ at point $I \in \Psi_v^{\mathrm{in,s}}$ is initialized with the same control signal as $v(J)$ at $J \in \Phi_v^{\mathrm{in}}$, where $I \xrightarrow{v} J$, and $v(I)$ at point $\Psi_v^{\mathrm{in},\perp}$ is initialized with $\perp$.

Definition 10 (UCREs for the separation control variables over the original index space)
Evolution control variable E:

$$\begin{array}{rcrcl}
I \in \Phi_E^{\mathrm{in}} & \rightarrow & E(I) & = & e_s \\
I \in \Phi & \rightarrow & E(I) & = & \textbf{if } E(I-\vartheta_E)=e_s \wedge F(I-\vartheta_F)=f \wedge L(I-\vartheta_L)\neq \ell \;\rightarrow\; e_c \\
 & & & & [\!]\ ((E(I-\vartheta_E)=e_s \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_c)) \\
 & & & & \quad \wedge L(I-\vartheta_L)=\ell \;\rightarrow\; e_d \\
 & & & & [\!]\ \textbf{else} \;\rightarrow\; E(I-\vartheta_E) \\
 & & & & \textbf{fi}
\end{array}$$

Initialization control variable F:

$$\begin{array}{rcrcl}
I \in \Phi_F^{\mathrm{in}} \backslash \Phi_F^f & \rightarrow & F(I) & = & \perp \\
I \in \Phi_F^f & \rightarrow & F(I) & = & f \\
I \in \Phi & \rightarrow & F(I) & = & F(I-\vartheta_F)
\end{array}$$

Termination control variable L:

$$\begin{array}{rcrcl}
I \in \Phi_L^{\mathrm{in}} \backslash \Phi_L^\ell & \rightarrow & L(I) & = & \perp \\
I \in \Phi_L^\ell & \rightarrow & L(I) & = & \ell \\
I \in \Phi & \rightarrow & L(I) & = & L(I-\vartheta_L)
\end{array}$$

Action control variable Z:

$$\begin{array}{rcrcl}
I \in \Phi & \rightarrow & Z(I) & = & \textbf{if } (E(I-\vartheta_E)=e_s \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_c \;\rightarrow\; z_c \\
 & & & & [\!]\ \textbf{else} \;\rightarrow\; z_p \\
 & & & & \textbf{fi}
\end{array}$$

where $\Phi_F^f = \{I \mid I \xrightarrow{F} J,\ I \in \Phi_F^{\mathrm{in}},\ J \in \mathsf{in}(\Phi,\vartheta_E)\}$ and $\Phi_L^\ell = \{I \mid I \xrightarrow{L} J,\ I \in \Phi_L^{\mathrm{in}},\ J \in \mathsf{out}(\Phi,\vartheta_E)\}$. $\square$

7.2 The Specification of Computation Control Variables

First, we state a correctness criterion for the computation control variables. Let $\{\Phi_i \mid 0 \leqslant i < r\}$ be a partitioning of Φ such that two points are contained in Φ_i iff both points are of the same type. We write $\xi(I)$ for the vector of control signals at point I; each component of the vector corresponds to one computation control variable:

$$\xi(I) \;=\; (v(I-\vartheta_v), \cdots)$$

Definition 11 *The computation control variables are correct if control signals at two computation points differ if the two points are in different partitions, i.e., if*

$$(\forall\, i,j : 0 \leqslant i,j < r \wedge i \neq j : (\forall\, I,J : I \in \Phi_i \wedge J \in \Phi_j : \xi(I) \neq \xi(J))) \qquad \Box$$

Computation control variables are pipelining variables. The specification of a computation control variable amounts to defining its control dependence vector and its input equations. As in the construction of the separation control variables, we use computation control variables to identify the boundaries between the partitions of the index space. The number of computation control variables required depends on the number of partitions and the shape of each partition.

We formulate the specification of computation control variables as a *lattice colouring* problem (Sect. 7.2.1). This gives rise to a method that we call the *separating hyperplane method* and that allows a provably correct construction of the computation control variables (Sect. 7.2.2).

7.2.1 The Formulation: Lattice Colouring

Remember that $p(I, v, \Phi)$ denotes a ϑ_v-path in the index space Φ that passes through point I, and that $P(v, \Phi)$ denotes the set of all ϑ_v-paths of v.

The specification of the computation control variables can be viewed as the problem of colouring a lattice, namely, the index space Φ. Computation control signals represent colours. There is one colouring rule: all points in the same path must have the same colour. The rationale behind this rule is that computation control variables are pipelining variables, and pipelining variables have only one value along a fixed path.

Let $c(x)$ stand for the colour of x. $C(M) = \{c(x) \mid x \in M\}$. Now, we can explicitly define the set of control signals $\xi(I)$ at computation point I:

$$\xi(I) = (c(p(I, v_0, \Phi)), \cdots, c(p(I, v_{k-1}, \Phi)))$$

where $\{v_i \mid 0 \leqslant i < k\}$ is the set of computation control variables. We write $\Xi(\Phi_i)$ for the set of control signals at the points in partition Φ_i:

$$\Xi(\Phi_i) = \{\xi(I) \mid I \in \Phi_i\}$$

The specification of the computation control variables can be represented by a *colouring*. It consists of the set of computation control variables and, for every variable v in the set, the definition of $C(P(v, \Phi))$. The correctness of a colouring is characterized by Def. 11.

The construction of the computation control variables relies on the identification of pipelining points by the separation control variables. Let us now extend the specification of action control variable Z. We use control signal z_i to denote the type of the points in partition Φ_i: $\mathcal{S}(Z) = \{z_0, \cdots, z_{r-1}, z_p\}$; the initial value z_c (Sect. 7.1.2) has been replaced by the values $\{z_i \mid 0 \leqslant i < r\}$.

