LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - The University of Edinburgh

The Synthesis of Control Signals for One-Dimensional Systolic Arrays

by

Jingling Xue
Christian Lengauer

LFCS Report Series **ECS-LFCS-91-156**

LFCS **June 1991**

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

Copyright © 1991, LFCS

Copyright © 1991, Laboratory for Foundations of Computer Science,
University of Edinburgh. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

The Synthesis of Control Signals for One-Dimensional 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–91–156
JUNE 1991

Abstract

We apply a previously reported synthesis method of control signals for systolic arrays [19]. In [19], a systolic array is described by an index transformation expressed as a non-singular square integer matrix, which happens to be a bijection from $\mathbb{Q}^n$ to $\mathbb{Q}^n$. The method expects uniform recurrence equations (source UREs) and returns a specification of control signals in terms of another set of UREs (control UREs); n is the number of indices in the source UREs. We apply this method to the special case of one-dimensional systolic arrays; they are described by index transformations from $\mathbb{Q}^n$ to $\mathbb{Q}^2$. This requires a modification of the part of the method that depends on the bijectivity of the space-time mapping.

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

Contents

1 Introduction 1

2 The Model 2

3 The Source: A System of UREs 3

4 The Target: A One-Dimensional Systolic Array 4
- 4.1 The Space-Time Mapping . . . 5
- 4.2 The Space-Time Diagram . . . 6
- 4.3 The Generation of Pipelining Points . . . 9

5 The Derivation of Control Signals 10

6 The Specification of Computation Control Variables 12

7 The Specification of Separation Control Variables... 13

8 ...for Three-Dimensional Source UREs 13
- 8.1 Initialization, Termination and Evolution Control Variables . . . 13
- 8.2 The Construction of Input and Computation Equations . . . 14
  - 8.2.1 The Host Program . . . 15
  - 8.2.2 The Cell Program . . . 16
- 8.3 The Construction of Control Dependence Vectors . . . 18
- 8.4 Adaptation of the Source . . . 21
- 8.5 Hardware Support . . . 23

9 ...for n-Dimensional Source UREs ($n>3$) 23
- 9.1 The Construction of Control Dependence Vectors . . . 23
- 9.2 The Construction of Input and Output Equations . . . 24
  - 9.2.1 The Host Program . . . 25
  - 9.2.2 The Cell Program . . . 25
- 9.3 The Proof of Correctness . . . 26
- 9.4 Adaptation of the Source . . . 26
- 9.5 Hardware Support . . . 27

10 The Elimination of Control Signals 27

11 Related Work 28

12 Conclusions 28

13 Acknowledgements 29

14 References 29

A Examples 30
A.1 Matrix Product 30
A.1.1 Three Moving Data Variables 31
A.1.2 Two Moving Data Variables 31
A.1.3 One Moving Data Variable 31
A.2 LU-Decomposition 31
A.2.1 The Specification of Computation Control Variables 32
A.2.2 The Specification of Separation Control Variables 33
A.2.2.1 Extension of Index Space 33
A.2.2.2 Separating Hyperplanes 33
A.2.2.3 Comparison of the Two Methods 34

1 Introduction

Elsewhere, we have presented a method on the synthesis of control signals for systolic arrays [19]. There, a systolic array is described by an index transformation that is expressed as a non-singular square integer matrix, where the first row is a scheduling vector and the remaining rows form an allocation matrix. The method complements previous work on data flow synthesis [12, 16]. Starting with a system of uniform recurrence equations (source UREs) for systolic design, our method constructs a specification of control signals for systolic arrays in terms of another set of UREs called *uniform control recurrence equations* (UCREs). An application of the standard space-time mapping to these two systems of equations delivers a specification of a systolic array; essentially, it describes both data and control signals in space and time.

Our method has the following advantages. First, we specify the control signals for systolic arrays at the source level; here, we are concerned with correctness. Then, we derive a systolic array by means of the space-time mapping; here, we are concerned with efficiency. A direct benefit from separating the two concerns of correctness and efficiency is that the UCREs, once proved correct at the source level, are independent of the space-time mapping and, therefore, not array-specific. Second, the correctness of the UCREs is easily preserved when the source UREs and the UCREs are partitioned and mapped to a fixed-size systolic array. Third, the method can be generalized in a systematic way to systolic arrays of reduced dimension. The case of dimension 1 is the topic of this paper.

To make the presentation more precise, we duplicate the terminology of UREs for UCREs and prefix the words "data" and "control", respectively. That is, we speak of data variables vs. control variables, and so on. When we speak of UCREs we always mean the UCREs derived for a given system of source UREs.

The UCREs consists of two types of control variables: *computation control variables* determine how the points in the index space are evaluated, and *separation control variables* determine the pipelining of input values from border cells to internal cells and output values from internal cells to border cells.

Conventionally, two distinct points of the index space must not be scheduled simultaneously at the same cell, regardless of the dimensions of the systolic array. Based on this premiss, we have developed a provably correct construction that provides the UCREs for the computation control variables. This construction works also for one-dimensional systolic arrays. The separation control variables must indicate correctly when data need to be pipelined at what cells. This information depends on the space-time mapping. 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 but that expect the index space to be in a certain shape.

We prefer the latter. Index spaces that do not conform to the expected shape can be made to do so (Sects. 8.4 and 9.4). To make the UCREs for separation control variables correct, we must choose their control dependence vectors wisely. In [19], the construction of the UCREs

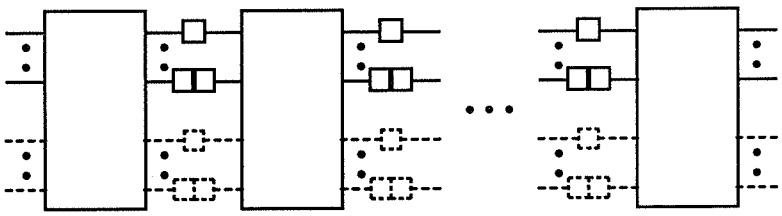


Figure 1: The one-dimensional systolic array model. The larger solid boxes represent cells. The lines represent connecting channels; solid lines carry data streams and dashed lines control streams. A sequence of small boxes symbolizes a sequence of delay buffers.

for the separation control variables relies on the bijectivity of the space-time mapping. In the present paper, we are concerned with the synthesis of control signals for one-dimensional systolic arrays. Their space-time mappings are, in general, not bijective (Lemma 2).

Sect. 6 contains a brief description of a method reported elsewhere on the construction of UCREs for the computation control variables [19]. Sect. 7 focuses on the construction of the UCREs for the separation control variables. Beside the final product, i.e., the UCREs for the separation control variables, the construction itself is interesting in its own right. It demonstrates how a system of UREs that respects some pre-defined specification can be constructed at the source level, independent of the space-time mapping, by exploiting certain properties common to all valid space-time mappings (Thm. 1).

The combination of this paper and [18] for data flow provides a complete framework for the synthesis of one-dimensional systolic arrays.

We do not consider two-dimensional UREs here; they can already be treated with the simpler method [19]. We present our method first for three-dimensional UREs and generalize it subsequently to n-dimensional UREs. To avoid unduly complex notation, the UCREs are expressed in the form of programs: one program for the host corresponds to the input equations, one program for the cells corresponds to the computation equations.

2 The Model

Definition 1 A *one-dimensional systolic array* consists of a finite sequence of cells with the following properties (Fig. 1).

1. The array is driven by a global clock that ticks in unit time. Each cell is active in every cycle.

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

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

4. There are two types of streams: data and control streams. A stream moves along a single dedicated channel between any two neighbouring cells. All channels associated

with a fixed stream have the same non-negative number of buffers. A buffer retains a value for one cycle. This means that a stream moves with a constant velocity.

5. A cell consists of a functional unit, a control unit and memory. □

We refer to the sequence of channels dedicated to a stream as the *link* for that stream. This model has been used previously in the synthesis of data flow for one-dimensional arrays [9, 14].

3 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 write $\dim(S)$ for the dimension of an integral polytope $S \subset \mathbb{Z}^n$.

Following [4], we denote quantification by $(\mathrm{q}\ v : \mathrm{r}(v) : \mathrm{e}(v))$; q is a quantifier, v the dummy variable that is bound by the quantification, $\mathrm{r}(v)$ is a predicate that specifies the range of v and $\mathrm{e}(v)$ is an expression in v.

We write a URE in the format: domain predicate $\longrightarrow$ recurrence equation [12]. All variables have n indices. The domain for which variable v is defined is denoted by Φ_v. The index space Φ is a union of all these domains; it is assumed to be a (convex) polytope of n dimensions. If these conditions are satisfied, we say that the source UREs are *n-dimensional.*

We use the conventional concept of a data dependence graph and a data dependence matrix [16]. For notational convenience, we assume that, for each data variable, there is only one associated data dependence vector, denoted ϑ_v. When we write $\vartheta_v \in D_\mathrm{d}$, we mean that ϑ_v is a data dependence vector, which is a column of the data dependence matrix D_d. The subscript d stands for "data". D_c denotes the control dependence matrix for the UCREs; the subscript c stands for "control". D denotes the dependence matrix whose columns are taken from both D_d and D_c.

The points in the index space are called *computation points.* If I is inside and $I-\vartheta_v$ is outside the index space, then I is called a *first* computation point of v and variable $v(I-\vartheta_v)$ defines an *input value.* If I is inside and $I+\vartheta_v$ is outside the index space, then I is called a *last* computation point of v and variable $v(I)$ defines an *output value.*

Definition 2 The set $\mathsf{in}(\Phi, \vartheta_v)$ of first computation points of v and the set $\mathsf{out}(\Phi, \vartheta_v)$ of last computation points of v are given by:

$$\mathsf{in}(\Phi, \vartheta_v) = \{I \mid I \in \Phi,\ I - \vartheta_v \notin \Phi\} \qquad \mathsf{out}(\Phi, \vartheta_v) = \{I \mid I \in \Phi,\ I + \vartheta_v \notin \Phi\}$$

The set of input values of v is given by $\Phi_v^{\mathsf{in}} = \{I \mid (I+\vartheta_v) \in \mathsf{in}(\Phi, \vartheta_v)\}$. □

Def. 2 implies that input values are used and output values are defined at the boundary of the index space. If this is not the case, they can be made so by adding pipelining equations [15]; a *pipelining equation* is of the form $v(I) = v(I-\vartheta_v)$. 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 values, 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.* We abbreviate $(\exists\ m : m \in \mathbb{Z}_0^+ : J = I + m\vartheta_v)$ to $I \xrightarrow{v} J$ and its negation to $I \not\xrightarrow{v} J$.

Example: Matrix Product

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

UREs:

$$\begin{array}{llll}
0<i\leqslant m, & 0<j\leqslant m, & k=m & \rightarrow\ c_{i,j} = C(i,j,k) \\
0<i\leqslant m, & 0<j\leqslant m, & 0<k\leqslant m & \rightarrow\ C(i,j,k) = C(i,j,k-1) + A(i,j-1,k)B(i-1,j,k) \\
0<i\leqslant m, & 0<j\leqslant m, & 0=k & \rightarrow\ C(i,j,k) = 0 \\
0<i\leqslant m, & 0<j\leqslant m, & 0<k\leqslant m & \rightarrow\ A(i,j,k) = A(i,j-1,k) \\
0<i\leqslant m, & 0=j, & 0<k\leqslant m & \rightarrow\ A(i,j,k) = a_{i,k} \\
0<i\leqslant m, & 0<j\leqslant m, & 0<k\leqslant m & \rightarrow\ B(i,j,k) = B(i-1,j,k) \\
0=i, & 0<j\leqslant m, & 0<k\leqslant m & \rightarrow\ B(i,j,k) = b_{k,j}
\end{array}$$

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

Variables: A, B, C

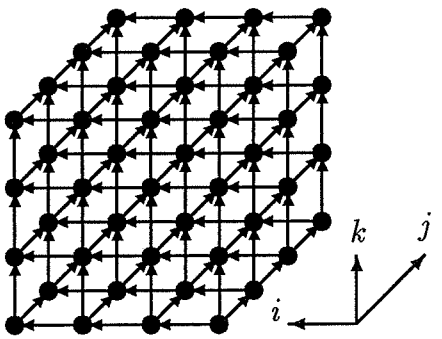
Data Dependence Matrix: $D_{\mathrm{d}} = [\vartheta_A, \vartheta_B, \vartheta_C] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

First Computation Points:

$$\begin{aligned}
\mathsf{in}(\Phi, \vartheta_A) &= \{(i,1,k) \mid 0<i\leqslant m,\ 0<k\leqslant m\} \\
\mathsf{in}(\Phi, \vartheta_B) &= \{(1,j,k) \mid 0<j\leqslant m,\ 0<k\leqslant m\} \\
\mathsf{in}(\Phi, \vartheta_C) &= \{(i,j,1) \mid 0<i\leqslant m,\ 0<j\leqslant m\}
\end{aligned}$$

Last Computation Points: $\mathsf{out}(\Phi, \vartheta_C) = \{(i,j,m) \mid 0<i\leqslant m,\ 0<j\leqslant m\}$

Data Dependence Graph $(m=4)$*:*



4 The Target: A One-Dimensional Systolic Array

This section describes the synthesis of the data flow in one-dimensional systolic arrays from the source UREs by means of the space-time mapping. In particular, it defines the validity

of the space-time mapping and contains a study of the space-time behaviour of the resulting systolic array. This provides a basis for the derivation of the UCREs that will be described in Sect. 5.

4.1 The Space-Time Mapping

The synthesis of systolic arrays amounts to finding a suitable *space-time mapping*, a linear transformation of the index space [12, 16]. In the transformed UREs, one index represents time and the remaining indices represent processor coordinates.

Definition 3 A space-time mapping consists of two components: step and place. Useful functions defined in terms of step and place are flow, input and output.

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

- $\mathsf{place} : \Phi \longrightarrow \mathbb{Z}$, $\mathsf{place}(I) = \sigma I$, $\sigma \in \mathbb{Z}^n$. place specifies the spatial distribution. σ is the *allocation vector*. $\{\sigma I \mid I \in \Phi\}$ is called the *processor space*. I is computed at location σI. $p_{\min}$ and $p_{\max}$ are the coordinate of the leftmost and rightmost cell: $p_{\min} = \min\{\sigma I \mid I \in \Phi\}$ and $p_{\max} = \max\{\sigma I \mid I \in \Phi\}$.

- $\mathsf{flow} : V \longrightarrow \mathbb{Q}$, $\mathsf{flow}(v) = \sigma\vartheta_v / \lambda\vartheta_v$, $\vartheta_v \in D$. flow specifies the velocity at which elements of a variable travel at each step. V denotes the set of data variables. Variable v is called *moving* if $\mathsf{flow}(v) \neq 0$ and *stationary* if $\mathsf{flow}(v) = 0$.

