J. VLSI Signal Processing, 10(2):181-199, 1995

Closed-Form Mapping Conditions for the Synthesis of Linear Processor Arrays

Jingling Xue
Department of Mathematics, Statistics & Computing Science,
University of New England
Armidale, NSW 2351, Australia

Abstract. This paper addresses the problem of mapping algorithms with constant data dependences to linear processor arrays. The closed-form necessary and sufficient mapping conditions are derived to identify mappings without computational conflicts and data link collisions. These mapping conditions depend on the space-time mapping matrix and the problem size parameters only. Their correctness can be verified in constant time that is independent of problem size. The design of optimal linear processor arrays is formulated as a mathematic programming problem, which can be solved efficiently by a systematic enumeration of a polynomial search space.

1 Introduction

The potential of VLSI for MIMD parallelism was first recognised by H. T. Kung and C. E. Leiserson [1], when they introduced the term *systolic array* to describe a processor network that was suitably constrained to meet the requirements of VLSI technology. The characteristic features of a systolic array are synchrony of computations, local and regular connections between processors and massive decentralised parallelism. This special-purpose computing paradigm supports the parallel implementation of iterative algorithms in a variety of areas, e.g., numerical analysis, signal or image processing, and graph theory.

A lot of research has been devoted to systematically synthesise systolic arrays from an algorithm description [2, 3, 4, 5, 6, 7, 8]. The basic idea is to apply a space-time mapping (transformation) to the algorithm so that the transformed algorithm represents a systolic array. A taxonomy of many synthesis methods can be found in [9]. Recently, the space-time mapping technique has been extended to deal with the optimisation of the array's latency [10, 11, 12], the optimisation of the processor count [13], the synthesis of fixed-size systolic arrays [14, 15, 16, 17, 18], the synthesis of spacetime-minimal systolic arrays [19, 20, 21], and the synthesis of control signals for systolic arrays [22, 23]. Most of these methods are restricted to the case where n-dimensional algorithms are mapped into $(n-1)$-dimensional systolic arrays. In this case, a space-time mapping that describes an $(n-1)$-dimensional array must satisfy two general mapping conditions:

Precedence Constraint: The execution of the array respects the dependence relations of the algorithm. That is, the time step assigned to a computation is smaller than those assigned to all its dependent computations.

Computation Constraint: No two computations are mapped to the same processor and the same time step.

Additional mapping conditions can be introduced if the limitations of resource are taken into account. One such a condition in [24] ensures that the space-time mapping can be implemented in a fixed architecture with predefined data links of processors.

In the context of $(n-1)$-dimensional array synthesis, the space-time mapping is an $n \times n$ integer matrix. The closed-form precedence and computation constraints exist. The precedence constraint is satisfied if the dot product of the space-time mapping and every dependence vector of the algorithm is positive. The computation constraint is satisfied if the space-time mapping is non-singular. Since these two constraints are linear constraints, the design of optimal $(n-1)$-dimensional arrays is an integer programming problem [8], which can also be solved efficiently by a systematic enumeration of its search space [25].

The synthesis of r-dimensional arrays from n-dimensional algorithms is a more difficult problem when $0 < r < n-1$. (In this case, the processor arrays are often referred to as *lower dimensional arrays*.) There are two reasons. Firstly, the space-time mapping is a $(r+1) \times n$ non-square matrix. We cannot verify the computation constraint by checking the non-singularity of the space-time mapping. A closed-form computation constraint needs to be developed. Secondly, as pointed out by Lee and Kedem [26], a space-time mapping that satisfies both the precedence and computation constraints may not describe a lower dimensional array. Sect. 5 provides an example space-time mapping that justifies this claim. To describe a lower dimensional array, a space-time mapping must satisfy one more general mapping condition:

Communication Constraint: The two data elements never travel together on the same data link.

Finding a closed-from communication constraint is not an easy task.

Following [24], we say that a *computational conflict* occurs if more than one computation of an algorithm are mapped to the same processor and the same time step, and that a *link conflict* (*data link collision*) occurs if more than one datum are so mapped that they travel along the same data link at the same time step.