With $G(F(I),L(I),E(I)) = ((E(I-\vartheta_E)=e_s \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_c)$

$$\begin{array}{rcl}
I\in\Psi \rightarrow Z(I) & = & \textbf{if}\, G(F(I),L(I),E(I) \;\wedge\; \xi(I)\in\Xi(\Phi_0) \;\rightarrow\; z_0 \\
& & [\,]\; G(F(I),L(I),E(I) \;\wedge\; \xi(I)\in\Xi(\Phi_1) \;\rightarrow\; z_1 \\
& & \quad\vdots \\
& & [\,]\; G(F(I),L(I),E(I) \;\wedge\; \xi(I)\in\Xi(\Phi_{r-1}) \;\rightarrow\; z_{r-1} \\
& & [\,]\; \textbf{else} \;\rightarrow\; z_{\mathrm{p}} \\
& & \textbf{fi}
\end{array}$$

The i-th guard of Z applies to the points of partition Φ_i, because $\xi(I) \in \Xi(\Phi_i)$ holds at a point iff the point is in Φ_i. This completes the formulation of the lattice colouring problem.

There may be more than one solution to the lattice colouring problem. In other words, there may be more than one correct computation control scheme. As is, our method provides no further help in the selection of the best colouring with respect to certain cost functions. One possible cost function might be $(\sum v : v \in V_c : |C(P(v,\Phi))|)$, where V_c denotes the set of computation control variables and $|C(P(v),\Phi)|$ represents the maximal number of control signals of variable v that may pass through channel $\sigma\vartheta_v$. A best colouring minimizes this function, which represents the total bit width of the computation control variables. Note that an optimization of the colouring may compete with an optimization of the space-time mapping.

7.2.2 The Construction: Separating Hyperplanes

Remember that $\{\Phi_i \mid 0 \leqslant i < r\}$ is a partitioning of the index space Φ. Each partition is a union of a finite set of polyhedra. Without lost of generality, we assume that each partition is itself a polyhedron (if not, choose a finer partitioning).

Definition 12 $S_{i,j}$ is a *separating hyperplane* between two partitions Φ_i and Φ_j iff $S_{i,j} = \{I \mid \pi_{i,j}I = \delta_{i,j}\}$, such that Φ_i is in the closed half-space $\{I \mid \pi_{i,j}I \leqslant \delta_{i,j}\}$ and Φ_j is in the open half-space $\{I \mid \pi_{i,j}I > \delta_{i,j}\}$. The two partitions Φ_i and Φ_j are called *neighbouring partitions* if all of their separating hyperplanes that contain at least one point of the index space are supporting hyperplanes to Φ_i; these separating hyperplanes are called *neighbouring separating hyperplanes.* □

By Def. 12, if $N_{i,j} = \{I \mid \pi_{i,j}I = \delta_{i,j}\}$ is a neighbouring separating hyperplane that separates Φ_i and Φ_j, $\{I \mid \pi_{i,j}I \leqslant \delta_{i,j}\}$ is a supporting half-space of Φ_i. Since every partition is a polyhedron, the existence of at least one separating hyperplane between any two neighbouring partitions is guaranteed [21]. In fact, a neighbouring separating hyperplane can be directly obtained from the domain predicates that appear in the source UREs; these domain predicates have induced the partitioning $\{\Phi_i \mid 0 \leqslant i < r\}$ (Fig. 5).

We associate a distinct computation control variable, denoted $C_{i,j}$, with every $N_{i,j}$. The idea is that, instead of computing two inequalities $\{I \mid \pi_{i,j}I \leqslant \delta_{i,j}\}$ and $\{I \mid \pi_{i,j}I > \delta_{i,j}\}$ at every cell, the result of evaluating each conditional can be shared by the pipelining of a control signal that represents the result from the boundary of the array. Each computation control variable takes on two different control signals, one in the half-space $\{I \mid \pi_{i,j}I \leqslant \delta_{i,j}\}$ and the other in the half-space $\{I \mid \pi_{i,j}I > \delta_{i,j}\}$.

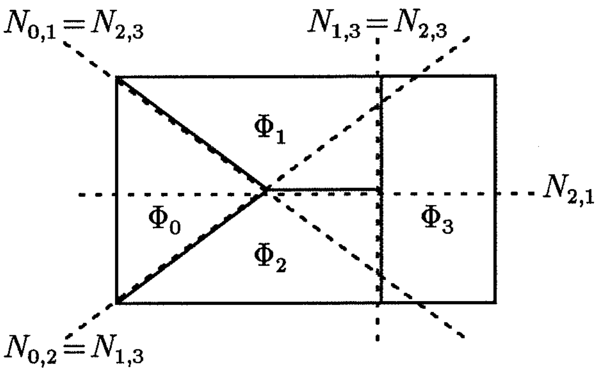


Figure 5: Illustration of the concept of neighbouring separating hyperplane. The bounding box depicts a two-dimensional index space; it has four partitions, separated by solid lines. There are four neighbouring hyperplanes, highlighted by dashed lines.

Definition 13 (UCREs for computation control variable $C_{i,j}$)

1. Its control dependence vector $\vartheta_{C_{i,j}}$ is any solution of $\vartheta_{C_{i,j}} \pi_{i,j} = 0$.
2. Let $\{\Phi^{\text{in},\leqslant}_{C_{i,j}}, \Phi^{\text{in},>}_{C_{i,j}}\}$ be the partitioning of $\Phi^{\text{in}}_{C_{i,j}}$ such that all points in $\Phi^{\text{in},\leqslant}_{C_{i,j}}$ satisfy $\pi_{i,j} I \leqslant \delta_{i,j}$ and all points in $\Phi^{\text{in},>}_{C_{i,j}}$ satisfy $\pi_{i,j} I > \delta_{i,j}$. With $\mathcal{S}(C_{i,j}) = \{c_{\leqslant}, c_{>}\}$, the input equations of $C_{i,j}$ are defined as follows:

$$\begin{array}{lcl} I \in \Phi^{\text{in},\leqslant}_{C_{i,j}} & \to & C_{i,j}(I) = c_{\leqslant} \\ I \in \Phi^{\text{in},>}_{C_{i,j}} & \to & C_{i,j}(I) = c_{>} \end{array}$$