- $\mathsf{input} : V \longrightarrow (\Phi_{\text{in}} \longrightarrow \mathbb{Z})$, $\Phi_{\text{in}} = (\bigcup\ v : v \in V : \mathsf{in}(\Phi, \vartheta_v))$,

$$\mathsf{input}(v(I)) \;=\; \begin{cases} \mathsf{step}(I) - (\mathsf{place}(I) - p_{\min})/\mathsf{flow}(v) & \text{if } \mathsf{flow}(v) > 0 \\ \mathsf{step}(I) - (\mathsf{place}(I) - p_{\max})/\mathsf{flow}(v) & \text{if } \mathsf{flow}(v) < 0 \end{cases}$$

  input specifies the steps at which input values are injected into the array. t_{fst} is the first such step: $t_{\text{fst}} = \min\{\mathsf{input}(v(I)) \mid I \in \Phi_{\text{in}}, v \in V\}$.

- $\mathsf{output} : V \longrightarrow (\Phi_{\text{out}} \longrightarrow \mathbb{Z})$, $\Phi_{\text{out}} = (\bigcup\ v : v \in V : \mathsf{out}(\Phi, \vartheta_v))$,

$$\mathsf{output}(v(I)) \;=\; \begin{cases} \mathsf{step}(I) - (\mathsf{place}(I) - p_{\max})/\mathsf{flow}(v) & \text{if } \mathsf{flow}(v) > 0 \\ \mathsf{step}(I) - (\mathsf{place}(I) - p_{\min})/\mathsf{flow}(v) & \text{if } \mathsf{flow}(v) < 0 \end{cases}$$

  output specifies the steps at which output values are ejected from the array. t_{lst} is the last such step: $t_{\text{lst}} = \max\{\mathsf{output}(v(I)) \mid I \in \Phi_{\text{out}}, v \in V\}$. □

We make the following assumptions (gcd stands for the greatest common denominator):

- $(\exists\ v : v \in V : \mathsf{flow}(v) \neq 0)$
- $(\text{gcd}\ i : 0 < i \leqslant n : \sigma_i) = 1$
- $(\text{gcd}\ v : v \in V : \textbf{if}\ \mathsf{flow}(v) = 0 \rightarrow \lambda\vartheta_v\ []\ \mathsf{flow}(v) \neq 0 \rightarrow \lambda\vartheta_v / \sigma\vartheta_v\ \textbf{fi}) = 1$

The conditional statement is from [3]. The first assumption is reasonable because an array that has no moving variables cannot be called systolic. The second and third assumption are for convenience: they normalize **step** and **place**.

Not every space-time mapping describes a systolic array. The following theorem [18] states necessary and sufficient conditions for validity of a space-time mapping.

Theorem 1 *Assume that evaluation of a point takes unit time. A space-time mapping is valid iff the following conditions are satisfied.*

1. *Precedence Constraint:* $(\forall\, v : v \in V : \lambda\vartheta_v \geqslant 1)$.
2. *Delay Constraint:* $(\forall\, v : v \in V : \mathsf{flow}(v) \neq 0 \implies |\lambda\vartheta_v / \sigma\vartheta_v| \in \mathbb{Z}^+)$.
3. *Communication Constraint:*
   $(\forall\, v : v \in V : \mathsf{flow}(v) \neq 0 \implies (\forall\, I, J : I, J \in \mathsf{in}(\Phi, \vartheta_v) \wedge I \neq J : \mathsf{input}(v(I)) \neq \mathsf{input}(v(J))))$.

The image of some x under a given space-time mapping is denoted by an overbar: $\overline{x}$. We sometimes write Π for λ and σ in combination:

$$\Pi = \begin{bmatrix} \lambda \\ \sigma \end{bmatrix} = \begin{bmatrix} \lambda_1, \ldots, \lambda_n \\ \sigma_1, \ldots, \sigma_n \end{bmatrix}$$

If one allows non-neighbouring connections, the image $\overline{\vartheta}_v$ of ϑ_v is $\begin{bmatrix} \lambda\vartheta_v \\ \sigma\vartheta_v \end{bmatrix}$, where $\lambda\vartheta_v - 1$ represents the number of delay buffers associated with channel $\sigma\vartheta_v$. Because of the restriction to neighbouring communication, the image $\overline{\vartheta}_v$ of ϑ_v is given by:

$$\overline{\vartheta}_v = \textbf{if } \mathsf{flow}(v) = 0 \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} \;[]\; \mathsf{flow}(v) \neq 0 \rightarrow \begin{bmatrix} \lambda\vartheta_v / |\sigma\vartheta_v| \\ \mathsf{sign}(\mathsf{flow}(v)) \end{bmatrix} \textbf{ fi}$$

where **sign** is defined as usual: $\mathsf{sign}(x) = \textbf{if } x > 0 \rightarrow 1 \;[]\; x = 0 \rightarrow 0 \;[]\; x < 0 \rightarrow -1 \textbf{ fi}$. $\lambda\vartheta_v / |\sigma\vartheta_v| - 1$ represents the number of delay buffers associated with channel $\mathsf{sign}(\mathsf{flow}(v))$ between any two neighbouring cells. If v is stationary, this interpretation is still valid if we presume the existence of a loop channel, for v, at every cell.

4.2 The Space-Time Diagram

Syntactically, a systolic array is a set, Υ, of points called *space-time points*:

$$\Upsilon = \{(t, x) \mid t_{\mathrm{fst}} \leqslant t \leqslant t_{\mathrm{lst}},\ p_{\min} \leqslant x \leqslant p_{\max},\ t, x \in \mathbb{Z}\}$$

where t represents time and x space. Because of the restrictions of our systolic array model, cell x is active at step t iff $(t, x) \in \Upsilon$. Semantically, space-time points fall into two categories:

- The set $\overline{\Phi}$ of *computation points* is the image of Φ. We call both the points in $\overline{\Phi}$ and the points in Φ computation points. This is justified because the communication constraint ensures that a valid space-time mapping is a bijection from Φ to $\overline{\Phi}$ [18]. We say that two computation points are of the same *type* if, for every data variable of the source UREs, the defining equation at both points are the same.

- The set Υ^{p} of *pipelining points* is the complement of $\overline{\Phi}$ in Υ. These points arise due to the restrictions of our model. Pipelining points are specified by pipelining equations. This follows from the nature of the space-time mapping.

It is often convenient to represent a one-dimensional systolic array as a *space-time diagram*, which is obtained by viewing the space-time points as a two-dimensional lattice, where the horizontal axis represents time and the vertical axis represents space.

Let us introduce some concepts that will emphasize the regular distribution of pipelining points relative to computation points in the space-time diagram.

Definition 4 Consider an integral polytope $S \subset \mathbb{Z}^n$. A ϑ_v*-path* in S consists of all points $I \in S$ such that 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 the ϑ_v-paths (of v): $P(v, S) = \{p(I, v, S) \mid I \in S\}$. □

In Def. 4, $p(I, v, S)$ and $p(J, v, S)$ denote the same ϑ_v-path if $I - J$ is an integral multiple of ϑ_v. Def. 4 implies that the union of all ϑ_v-paths in S, for any fixed v, is S. When we refer to a $\overline{\vartheta}_v$-path, we mean a path in the space-time diagram Υ.

The first point (i.e. the source) of a $\overline{\vartheta}_v$-path is called an *input point* of v. The last point (i.e., the target) of a $\overline{\vartheta}_v$-path is called an *output point* of v. The elements of a variable are input at its input points and output at its output points. Of course, only one element can be input to and output from any $\overline{\vartheta}_v$-path.

There are two types of $\overline{\vartheta}_v$-paths (Fig. 2):

- Paths that contain computation points. These paths can be viewed as consisting of the following three consecutive segments:
  - The first segment, called the *soaking path* and denoted $s(\overline{I}, v, \Upsilon)$, consists of pipelining points. The points of this segment are called *soaking points.*
  - The second segment, called the *computation-relaying path* and denoted $c(\overline{I}, v, \Upsilon)$, consists of both computation and pipelining points. The first point of the segment is the image of a first, the last point the image of a last computation point of v. We also call the points in $\overline{\mathsf{in}(\Phi, \vartheta_v)}$ ($\overline{\mathsf{out}(\Phi, \vartheta_v)}$) the first (last) computation points of v, because a valid space-time mapping preserves dependence and, thus, order (Thm. 1 and Lemma 3). The pipelining points of the segment are called *relaying points.* There are $\Gamma_v - 1$ relaying points between every two neighbouring computation points (e.g., see the ϑ_B-paths in Fig. 2; in this case, $\Gamma_B = 3$):

$$\Gamma_v = \textbf{if } \mathsf{flow}(v) = 0 \rightarrow \lambda \vartheta_v \ [] \ \mathsf{flow}(v) \neq 0 \rightarrow |\sigma \vartheta_v| \textbf{ fi}$$

  - The third segment, called the *draining path* and denoted $d(\overline{I}, v, \Upsilon)$, consists of pipelining points. The points of this segment are called *draining points.*
- Paths that do not contain computation points. The points of these paths are called *undefined points* (e.g., see the dashed arrow in Fig. 2).

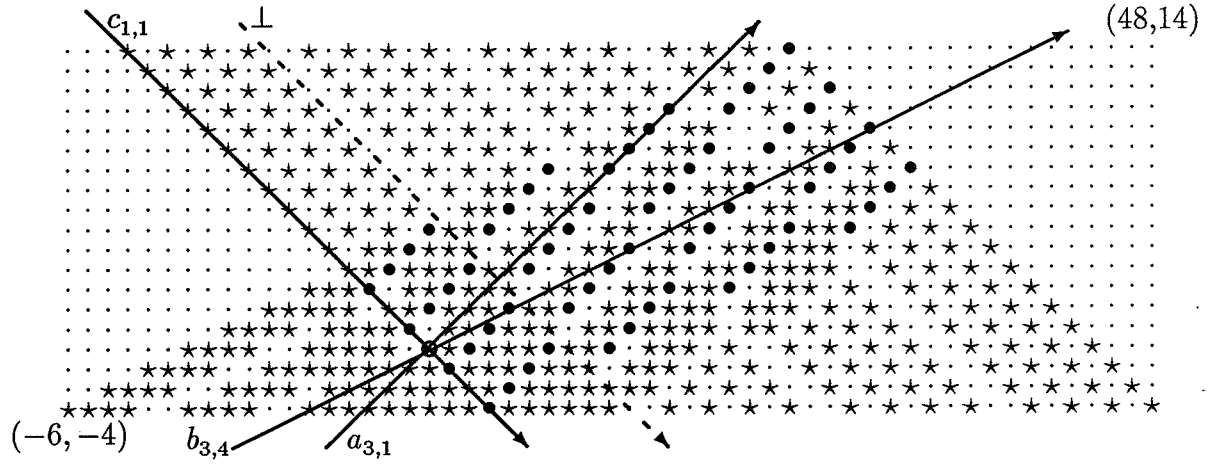


Figure 2: The space-time diagram for Ramakrishnan and Varman's one-dimensional systolic array ($m=4$). The array is described by the following space-time mapping [9]:

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

The point at the bottom-left corner is $(t_{\text{fst}}, p_{\text{min}}) = (-6, -4)$, that at the top-right corner is $(t_{\text{lst}}, p_{\text{max}}) = (48, 14)$. The fat dots represent computation points. The regular dots and stars represent pipelining points; the stars are the points at which at least one data element is propagated. The output of A and B is of no interest and is not depicted. Three solid arrows denote three paths: a $\overline{\vartheta}_A$-path, a $\overline{\vartheta}_B$-path and a $\overline{\vartheta}_C$-path, to which $a_{3,1}$, $b_{3,4}$ and $c_{1,1}$ are input, respectively. The dashed arrow denotes a ϑ_C-path, to which $\perp$ is input.

Note that the definitions of soaking, draining, relaying and undefined points are associated with a particular variable. For example, a pipelining point that is a soaking point for one variable may be either a soaking or draining or relaying or undefined point for another variable.

The soaking points of v serve to propagate the input values of v from its input points to its first computation points; the draining points of v serve to propagate the output values of v from its last computation points to its output points; the relaying points of v serve to employ the intermediate cells as delay buffers for relaying the values of v between non-neighbouring cells (e.g., see the ϑ_B-path highlighted in Fig. 2). The concept of soaking and draining serves to satisfy the restriction of border communication; the concept of relaying serves to satisfy the restriction of neighbouring communication.

We denote the set of soaking points, the so-called *soaking space*, of v by Υ_v^{s}, the set of draining points, the so-called *draining space*, of v by Υ_v^{d} and the set of undefined points, the so-called *undefined space*, of v by $\Upsilon_v^{\perp}$. For convenience, we call the set of computation points the *computation space*; it is the same for all variables. By definition, $\{\Upsilon_v^{\text{s}}, \Upsilon_v^{\text{d}}, \Upsilon_v^{\perp}\}$ is a partitioning of Υ^{p}. Therefore, $\{\Upsilon_v^{\text{s}}, \Upsilon_v^{\text{d}}, \Upsilon_v^{\perp}, \overline{\Phi}\}$ is a partitioning of Υ.

We can rephrase the communication constraint with respect to the concept of a $\overline{\vartheta}_v$-path.

Lemma 1 *The communication constraint is satisfied iff*

$$(\forall\, v : v \in V : \mathsf{flow}(v) \neq 0 \implies (\forall\, I, J : I, J \in \mathsf{in}(\Phi, \vartheta_v) \wedge I \neq J : p(\overline{I}, v, \Upsilon) \neq p(\overline{J}, v, \Upsilon))).$$

Proof. If, for any $I, J \in \mathsf{in}(\Phi, \vartheta_v)$, $p(\overline{I}, v, \Upsilon)$ and $p(\overline{J}, v, \Upsilon)$ denote the same $\overline{\vartheta}_v$-path, then $v(I)$ and $v(J)$ must be input at the same time, and vice versa. □

Remark.
A $\overline{\vartheta}_v$-path of data variable v may start at an internal cell in the following two cases.

1. If v is a stationary, all $\overline{\vartheta}_v$-paths start at internal cells. The elements of the variable must be loaded before the computation starts, i.e., before step t_{fst}.

2. If v is a moving, some $\overline{\vartheta}_v$-paths may start at internal cells. In this case, the input value must be $\perp$; it can be ignored, i.e., need not be supplied by the host. □

4.3 The Generation of Pipelining Points

This section explains how pipelining points are generated by means of a two-step extension of the index space. The presence of pipelining points complicates the specification of control signals; we introduce separation control variables to deal with this complication.

First, to impose the restriction of border communication, the index space is extended in such a way that the inverse images of input and output points are at the boundary of the extended index space. The basic idea is to replicate a data dependence vector in the data dependence graph forward and backward until the images of points so created are outside the processor space [15]. This method works only for moving variables.

Next, to impose the restriction of neighbouring communication, the directed arc represented by ϑ_v between any two nodes, say I and J with $J = I + \vartheta_v$, in the data dependence graph that was created in the first step is sliced into Γ_v consecutive directed arcs ϑ_v/Γ_v. This creates $\Gamma_v - 1$ points in the original arc. Consider one such point K; the cell at location $\mathsf{place}(K)$ serves at $\mathsf{step}(K)$ as a delay buffer to propagate an element of v from the cell at location $\mathsf{place}(I)$ to the cell at location $\mathsf{place}(J)$.