There have been several attempts on mapping n-dimensional algorithms into lower dimensional processor arrays [27, 28, 29, 26, 8, 30, 31, 24]. In particular, important steps towards a formal solution were made in [29]. Lee and Kedem [29] gave a set of necessary and sufficient conditions on mappings without computational and link conflicts. However, the computation and communication constraints they proposed do not have closed-form expressions (see Def. 1). Their verification is done by analysis of all computations in the index space of the algorithm, which is too expensive to be practically feasible. Shang and Fortes [24] presented a set of closed-form conditions on computational conflict-free mappings. In their methodology, the issue of avoiding link conflicts was not addressed. Ganapathy and Wah [28] proposed a parameter-based method in which the operations of the array are captured by a set of parameters, and the constraints are derived to avoid computational and link conflicts. The recurrence equations they considered are uniform, i.e., all equations are defined at the same domain, i.e., the index space. For this class of algorithms, the communication constraint always implies the computation constraint (see Thm. 1). Hence, the issue of making the computation constraint closed-form was not discussed. The communication constraint was stated in a theorem ([28, Thm. 3]). It does not have closed-form expressions since it contains a few parameters that need to be further defined every time when it is applied to individual variables. The closed-form condition was given in the context of algebraic path problem. But how this is done in general was not mentioned.

This paper is concerned with the derivation of closed-form computation and communication constraints for mapping three-dimensional algorithms to linear processor arrays. Depending on the shapes of the domains of input and output data, different closed-form communication constraints are provided. Often the absence of link conflicts on a certain data link implies the absence of computational conflicts. Conditions are provided that check for this situation so that the computation constraint can be eliminated. If the computation constraint cannot be eliminated this way, we simply replace the computation constraint by the communication constraint for an imaginary variable. So the focus of this paper is on the derivation of closed-form communication constraints.

2 Algorithm Model

$\mathbb{Z}$ ($\mathbb{Q}$) denotes the set of integers (rationals). $\mathbb{Z}^n$ ($\mathbb{Q}^n$) denotes its n-fold Cartesian product. The starting point of our synthesis method is a single-assignment algorithm, which is a set of perfectly nested **for** loops adhering to the following format:

INPUT: $\mathrm{in}_{V_1} = P_1$
$\mathrm{in}_{V_2} = P_2$
$\cdots$
$\mathrm{in}_{V_\ell} = P_\ell$

OUTPUT: $\mathrm{out}_{V_1} = Q_1$
$\mathrm{out}_{V_2} = Q_2$
$\cdots$
$\mathrm{out}_{V_\ell} = Q_\ell$

LOOPS: **for** x_1 **from** lb_1 **to** ub_1 **do**
for x_2 **from** lb_2 **to** ub_2 **do**
$\cdots$
for x_n **from** lb_n **to** ub_n **do**
S_1
S_2
$\cdots$
S_r

where

- $V_1, V_2, \cdots, V_\ell$ are the variables of the algorithm; they form the variable set $\mathcal{V}$.
- Set $\mathrm{in}_{V_i} \in \mathbb{Z}^n$ ($\mathrm{out}_{V_i} \in \mathbb{Z}^n$) is called the *input* (*output*) *space* of variable V_i. $V_i(I)$ for all vectors I in in_{V_i} (out_{V_i}) represent the input (output) data of V_i. By convention, empty input and output spaces need not be declared in the algorithm. Both input and output spaces are sometimes referred to as *I/O spaces*.
- The loops define a space Φ of points $(x_1, x_2, \cdots, x_n)$ called the *index space* ($\Phi \subset \mathbb{Z}^n$):

$$\Phi \quad = \quad \{(x_1, x_2, \cdots, x_n) \mid (\forall\ i : 0 < i \leq n : lb_i \leq x_i \leq ub_i)\}$$

The bounds lb_i and ub_i are linear expressions in the loop indices x_1 to x_{i-1} and in additional variables that specify the problem size. The loops have unit stride.