□

Def. 13 provides a simple construction of the computation control variables. It differs from the method of [2] in two aspects. First, by specifying the computation control variables at the source level, our method is capable of transforming all domain predicates at the source to pipelining equations. The method of [2] works at the target level; it only transforms time-dependent transformed domain predicates to pipelining equations. Second, the method of [2] may fail in some cases. Assume that the index space has two dimensions and that $t \geqslant px + q$ ($p, q \in \mathbb{Q}$) is a transformed domain predicate. By the method of [2], a control signal is first input at border cell x_0 at step $px_0 + q$ and then pipelined through the array with a velocity of $1/p$. However, if $p < 1$, this predicate cannot be pipelined, because a control signal cannot travel farther than unit distance in one clock tick. Our method works always, because the space-time mapping is chosen after the UCREs have been constructed.

7.2.3 The Proof of Correctness

Theorem 7 *The computation control variables as specified in Def. 13 are correct.*

Proof. Pick any two points I and J which are in different partitions Φ_i and Φ_j.

Case 1. Φ_i and Φ_j are neighbouring partitions. $N_{i,j}$ is the neighbouring separating hyperplane. $C_{i,j}$ is the corresponding computation control variable. By Def. 13, $C_{i,j}(I) \neq C_{i,j}(J)$. That is, $c(p(I, C_{i,j}, \Phi) \neq c(p(J, C_{i,j}, \Phi)$, which implies $\xi(I) \neq \xi(J)$. An application of Def. 11 completes the proof.

Case 2. Φ_i and Φ_j are not neighbouring partitions. Because all partitions are polyhedra, there must exist a partition Φ_k $(k \neq i \wedge k \neq j)$ such that either Φ_i and Φ_k are neighbouring partitions and the neighbouring separating hyperplane $N_{i,k}$ is also a separating hyperplane between Φ_i and Φ_j or Φ_j and Φ_k are neighbouring partitions and the neighbouring separating hyperplane $N_{j,k}$ is also a separating hyperplane between Φ_i and Φ_j. Then, an argument similar to the previous case completes the proof. □

A computation control variable takes on two values. If the input of the variable is undefined at some step, one of the two values can be input arbitrarily – the proof of Thm. 7 requires no specific value in this case.

8 Generalizing the Specification of Termination Control

Termination control can be specified similarly to computation control, but we may want to use more than one termination control variable. To do so, we make use of the following fact. Once first computation points of E are correctly identified, a point is a last computation point of E only if it is a computation point. This is expressed by the presence of condition $(E(I-\vartheta_E)=e_s \wedge F(I-\vartheta_F)=f) \vee E(I-\vartheta_E)=e_c$ in the second guard of E (Def. 8).

We phrase the problem of specifying termination control variables as a lattice colouring problem. Let $\{L_i \mid 0 \leqslant i < p\}$ be the termination control variables needed. We write $\psi(I)$ for the vector of control signals at point I; each component of the vector corresponds to one termination control variable:

$$\psi(I) \;=\; (L_0(I-\vartheta_{L_0}), \cdots, L_{p-1}(I-\vartheta_{L_{p-1}}))$$

Theorem 8 *Assume* $\Psi_E^{\mathrm{s}} \cap \mathsf{sig.in} = \{\}$. *Spec. 2 is satisfied if the following holds:*

$$(\forall\; I, J : I \in \Phi \backslash \mathsf{out}(\Phi, \vartheta_E) \wedge J \in \mathsf{out}(\Phi, \vartheta_E) : \psi(I) \neq \psi(J))$$

Proof. Def. 8. □

This theorem states that, to specify termination control, we must only find a set of separating hyperplanes that separate the set $\Phi \backslash \mathsf{out}(\Phi, \vartheta_E)$ from the set $\mathsf{out}(\Phi, \vartheta_E)$.

To specify the separation control variables accordingly, the conditions in the guards of the evolution control variable E that refer to termination control variable L (Defs. 8 and 10) must be replaced by new conditions referring to the termination control variables $\{L_i \mid 0 \leqslant i < p\}$. This change in the specification makes the UCREs for separation control variables dependent on the source UREs.

9 Optimization

9.1 Elimination Termination Control

The points in Ψ^{d}_{E} are pipelining points. If, at these points, all data variables are undefined, i.e., if

$$(\forall\, v : v \in V : I \in \Psi^{\mathrm{d}}_{E} \rightarrow v(I) = \bot)$$

we can eliminate termination control: computations at these points are free of interpretation.

Our method presumes the existence of at most one extreme ray in the index space. If the index space contains a ray, the projection vector must be parallel to that ray, if we are to obtain a systolic array with a finite number of cells.[3] The presence of a ray means that the index space is infinite, and so is the execution of the systolic array (if it has finitely many cells).

If ϑ_E is parallel to the extreme ray,[4] the termination control variable has no rôle: it would be expected to indicate the last of an infinite sequence of computations points. It can be easily verified that the separation control variables without termination control are correct.

One specification of 1-D convolution requires that input sequence X and output sequence Y be infinite [13]. The extreme ray is given by vector $(1,0)$. The projection vector must be $u=(1,0)$. If we choose $\ell.E=W$, i.e., $\vartheta_E=\vartheta_W=(1,0)$, we can eliminate the termination control variable L. One choice for initialization control variable F is $\ell.F=Y$.

9.2 Eliminating all Control

The control signals can be totally eliminated iff the following two conditions hold:

1. All computation points that are mapped to the same cell must be of the same type.

2. A pipelining point mapped to a cell can be treated as a computation point that is mapped to the same cell without violating the semantics of the source UREs.

9.2.1 Eliminating Computation Control

The computation control variables can be eliminated iff all computation points mapped to the same cell are of the same type. Obviously, if all points in the index space are of the same type, no computation control variables are required. Examples are convolution and matrix multiplication. If the index space contains different types of computation points, an appropriate choice of the space-time mapping might still avoid the necessity of computation control (App. A.4).