Definition 5 The *extended index space* is defined as follows:

$$\Psi_v = \begin{cases} \{I \mid I = J + m\vartheta_v/\lambda\vartheta_v,\ J \in \Phi,\ J + \vartheta_v \in \Phi,\ m \in \mathbb{Z}_0^+,\ 0 \leqslant m \leqslant \lambda\vartheta_v\} & \text{if } \mathsf{flow}(v) = 0 \\ \{I \mid I = J + m\vartheta_v/\sigma\vartheta_v,\ J \in \Phi,\ m \in \mathbb{Z},\ p_{\min} \leqslant \sigma I \leqslant p_{\max}\} & \text{if } \mathsf{flow}(v) \neq 0 \end{cases}$$
$$\Psi = (\bigcup\ v : v \in V : \Psi_v)$$

The portion attributed to variable v is Ψ_v. □

The points in $\Psi \setminus \Phi$ may be rational rather than integral: $\Psi \subset \mathbb{Q}^n$. The images of the points in $\Psi \setminus \Phi$ constitute all space-time points at which at least one datum is propagated. The defining equation of data variable v at points $\Psi_v \setminus \Psi$ is a pipelining equation.

We rephrase the communication constraint by considering the space-time mapping as a mapping from Ψ_v to Υ.

Lemma 2 *The communication constraint is satisfied iff*

$$(\forall\ v : v \in V : (\forall\ I, J : I, J \in \Psi_v \wedge I \neq J : \mathsf{place}(I) = \mathsf{place}(J) \implies \mathsf{step}(I) \neq \mathsf{step}(J))).$$

Proof. The definition of **input** and the communication constraint. □

In general, valid space-time mappings from Ψ to Υ are not injective; one example is the mapping displayed in Fig. 2.

The communication constraint can also be restated with emphasis on the correspondence between ϑ_v-paths in the extended index space and $\overline{\vartheta}_v$-paths in the space-time diagram.

Lemma 3 *The communication constraint is satisfied iff*

$$(\forall\ v : v \in V : \mathsf{flow}(v) \neq 0 \implies (\forall\ I, J : I, J \in \Psi : p(I, v, \Psi) \neq p(J, v, \Psi) \implies p(\overline{I}, v, \Upsilon) \neq p(\overline{J}, v, \Upsilon))).$$

Proof. Lemma 2. □

Lemma 3 states that a valid space-time mapping is an injection from $P(v, \Psi)$ to $P(v, \Upsilon)$, but it is in general not a surjection. This is the reason for the existence of $\overline{\vartheta}_v$-paths that contain no computation points (Fig. 2).

Let us now examine the complication caused by the presence of pipelining points in the synthesis of control signals for one-dimensional systolic arrays.

A valid space-time mapping is injection from Ψ_v to Υ, for any fixed v (Lemma 2). But, it is not an injection from Ψ to Υ. Two points, one in $\Psi_v \backslash \Phi$ and another in $\Psi_w \backslash \Phi$, for different v and w, may share the same image. Take the pipelining point $(12, -1)$ highlighted in Fig. 2 by a circle. Elements $a_{3,1}$, $b_{3,4}$, and $c_{1,1}$ are propagated at this point. A simple calculation shows that they are specified by variables $A(3, -8, 1)$, $B(1/3, 4, 3)$ and $C(1, 1, 5/2)$ in the extended index space. The problem is that the inverse images of pipelining points depend on the space-time mapping.

Hence, a correctness criterion for the UCREs for the separation control variables has to involve the space-time mapping (Sect. 7). But our method requires that the UCREs be specified at the source level, independent of the space-time mapping. We accomplish this by relying on certain properties common to all valid space-time mappings (Lemma 7) and by exploiting the uniformity of pipelining points in the space-time diagram with an additional, so-called evolution control scheme.

5 The Derivation of Control Signals

The specification of a control variable, v, proceeds in three steps:

1. Choose a non-zero vector $\vartheta_v \in \mathbb{Z}^n$ that satisfies the restriction $(\forall\ I, J : I, J \in \Phi : (\exists\ m : m \in \mathbb{Q}^+ : I - J = m\vartheta_v \implies m \geqslant 1))$ as the control dependence vector. ϑ_v is the smallest index distance between any two points in the index space with a common orientation. This is to enforce the restriction to one link per variable (Def. 1).

2. Define one or more input equations that specify the initialization of v by control signals.

3. Define the computation equation that defines the value of $v(I)$ in terms of the value of variable $v(I - \vartheta_v)$.

The output of control signals is irrelevant.

Several factors that may influence the choice of a control dependence vector for variable v are discussed in [19]. It is best to choose one of the data dependence vectors; we write $\ell.v$ for the corresponding data variable: $\vartheta_v = \vartheta_{\ell.v}$. This implies $\mathsf{flow}(v) = \mathsf{flow}(\ell.v)$. There are two advantages. First, a step function that is valid for the source UREs is also valid for the UCREs. Second, the latency of the array (i.e., the number of steps required) is retained.

Control variable v is specified by one of two types of computation equations: a pipelining equation, which is of the form $v(I) = v(I - \vartheta_v)$ (Sect. 3), or a *data-dependent equation*, which is of the following form:

$$
\begin{array}{rcl}
v(I) & = & \textbf{if } \mathcal{B}_0(w(I-\vartheta_w),\cdots) \rightarrow f_0(v(I-\vartheta_v)) \\
& & [] \ \mathcal{B}_1(w(I-\vartheta_w),\cdots) \rightarrow f_1(v(I-\vartheta_v)) \\
& & \quad \vdots \\
& & [] \ \mathcal{B}_{r-1}(w(I-\vartheta_w),\cdots) \rightarrow f_{r-1}(v(I-\vartheta_v)) \\
& & \textbf{fi}
\end{array}
$$

$\mathcal{B}_i(w(I-\vartheta_w),\cdots) \rightarrow f_i(I-\vartheta_v)$ is called a *guarded command* [3]; w is a control variable. The three dots stands for an arbitrary fixed number of control variables. ϑ_v and ϑ_w are the control dependence vectors of v and w, respectively. The *guard* $\mathcal{B}_i(w(I_w - \vartheta_w),\cdots)$ is a Boolean expression. A guard can always be written in disjunctive normal form, where each disjunct consists of a conjunction of tests of an argument for 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 $v(I-\vartheta_v)$. The evaluation of guards is deterministic. We require that one and only one guard is true at a point.

A control variable is called a *pipelining variable* (*data-dependent variable*) if its computation equation is a pipelining (data-dependent) equation.

Both the specification of separation and computation control variables are constructed for the index space Φ of the source UREs. Pick a control variable v.

- The domain of its input equations is Φ_v^{in}.
- The domain of its computation equation is Φ.

To obtain both data and control flow, we apply the same space-time mapping to both the source UREs and the UCREs. Therefore, we must ensure the validity of the space-time mapping for both the source UREs and the UCREs (Thm. 1). When we refer to a valid space-time mapping from now on, we mean a space-time mapping that is valid for both the source UREs and the UCREs. Note that the first step t_{fst} and the last step t_{lst} are now defined with respect to both data and control variables.

The concept of an extended index space applies also for the UCREs. In contrast to the source UREs, there are data-dependent variables in the UCREs. At the points added in the extension, the defining equation of a data-dependent variable is the equation at the points in the index space, i.e., it is not a pipelining equation.

Similarly, the concept of a space-time diagram applies also to the UCREs. At pipelining points, the distribution of control signals can be obtained in a similar manner as that of data, except that data-dependent variables must be treated in addition:

- The defining equation of a control variable at a pipelining point is the same as that at computation points.
- If the input of a control variable is undefined at some step, then the undefined value $\perp$ is injected.

Computation control serves to distinguish different types of computation points. The distribution of computation control signals at pipelining points is irrelevant. Separation control serves to distinguish pipelining points from computation points. The distribution of separation control signals at pipelining points is of relevance.

6 The Specification of Computation Control Variables

This section summarizes the *separating hyperplane method* for the construction of the computation control variables [19].

Let $\{\Phi_i \mid 0 \leqslant i < r\}$ be a partitioning of Φ such that two points are contained in Φ_i iff they are of the same type. The computation control variables are correct if, under a valid space-time mapping, the combinations of their values at two computation points differ if the two points are in different partitions.

Computation control variables are specified for the index space. They are pipelining variables. Thus, their specification amounts to defining their control dependence vectors and input equations. Each partition is a union of a finite set of polytopes. Without loss of generality, we assume that each partition is itself a polytope (if not, choose a finer partitioning). Since every partition is a polytope, the existence of at least one separating hyperplane between any two neighbouring partitions is guaranteed [17]. We want to find a set of separating hyperplanes that contains at least one separating hyperplane for any two partitions. We write $N_{i,j}$ for the separating hyperplane that separates Φ_i from Φ_j. This set is calculated from the domain predicates that appear in the source UREs. These domain predicates induce the partitioning $\{\Phi_i \mid 0 \leqslant i < r\}$.

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 $\pi_{i,j} I \leqslant \delta_{i,j}$ and $\pi_{i,j} I > \delta_{i,j}$ at every cell, the result of evaluating each inequality 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 $\{I \mid \pi_{i,j} I > \delta_{i,j}\}$.

Definition 6 (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 $\mathcal{S}(C_{i,j}) = \{c^{\leqslant}_{i,j}, c^{>}_{i,j}\}$, the input equations of $C_{i,j}$ are defined as follows:

$$I \in \Phi^{\text{in},\leqslant}_{C_{i,j}} \;\rightarrow\; C_{i,j}(I) = c^{\leqslant}_{i,j} \quad \text{and} \quad I \in \Phi^{\text{in},>}_{C_{i,j}} \;\rightarrow\; C_{i,j}(I) = c^{>}_{i,j} \qquad \square$$

Theorem 2 [19] *The computation control variables as specified previously are correct.*

7 The Specification of Separation Control Variables...

A separation control variable, v, can always be interpreted to label a data variable. If it is not already, the following augmentation of the source UREs makes $\ell.v$ a data variable:

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

For convenience, we assume in subsequent sections that the extended index space Ψ and the space-time diagram Υ are defined for the thus augmented UREs. When we refer to Ψ_v, Υ_v^{s}, Υ_v^{d}, $\Upsilon_v^{\perp}$, $\mathsf{in}(\Phi, \vartheta_v)$ and $\mathsf{out}(\Phi, \vartheta_v)$, for control variable v, we mean $\Psi_{\ell.v}$, $\Upsilon_{\ell.v}^{\text{s}}$, $\Upsilon_{\ell.v}^{\text{d}}$, $\Upsilon_{\ell.v}^{\perp}$, $\mathsf{in}(\Phi, \vartheta_{\ell.v})$ and $\mathsf{out}(\Phi, \vartheta_{\ell.v})$, for the corresponding data variable $\ell.v$, respectively. We present the specification of separation control variables with respect to the space-time diagram.

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

$$\varphi(\overline{I}) \;=\; (v(\overline{I} - \overline{\vartheta}_v), \cdots).$$

Definition 7 The separation control variables are *correct* if, under a valid space-time mapping, control signals at a computation point and a pipelining point differ, i.e., if

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

To avoid unduly complex notation, we present the specification of control signals in the form of programs: a program for the host that corresponds to the input equations and a program for the array cells that corresponds to the computation equations.

In the next section, we cover the case of three-dimensional source UREs. The section after that discusses source UREs of higher dimension.

8 ...for Three-Dimensional Source UREs

8.1 Initialization, Termination and Evolution Control Variables

For any data variable v, $\{\Upsilon_v^{\text{s}}, \Upsilon_v^{\text{d}}, \Upsilon_v^{\perp}, \overline{\Phi}\}$ is a partitioning of the space-time diagram. 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. $\overline{\mathsf{in}(\Phi, \vartheta_v)}$ is the boundary between Υ_v^{s} and $\overline{\Phi}$, and $\overline{\mathsf{out}(\Phi, \vartheta_v)}$ is the boundary between Υ_v^{d} and $\overline{\Phi}$. $\Upsilon_v^{\perp}$ is the boundary between $\Upsilon_v^{\perp}$ and the remaining three partitions. 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 values of control variable v. A control variable is denoted by an upper-case letter with or without a subscript. Its values, excluding $\perp$, are denoted by the corresponding lower-case letter with subscripts. There are five separation control variables:

- The *initialization control variables*, F_0 and F_1. They are the pipelining variables that together identify the first computation points of E, i.e., the boundary $\overline{\mathsf{in}(\Phi, \vartheta_E)}$ between Υ_E^{s} and $\overline{\Phi}$. $\mathcal{S}(F_0) = \{f_0, \perp\}$ and $\mathcal{S}(F_1) = \{f_1, \perp\}$.

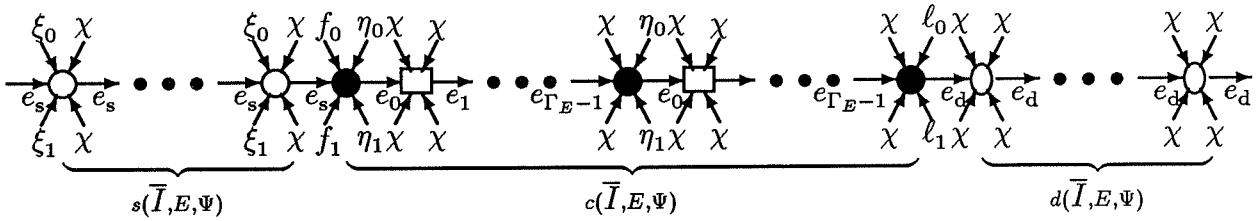


Figure 3: Illustration of the construction of the separation control variables. Fat dots are computation points, circles are soaking points, boxes are relaying points and ovals are draining points. The relevant control signals at a point are shown. Arrows pointing southeast are for ϑ_{F_0}, arrows pointing northeast for ϑ_{F_1}, arrows pointing southwest for ϑ_{L_0}, arrows pointing northwest for ϑ_{L_1} and arrows pointing east for ϑ_E. χ stands for "don't-care". ξ_0 and ξ_1 satisfy $(\xi_0 \neq f_0 \vee \xi_1 \neq f_1)$. η_0 and η_1 satisfy $(\eta_0 \neq \ell_0 \vee \eta_1 \neq \ell_1)$.

- The *termination control variables*, L_0 and L_1. They are the pipelining variables that together identify the last computation points of E, i.e., the boundary $\overline{\mathsf{out}(\Phi, \vartheta_E)}$ between Υ_E^{d} and $\overline{\Phi}$. $\mathcal{S}(L_0) = \{\ell_0, \perp\}$ and $\mathcal{S}(L_1) = \{\ell_1, \perp\}$.
- The *evolution control variable*, E. It is a data-dependent variable that depends on variables F_0, F_1, L_0 and L_1. E by itself identifies the points in $\Upsilon_E^{\perp}$, i.e., the boundary between $\Upsilon_E^{\perp}$ and the remaining three partitions. $\mathcal{S}(E) = \{e_{\mathrm{s}}, e_0, \cdots, e_{\Gamma_E-1}, e_{\mathrm{d}}, \perp\}$.