- Statement S_i in the loops has the following form:

$$V(I) \;=\; \begin{cases} \textbf{if } I \in \Phi_1 \rightarrow \; f_1(W_1(I-\vartheta_1), \ldots) \\ [\,]\; I \in \Phi_2 \rightarrow \; f_2(W_2(I-\vartheta_2), \ldots) \\ \qquad\qquad \cdots \\ [\,]\; I \in \Phi_s \rightarrow \; f_s(W_s(I-\vartheta_s), \ldots) \\ \textbf{fi} \end{cases}$$

V and W_i are some variables in the variable set $\mathcal{V}$. $\Phi_j \in \mathbb{Z}^n$ and $\Phi_k \in \mathbb{Z}^n$ are disjoint if j and k differ. The three dots in the argument list of f_i stand for an arbitrary but fixed number of similar arguments. The constant vectors $\vartheta_i \in \mathbb{Z}^n$ are called *dependence vector*. Following [29], we say that a dependence vector is *associated with* a variable if it is the difference of indices where that variable is used and where that variable is generated. By this definition, a variable may be associated with more than one dependence vector. The same dependence vector may be associated with more than one variable. It is assumed that each variable is associated with at least one dependence vector [29]. This assumption is reasonable, since a variable with no associated dependence vector represents input/output data only. The graph whose vertices are the points of the index space and whose edges are the dependences between the points is called the *dependence graph*. If V is defined for the entire index space, we write

$$V(I) \;=\; f(W(I-\vartheta), \cdots)$$

We conclude this section by presenting an algorithm for the multiplication of two square matrices of size $m \times m$ (thereafter referred to as matrix product).

$$\begin{array}{rl} \text{Specification:} & (\forall\, i,j : 0 < i,j \leq m : c_{i,j} = (\sum\, k : 0 < k \leq m : a_{i,k} b_{k,j})) \\ \textbf{INPUT}: & \text{in}_A = \{(i,0,k) \mid 0 < i,k \leq m\} \\ & \text{in}_B = \{(0,j,k) \mid 0 < j,k \leq m\} \\ & \text{in}_C = \{(i,j,0) \mid 0 < i,j \leq m\} \\ \textbf{OUTPUT}: & \text{out}_C = \{(i,j,m) \mid 0 < i,j \leq m\} \\ \textbf{LOOPS}: & \textbf{for } i \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & \textbf{for } j \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & \textbf{for } k \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & \quad A(i,j,k) = A(i,j-1,k) \\ & \quad B(i,j,k) = B(i-1,j,k) \\ & \quad C(i,j,k) = C(i,j,k-1) + A(i,j-1,k) B(i-1,j,k) \end{array}$$

Where $A(i,0,k) = a_{i,k}$, $B(0,j,k) = b_{k,j}$, $C(i,j,0) = 0$, and $c_{i,j} = C(i,j,m)$.

The corresponding dependence graph is depicted in Fig. 1. The index space is a cube of size m:

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

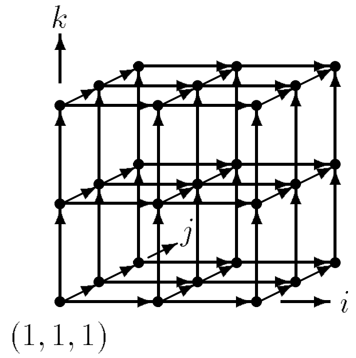

Figure 1: The dependence graph for matrix product ($m = 3$).

The dependence vectors associated with variables A, B and C are:

$$\vartheta_A \;=\; (0,1,0), \quad \vartheta_B \;=\; (1,0,0), \quad \vartheta_C \;=\; (0,0,1)$$

The input spaces of variables A, B and C are translations of the front, left and bottom faces of the index space by $-\vartheta_A$, $-\vartheta_B$ and $-\vartheta_C$, respectively. They are not part of the index space. The output space of variable C is the top face of the index space. The output data of variables A and B are of no interest. Hence, the corresponding output spaces are empty; their declarations are omitted.