Remember that the space-time mapping has to minimize other factors beside hardware costs for control, e.g., latency, the size of the array and so on. Any one of these may take priority over the savings in hardware due to an elimination of control.

[3]Quinton [13] shows how systolic rings can be obtained for 1-D convolution by choosing projection vectors that are not parallel to the extreme ray in the index space. Ring layouts have the mod operator in the place function.

[4]Note that, in this case, E is stationary.

9.2.2 Eliminating Separation Control

We can exploit algebraic properties for a minimization of the number of separation control variables. For the array shown in Fig. 4, we can do entirely without control signals. We can simply input 0 whenever $\perp$ is input for W. Because $Y_{\text{out}} = Y_{\text{in}} + X_{\text{in}} 0$ is equivalent to $Y_{\text{out}} = Y_{\text{in}}$, the semantics of the source is preserved.[5] We still derive separation control variables in our example, purely for the sake of illustration.

In fact, we can state a more general result: for inner-product-like source UREs in which all equations are defined for the same index space and all equations but one are pipelining equations [13], no computation control variables are needed.

Theorem 9 *Consider a system of UREs whose computation equations are as follows:*

$$\begin{array}{lcll} I \in \Phi & \rightarrow & V(I) = V(I - \vartheta_V) & \text{(E1)} \\ I \in \Phi & \rightarrow & W(I) = W(I - \vartheta_W) & \text{(E2)} \\ I \in \Phi & \rightarrow & U(I) = U \oplus V(I - \vartheta_V) \otimes W(I - \vartheta_W) & \text{(E3)} \end{array}$$

Let $U(I), V(I), W(I) \in \mathsf{M}$. $(\mathsf{M}, \oplus, \otimes, \emptyset)$ forms an algebra with two binary operators $\oplus$ and $\otimes$, with $\otimes$ binding stronger than $\oplus$, and a zero element $\emptyset \in \mathsf{M}$. If the following conditions are satisfied, separation control variables are unnecessary:

1. $(\exists\, x : x \in \{V, W\} : (\mathsf{in}(\Phi, \vartheta_x)$ and $\mathsf{out}(\Phi, \vartheta_x)$ are facets of $\Phi) \wedge \vartheta_x \nparallel \vartheta_U)$.
2. $\mathsf{in}(\Phi, \vartheta_U)$ *and* $\mathsf{out}(\Phi, \vartheta_U)$ *are facets of* Φ.
3. $(\forall\, r, s : r, s \in \mathsf{M} : \emptyset \otimes r = r \otimes \emptyset = \emptyset \wedge s \oplus \emptyset = s)$.

Proof. Because V and W are pipelining equations at all points in the extended index space Ψ, we only need to prove that, at the pipelining points in Ψ^{p}, U, which is initially specified by a pipelining equation, can be redefined according to Equ. E3 without violating the semantics of the source UREs. We consider two cases, depending on how points in Ψ^{p} are generated:

Case 1. Points in $\Psi^{\mathrm{s}}_U \cup \Psi^{\mathrm{d}}_U$, which are generated in the extension of U. Without loss of generality, assume that $\mathsf{in}(\Phi, \vartheta_V)$ and $\mathsf{out}(\Phi, \vartheta_V)$ are facets of Φ and $\vartheta_V \nparallel \vartheta_U$ of the first hypothesis. By Def. 4 and the second hypothesis, $\Psi_U \cap \Psi_V = \Phi$. Thus, V is undefined at all points in $\Psi^{\mathrm{s}}_U \cup \Psi^{\mathrm{d}}_U$. Hence, we can introduce the following two equations:

$$\begin{array}{rcl} I \in \Psi_V^{\mathrm{in},\perp} & \rightarrow & V(I) = \emptyset \\ I \in \Psi^{\mathrm{s}}_U \cup \Psi^{\mathrm{d}}_U & \rightarrow & U(I) = U(I - \vartheta_U) \oplus V(I - \vartheta_V) \otimes W(I - \vartheta_W) \end{array}$$

An application of the third hypothesis yields that $U(I) = U(I - \vartheta_U) \oplus \emptyset \otimes W(I - \vartheta_W)$ is equivalent to $U(I) = U(I - \vartheta_U)$. Hence, the semantics of the source UREs at the points in $\Psi^{\mathrm{s}}_U \cup \Phi^{\mathrm{d}}_U$ is preserved.

Case 2. Points in $\Psi^{\perp}_U$, which are generated in the extension of V and W. U is undefined at these points and is free of interpretation. Hence, it can be specified by Equ. E3 without violating the semantics of the source:

$$I \in \Psi^{\perp}_U \;\rightarrow\; U(I) = U(I - \vartheta_U) \oplus V(I - \vartheta_V) \otimes W(I - \vartheta_W) \qquad \square$$

[5]This property is called *neutrality* [5].

Examples to which Thm. 9 applies are convolution and matrix multiplication. Other examples, like sorting, can be treated similarly. The two operators in sorting [19] are min and max, and we exploit the properties $\min(+\infty, x) = x$ and $\max(m, -\infty) = m$. By injecting $-\infty$ and $+\infty$ appropriately, control signals can be completely eliminated. We expect that other common operators also have algebraic properties that help in the elimination of control signals.

For further uses of algebraic transformations in systolic design, see [5, 10, 17].

Thm. 9 does not generalize directly to UREs that contain more than one non-pipelining equation, but similar techniques may apply. First, we might divide the UREs into several groups such that the equations in each group satisfy some variant of Thm. 9. If necessary, we can rewrite them by appropriately introducing new variables for pipelining the zero element. Second, we might classify the UREs according to shape and look for solutions for each class.