Now, we can explicitly define the control dependence matrix:

$$D_{\mathrm{c}} = [\vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{L_0}, \vartheta_{L_1}, \vartheta_E]$$

We can also explicitly define the set of control signals $\varphi(\overline{I})$ at space-time point $\overline{I}$:

$$\varphi(\overline{I}) = (F_0(\overline{I} - \overline{\vartheta}_{F_0}), F_1(\overline{I} - \overline{\vartheta}_{F_1}), L_0(\overline{I} - \overline{\vartheta}_{L_0}), L_1(\overline{I} - \overline{\vartheta}_{L_1}), E(\overline{I} - \overline{\vartheta}_E))$$

8.2 The Construction of Input and Computation Equations

In this section, we present a construction of the UCREs for the separation control variables, assuming an arbitrary choice of control dependence vectors. Sect. 8.3 describes how to choose the control dependence vectors appropriately.

Let us first consider the specification of evolution control variable E. We consider two types of $\overline{\vartheta}_E$-paths in the space-time diagram:

1. Path $p(\overline{I}, E, \Upsilon)$ contains computation points (Fig. 3). We input e_{s} at the input point of $s(\overline{I}, v, \Upsilon)$ and adopt this value for all soaking points of E. We shall construct F_0 and F_1 such that the first computation point of $c(\overline{I}, E, \Upsilon)$ satisfies $F_0(\overline{I} - \overline{\vartheta}_{F_0}) = f_0 \wedge F_1(\overline{I} - \overline{\vartheta}_{F_1}) = f_1$ but the soaking points do not. There, e_{s} is converted to e_0, the value for all computation points (but the last). Then, if a point receives element e_k of E, it sends $e_{(k+1) \bmod \Gamma_E}$; k is the distance of a relaying point from its preceding computation point. Thus, the considered element of E periodically adopts the values $e_0, e_1, \cdots, e_{\Gamma_E-1}$. We shall construct L_0 and L_1 such that the last computation point of $c(\overline{I}, E, \Upsilon)$ satisfies

$L_0(\overline{I}-\overline{\vartheta}_{L_0})=\ell_0 \wedge L_1(\overline{I}-\overline{\vartheta}_{L_1})=\ell_1$ but all other computation points do not. There, e_{Γ_E-1} is converted to e_d, the value for all draining points. If path $p(\overline{I}, E, \Upsilon)$ contains only one computation point, which is, therefore, both a first and last computation point of E, e_s changes to e_d at that point.

2. Path $p(\overline{I}, E, \Upsilon)$ contains no computation points; it contains only undefined points of E. We input $\perp$ at the input point of $p(\overline{I}, E, \Upsilon)$ and adopt this value for all points of the path.

By the construction of E, we inject e_s at steps $\mathsf{input}(E(I))$, where $I \in \mathsf{in}(\Phi, \vartheta_E)$, and $\perp$ at the remaining steps. The specification of E must consist of three guarded commands. E can be viewed as a piecewise pipelining variable.

Next, we consider the specification of initialization control variables F_0 and F_1 and termination control variables L_0 and L_1. To establish $F_0(\overline{I}-\overline{\vartheta}_{F_0})=f_0 \wedge F_1(\overline{I}-\overline{\vartheta}_{F_1})=f_1$ at the first computation points I of E but not at the soaking points of E, it is necessary (but not sufficient) to inject f_i $(i=0,1)$ at $\mathsf{input}(F_i(I))$, where $I \in \mathsf{in}(\Phi, \vartheta_E)$, and $\perp$ at the remaining steps. This is the only way of establishing $F_0(\overline{I}-\overline{\vartheta}_{F_0}) = f_0 \wedge F_1(\overline{I}-\overline{\vartheta}_{F_1}) = f_1$ at the first computation points of E since F_0 and F_1 are pipelining variables. It needs not be sufficient: $F_0(\overline{I}-\overline{\vartheta}_{F_0})=f_0 \wedge F_1(\overline{I}-\overline{\vartheta}_{F_1})=f_1$ may also hold at some soaking points of E because a valid a space-time mapping is not an injection from Ψ to Υ (Sect. 4.3).

Similarly, to establish $L_0(\overline{I}-\overline{\vartheta}_{L_0})=\ell_0 \wedge L_1(\overline{I}-\overline{\vartheta}_{L_1})=\ell_1$ but not at the remaining computation points, it is necessary (but not sufficient) to inject ℓ_i $(i=0,1)$ at $\mathsf{input}(L_i(I))$, where $I \in \mathsf{out}(\Phi, \vartheta_E)$, and $\perp$ at the remaining steps.

For sufficiency, we must choose the control dependence matrix D_c appropriately (Sect. 8.3). This completes the specification of F_0, F_1, L_0 and L_1. Because they are pipelining variables, no computation equations are required. Now, we have all necessary details for the specification of the host and the cell program.

8.2.1 The Host Program

The notation in this program is explained on the following page.

PROGRAM: $HostProg_{n=3}(D_c)$
for t **from** t_{fst} **to** t_{lst} **do**
  for k **from** 0 **to** 1 **do**
    if $t \in \{\mathsf{input}(F_k(I)) \mid I \in \mathsf{in}(\Phi, \vartheta_E)\}$ $\rightarrow$ **inject**$(f_k, PE(F_k))$
    [] **else** $\rightarrow$ **inject**$(\perp, PE(F_k))$
    fi
    if $t \in \{\mathsf{input}(L_k(I)) \mid I \in \mathsf{out}(\Phi, \vartheta_E)\}$ $\rightarrow$ **inject**$(\ell_k, PE(L_k))$
    [] **else** $\rightarrow$ **inject**$(\perp, PE(L_k))$
    fi
  if $t \in \{\mathsf{input}(E(I)) \mid I \in \mathsf{in}(\Phi, \vartheta_E)\}$ $\rightarrow$ **inject**$(e_s, PE(E))$
  [] **else** $\rightarrow$ **inject**$(\perp, PE(E))$
  fi

$PE(v)$ stands for the coordinate of the border cell to which elements of moving variable v are injected:

$$PE(v) = \textbf{if } \mathsf{flow}(v) > 0 \rightarrow p_{\min} \ [] \ \mathsf{flow}(v) < 0 \rightarrow p_{\max} \textbf{ fi}$$

$\textbf{inject}(s, PE(v))$ stands for the injection of control signal $s \in \mathcal{S}(v)$ to border cell $PE(v)$. The host executes the following program per step:

We subscribe the name of program *HostProg* with $n=3$ to indicate that the program is for three-dimensional UREs. This program depends on the control dependence matrix D_c.

8.2.2 The Cell Program

We name the input (output) channel for control variable v at the cell v^{in} (v^{out}). $\mathcal{B}(F)$ and $\mathcal{B}(L)$ are defined as follows:

$$\begin{aligned} \mathcal{B}(F) &= (F_0^{\text{in}} = f_0) \wedge (F_1^{\text{in}} = f_1) \\ \mathcal{B}(L) &= (L_0^{\text{in}} = \ell_0) \wedge (L_1^{\text{in}} = \ell_1) \end{aligned}$$

We use an auxiliary control variable Z, called the *action control variable*, with no associated control dependence vector: $\mathcal{S}(Z) = \{z_c, z_p\}$. If $\overline{I}$ is a computation point, then $F(\overline{I}) = z_c$; if $\overline{I}$ is a pipelining point, then $Z(\overline{I}) = z_p$. The cell executes the following program per step:

PROGRAM: $CellProg_{n=3}(D_c)$

$$\begin{aligned}
&(\forall\, i : 0 \leqslant i \leqslant 1 : F_i^{\text{out}} = F_i^{\text{in}}) \\
&(\forall\, i : 0 \leqslant i \leqslant 1 : L_i^{\text{out}} = L_i^{\text{in}}) \\
&E^{\text{out}} = \textbf{if } E^{\text{in}} = e_s \wedge \mathcal{B}(F) \wedge \neg\mathcal{B}(L) \rightarrow e_0 \\
&\quad [] \ ((E^{\text{in}} = e_s \wedge \mathcal{B}(F)) \vee E^{\text{in}} = e_{\Gamma_E - 1}) \wedge \mathcal{B}(L) \rightarrow e_d \\
&\quad ([] \ k : 0 \leqslant k < \Gamma_E : E^{\text{in}} = e_k \rightarrow e_{(k+1) \bmod \Gamma_E}) \\
&\quad [] \textbf{ else} \rightarrow E^{\text{in}} \\
&\quad \textbf{fi} \\
&Z^{\text{out}} = \textbf{if } (E^{\text{in}} = e_s \wedge \mathcal{B}(F)) \vee E^{\text{in}} = e_{\Gamma_E - 1} \rightarrow z_c \\
&\quad [] \textbf{ else} \rightarrow z_p \\
&\quad \textbf{fi}
\end{aligned}$$

The first guard of E^{out} selects first computation points of E but excludes points that are both a first and last computation point of E. $E^{\text{in}} = e_s \wedge \mathcal{B}(F)$ holds at a point if the point is a first computation point of E. If it is also a last computation point of E, $\mathcal{B}(L)$ holds also. The second guard of E^{out} establishes whether a point is a last computation point of E. $E^{\text{in}} = e_s \wedge (\mathcal{B}(F)$ holds at first computation points of E and $E^{\text{in}} = e_{\Gamma_E - 1}$ holds at computation points that are not first computation points of E. Thus, if $\mathsf{in}(\Phi, \vartheta_E)$ and $\mathsf{out}(\Phi, \vartheta_E)$ are disjoint, the first guard of E can be simplified to $E^{\text{in}} = e_s \wedge \mathcal{B}(F)$ and the second to $E^{\text{in}} = e_{\Gamma_E - 1} \wedge \mathcal{B}(L)$. The third guarded command of E^{out} is a quantification over the choice operator $[]$. It handles points that are relaying or computation points but neither first nor last computation points of E. The quantification represents Γ_E separate guarded commands. The **else** guard of E^{out} captures the undefined points of E.

Like program $HostProg_{n=\mathcal{I}}(D_c)$, program $CellProg_{n=\mathcal{I}}(D_c)$ depends on the control dependence matrix D_c (in fact, it only depends on ϑ_E).

The host and cell program can be equivalently represented by the control dependence graph defined for the extended index space Ψ:

- The nodes are the points in the extended index space; the equations defined at each node are represented by program $CellProg_{n=\mathcal{I}}(D_c)$.
- The dependence vectors are generated by duplicating control dependence vectors ϑ_{F_0}, ϑ_{F_1}, ϑ_{L_0}, ϑ_{L_1} and ϑ_E throughout the extended index space.
- Let $\mathsf{value}(p(I,v,\Psi))$ be the control value initialized at the source point in $\mathsf{in}(\Psi,\vartheta_v)$ of the ϑ_v-path $p(I,v,\Psi)$:

$$\begin{array}{lrcl}
(\forall\, k : 0 \leqslant k \leqslant 1 : & \mathsf{value}(p(I,F_k,\Psi)) & = & \textbf{if } I \in \mathsf{in}(\Phi,\vartheta_E) \rightarrow f_k \;\; [] \textbf{ else} \rightarrow \bot \textbf{ fi}) \\
(\forall\, k : 0 \leqslant k \leqslant 1 : & \mathsf{value}(p(I,L_k,\Psi)) & = & \textbf{if } I \in \mathsf{out}(\Phi,\vartheta_E) \rightarrow \ell_k \;\; [] \textbf{ else} \rightarrow \bot \textbf{ fi}) \\
& \mathsf{value}(p(I,E,\Psi)) & = & \textbf{if } I \in \mathsf{in}(\Phi,\vartheta_E) \rightarrow e_s \;\; [] \textbf{ else} \rightarrow \bot \textbf{ fi}
\end{array}$$

The definition of value follows directly from program $HostProg_{n=\mathcal{I}}(D_c)$. Because F_0, F_1, L_0 and L_1 are pipelining variables, we know that, for every control variable $v \in \{F_0, F_1, L_0, L_1\}$ and every point J in ϑ_v-path $p(I,v,\Psi)$, $v(J) = \mathsf{value}(p(I,v,\Psi))$.

Remark.
A $\overline{\vartheta}_v$-path of control variable v may start at an internal cell in the following two cases.

1. If v is stationary, all $\overline{\vartheta}_v$-paths start at internal cells. The control signals of the variable must be loaded before the computation starts, i.e., before step t_{fst}.
2. If v is moving, some $\overline{\vartheta}_v$-paths may start at internal cells. In this case, the input value must be $\bot$. In contrast to data variables (Sect. 4.2), $\bot$ is a meaningful control signal and must be input. The input of $\bot$ before step t_{fst} can be implemented by system reset or by pipelining. Our separation control scheme requires that the initialization and termination control variables be initialized at all cells with $\bot$ before step t_{fst}. □

The previous construction of the evolution control variable requires that separation control variables satisfy the following specification (Fig. 3):

$$\begin{array}{lrcl}
\text{Spec. 1.} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (f_0, f_1, \chi, \chi, e_s) & \Longleftrightarrow & \overline{I} \in \overline{\mathsf{in}(\Phi,\vartheta_E)}) \\
\text{Spec. 2} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (f_0, f_1, \ell_0, \ell_1, e_s) & \Longleftrightarrow & \overline{I} \in \overline{\mathsf{out}(\Phi,\vartheta_E)} \cap \overline{\mathsf{in}(\Phi,\vartheta_E)}) \;\wedge \\
& (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (\chi, \chi, \ell_0, \ell_1, e_c) & \Longleftrightarrow & \overline{I} \in \overline{\mathsf{out}(\Phi,\vartheta_E)} \setminus \overline{\mathsf{in}(\Phi,\vartheta_E)}) \\
\text{Spec. 3.} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (\xi_0, \xi_1, \chi, \chi, e_s) & \Longleftrightarrow & \overline{I} \in \Upsilon^s_E) \\
\text{Spec. 4.} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (\chi, \chi, \eta_0, \eta_1, e_c) & \Longleftrightarrow & \overline{I} \in \overline{\Phi} \setminus (\overline{\mathsf{in}(\Phi,\vartheta_E)} \cup \overline{\mathsf{out}(\Phi,\vartheta_E)})) \\
\text{Spec. 5.} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (\chi, \chi, \chi, \chi, e_d) & \Longleftrightarrow & \overline{I} \in \Upsilon^d_E) \\
\text{Spec. 6.} & (\forall\, \overline{I} : \overline{I} \in \Upsilon : \varphi(\overline{I}) = (\chi, \chi, \chi, \chi, \bot) & \Longleftrightarrow & \overline{I} \in \Upsilon^{\bot}_E)
\end{array}$$

where χ stands for "don't care", ξ_0 and ξ_1 satisfy $(\xi_0 \neq f_0 \vee \xi_1 \neq f_1)$, and η_0 and η_1 satisfy $(\eta_0 \neq \ell_0 \vee \eta_1 \neq \ell_1)$. The satisfaction of this specification trivially ensures the correctness of the separation control variables (Def. 7). Note that Spec. 2 is a conjunction: one conjunct captures the points in both $\overline{\mathsf{out}(\Phi, \vartheta_E)}$ and $\overline{\mathsf{in}(\Phi, \vartheta_E)}$ and the other the points only in $\overline{\mathsf{out}(\Phi, \vartheta_E)}$ (Fig. 3).