3 Array Model

We use the linear array model proposed in [29, 30]. A *linear systolic array* consists of a finite sequence of processors with the following properties (Fig. 2):

- The execution of the array is synchronous, i.e., is governed by a global clock that ticks in unit time.
- Only the two border processors are connected to the host.
- Only neighbouring processors are connected directly with each other.
- A (*data*) *link* is a line of channels all of which have the same number of delay buffers. A buffer retains a value for one time step, i.e., one clock tick. Hence, all the elements move along a common link with a constant velocity.

Two restrictions imposed on the linear array model are worth emphasising here. One restriction requires that I/O with the host be performed at the two border processors of the array. As will be explained in Sect. 6, this actually limits the class of algorithms that can be mapped to the linear array model. Another restriction disallows non-neighbouring channels. As will be explained in Sect. 4.3, this reduces the space of correct space-time mappings. The correctness of the space-time mapping will be discussed in Sect. 4.4.

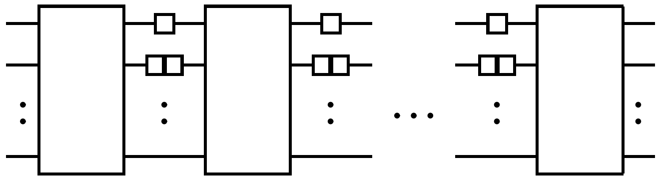

Figure 2: The linear systolic array model. The large solid boxes represent processors. The small boxes represent delay buffers. The lines represent connecting channels.

4 The Space-Time Mapping

Many methods have been proposed to map algorithms into processor arrays. The space-time mapping (transformation) method proposed in [32, 5, 8] is used in this paper. A space-time mapping, T, that describes a linear array is of the form:

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

where $\lambda \in \mathbb{Z}$ is called the *scheduling vector* and $\sigma \in \mathbb{Z}$ the *allocation vector*. A space-time mapping provides a complete description of the underlying processor array. In particular, we can extract the following information from a space-time mapping: the processors and time steps at which computations are performed, the velocities of data, the number of buffers on a channel, the direction of data links, and input/output characteristics, etc.

4.1 Mapping Index Points to Processors and Time Steps

A point I in the index space is computed at processor σI at time step λI. The mapping of index points to processors and time steps can be interpreted geometrically. We can view the index space as being sliced by two types of hyperplanes. A *temporal hyperplane* is a hyperplane whose normal is the scheduling vector λ. A *spatial hyperplane* is a hyperplane whose normal is the allocation vector λ. All the points in a temporal hyperplane are mapped to the same time step. All the points in a spatial hyperplane are mapped to the same processor.

4.2 Mapping Dependence Vectors to Data Links

In our linear array model, the number of data links is not predefined. It is assumed that as many links are available as required. We follow [29] in associating dependence vectors with data links. If a variable is associated with K dependence vectors, we dedicate K different links for the variable, one per dependence vector. Therefore K copies of the same datum may travel through the array, each in a dedicated link and possibly with K different velocities. The velocity and input and output time steps of data traveling on a data link are completedly determined by the space-time mapping. This will be discussed shortly.

Instead of thinking of a variable with K dependence vectors, we can simplify our presentation with no loss of generality by proceeding as if there were K variables each of which is associated with one distinct vector from among the K dependence vectors. The I/O spaces of these K variables inherit the the I/O space of the original variable. In what follows, we assume that each variable in the algorithm is associated with one distinct dependence vector. We shall write ϑ_V for the dependence vector associated with a variable V. Under this assumption, each variable is associated with one link in the array. That is, all elements of a variable move along a dedicated link in the array.