Finally, we point out that, when the systolic array is of reduced dimension (e.g., due to a projection), Thm. 9 is invalid because the space-time mapping is not a bijection. That is, two variables with different index vectors (which are in $\Psi \setminus \Phi$) may be scheduled at the same cell simultaneously. Take matrix multiplication as an example: $C = A \cdot B$. Say a cell is performing a pipelining operation. If the array has two dimensions, the cell deals with one element of only one of the three variables A, B and C per step. But if the array is one-dimensional, the cell may have to deal with one element of each of the three variables. For elements of C the question arises, whether to propagate them or compute with them [23].

9.3 Multipurpose Control

A control variable can serve more than one purpose. For example, it can govern both termination control and computation control. The objective in constructing UCREs is to associate a unique set of control signals with different types of points that have the same spatial coordinates.

9.4 Combining Control Variables

If two control variables are associated with the same control dependence vector and, in addition, the domains of input equations at which $\perp$ is not an input value are disjoint, we can merge the two variables into one. This reduces the total bit width required. For example, we can merge initialization control variable F and termination control variable L of the array in Fig. 4. In this particular case, there is no benefit.

If there are k parallel neighbouring separating hyperplanes, denoted $(\forall\ j : 0 \leqslant j < k : \{I \mid \pi I = \delta_j\})$, in the specification of the computation control variables, we can merge the k corresponding computation control variables into one, say, C. We write $\{\Phi_C^{\text{in},i} \mid 0 \leqslant i \leqslant k\}$ for the partitioning of Φ_C^{in} such that the points of $\Phi_C^{\text{in},i}$ satisfy the inequalities $\pi I > \delta_{i-1}$ and $\pi I \leqslant \delta_i$ ($\delta_{-1} = -\infty$ and $\delta_k = +\infty$). With $\mathcal{S}(C) = \{c_0, \cdots, c_k\}$, the input equations of C are defined as follows:

$$(\forall\ i : 0 \leqslant i \leqslant k : I \in \Phi_C^{\text{in},i} \ \rightarrow\ C(I) = c_i).$$

This merge reduces the number of control signals from $2k$ for the k variables to $k+1$ for the variable that replaces them.

9.5 Simplification of Action Control

In general, the action control variable is overspecified. We do not know the distribution of control signals until the space-time mapping is fixed. Because our specification is at the source level, it allows for the possibility that any control signal may encounter any cell. In a given application, a control signals may be known to encounter only a subset of cells. This makes certain disjuncts in the specification unnecessary. For example, if all points of partition Φ_i are not mapped to a cell, then the i-th guard of action control variable Z can be eliminated for that cell. We expect that these disjuncts can be eliminated mechanically after the space-time mapping is fixed.

If all computation points mapped to the same cell are of the same type, no action control needs to be specified for that cell.

9.6 Bit Width vs. Time Dependence

In the specification of computation control, we insist that all control signals be pipelined (all computation control variables are pipelining variables). We may allow for a number of non-pipelined control variables. In the absence of pipelining, the notion of time must become explicit in the specification (in time-dependent guards), and cells need to be aware of time (by access to the global clock). This increases the hardware complexity of the cell but reduces the total bit width. For example, if a cell has to perform a particular computation at steps ranging from t_0 to t_1, we can use a counter at that cell to keep track of the state of the global clock.

10 Related Work

To our knowledge, Guibas, Kung and Thompson [4] were the first to point out that control in a systolic array may be implemented by pipelining of control signals analogous to the pipelining of data.

Chen was the first to give a formal treatment of this idea [2]. Her method is to replace the test of some time-dependent domain predicate appearing in a transformed equation by a one-bit control signal, which is pipelined from the boundary of the array. A time-dependent domain predicate is a domain predicate in which one index represents time and the other indices represent processor coordinates.

Rajopadhye [16] describes a method that is based on the same suggestion of data pipelining. He considers two types of domain predicates: one consists of a conjunction of equalities, the other includes inequalities. For the former case, he presents a method that transforms a domain predicate at the source level to a pipelining equation. For the latter case, which is more prevalent in practical applications, he merely points out how the method for the former case can be extended. The idea is similar to Chen's [2]; both depend on the space-time mapping.

The methods of Chen and Rajopadhye represented significant progress but have the following limitations. First, they explicitly rely on bijectivity of a space-time mapping. They do not readily generalize to arrays of reduced dimension [23]. Second, the pipelining of domain predicates is determined after the space-time mapping has been selected. If the array has a fixed size, these methods do not apply. Third, these methods require that the transformed UREs fully specify the space-time behaviour of the array. Consequently, the specification of pipelining points must be included by an explicit input and output extension.

Let us mention other work related to this subject. Ramakrishnan and Varman [18] present a one-dimensional systolic array for matrix multiplication. They supply the flow of data and control and use linear algebra to prove the correctness of the array. Kumar and Tsai [7] propose a more general method that requires an explicit choice of the communication topology and, more seriously, of the sequencing of input variables. The flow of data and control is then derived by solving a set of constraint equations on timing. Lang [9] shows how control signals (i.e., instructions) can be pipelined to solve problems such as matrix multiplication and merging two arrays. Bu, Deprettere and Dewilde [1] consider the synthesis of control circuitry that arises in array partitioning.

11 Conclusions

We have discussed methods for the synthesis of control signals from UREs for the systolic array model described in Sect. 3. Our synthesis yields two types of control variables. Separation control variables identify pipelining and computation points, and computation control variables identify different types of computation points. We have presented UCREs for the separation control variables and a provably correct construction of UCREs for the computation control variables, which apply for any source UREs. To guarantee the correctness of the separation control variables, we must find control dependence vectors that satisfy the initialization and termination theorems. A method that does so has been described in Sect. 7.1.4; it may require an extension of the index space.

Our method has the following advantages:

- We specify the control signals at the source level and also prove their correctness at the source level. In the specification, we are concerned with correctness (Sects. 5–8). In the derivation of a systolic array by means of a space-time mapping, we are concerned with efficiency (Sect. 9).

- The source UREs and the associated UCREs provide a complete description of both data and control flow at the source level. The correctness of the control signals can be easily preserved if the source UREs and the associated UCREs are partitioned and mapped to a fixed-size array [11].