The following theorem describes the essential rôle of the evolution control variable E in the correctness of the separation control variables.

Theorem 3 *Specs. 1–2 are satisfied iff Specs. 3–6 are satisfied.*

Proof. $HostProg_{n=3}(D_c)$, $CellProg_{n=3}(D_c)$ and Specs. 1–6. □

Thm. 3 states that the correctness of the separation control variables is fully established by the satisfaction of Specs. 1–2. Next, we state a necessary and sufficient condition for the satisfaction of Spec. 1.

Lemma 4 *Spec. 1 is satisfied iff, for every $\overline{\vartheta}_E$-path containing computation points, the first computation point of E satisfies $\mathcal{B}(F)$ but the soaking points do not.*

Proof. $HostProg_{n=3}(D_c)$, $CellProg_{n=3}(D_c)$ and Spec. 1. □

A similar necessary and sufficient condition exists for the satisfaction of Spec. 2.

Lemma 5 *Assume Spec. 1 is satisfied. Spec. 2 is satisfied iff, for every $\overline{\vartheta}_E$-path containing computation points, the last computation point of E satisfies $\mathcal{B}(L)$ but the remaining computation points do not.*

Proof. $HostProg_{n=3}(D_c)$, $CellProg_{n=3}(D_c)$ and Specs. 1–2. □

In Sect. 8.3, we present sufficient conditions that ensure the satisfaction of Specs. 1–2 and that provide a construction of control dependence matrix D_c. These conditions impose constraints on the shape of the index space.

8.3 The Construction of Control Dependence Vectors

The construction of initialization control variables F_0 and F_1 aims at identifying the points in $\overline{\mathsf{in}(\Phi, \vartheta_E)}$, and similarly, the construction of termination control variables L_0 and L_1 aims at identifying the points in $\overline{\mathsf{out}(\Phi, \vartheta_E)}$.

We write $\mathsf{sig.in}$ for the set of the intersection points between a ϑ_{F_0}-path $p(I, F_0, \Psi)$ such that $\mathsf{value}(p(I, F_0, \Psi)) = f_0$ and a ϑ_{F_1}-path $p(J, F_1, \Psi)$ such that $\mathsf{value}(p(J, F_1, \Psi)) = f_1$:

$$\mathsf{sig.in} = \{K \mid K \in p(I, F_0, \Psi) \cap p(J, F_1, \Psi),\ \mathsf{value}(p(I, F_0, \Psi)) = f_0,\ \mathsf{value}(p(J, F_1, \Psi)) = f_1\}.$$

Similarly, we write $\mathsf{sig.out}$ for the set of the intersection points between a ϑ_{L_0}-path $p(I, L_0, \Psi)$ such that $\mathsf{value}(p(I, L_0, \Psi)) = \ell_0$ and a ϑ_{L_1}-path $p(J, L_1, \Psi)$ such that $\mathsf{value}(p(J, L_1, \Psi)) = \ell_1$:

$$\mathsf{sig.out} = \{K \mid K \in p(I, L_0, \Psi) \cap p(J, L_1, \Psi),\ \mathsf{value}(p(I, L_0, \Psi)) = \ell_0,\ \mathsf{value}(p(J, L_1, \Psi)) = \ell_1\}.$$

Lemma 6 $\mathsf{in}(\Phi, \vartheta_E) \subseteq \mathsf{sig.in}$ *and* $\mathsf{out}(\Phi, \vartheta_E) \subseteq \mathsf{sig.out}$.

Proof. Definitions of sig.in, sig.out, $\mathsf{in}(\Phi, \vartheta_E)$ and $\mathsf{out}(\Phi, \vartheta_E)$. □

Let us consider the initialization control variables first. $\mathcal{B}(F)$ holds for the points in $\overline{\mathsf{sig.in}}$, but it may also hold for a space-time point that is not in $\overline{\mathsf{sig.in}}$. This may happen when two points – one in a ϑ_{F_0}-path $p(I, F_0, \Psi)$ with $\mathsf{value}(p(I, F_0, \Psi)) = f_0$ and another in a ϑ_{F_1}-path $p(J, F_1, \Psi)$ with $\mathsf{value}(p(J, F_1, \Psi)) = f_1$ that does not intersect the ϑ_{F_0}-path – share the same image (Sect. 4.3). A similar argument applies for the termination control variables. The basic idea in constructing the control dependence matrix is to eliminate this possibility. Since the definitions of sig.in and sig.out do not depend on the space-time mapping, we can then establish the correctness of the separation control variables at the source level.

Before presenting a construction of the control dependence matrix, we introduce a lemma, on which the construction depends.

Lemma 7 *Assume* $0 \neq \mathsf{flow}(v) \neq \mathsf{flow}(w) \neq 0$. *Assume* Π *satisfies Thm. 1.*

$$(\forall\, I, J, K : I, J, K \in \mathbb{Z}^n \wedge I \xrightarrow{v} K \wedge J \xrightarrow{w} K : \overline{I} = \overline{J} \implies I = J = K).$$

Proof. $\overline{I} = \overline{J} \iff \mathsf{step}(I) = \mathsf{step}(J) \wedge \mathsf{place}(I) = \mathsf{place}(J) \iff \lambda I = \lambda J \wedge \sigma I = \sigma J$. $I \xrightarrow{v} K \iff (\exists\, m : m \in \mathbb{Z}_0^+ : K = m\vartheta_v + I)$ and $J \xrightarrow{w} K \iff (\exists\, n : n \in \mathbb{Z}_0^+ : K = n\vartheta_w + J)$. A simple algebraic calculation establishes $m\lambda\vartheta_v = n\lambda\vartheta_w \;\wedge\; m\sigma\vartheta_v = n\sigma\vartheta_w$.

We consider all four possible geometric relationships between I and J relative to K (the proof proceeds to show that only $I = J = K$ can hold):

Case 1: $I = K \wedge J \neq K$. This implies $m = 0$ and consequently $\lambda\vartheta_w = 0$, contradicting Thm. 1.

Case 2: $I \neq K \wedge J = K$. This implies $n = 0$ and consequently $\lambda\vartheta_v = 0$, contradicting Thm. 1.

Case 3: $I \neq K \wedge J \neq K$. By hypothesis $0 \neq \mathsf{flow}(v) \neq \mathsf{flow}(w) \neq 0$, we infer $m\sigma\vartheta_v = n\sigma\vartheta_w \neq 0$. Dividing $m\sigma\vartheta_v$ by $m\lambda\vartheta_v$ and $n\sigma\vartheta_w$ by $n\lambda\vartheta_w$ (note that $m\lambda\vartheta_v = n\lambda\vartheta_w$) yields $\sigma\vartheta_v/\lambda\vartheta_v = \sigma\vartheta_w/\lambda\vartheta_w$, i.e., $\mathsf{flow}(v) = \mathsf{flow}(w)$, contradicting the hypothesis.

Case 4: $I = K \wedge J = K$. Trivially true. □

This proof does not require the containment of I, J and K in the index space; it only relies on the geometric relationship of the three points. The lemma states that two ϑ-paths of variables moving with different velocities only share an image at the intersection point. We choose ϑ_{F_0} and ϑ_{F_1} in such a way that every ϑ_{F_0}-path with $\mathsf{value}(p(I, F_0, \Psi)) = f_0$ intersects all ϑ_{F_1}-paths with $\mathsf{value}(p(J, F_1, \Psi)) = f_1$. This ensures $\mathcal{B}(F)$ at a space-time point iff the point is in $\overline{\mathsf{sig.in}}$. But sig.in may be a proper superset of $\mathsf{in}(\Phi, \vartheta_E)$ (Lemma 6). To ensure the satisfaction of Spec. 1, we require $\mathsf{sig.in} = \mathsf{in}(\Phi, \vartheta_E)$. This restricts the shape of the index space.

Similarly, we choose ϑ_{L_0} and ϑ_{L_1} such that every ϑ_{L_0}-path with $\mathsf{value}(p(I, L_0, \Psi)) = \ell_0$ intersects all ϑ_{L_1}-paths with $\mathsf{value}(p(J, L_1, \Psi)) = \ell_1$ and require $\mathsf{sig.out} = \mathsf{out}(\Phi, \vartheta_E)$.

Theorem 4 *(Initialization Theorem)* $0 \neq \mathsf{flow}(F_0) \neq \mathsf{flow}(F_1) \neq 0$. *Spec. 1 is satisfied if* $\mathsf{in}(\Phi, \vartheta_E)$ *is a parallelogram; two edges are parallel to vector* ϑ_{F_0} *and the other two to* ϑ_{F_1}.

Proof. By the hypotheses, every ϑ_{F_0}-path $p(I, F_0, \Psi)$ such that $\mathsf{value}(p(I, F_0, \Psi)) = f_0$ must intersect all ϑ_{F_1}-paths $p(J, F_1, \Psi)$ such that $\mathsf{value}(p(J, F_1, \Psi)) = f_1$ and the set of intersection points $\mathsf{sig.in}$ satisfies $\mathsf{sig.in} = \mathsf{in}(\Phi, \vartheta_E)$. By applying Lemma 7 and Thm. 1, we conclude that $\mathcal{B}(F)$ holds at a space-time point iff the point is in $\overline{\mathsf{sig.in}}$, i.e., in $\overline{\mathsf{in}(\Phi, \vartheta_E)}$. Thus, for every ϑ_E-path containing computation points, $\mathcal{B}(F)$ holds for the first computation point of E but not for the soaking points of the path. An application of Lemma 4 completes the proof. □

By symmetry, there is an analogue of Thm. 4 for termination control variables.

Theorem 5 *(Termination Theorem) Assume that Spec. 1 is satisfied and* $0 \neq \mathsf{flow}(L_0) \neq \mathsf{flow}(L_1) \neq 0$. *Spec. 2 is satisfied if the* $\mathsf{out}(\Phi, \vartheta_E)$ *is a parallelogram; two edges are parallel to vector* ϑ_{L_0} *and the other two to* ϑ_{L_1}.

Proof. Similar to the proof of Thm. 4. □

Let us introduce some concepts that are used in this and subsequent sections.

- $\pi x \leqslant \pi_0$ $(\pi, \pi_0 \in \mathbb{Q}^n)$ is a *valid inequality* for a polytope $S \subset \mathbb{Q}^n$, if it is satisfied at all points of S. If $\mathcal{F} = \{x \mid x \in S, \pi x = \pi_0\}$ and $\pi x \leqslant \pi_0$ is a valid inequality, $\mathcal{F}$ is called a *face* of S. A face $\mathcal{F}$ of S is called a k-face of S if $\dim(\mathcal{F}) = k$. A k-face is a *facet* of S if $k = \dim(S) - 1$ [11, Sect. I. 4].

- Let $\{\vartheta_i \mid 0 \leqslant i < n\}$ be n linearly independent n-vectors. Let $\{\delta_j \mid 0 \leqslant j < n\}$ be n linearly independent n-vectors such that $(\forall\ i : 0 \leqslant i < n \wedge i \neq j : \delta_j \vartheta_i = 0)$. That is, δ_j is the normal to the hyperplane formed by $(n-1)$ linearly independent vectors $\{\vartheta_i \mid 0 \leqslant i < n \wedge i \neq j\}$. The *smallest enclosing parallelepiped* $\mathcal{P}(S, \vartheta_0, \cdots, \vartheta_{n-1})$ of a convex polytope $S \subset \mathbb{Q}^n$ that is generated by n linearly independent n-vectors $\{\vartheta_i \mid 0 \leqslant i < n\}$ is defined as follows:

$$\mathcal{P}(S, \vartheta_0, \cdots, \vartheta_{n-1}) = \{I \mid H^- \leqslant [\delta_0, \cdots, \delta_{n-1}] I \leqslant H^+\}.$$

  where $H^- = (H_0^-, \cdots, H_{n-1}^-)$ and $H^+ = (H_0^+, \cdots, H_{n-1}^+)$ are n-vectors:

$$H_j^- = \min\{\delta_j I \mid I \in S\} \qquad H_j^+ = \max\{\delta_j I \mid I \in S\}.$$

  $\mathcal{P}(S, \vartheta_0, \cdots, \vartheta_{n-1})$ has $2n$ facets. $\mathsf{in}(\mathcal{P}(S, \vartheta_0, \cdots, \vartheta_{n-1}), \vartheta_i)$ and $\mathsf{out}(\mathcal{P}(S, \vartheta_0, \cdots, \vartheta_{n-1}), \vartheta_i)$ are the two facets associated with ϑ_i.

The condition of Thm. 4 can be expressed as: $\mathcal{P}(\mathsf{in}(\Phi, \vartheta_E), \vartheta_{F_0}, \vartheta_{F_1}) = \mathsf{in}(\Phi, \vartheta_E)$. The condition of Thm. 5 can be expressed as: $\mathcal{P}(\mathsf{out}(\Phi, \vartheta_E), \vartheta_{L_0}, \vartheta_{L_1}) = \mathsf{out}(\Phi, \vartheta_E)$.

Thm. 4 (Thm. 5), or more precisely, Lemma 7, on which the proof of Thm. 4 (Thm. 5) depends, justifies the need for two initialization (termination) control variables.

The satisfaction of Spec. 2 relies on the satisfaction of Spec. 1. This is the reason for the prerequisite "Spec. 1 is satisfied" in Lemma 5 and Thm. 5. We can exploit this asymmetry to advantage in specifying the termination control variables. Once the first computation points of E are correctly identified, a space-time point is a last computation point of E only if it is a computation point. It turns out that termination control can be specified like computation control by the separating hyperplane method (Def. 6).

The basic idea in constructing termination control variables is to make sure that the combination of control signals of the termination control variables at the last computation points of E and the remaining computation points are disjoint. The number of termination control variables needed may vary for different source UREs.