4.3 Data's Velocities, Input and Output Characteristics

We write $\mathsf{flow}(V)$ for the velocity with which the elements of a variable V travel:

$$\mathsf{flow}(V) \quad = \quad \frac{\sigma\vartheta_V}{\lambda\vartheta_V}$$

A variable V is *moving* if $\mathsf{flow}(V) \neq 0$ and is *stationary* if $\mathsf{flow}(V) = 0$. We make the convention that a variable V moves to the right if $\mathsf{flow}(V) > 0$ and to the left if $\mathsf{flow}(V) < 0$. Stationary variables do not require data links. Instead, local processor memories are needed to store the data elements of stationary variables. A space-time mapping does not provide any help for handling stationary variables. A scheme for loading and draining stationary data elements needs to be developed. One such a scheme can be found in [33].

In our array model, all the elements of the same variable move on a data link with a constant velocity. This is implemented by requiring that the same number of buffers be associated with all the channels of a data link. The number of buffers on a channel for a variable V is determined as follows. Let I and J be two index points such that $I = J + \vartheta_V$. Data element $V(J)$ is used to compute data element $V(I)$. Under the space-time mapping, we have $\sigma I = \sigma J + \sigma\vartheta_V$. That is, I and J are mapped to processors at locations σI and σJ, respectively. The element $V(J)$ computed at processor σJ travels with a velocity of $\mathsf{flow}(V)$ to processor σI where $V(I)$ is to be computed. To implement this velocity, the number of buffers associated with a channel for V is

$$\mathrm{buf}(V) \quad = \quad \frac{1}{|\mathsf{flow}(V)|} - 1$$

where the "1" accounts for the fact that evaluation of a point takes unit time. To ensure that the mapping describes a linear array with neighbouring channels, we must exclude the case where buf(V) is not an integer. We introduce a general mapping condition that needs to be satisfied by a space-time mapping:

Delay Constraint: The number of buffers on a channel is an integer.

If the restriction on neighbouring communication is relaxed, the delay constraint can be disregarded. This is why we mentioned in Sect. 3 that the space of correct space-time mappings is reduced if non-neighbouring communication is not permitted.

Once a space-time is given, the time steps at which data are input and output are determined:

- Function input specifies the steps at which input data are input into the array:

$$\mathsf{input}: \ \mathcal{V} \to \mathrm{in}_V \to \mathbb{Z}$$
$$\mathsf{input}(V)(I) \ = \ \begin{cases} \textbf{if} & \mathsf{flow}(V)>0 \ \to \ \lambda I - (\sigma I - p_{\min})/\mathsf{flow}(V) \\ [] & \mathsf{flow}(V)<0 \ \to \ \lambda I - (\sigma I - p_{\max})/\mathsf{flow}(V) \\ \textbf{fi} & \end{cases}$$

- Function output specifies the steps at which output data are output from the array:

$$\mathsf{output}: \ \mathcal{V} \to \mathrm{out}_V \to \mathbb{Z}$$
$$\mathsf{output}(V)(I) \ = \ \begin{cases} \textbf{if} & \mathsf{flow}(V)>0 \ \to \ \lambda I - (\sigma I - p_{\max})/\mathsf{flow}(V) \\ [] & \mathsf{flow}(V)<0 \ \to \ \lambda I - (\sigma I - p_{\min})/\mathsf{flow}(V) \\ \textbf{fi} & \end{cases}$$

Let us explain why the definitions of input and output are correct. For reasons of symmetry, we only consider input when $\mathsf{flow}(V)>0$. Let $V(I)$ be an input datum of variable V. This datum is input from the border processor $p_{\min}$ and travels with a velocity of $\mathsf{flow}(V)$. By the space-time mapping, $V(I)$ must be available at the processor σI at the step λI. The distance between this processor and the border processor $p_{\min}$ is $\sigma I - p_{\min}$. In order for it to reach the processor σI at step λI, the input datum $V(I)$ must be input $(\sigma I - p_{\min})/\mathsf{flow}(V)$ steps earlier at the border processor $p_{\min}$. That is, it must be input at the step $\lambda I - (\sigma I - p_{\min})/\mathsf{flow}(V)$.