- Our method can be extended to arrays of reduced dimension. The construction of computation control variables remains the same. The construction of the UCREs for the separation control variables becomes more complex. In the case of one-dimensional arrays, for example, a construction of UCREs for three-dimensional UREs can be given directly but must be generalized for n-dimensional UREs; the generalization relies on a hierarchical decomposition of the original index space into three-dimensional subdomains [23].

- The UCREs constructed for the source UREs do not depend on the space-time mapping, i.e., are not array-specific.

The main question that remains open is the treatment of stationary variables. We must derive control circuitry for their reading, writing, loading and recovery. After these issues have been resolved, we can focus on the minization of control hardware.

A more general class of recurrence equations that subsumes UREs is the class of *affine* recurrence equations (AREs). Only uniform recurrence equations can be mapped directly to systolic arrays. Methods for the transformation of AREs to UREs have been proposed recently [15, 22]. Our method expects UREs.

12 Acknowledgement

We are grateful to Björn Lisper for many constructive and supportive exchanges. Thanks to Marina Chen and Ping F. Yeung for discussions.

13 References

[1] 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.

[2] M. C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *J. Parallel and Distributed Computing*, 3(4):461–491, 1986.

[3] E. W. Dijkstra. *A Discipline of Programming.* Series in Automatic Computation. Prentice-Hall, 1976.

[4] L. J. Guibas, H. T. Kung, and C. D. Thompson. Direct VLSI implementation of combinatorial algorithms. In *Proc. Caltech Conf. on VLSI*, pages 509–525, 1979.

[5] C.-H. Huang and C. Lengauer. The derivation of systolic implementations of programs. *Acta Informatica*, 24(6):595–632, Nov. 1987.

[6] 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.

[7] V. K. Prasanna Kumar and Y.-C. Tsai. Designing linear systolic arrays. *J. Parallel and Distributed Computing*, 7(3):441–463, Nov. 1989.

[8] S. Y. Kung. *VLSI Processor Arrays.* Prentice-Hall Int., 1988.

[9] H. W. Lang. The instruction systolic array – a parallel architecture for VLSI. *Integration*, 4:65–74, 1986.

[10] C. Lengauer and J. Xue. A systolic array for pyramidal algorithms. Technical Report ECS-LFCS-90-114, Department of Computer Science, University of Edinburgh, June 1990. To appear in *J. VLSI Signal Processing.*

[11] D. I. Moldovan and J. A. B. Fortes. Partitioning and mapping algorithms into fixed-size systolic arrays. *IEEE Trans. on Computers*, C-35(1):1–12, Jan. 1986.

[12] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.

[13] 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.

[14] P. Quinton. The systematic design of systolic arrays. In *Automata Networks in Computer Science*, chapter 9, pages 229–260. Princeton University Press, 1987. Also: Technical Reports 193 and 216, IRISA (INRIA-Rennes), 1983.

[15] 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.

[16] S. V. Rajopadhye. Synthesizing systolic arrays with control signals from recurrence equations. *Distributed Computing*, 3:88–105, 1989.

[17] S. V. Rajopadhye. Algebraic transformations in systolic array synthesis: A case study. In L. J. M. Claesen, editor, *Formal VLSI Specification and Synthesis (VLSI Design Methods-I)*, pages 361–370. North-Holland, 1990.

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

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

[20] S. K. Rao and T. Kailath. Regular iterative algorithms and their implementations on processor arrays. *Proc. IEEE*, 76(3):259–282, Mar. 1988.

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

[22] Y. Wong and J.-M. Delosme. Broadcast removal in systolic algorithms. In K. Bromley, S.-Y. Kung, and E. E. Swartzlander, editors, *Proc. Int. Conf. on Systolic Arrays*, pages 403–412. IEEE Computer Society, 1988.

[23] J. Xue and C. Lengauer. The synthesis of control signals for one-dimensional systolic arrays. In P. Quinton and Y. Robert, editors, *Algorithms and Parallel VLSI Architectures II.* North-Holland, 1991. To appear.

A Example: LU-Decomposition

A.1 The Problem Specification

LU-decomposition is the unique decomposition of a non-singular $m \times m$ matrix C into a lower-triangular matrix A and an upper-triangular matrix B such that $A \cdot B = C$. The elements of the upper triangle of A and the elements of the lower triangle of B (excluding the diagonal) are 0; the diagonal elements of A are 1.

Specification: $(\forall\, i,j : 0<i,j\leqslant m :\ \sum\, k : 0<k\leqslant m : a_{i,k}b_{k,j} = c_{i,j})$

UREs:

$$\begin{array}{llll}
0<i\leqslant m, 0<j\leqslant m, 0=k & \rightarrow & C(i,j,k) = c(i,j) & \text{(T0)}\\
k<i\leqslant m, m=j \quad , 0<k\leqslant m & \rightarrow & a_{i,k} = A(i,j,k) & \text{(T1)}\\
m=i \quad , k\leqslant j\leqslant m, 0<k\leqslant m & \rightarrow & b_{k,j} = B(i,j,k) & \text{(T2)}\\
k=i \quad , k=j \quad , 0<k\leqslant m & \rightarrow & B(i,j,k) = B(i,j,k-1)^{-1} & \text{(T3)}\\
k<i\leqslant m, k=j \quad , 0<k\leqslant m & \rightarrow & A(i,j,k) = C(i,j,k-1)B(i,j,k) & \text{(T4)}\\
k=i \quad , k<j\leqslant m, 0<k\leqslant m & \rightarrow & B(i,j,k) = C(i,j,k-1) & \text{(T5)}\\
k<i\leqslant m, k<j\leqslant m, 0<k\leqslant m & \rightarrow & A(i,j,k) = A(i,j-1,k) & \multirow{2}{*}{\} (T6)}\\
k<i\leqslant m, k<j\leqslant m, 0<k\leqslant m & \rightarrow & B(i,j,k) = B(i-1,j,k) & \\
k\leqslant i\leqslant m, k\leqslant j\leqslant m, 0<k\leqslant m & \rightarrow & C(i,j,k) = C(i,j,k-1)-A(i,j-1,k)B(i-1,j,k) & \text{(T*)}
\end{array}$$