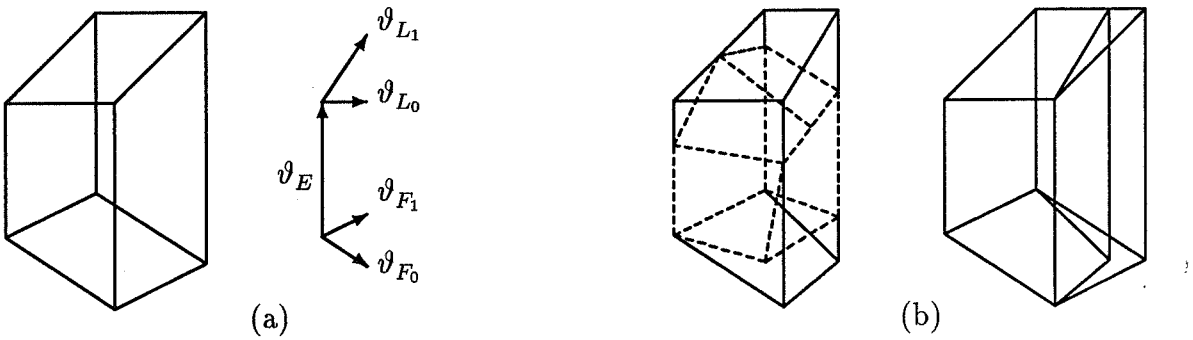


Figure 4: (a) The shape of the three-dimensional index space Φ, as required by programs $HostProg_{n=3}(D_c)$ and $CellProg_{n=3}(D_c)$. The five control dependence vectors ϑ_E, ϑ_{F_0}, ϑ_{F_1}, ϑ_{L_0} and ϑ_{L_1} are shown. The top facet $\mathsf{in}(\Phi, \vartheta_E)$ and the bottom facet $\mathsf{out}(\Phi, \vartheta_E)$ must be parallelograms. (b) Illustration of the adaption index space Φ. The dashed polytope depicts Φ. Assume that the control dependence vectors of (a) are chosen. The polytope enclosing Φ on the left depicts the domain Φ_E. It is duplicated on the right. The polytope enclosing Φ_E on the right depicts the adapted index space Φ_A.

8.4 Adaptation of the Source

By Thms. 4 and 5, the correctness of $HostProg_{n=3}(D_c)$ and $CellProg_{n=3}(D_c)$ relies on the fact that the index space Φ is of the following shape (Fig. 4(a)):

- $\mathsf{in}(\Phi, \vartheta_E)$ is a parallelogram; two edges are parallel to ϑ_{F_0} and the other two to ϑ_{F_1}.
- $\mathsf{out}(\Phi, \vartheta_E)$ is a parallelogram; two edges are parallel to ϑ_{L_0} and the other two to ϑ_{L_1}.

This implies that $\vartheta_{L_0}(\vartheta_E \times \vartheta_{F_0}) = 0$ ($\times$ denotes the vector product), i.e., that ϑ_{L_0} is perpendicular to the normal to the hyperplane formed by vectors ϑ_E and ϑ_{F_0}. Similarly, $\vartheta_{L_1}(\vartheta_E \times \vartheta_{F_1}) = 0$.

Once control dependence vectors ϑ_E, ϑ_{F_0}, ϑ_{F_1}, ϑ_{L_0} and ϑ_{L_1} satisfying $\vartheta_{L_0}(\vartheta_E \times \vartheta_{F_0}) = 0$ and $\vartheta_{L_1}(\vartheta_E \times \vartheta_{F_1}) = 0$ are given, the index space can be adapted to be of the required form with the concept of a supporting hyperplane (Fig. 4(b)).

A *supporting hyperplane* $\{I \mid \pi I = \delta\}$ to a polytope $C \subset \mathbb{Q}^n$ is a hyperplane that contains at least one point in C such that either all points I in C satisfy $\pi I \leqslant \delta$ or all satisfy $\pi I \geqslant \delta$. The *adapted index space*, denoted Φ_A, is the closed polytope (i.e., including the boundaries) bounded by the following six supporting hyperplanes to the index space Φ:

- The two supporting hyperplanes $\{I \mid (\vartheta_E \times \vartheta_{F_0})I = H_0^-\}$ and $\{I \mid (\vartheta_E \times \vartheta_{F_0})I = H_0^+\}$ where $H_0^- = \min\{(\vartheta_E \times \vartheta_{F_0})I \mid I \in \Phi\}$ and $H_0^+ = \max\{(\vartheta_E \times \vartheta_{F_0})I \mid I \in \Phi\}$.
- The two supporting hyperplanes $\{I \mid (\vartheta_E \times \vartheta_{F_1})I = H_1^-\}$ and $\{I \mid (\vartheta_E \times \vartheta_{F_1})I = H_1^+\}$ where $H_1^- = \min\{(\vartheta_E \times \vartheta_{F_1})I \mid I \in \Phi\}$ and $H_1^+ = \max\{(\vartheta_E \times \vartheta_{F_1})I \mid I \in \Phi\}$.
- The supporting hyperplane $\{I \mid (\vartheta_{F_0} \times \vartheta_{F_1})I = \beta_F\}$ that contains points of $\mathsf{in}(\Phi, \vartheta_E)$ and the supporting hyperplane $\{I \mid (\vartheta_{L_0} \times \vartheta_{L_1})I = \beta_L\}$ that contains points of $\mathsf{out}(\Phi, \vartheta_E)$, where β_F (β_L) is completely determined by Φ, ϑ_{F_0} (ϑ_{L_0}) and ϑ_{F_1} (ϑ_{L_1}).

If ϑ_{F_0} is parallel to one of ϑ_{L_0} and ϑ_{L_1}, and ϑ_{F_1} to the other, $\Phi_{\mathrm{E}}=\mathcal{P}(\Phi,\vartheta_{F_0},\vartheta_{F_1},\vartheta_E)$.

Φ_{E} is the union of all ϑ_E-paths in the adapted index space Φ_{A} that contain computation points of the original index space Φ:

$$\Phi_{\mathrm{E}}=(\bigcup\ I : I\in\Phi : p(I,E,\Phi_{\mathrm{A}}))$$

The UREs defined for Φ_{E} can be obtained straight-forwardly from the source UREs; the points that are not in the original index space are specified by pipelining equations. The control signals must be specified for the UREs defined at Φ_{E}; the two programs $HostProg_{n=3}(D_{\mathrm{c}})$, with Φ replaced by Φ_{E}, and $CellProg_{n=3}(D_{\mathrm{c}})$ constitute the specification of the separation control. The computation control is then specified for Φ_{E} by the separating hyperplane method (Def. 6). The space-time diagram is calculated for Φ_{E}, but the space-time mapping must be valid for the adapted index space Φ_{A}.

Consider the UREs defined for Φ_{E}. The correctness of the computation control follows from Thm. 2. The correctness of the separation control follows from the fact that $\mathcal{B}(F)$ ($\mathcal{B}(L)$) in program $CellProg_{n=3}(D_{\mathrm{c}})$ holds at a space-time point iff the point is in the image of $\mathsf{in}(\Phi_{\mathrm{A}},\vartheta_E)$ ($\mathsf{out}(\Phi_{\mathrm{A}},\vartheta_E)$) because the space-time mapping is valid for the adapted index space, and the fact that $\mathsf{in}(\Phi_{\mathrm{E}},\vartheta_E)\subseteq\mathsf{in}(\Phi_{\mathrm{A}},\vartheta_E)$ ($\mathsf{out}(\Phi_{\mathrm{E}},\vartheta_E)\subseteq\mathsf{out}(\Phi_{\mathrm{A}},\vartheta_E)$) (Thms. 4 and 5).

The termination control variables can also be specified by the separating hyperplane method (Def. 6); no restriction is imposed on the shape of $\mathsf{out}(\Phi,\vartheta_E)$. A similar adaptation of the index space applies with a domain Φ_{E}^- in place of Φ_{E} and an adapted index space Φ_{A}^- in place of Φ_{A}. Φ_{E}^- is a subset of Φ_{E}; it excludes the points generated in the extension of E along direction ϑ_E and $\Phi_{\mathrm{A}}^-=\Phi_{\mathrm{E}}^-\cup\mathsf{in}(\Phi_{\mathrm{A}},\vartheta_E)$.

Let us compare the two methods of the specification of separation control. They differ in the specification of termination control. For a fixed space-time mapping, the method that relies on the Termination Theorem (Thm. 5) generally incurs more hardware cost and/or higher latency than the method that relies on separating hyperplanes since $\Phi_{\mathrm{A}}^-\subseteq\Phi_{\mathrm{A}}$ and $\Phi_{\mathrm{E}}^-\subseteq\Phi_{\mathrm{E}}$. This seems to favour separating hyperplanes but, to apply separating hyperplanes, $\mathsf{out}(\Phi,\vartheta_{\mathrm{E}})$ must be partitioned into convex polytopes. The more partitions there are, the more separating hyperplanes (and, consequently, computation control variables) are needed. This induces more disjuncts in $\mathcal{B}(L)$ of the cell program (App. 2.2.2). The advantage of the method that relies on the Termination Theorem is the simplicity of the generated programs: $\mathcal{B}(L)$ is fixed. It is more suitable if the exact shape of $\mathsf{out}(\Phi,\vartheta_{\mathrm{E}})$ is complex but close to parallelogram.

The cell program that relies on the Termination Theorem can be optimized in the following cases. First, if $\mathsf{in}(\Phi,\vartheta_E)$ satisfies Thm. 4, i.e., if $\mathsf{sig.in}=\mathsf{in}(\Phi,\vartheta_E)$, $E^{\mathrm{out}}=e_{\mathrm{s}}$ can be disregarded. This is because $\mathcal{B}(F)$ implies $E^{\mathrm{out}}=e_{\mathrm{s}}$ at space-time points. Second, if $\mathsf{out}(\Phi,\vartheta_E)$ satisfies Thm. 5, i.e., if $\mathsf{sig.out}=\mathsf{out}(\Phi,\vartheta_E)$, there are two alternative optimizations. Either we can disregard $(E^{\mathrm{in}}=e_{\mathrm{s}}\wedge\mathcal{B}(F))\vee E^{\mathrm{in}}=e_{\Gamma_E-1}$ because it is implied by $\mathcal{B}(L)$ at space-time points, or we can eliminate one termination control variable because, in this case, we can separate a parallelogram from the rest of the index space by a single separating hyperplane.

8.5 Hardware Support

The bit width required to communicate values of v is $\lceil \log_2 |\mathcal{S}(v)| \rceil$. For example, E needs $\lceil \log_2(\Gamma_E+3) \rceil$ bits – Γ_E-1 for relaying points, one for computation points, one for soaking points, one for draining points and one for $\perp$. The bit width of E depends on the choice of both control dependence vector ϑ_E and the space-time mapping. We should always choose ϑ_E to minimize Γ_E. If possible, Γ_E should be a constant. This is the case in practical situations. A useful space-time mapping should make Γ_w constant for at least one data variable w; then, we choose $\ell.E = w$. The modulo operation in $CellProg_{n=3}(D_c)$ can be implemented by a circular shift (or rotate) operation if Γ_E is a power of 2. It is superfluous if $\Gamma_E=1$.

We reiterate that the correctness of the UCREs does not depend on the space-time mapping. When we address issues of efficiency, we need to consider the merits and demerits of different space-time mappings with respect to some cost function.

9 ...for n-Dimensional Source UREs $(n>3)$

Unfortunately, Thm. 4 does not generalize to n-dimensional UREs. In the general case, $\mathsf{in}(\Phi, \vartheta_E)$ must be of $n-1$ dimensions in order to be a facet. Because some ϑ_{F_0}-paths and ϑ_{F_1}-paths will not intersect, Lemma 7, on which the proof of Thm. 4 depends, does not apply and there is no analogue that applies for more than two control variables. The situation for Thm. 5 is similar.

The separation control scheme for n-dimensional UREs exploits the essential rôle of evolution control variable E. We have the following corollary of Thm. 3.

Corollary 1 *Consider the previous specification of evolution control variable E and an initialization and termination control scheme. The separation control variables are correct iff the initialization and termination control scheme can correctly identify the first and last computation points of E.*

Proof. Thm. 3. □

9.1 The Construction of Control Dependence Vectors

The basic idea in constructing the separation control variables for n-dimensional UREs is a hierarchical decomposition of the index space down to 3-faces such that

- the method for three-dimensional UREs applies to the 3-faces, and
- the evolution control scheme applies to the remaining faces.

We need $n-3$ more evolution control variables – one for each extra dimension. We name them $\{E_i \mid 0 \leqslant i < n-3\}$; E takes on the new name E_{n-3}.

The control dependence matrix is now redefined as follows:

$$D_c = [\vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{L_0}, \vartheta_{L_1}, \vartheta_{E_0}, \cdots, \vartheta_{E_{n-3}}].$$

Our method imposes the following restrictions on the control dependence vectors and on the shape of the index space:

- $\vartheta_{F_0} = \vartheta_{L_0} \wedge \vartheta_{F_1} = \vartheta_{L_1}$.
- $\vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{E_0}, \cdots, \vartheta_{E_{n-3}}$ are linearly independent.
- $\mathcal{P}(\Phi, \vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{E_0}, \cdots, \vartheta_{E_{n-3}}) = \Phi$.

Therefore, the index space must be a parallelepiped generated by the control dependence vectors. $\mathsf{in}(\Phi, \vartheta_i)$ and $\mathsf{out}(\Phi, \vartheta_i)$, for any $\vartheta_i \in D_c$, are facets of the index space.

Next, we describe a procedure for decomposing the index space Φ. Actually, we decompose the index space further, into 2-faces (not 3-faces). The 2-faces will be used in the construction of the host program. If we can identify the set $\mathsf{in}(\Phi, \vartheta_{E_0})$ of first computation points and the set $\mathsf{out}(\Phi, \vartheta_{E_0})$ of last computation points of E_0 in Φ, we are done, by Cor. 1. To identify the points in $\mathsf{in}(\Phi, \vartheta_{E_0})$, it suffices to identify the set $\mathsf{in}(\mathsf{in}(\Phi, \vartheta_{E_0}), \vartheta_{E_1})$ of first computation points and the set $\mathsf{out}(\mathsf{in}(\Phi, \vartheta_{E_0}), \vartheta_{E_1})$ of last computation points of E_1 in $\mathsf{in}(\Phi, \vartheta_{E_0})$. Similarly, to identify the points in $\mathsf{out}(\Phi, \vartheta_{E_0})$, it suffices to identify the set $\mathsf{in}(\mathsf{out}(\Phi, \vartheta_{E_0}), \vartheta_{E_1})$ of first computation points and the set $\mathsf{out}(\mathsf{out}(\Phi, \vartheta_{E_0}), \vartheta_{E_1})$ of last computation points E_1 in $\mathsf{out}(\Phi, \vartheta_{E_0})$. We repeat this procedure for subsequent E_i; we decompose each of the 2^i $(n-i)$-faces obtained in the call for E_{i-1} into the set of first computation points and the set of last computation points of E_i until we obtain 2-faces for E_{n-3}.

This decomposition produces a tree structure: there are 2^i $(n-i)$-faces at level i (the top level is 0). The following recursive equation assigns a unique identifier to each of these faces.

$$\begin{array}{lll} & \mathcal{F}(0,0) & = \Phi \\ (\forall\, i,j : 0 < i \leqslant n-2 \wedge 0 \leqslant j < 2^{i-1} : & \mathcal{F}(i,2j) & = \mathsf{in}(\mathcal{F}(i-1,j), \vartheta_{E_{i-1}})) \\ (\forall\, i,j : 0 < i \leqslant n-2 \wedge 0 \leqslant j < 2^{i-1} : & \mathcal{F}(i,2j+1) & = \mathsf{out}(\mathcal{F}(i-1,j), \vartheta_{E_{i-1}})) \end{array}$$

Node $\mathcal{F}(i-1,j)$ in the tree has two children: $\mathcal{F}(i,2j)$ and $\mathcal{F}(i,2j+1)$. The leaf nodes of the tree are 2-faces $\mathcal{F}(n-2,j)$; for even j, $\mathcal{F}(n-2,j)$ corresponds to $\mathsf{in}(\Phi, \vartheta_E)$, for odd j, it corresponds to $\mathsf{out}(\Phi, \vartheta_E)$ for a three-dimensional index space Φ.