4.4 The Correctness of the Space-Time Mapping

The mapping conditions for the correctness of space-time mappings for linear systolic arrays are given below. The notation $f : D \hookrightarrow R$ denotes that function f is injective from domain D to range R. The notation for a quantified expression is $(\mathit{quant}: \ \mathit{range}: \ \mathit{term})$, where *quant* specifies a quantifier and a list of dummy variables for the quantification, *range* specifies the range of the dummies and *term* is some function or predicate on the dummies [34]. For $x, y \in \mathbb{Z}$, we write $x|y$ iff x divides y. Recall that $\mathcal{V}$ denotes the set of all variables of the algorithm.

Definition 1 (Mapping conditions) A space-time mapping, T, is *correct* iff

- $(\forall \ V : V \in \mathcal{V} : \lambda \vartheta_V > 0)$ (Precedence Constraint)
- $T : \Phi \ \hookrightarrow \ \mathbb{Z}^2$ (Computation Constraint)
- $(\forall \ V : V \in \mathcal{V} : \mathsf{flow}(V) \neq 0 \implies \sigma \vartheta_V | \lambda \vartheta_V)$ (Delay Constraint)
- $(\forall \ V : V \in \mathcal{V} : \mathsf{flow}(V) \neq 0 \implies \mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z})$
  $(\forall \ V : V \in \mathcal{V} : \mathsf{flow}(V) \neq 0 \implies \mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z})$ (Communication Constraint)

The precedence constraint ensures that the dependences prescribed in the algorithm are respected. It remains the same as in the synthesis of $(n-1)$-dimensional arrays [6, 8]. The computation constraint ensures that the mapping is computational conflict-free. The delay constraint states that the number of buffers on any channel for a moving variable is a non-negative integer. It serves to enforce the neighbouring communication required in the linear array model. The communication constraint ensures that the mapping is link conflict-free.

These mapping conditions are essentially the same as those proposed by Lee and Kedem [26], except that our communication constraint is both a necessary and sufficient condition on link conflict-free mappings. Lee and Kedem's algorithm model does not have the notion of I/O spaces. They stated the communication constraint with respect to the index space, which amounts to defining the input and output spaces of a variable V as follows:

$$\begin{array}{rcl} \mathrm{in}_V & = & \{I-\vartheta_V \mid I\in\Phi \wedge (I-\vartheta_V)\notin\Phi\} \\ \mathrm{out}_V & = & \{I \mid I\in\Phi \wedge (I+\vartheta_V)\notin\Phi\} \end{array} \tag{1}$$

The I/O spaces defined this way are generally larger than the actual I/O spaces. So the communication constraint thus defined is generally over-stated.

Being dependent on the index vectors, both the computation and communication constraints in Def. 1 do not have closed-form expressions. In the synthesis procedure proposed by Lee and Kedem, the two constraints were verified by analysis of all computations in the index space. The time taken is therefore dependent on the problem size. The procedure is only applicable to problems of small sizes. In what follows, we focus on the derivation of the closed-form computation and communication constraints. The objective is to transform the computation and communication constraints so that the transformed constraints depend on the space-time mapping matrix and the problem size parameters only. As a result, these transformed constraints can be verified in constant time that is independent of the problem size.

We shall drop the antecedent $\mathsf{flow}(V)$ in the delay and the communication constraints. It is understood that the two constraints are only applicable for moving variables. Similarly, whenever we apply functions input and output to a variable V or write $\sigma\vartheta_V|\lambda\vartheta_V$, we mean that V is a moving variable.