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

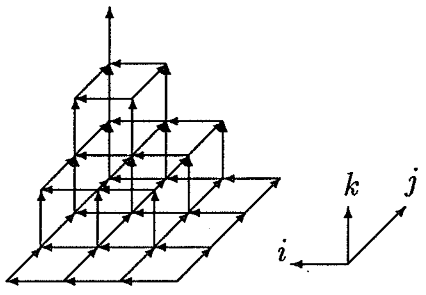
Variables: A, B, C

Data Dependence Matrix: $D = [\vartheta_A, \vartheta_B, \vartheta_C] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Input Space: $\mathsf{in}(\Phi, \vartheta_C) = \{(i,j,1) \mid 0<i\leqslant m, 0<j\leqslant m\}$

Output Spaces: $\mathsf{out}(\Phi, \vartheta_A) = \{(i,m,k) \mid k<i\leqslant m,\ 0<k\leqslant m\}$
$\mathsf{out}(\Phi, \vartheta_B) = \{(m,j,k) \mid k\leqslant j\leqslant m,\ 0<k\leqslant m\}$

Data Dependence Graph $(m=4)$:



A.2 The Construction

A.2.1 Specification of Computation Control

There are four partitions of computation points in the index space:

$$
\begin{aligned}
\Phi_{T3} &= \{(i,j,k) \mid k=i, & k=j, & \quad 0<k\leqslant m\} \\
\Phi_{T4} &= \{(i,j,k) \mid k<i\leqslant m, & k=j, & \quad 0<k\leqslant m\} \\
\Phi_{T5} &= \{(i,j,k) \mid k<i\leqslant m, & k<j\leqslant m, & \quad 0<k\leqslant m\} \\
\Phi_{T6} &= \{(i,j,k) \mid k=i, & k<j\leqslant m, & \quad 0<k\leqslant m\}
\end{aligned}
$$

The points in Φ_{Ti} are specified by Equ. T$*$ and Equ(s). Ti ($3\leqslant i\leqslant 6$).

We interpret the construction of the computation control variables in two different ways.

First, we characterize it by a lattice colouring (Sect. 7.2). We colour the index space as follows:

- To colour paths $p((i,j,k),A,\Phi)$:
  - Use p_0 to colour the paths whose sources (i,j,k) satisfy $i=k \wedge j=k$.
  - Use p_1 for the remaining paths.
- To colour paths $p((i,j,k),B,\Phi)$:
  - Use q_0 to colour the paths whose sources (i,j,k) satisfy $i=k \wedge j=k$.
  - Use q_1 for the remaining paths.

We obtain $C(\Phi_{T3}) = \{(p_0,q_0)\}$, $C(\Phi_{T4}) = \{(p_1,q_0)\}$, $C(\Phi_{T5}) = \{(p_0,q_1)\}$ and $C(\Phi_{T6}) = \{(p_1,q_1)\}$. By Def. 11, the computation control variables are correct. We write P and Q for control variables implementing the colouring of A and B, respectively: $\ell.P = A$, $\mathcal{S}(P)=\{p_0,p_1\}$, $\ell.Q=B$ and $\mathcal{S}(Q)=\{q_0,q_1\}$.

Second, we characterize the construction of computation control variables by separating hyperplanes (Def. 13). The four partitions are formed by dividing the computation points with the two separating hyperplanes $\{(i,j,k) \mid i=k\}$ and $\{(i,j,k) \mid j=k\}$. Thus, we need two computation control variables. Def. 13, with a choice of $(0,1,0)$ and $(0,1,0)$ for the corresponding control dependence vectors yields the previous computation control variables. Clearly, we have more freedom in choosing a control dependence vector by applying Def. 13 than by solving the lattice colouring problem. In the former, we can choose a control dependence vector that is not a data dependence vector of the source UREs.

A.2.2 Specification of Separation Control

We choose $\ell.E=C$. A, B and C are undefined for draining space Ψ_E^{d}. Hence, no termination control variable is needed (Sect. 9.1). P and Q could also serve for termination control if termination control were necessary (Thm. 8). If we chose E to label either A or B, an extension of the index space as described in Sect. 7.1.4 would be necessary. This is because neither $\mathsf{in}(\Phi,\vartheta_A)$ nor $\mathsf{in}(\Phi,\vartheta_B)$ is a facet of Φ. To choose the initialization control variable, we simply find a dependence vector in the space $\mathsf{linear}(\mathsf{in}(\Phi,\vartheta_E))$ (Thm. 3):

$$\mathsf{in}(\Phi,\vartheta_E) = \{(i,j,1) \mid 0<i\leqslant m, 0<j\leqslant m\}$$

Both ϑ_A and ϑ_B are reasonable choices, as data dependence vectors in this space. Let us arbitrarily choose $\ell.F=A$. Spec. 1 is satisfied (Thm. 3). Since termination control is not needed, the separation control variables are correct (Thm. 1).

Note that $\ell.P=\ell.F$. It is possible to combine P and F to one control variable (Sect. 9.4). For this example, this optimization does not save anything and is omitted.

A.3 The UCREs

With $\mathcal{S}(Z)=\{z_{T3}, z_{T4}, z_{T5}, z_{T6}, z_p\}$, where z_{Ti} denotes the points in partition Φ_{Ti} $(3 \leqslant i \leqslant 6)$.