Next, we present our construction of the separation control variables. For ease of presentation, we consider each face that is a non-leaf node in the decomposition tree separately, thus introducing additional control variables, which are eliminated again in Sect. 9.5.

9.2 The Construction of Input and Output Equations

The specification of $\mathcal{F}(n-3,j)$ follows from the specification of programs $\mathit{HostProg}_{n=3}(D_c)$ and $\mathit{CellProg}_{n=3}(D_c)$. To distinguish instances with different j, we add the arguments $n-3$ and j to the five control variables F_0, F_1, L_0, L_1 and E and all their control signals. In $\mathit{HostProg}_{n=3}(D_c)$, we replace $\mathsf{in}(\Phi, \vartheta_E)$ by $\mathcal{F}(n-2,2j)$ and $\mathsf{out}(\Phi, \vartheta_E)$ by $\mathcal{F}(n-2,2j+1)$.

We use one evolution control variable, denoted $E(i,j)$, for every face $\mathcal{F}(i,j)$: $\mathcal{S}(E(i,j)) = \{e(i,j)_s, e(i,j)_0, \cdots, e(i,j)_{\Gamma_{E(i,j)}-1}, e(i,j)_d, \perp\}$. $\mathcal{F}(i+1,2j)$ is the set of first computation points of $E(i,j)$ and $\mathcal{F}(i+1,2j+1)$ is the set of last computation points of $E(i,j)$ with respect to $\mathcal{F}(i,j)$. $\mathcal{F}(i+1,2j)$ will be identified by evolution control variable $E(i+1,2j)$, and $\mathcal{F}(i+1,2j+1)$ will be identified by evolution control variable $E(i+1,2j+1)$. $E(i+1,2j)$ plays the same rôle as, for three-dimensional UREs, F_0 and F_1 together. $E(i+1,2j+1)$ plays the same rôle as, for three-dimensional UREs, L_0 and L_1 together.

9.2.1 The Host Program

PROGRAM: $HostProg_{n>3}(D_c)$
for t **from** t_{fst} **to** t_{lst} **do**
  for i **from** 0 **to** $2^{n-3}-1$ **do**
    for k **from** 0 **to** 1 **do**
      if $t \in \{\mathsf{input}(F(n-3,i)_k(I)) \mid I \in \mathcal{F}(n-2,2i)\}$
        $\rightarrow$ **inject**$(f(n-3,i)_k, PE(F(n-3,i)_k))$
      [] **else** $\rightarrow$ **inject**$(\perp, PE(F(n-3,i)_k))$
      fi
      if $t \in \{\mathsf{input}(L(n-3,i)_k(I)) \mid I \in \mathcal{F}(n-2,2i+1)\}$
        $\rightarrow$ **inject**$(\ell(n-3,i)_k, PE(L(n-3,i)_k))$
      [] **else** $\rightarrow$ **inject**$(\perp, PE(L(n-3,i)_k))$
      fi
  for i **from** 0 **to** $n-3$ **do**
    for j **from** 0 **to** 2^i-1 **do**
      if $t \in \{\mathsf{input}(E(i,j)(I)) \mid I \in \mathcal{F}(i+1,2j)\}$ $\rightarrow$ **inject**$(e(i,j)_s, PE(E(i,j)))$
      [] **else** $\rightarrow$ **inject**$(\perp, PE(E(i,j)))$
      fi

9.2.2 The Cell Program

PROGRAM: $CellProg_{n>3}(D_c)$

$$
\begin{array}{l}
(\forall\, i,j : 0 \leqslant i < 2^{n-3} \wedge 0 \leqslant j \leqslant 1 : \\
\quad F(n-3,i)^{\text{out}}_j = F(n-3,i)^{\text{in}}_j \\
\quad L(n-3,i)^{\text{out}}_j = L(n-3,i)^{\text{in}}_j \\
\quad E(n-3,i)^{\text{out}} = \textbf{if}\ \ E(n-3,i)^{\text{in}} = e(n-3,i)_s \wedge \mathcal{B}(F(n-3,i)) \wedge \neg\mathcal{B}(L(n-3,i)) \rightarrow e(n-3,i)_0 \\
\qquad\qquad [] \ \ ((E(n-3,i)^{\text{in}} = e(n-3,i)_s \wedge \mathcal{B}(F(n-3,i))) \\
\qquad\qquad\quad \vee E(n-3,i)^{\text{in}} = e(n-3,i)_{\Gamma_{E(n-3,i)}-1}) \wedge \mathcal{B}(L(n-3,i)) \rightarrow e(n-3,i)_d \\
\qquad\qquad (\![] \ \ k : 0 \leqslant k < \Gamma_{E(n-3,i)} : E(n-3,i)^{\text{in}} = e(n-3,i)_k \\
\qquad\qquad\quad \rightarrow e(n-3,i)_{(k+1) \bmod \Gamma_{E(n-3,i)}}) \\
\qquad\qquad [] \ \ \textbf{else} \rightarrow E(n-3,i)^{\text{in}} \\
\qquad\qquad \textbf{fi} \qquad\qquad) \\
(\forall\, i,j : 0 \leqslant i < n-3 \wedge 0 \leqslant j < 2^i : \\
\quad E(i,j)^{\text{out}} = \textbf{if}\ \ E(i+1,2j)^{\text{in}} = e(i+1,2j)_0 \rightarrow e(i,j)_0 \\
\qquad\qquad [] \ \ E(i+1,2j+1)^{\text{in}} = e(i+1,2j+1)_0 \rightarrow e(i,j)_d \\
\qquad\qquad (\![] \ \ k : 0 \leqslant k < \Gamma_{E(i,j)} : E(i,j)^{\text{in}} = e(i,j)_k \rightarrow e(i,j)_{(k+1) \bmod \Gamma_{E(i,j)}}) \\
\qquad\qquad [] \ \ \textbf{else} \rightarrow E(i,j)^{\text{in}} \\
\qquad\qquad \textbf{fi} \qquad\qquad) \\
\quad Z^{\text{out}} = \textbf{if}\ \ (\exists\, j : 0 \leqslant j < 2^{n-3} : E(n-3,j)^{\text{in}} = e(n-3,j)_s \wedge \mathcal{B}(F(n-3,j))) \vee \\
\qquad\qquad (\exists\, i,j : 0 \leqslant i \leqslant n-3 \wedge 0 \leqslant j < 2^i : E(i,j)^{\text{in}} = e(i,j)_{\Gamma_{E(i,j)}-1}) \rightarrow z_c \\
\qquad\qquad [] \ \ \textbf{else} \rightarrow z_p \\
\qquad\qquad \textbf{fi}
\end{array}
$$

$\mathcal{B}(F(n-3,i)$ and $\mathcal{B}(L(n-3,i)$ are defined as follows:

$$\begin{aligned}\mathcal{B}(F(n-3,i)) &= (F(n-3,i)_0^{\text{in}}=f(n-3,i)_0) \wedge (F(n-3,i)_1^{\text{in}}=f(n-3,i)_1)\\ \mathcal{B}(L(n-3,i)) &= (L(n-3,i)_0^{\text{in}}=\ell(n-3,i)_0) \wedge (L(n-3,i)_1^{\text{in}}=\ell(n-3,i)_1)\end{aligned}$$

9.3 The Proof of Correctness

Theorem 6 *Assume* $(\forall\ j : 0 \leqslant j < 2^{n-3} : 0 \neq \mathsf{flow}(F(n-3,j)_0) \neq \mathsf{flow}(F(n-3,j)_1) \neq 0 \wedge 0 \neq \mathsf{flow}(L(n-3,j)_0) \neq \mathsf{flow}(L(n-3,j)_1) \neq 0)$. *If* $\mathcal{P}(\Phi, \vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{E_0}, \cdots, \vartheta_{E_{n-3}}) = \Phi$, *the separation control variables specified by* $HostProg_{n>3}(D_c)$ *and* $CellProg_{n>3}(D_c)$ *are correct.*

Proof. The proof is conducted in two steps. First, we must show that Specs. 1–6 are satisfied for all 3-faces $\{\mathcal{F}(n-3,j) \mid 0 \leqslant j < 2^{n-3}\}$ of the index space. This follows from Thms. 3–5. Second, we must show that the set of first computation points and the set of last computation points of E_i in $\mathcal{F}(i,j)$ can be correctly identified. This is proved inductively using Cor. 1, by starting from $i=n-4$ and finishing at $i=0$ for faces $\{\mathcal{F}(i,j) \mid 0 \leqslant i < n-3,\ 0 \leqslant j < 2^i\}$. □

Thm. 6 is also valid if the assumption $\mathcal{P}(\Phi, \vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{E_0}, \cdots, \vartheta_{E_{n-3}}) = \Phi$ is replaced by the following restrictions:

- There exists $\vartheta(i-1,j)$, which is the control dependence vector for $E(i-1,j)$, such that $\mathcal{F}(i,2j)$ and $\mathcal{F}(i,2j+1)$ are facets of $\mathcal{F}(i-1,j)$, i.e., $\mathcal{F}(i,2j)=\mathsf{in}(\mathcal{F}(i-1,j),\vartheta(i-1,j))$ and $\mathcal{F}(i,2j+1)=\mathsf{out}(\mathcal{F}(i-1,j),\vartheta(i-1,j))$.
- For even j, $\mathcal{F}(n-2,j)$ is a parallelogram. Two of its edges are parallel to $\vartheta(n-3,j)_{f_0}$, which is the control dependence vector for $F(n-3,j)_0$; the other two are parallel to $\vartheta(n-3,j)_{f_1}$, which is the control dependence vector for $F(n-3,j)_1$.
- For odd j, $\mathcal{F}(n-2,j)$ is a parallelogram. Two of its edges are parallel to $\vartheta(n-3,j)_{\ell_0}$, which is the control dependence vector for $L(n-3,j)_0$; the other two are parallel to $\vartheta(n-3,j)_{\ell_1}$, which is the control dependence vector for $L(n-3,j)_1$.

9.4 Adaptation of the Source

The definition of $\mathcal{P}(\Phi, \vartheta_0, \cdots, \vartheta_{n-1})$ in Sect. 9.1 indicates how the index space can be adapted to $\mathcal{P}(\Phi, \vartheta_0, \cdots, \vartheta_{n-1})$. The control signals are then specified for the adapted UREs.

Similar to three-dimensional UREs, the termination control for n-dimensional UREs can be specified by the separating hyperplane method. Then, we can replace the previous specifications for $\mathcal{F}(1,1)$ (i.e., $\mathsf{out}(\Phi, \vartheta_{E_0})$) and all its descendants in the decomposition tree by the specification of $\mathcal{F}(1,1)$ derived with the separating hyperplane method (Def. 6). Therefore, no restriction is imposed on the shape of $\mathcal{F}(1,1)$.

A taxonomy of applications suitable for a systolic implementation is given in [5]. A large proportion of these applications have been specified as n-dimensional source UREs with n orthogonal data dependence vectors (in most applications, $n=3$), which are, in general, orthonormal bases of an n-dimensional vector space and can be chosen as control dependence vectors. An application of the adaptations described in Sect. 8.4 and this section can readily make the index space to be of required form.

9.5 Hardware Support

There are 2^i evolution control variables $E(i,j)$ at level i, 2^{n-3} initialization control variables $F(n-3,j)_0$ and $F(n-3,j)_1$ each, and 2^{n-3} termination control variables $L(n-3,j)_0$ and $L(n-3,j)_1$ each. If all evolution control variables of one level are associated with the same control dependence vector, they can be merged to variable E_i. This reduces the bit width. Similarly, we can merge the control variables $\{F(n-3,j)_0 \mid 0 \leqslant j < 2^i\}$ to F_0, $\{F(n-3,j)_1 \mid 0 \leqslant j < 2^i\}$ to F_1, $\{L(n-3,j)_0 \mid 0 \leqslant j < 2^i\}$ to L_0, $\{L(n-3,j)_1 \mid 0 \leqslant j < 2^i\}$ to L_1. Note that the value spaces of two control variables, say, v and w that are being merged must be distinct, i.e., $\mathcal{S}(v) \cap \mathcal{S}(w) = \{\}$. The resulting UCREs can be readily obtained from the original UCREs.

The bit width of E_i is $\lceil \log_2(2^i(\Gamma_{E_i}+2)+1) \rceil$. The bit width of v ($v \in \{F_0, F_0, L_1, L_1\}$) is $\lceil \log_2(2^{n-3}+1) \rceil$.

10 The Elimination of Control Signals

Computation control is not necessary if all computation points are of the same type. Separation control is not necessary if pipelining points can be treated as computation points without violating the semantics of the source UREs. We have proved that separation control is unnecessary in all two-dimensional systolic arrays for matrix product by making use of algebraic properties of addition and multiplication [19]. This is not true for one-dimensional systolic arrays. Consider Fig. 2. Elements $a_{3,1}$, $b_{3,4}$, and $c_{1,1}$ are propagated at the pipelining point highlighted by a circle. We cannot treat this point as a computation point; otherwise, $c_{1,1}$ would be recomputed, when it should just be propagated. Separation control plays an important rôle in systolic arrays of reduced dimension.

We illustrate the elimination of separation control with the example of matrix product.

Theorem 7 *Separation control is unnecessary iff elements of A and B are not present simultaneously at a pipelining point whenever an element of C is present.*

Proof.

$\Longleftarrow$ A and B are pipelining variables at both computation and pipelining points. We only need to consider variable C, by distinguishing two cases of pipelining points:

- A pipelining point that propagates no elements of C. The operation for C can be specified as $C^{\text{out}} = C^{\text{in}} + A^{\text{in}} B^{\text{in}}$ without violating the semantics of matrix product.
- A pipelining point that propagates some element of C. By the hypothesis, A or B is undefined at the point. Then, either the ϑ_A-path or the ϑ_B-path that passes through the point contains no computation points (Lemma 3). $C^{\text{out}} = C^{\text{in}}$ is semantically equivalent to $C^{\text{out}} = C^{\text{in}} + A^{\text{in}} 0$ and $C_{\text{out}} = C^{\text{in}} + 0 B^{\text{in}}$. If we inject 0 in place of $\perp$ for A or B, we can replace operation $C^{\text{out}} = C^{\text{in}}$ at pipelining points by operation $C^{\text{out}} = C^{\text{in}} + A^{\text{in}} B^{\text{in}}$ without violating the semantics of matrix product.

$\Longrightarrow$ Evaluating a pipelining point that propagates an element for each of three variables A, B and C as a computation point violates the semantics of matrix product. □

A synthesis procedure described in [18] takes an allocation vector σ and returns a scheduling vector such that the resulting space-time mapping satisfies the hypothesis of Thm. 7. One such space-time mapping is:

$$\Pi = \begin{bmatrix} 6m-1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

Control signals are eliminated at the cost of additional computation steps necessary for the satisfaction of the hypothesis of Thm. 7. In the presence of control signals, this hypothesis need not be satisfied. One valid according improvement of the time component is:

$$\Pi = \begin{bmatrix} 2m-2 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

The former space-time mapping requires $18m^2-18m+1$ steps, the latter $6m^2-9m+4$ steps.

11 Related Work

To our knowledge, Guibas, Kung and Thompson [6] were the first to point out that control in a systolic array may be implemented by pipelining control signals analogously to data. Later, Chen [1] gave a formal treatment of this idea. 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. Rajopadhye [13] presented a similar idea [13].

The methods of Chen and Rajopadhye represented significant progress but have the following limitations. First, they require the bijectivity of the space-time mapping. Thus, they do not readily generalize to arrays of reduced dimension. 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, the specification of control signals for pipelining points is not considered.

Let us mention other work related to this subject. Ramakrishnan and Varman [14] present a one-dimensional systolic array for matrix product. 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 [8] shows how control signals (i.e., instructions) can be pipelined to solve problems such as matrix product and merging two arrays.

12 Conclusions

It can be shown that the UCREs for the separation control variables presented in this paper also work for r-dimensional arrays ($1<r<n$) with σ being a scheduling matrix of size $r \times n$. This is mainly because Lemma 7, on which the construction of UCREs for the separation control variables depends, is still valid.

The representation of a one-dimensional systolic array by a space-time diagram has proved helpful. It would be useful to study the properties of space-time diagrams. Then we could choose space-time mappings in such a way that systolic arrays from a special class, e.g., with specific latency, throughput and control complexity, etc. are obtained. The work on systolic automata may be relevant here (e.g., [2]).

One limitation of our specification of separation control is the requirement that $\mathsf{in}(\Phi, \vartheta_E)$ be a facet of the index space. Although any index space can be brought into this form, an adaptation results in more points. This leads potentially to an increased latency or more processors in the resulting systolic array.

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

13 Acknowledgements

We are grateful to Björn Lisper for a reading of an early draft and for many constructive and supportive discussions. Thanks to Marina Chen and Ping F. Yeung for comments.

14 References

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

[2] C. Choffrut and K. Culik. On real-time cellular automata and trellis automata. *Acta Informatica*, 21:393–407, 1984.

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

[4] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[5] J. A. B. Fortes, K.-S. Fu, and B. W. Wah. Systematic design approaches for algorithmically specified systolic arrays. In V. M. Milutinović, editor, *Computer Architecture – Concepts and Systems*, chapter 11. North-Holland, 1988.

[6] 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.

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

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

[10] C. Lengauer and J. W. Sanders. The projection of systolic programs. *Formal Aspects of Computing*, 2:273–293, 1990.

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

[12] 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.

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

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

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

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

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

[18] J. Xue and C. Lengauer. On one-dimensional systolic arrays. In *Proc. ACM Int. Workshop on Formal Methods in VLSI Design.* Springer-Verlag, Jan. 1991. To appear.

[19] J. Xue and C. Lengauer. The systematic derivation of control signals for systolic arrays. Technical Report ECS-LFCS-91-152, Department of Computer Science, University of Edinburgh, May 1991.

A Examples

A.1 Matrix Product

As mentioned in Sect. 7, it is best to choose data dependence vectors as control dependence vectors whenever possible. For reasons of symmetry, it makes no difference which data variable we choose for a labelling by E: A, B or C. Let us choose $\ell.E = C$. Both $\mathsf{in}(\Phi, \vartheta_E)$ and $\mathsf{out}(\Phi, \vartheta_E)$ are facets of the index space Φ.

First, let us choose control dependence vectors ϑ_{F_0} and ϑ_{F_1}. By Thm. 4, the only choices are ϑ_A and ϑ_B. We choose $\ell.F_0 = A$ and $\ell.F_1 = B$. Symmetrically, choosing $\ell.L_0 = A$ and $\ell.L_1 = B$, Thm. 5 is satisfied. By Thm. 3, the separation control variables are correct. The control dependence matrix is:

$$D_c = [\vartheta_{F_0}, \vartheta_{F_1}, \vartheta_{L_0}, \vartheta_{L_1}, \vartheta_E] = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Because all computation points are of the same type, computation control is unnecessary. Hence, $\mathit{HostProg}_{n=3}(D_c)$ and $\mathit{CellProg}_{n=3}(D_c)$ specify the control signals for all one-dimensional systolic arrays for matrix product whose data variables A and B are moving.

A.1.1 Three Moving Data Variables

Let us choose the space-time mapping that describes Ramakrishnan and Varman's array (Fig. 2). Our array runs in $(9m^2-9m+2)/2$ steps. The array of Ramakrishnan and Varman runs in $(12m^2+7m+2)/2$ steps. Both arrays require the same hardware for data flow. Our array uses three control variables with a total bit width of six bits. Their array uses only five bits, but it needs an encoder and decoder per cell.

A.1.2 Two Moving Data Variables

Choose the space-time mapping:

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

Both data variable C and control variable E are stationary. Assume that C is initialized to zero and E to e_s or $\perp$ appropriately.

The result is a systolic array that runs in m^3+m^2-1 steps. The array of Kumar and Tsai [7] with the same processor space runs in $3m^2-m+3$ steps. However, it uses two links for data variables B (we use one). Elements of B must be frequently routed from one link to the other, complicating the control circuitry.

To compare the hardware required for the control flow: both arrays use a two-bit buffer for E in each cell. In Kumar and Tsai's array, there are three one-bit moving control variables; two control variables need two buffers per channel; one needs one buffer. In our array, there are four one-bit moving control variables: F_0, F_1, L_0 and L_1; F_0 and F_1 do not need buffers; L_0 and L_1 each needs m buffers per channel.

A.1.3 One Moving Data Variable

Lengauer and Sanders [10] proposed a one-dimensional systolic array for implementation on transputers. This array conforms to the following space-time mapping:

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

Data variables B and C are stationary. The array consists of m cells and runs in m^2-m-1 steps. $HostProg_{n=3}(D_c)$ and $CellProg_{n=3}(D_c)$ are correct if specifications for the access of stationary variables are provided.

A.2 LU-Decomposition

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 $AB = C$. The elements of the upper triangle of A and the lower triangle of B (excluding the diagonal) are 0; the diagonal elements of A are 1.

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

UREs:

$$
\begin{array}{lll}
0<i\leqslant m, 0<j\leqslant m, 0=k & \rightarrow & C(i,j,k) = c_{i,j} \\
k<i\leqslant m, m=j \quad , 0<k\leqslant m & \rightarrow & a_{i,k} = A(i,j,k) \\
m=i \quad , k\leqslant j\leqslant m, 0<k\leqslant m & \rightarrow & b_{k,j} = B(i,j,k) \\
k=i \quad , k=j \quad , 0<k\leqslant m & \rightarrow & B(i,j,k) = B(i,j,k-1)^{-1} \quad \blacksquare \\
k<i\leqslant m, k=j \quad , 0<k\leqslant m & \rightarrow & A(i,j,k) = C(i,j,k-1)B(i,j,k) \quad \blacktriangle \\
k=i \quad , k<j\leqslant m, 0<k\leqslant m & \rightarrow & B(i,j,k) = C(i,j,k-1) \quad \blacktriangledown \\
k<i\leqslant m, k<j\leqslant m, 0<k\leqslant m & \rightarrow & A(i,j,k) = A(i,j-1,k) \quad \} \bullet \\
k<i\leqslant m, k<j\leqslant m, 0<k\leqslant m & \rightarrow & B(i,j,k) = B(i-1,j,k) \quad \} \bullet \\
k\leqslant i\leqslant m, k\leqslant j\leqslant m, 0<k\leqslant m & \rightarrow & C(i,j,k) = C(i,j,k-1) - A(i,j-1,k)B(i-1,j,k) \quad (*)
\end{array}
$$

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

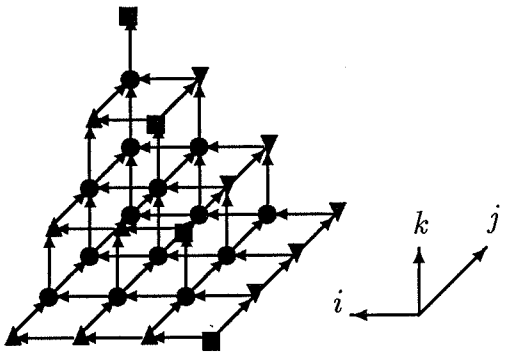
Variables: A, B, C

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

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

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

Data Dependence Graph ($m=4$)*:*



For $s \in \{\blacksquare, \blacktriangle, \blacktriangledown, \bullet\}$, the equation(s) labelled by s is defined at the points represented by s in the data dependence graph. The equation labelled by $(*)$ is defined at every point. Input and output equations are not depicted.

A.2.1 The Specification of Computation Control Variables

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

$$
\begin{aligned}
\Phi_{\blacksquare} &= \{(i,j,k) \mid k=i, \quad k=j, \quad 0<k\leqslant m\} \\
\Phi_{\blacktriangle} &= \{(i,j,k) \mid k<i\leqslant m, k=j, \quad 0<k\leqslant m\} \\
\Phi_{\blacktriangledown} &= \{(i,j,k) \mid k<i\leqslant m, k<j\leqslant m, 0<k\leqslant m\} \\
\Phi_{\bullet} &= \{(i,j,k) \mid k=i, \quad k<j\leqslant m, 0<k\leqslant m\}
\end{aligned}
$$

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. We denote them by P and Q and let $\mathcal{S}(P)=\{p_0,p_1\}$ and $\mathcal{S}(Q)=\{q_0,q_1\}$. Def. 6, with a choice of $\vartheta_P=(1,0,0)=\vartheta_A$ and $\vartheta_Q=(1,0,0)=\vartheta_B$ for the corresponding control dependence vectors yields the following specification of computation control variables:

$$\begin{array}{lll}
k=i \quad ,k-1=j \ ,0<k\leqslant m & \rightarrow & P(i,j,k) = p_0 \\
k<i\leqslant m,k-1=j \ ,0<k\leqslant m & \rightarrow & P(i,j,k) = p_1 \\
k\leqslant i\leqslant m,k\leqslant j\leqslant m,0<k\leqslant m & \rightarrow & P(i,j,k) = P(i,j-1,k) \\
k-1=i \ ,k=j \quad ,0<k\leqslant m & \rightarrow & Q(i,j,k) = q_0 \\
k-1=i \ ,k<j\leqslant m,0<k\leqslant m & \rightarrow & Q(i,j,k) = q_1 \\
k\leqslant i\leqslant m,k\leqslant j\leqslant m,0<k\leqslant m & \rightarrow & Q(i,j,k) = Q(i-1,j,k)
\end{array}$$

The pairs of control signals of P and Q at the four partitions $\Phi_{\blacksquare}$, $\Phi_{\blacktriangle}$, $\Phi_{\blacktriangledown}$ and $\Phi_{\bullet}$ is (p_0,q_0), (p_1,q_0), (p_0,q_1) and (p_1,q_1), respectively.

A.2.2 The Specification of Separation Control Variables

A.2.2.1 Adaptation of Index Space

The adaptation of the index space described in Sect. 8.4 produces the adapted index space $\Phi_{\mathrm{A}}=\Phi_{\mathrm{E}}$, the cubic index space of matrix product. Since ϑ_A, ϑ_B and ϑ_C are the same data dependence vectors as in matrix product, the specification of the separation control variables proceeds as for matrix product (App. A.1). Choosing $\ell.F_0=\ell.L_0=A$, $\ell.F_1=\ell.L_1=B$ and $\ell.E=C$, $HostProg_{n=3}(D_{\mathrm{c}})$, with Φ replaced by Φ_{E}, and $CellProg_{n=3}(D_{\mathrm{c}})$ constitute the specification for the separation control variables.

The computation control variables are then specified for Φ_{E}. If we further note that variables A, B and C are undefined at all points in $\Phi_{\mathrm{E}}\setminus\Phi$, only the original index space needs to be considered. The specification of the computation control variables is as in App. A.2.1.

The control dependence matrix D_{c} is as for matrix product (App. A.1).

A.2.2.2 Separating Hyperplanes

We choose $\ell.E=C$. The specification of the initialization control variables F_0 and F_1 proceeds exactly as for matrix product. We obtain $\ell.F_0=A$ and $\ell.F_1=B$. If we had chosen $\ell.E=A$ or $\ell.E=B$, an adaptation of the index space as described in Sect. 8.4 would be necessary, because neither $\mathsf{in}(\Phi,\vartheta_A)$ nor $\mathsf{in}(\Phi,\vartheta_B)$ is a facet of the index space Φ.

The termination control can be specified by the separating hyperplane method (Def. 6). It turns out that P and Q, as specified previously for computation control, can also serve as termination control variables. Since $\mathsf{out}(\Phi,\vartheta_E)=\Phi_{\blacksquare}\cup\Phi_{\blacktriangle}\cup\Phi_{\blacktriangledown}$ and $\Phi\setminus\mathsf{out}(\Phi,\vartheta_E)=\Phi_{\bullet}$, the pairs of control signals of P and Q at these two domains are distinct. Hence, the last computation points of E are correctly identified. We must add the termination control value $\perp$: $\mathcal{S}(P)=\{p_0,p_1,\perp\}$ and $\mathcal{S}(Q)=\{q_0,q_1,\perp\}$.

The control dependence matrix D_{c} is as for matrix product (App. A.1).

The previous specification of P and Q takes the place of the second conditional statement in $HostProg_{n=3}(D_c)$. Furthermore, $\mathcal{B}(L)$ in program $CellProg_{n=3}(D_c)$ must be replaced by:

$$\mathcal{B}(P,Q) \;=\; (P^{\text{in}}=p_0 \wedge Q^{\text{in}}=q_0) \vee (P^{\text{in}}=p_0 \wedge Q^{\text{in}}=q_1) \vee (P^{\text{in}}=p_1 \wedge Q^{\text{in}}=q_0)$$

A.2.2.3 Comparison of the Two Methods

The specification of separation control (either as in App. A.2.2.1 or as in App. A.2.2.2) and the specification of computation control apply for all one-dimensional systolic arrays for LU-decomposition whose data variables A and B are moving.

For a fixed space-time mapping, the number of links, buffers on the corresponding channels and the bit widths are independent of the method used for the specification of termination control. The number of cells and the latency turn out no larger with the separating hyperplane method than with the other method. The host and cell programs turn out slightly more complex with the separating hyperplane method. For example, pick the space-time mapping that describes Ramakrishnan and Varman's array for matrix product (Fig. 2) and let m be even. Both arrays run in $(9m^2-11m+4)/2$ steps. The array obtained with the separating hyperplane method has $(2m^2-2m+2)/2$ cells, the one obtained with the other method has $(3m^2-3m+2)/2$ cells.