Separation Control Variables F and E:

```
$k \leqslant i \leqslant m, k-1=j, 1<k \leqslant m \rightarrow F(i,j,k) = \perp$
$0<i \leqslant m, 0=j, 1=k \rightarrow F(i,j,k) = f$
$k<i \leqslant m, k<j \leqslant m, 0<k \leqslant m \rightarrow F(i,j,k) = F(i,j-1,k)$
$0<i \leqslant m, 0<j \leqslant m, 0=k \rightarrow E(i,j,k) = e_s$
$k \leqslant i \leqslant m, k \leqslant j \leqslant m, 0<k \leqslant m \rightarrow E(i,j,k) =$ if $E(i,j,k-1)=e_s \wedge F(i,j-1,k)=f \rightarrow e_c$
                                                   [] else $\rightarrow E(i,j,k-1)$
                                                   fi
```

Computation Control Variable P and Q:

```
$k=i, k-1=j, 0<k \leqslant m \rightarrow P(i,j,k) = p_0$
$k<i \leqslant m, k-1=j, 0<k \leqslant m \rightarrow P(i,j,k) = p_1$
$k \leqslant i \leqslant m, k \leqslant j \leqslant m, 0<k \leqslant m \rightarrow P(i,j,k) = P(i,j-1,k)$
$k-1=i, k=j, 0<k \leqslant m \rightarrow Q(i,j,k) = q_0$
$k-1=i, k<j \leqslant m, 0<k \leqslant m \rightarrow Q(i,j,k) = q_1$
$k \leqslant i \leqslant m, k \leqslant j \leqslant m, 0<k \leqslant m \rightarrow Q(i,j,k) = Q(i-1,j,k)$
```

Action Control Variable Z:

With $G(i,j,k) = (E(i,j,k-1)=e_s \wedge F(i,j-1,k)=f) \vee (E(i,j,k-1)=e_c)$

```
$k \leqslant i \leqslant m,\ k \leqslant j \leqslant m,\ 0<k \leqslant m \rightarrow Z(i,j,k) =$
        if $G(i,j,k) \wedge P(i,j-1,k)=p_0 \wedge Q(i-1,j,k)=q_0 \rightarrow z_{T3}$
        [] $G(i,j,k) \wedge P(i,j-1,k)=p_1 \wedge Q(i-1,j,k)=q_0 \rightarrow z_{T4}$
        [] $G(i,j,k) \wedge P(i,j-1,k)=p_0 \wedge Q(i-1,j,k)=q_1 \rightarrow z_{T5}$
        [] $G(i,j,k) \wedge P(i,j-1,k)=p_1 \wedge Q(i-1,j,k)=q_1 \rightarrow z_{T6}$
        [] else $\rightarrow z_p$
        fi
```

A.4 The Space-Time Mapping

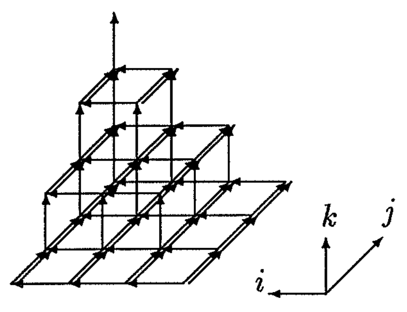
Choose the following space-time mapping:

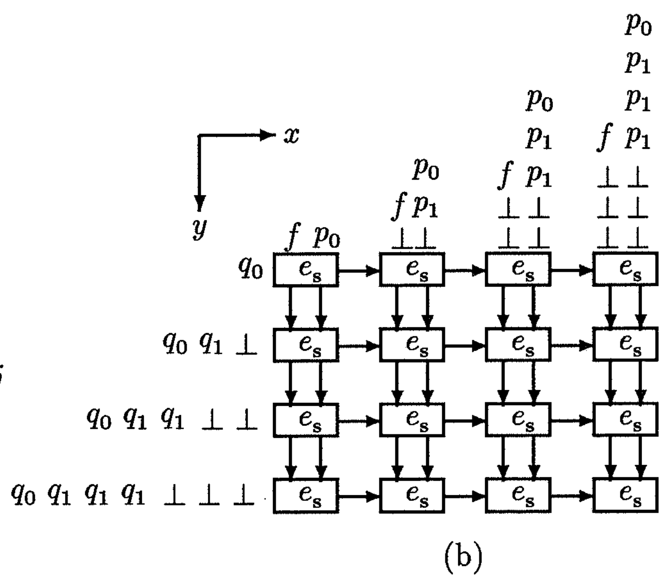
$$\Pi = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

Because every cell handles only one type of computation points, no computation control is required (Sect. 9.2.1).

Choose the following space-time mapping:

$$\Pi = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$





x
y
q_0
$q_0\ q_1\ \perp$
$q_0\ q_1\ q_1\ \perp\ \perp$
$q_0\ q_1\ q_1\ q_1\ \perp\ \perp\ \perp$
$f\ p_0$
$p_0\ f\,p_1\ \perp\perp$
$p_0\ p_1\ f\ p_1\ \perp\perp\ \perp\perp$
$p_0\ p_1\ p_1\ f\ p_1\ \perp\perp\ \perp\perp\ \perp\perp$
e_s
(b)

Figure 6: (a) The control dependence graph for LU-decomposition ($m = 4$). (b) The distribution of control signals at the first step ($m = 4$). Only the channels that carry control variables are shown. All input sequences extend to infinity with $\perp$, but only leading signals $\perp$ are shown. The control signals e_s are loaded before the start of the computation. The coordinate system for the systolic array conforms to $\Pi(i,j,k) = (x,y,t)$.

Some cells of the array handle different types of computation points. Hence, computation control is needed. See Fig. 6.

Note that only control signal (p_0, q_0) encounters cell $(1,1)$. Hence, the guarded command of the action control variable can be simplified for that cell (Sect. 9.5). Note also that only computation point $(1,1,1)$, which is in partition Φ_{T3}, is mapped to cell $(1,1)$. Thus, the specification of action control at that cell can be simplified to:

$$\begin{aligned} Z_{\text{out}} \;=\; & \textbf{if}\ P_{\text{in}} = p_0 \wedge Q_{\text{in}} = q_0 \rightarrow z_{\text{T3}} \\ & [\,]\ \textbf{else} \rightarrow z_{\text{p}} \\ & \textbf{fi} \end{aligned}$$

Similar optimizations can be performed for other cells.