The rest of the paper is organised as follows. Sect. 5 uses matrix product as an example to explain the necessity of the communication constraint. Sect. 6 discusses one restriction on the class of algorithms that can be mapped to our linear array model. Sect. 7 is concerned with the elimination of the computation constraint. We show that if a certain relationship exists between the index space and some I/O space, the computation constraint is implied by the communication constraint for that I/O space and can therefore be eliminated. Sect. 8 discusses the issue of simplifying the communication constraint. In general, there are two communication constraints associated with a variable, one for its input space and the other for its output space. The total number of these constraints can be twice as many as there are variables in the algorithm. We show that if a certain relationship exists between two I/O spaces, the constraint for one I/O space is implied by the constraint for the other I/O space and can therefore be eliminated. Sect. 9 considers the computation constraint further. If the computation constraint cannot be eliminated by the method discussed in Sect. 7, we simply add a communication constraint for an imaginary variable. By enforcing this added constraint, we are guaranteed that the computation constraint is enforced. Sect. 10 is the central part of the paper. It is concerned with the derivation of closed-form communication constraints. Depending on the shapes and dimensions of I/O spaces, different closed-form communication constraints are provided.

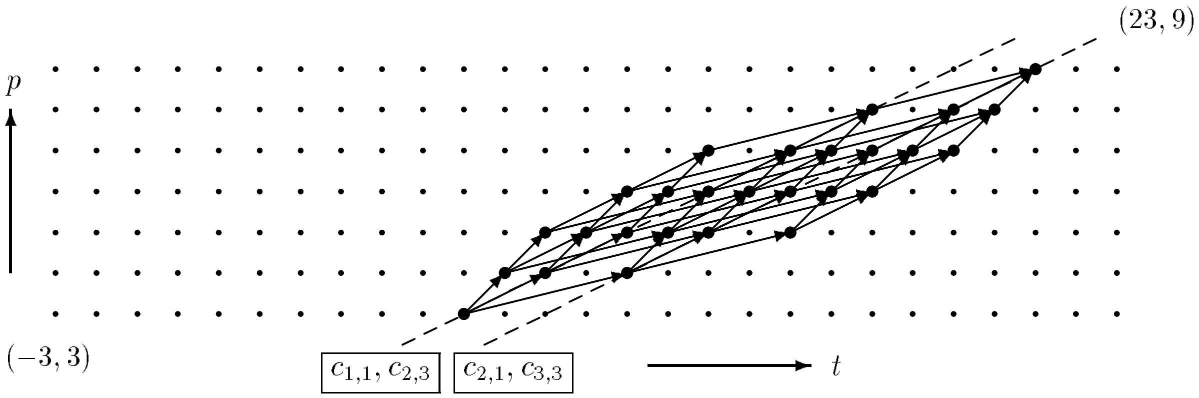

Figure 3: A linear systolic array for matrix product.

5 Why the Communication Constraint?

The importance of the communication constraint for the synthesis of linear processor arrays was first recognised in [30] and later formalised in [29]. To show the necessity of the communication constraint, we provide an example space-time mapping that satisfies the precedence and computation constraints but violates the communication constraint.

The following space-time mapping

$$T = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

describes the array depicted in terms of a space-time diagram in Fig. 3 ($m = 3$). (A *space-time diagram* is a two-dimensional pictorial representation of a linear array, where the temporal dimension is horizontal and the spatial dimension is vertical. There is a dot at the point (t, p) iff t is a time step number and p is a processor coordinate.) This array consists of seven processors and runs in 27 steps. Three links are needed to carry the elements of three variables A, B and C, respectively. We can see from the figure that the precedence and computation constraints are satisfied. The precedence constraint is satisfied because the projections of the three dependence vectors on the time axis all point towards the positive direction of the time axis. The computation constraint is satisfied because no two points of the index space overlap. However, as highlighted by the two dashed lines in the space-time diagram, both $c_{1,1}$ and $c_{2,3}$ are required to be input at time step $t = 7$, and both $c_{2,1}$ and $c_{3,3}$ at time step $t = 9$. Thus, the communication constraint is violated.

6 One Source Restriction

Our linear array models restricts I/O with the host at the two border processors. We mentioned in Sect. 3 that this restriction limits the class of algorithms that can be implemented in our linear array model. This section explains one restriction imposed on the class of algorithms acceptable by our method. We first present two technical lemmata, one of which makes explicit the motivation behind this source restriction. The correctness of the two lemmata can be proved by a simple algebraic calculation. These two lemmata are also useful in the proof of the results in Sects. 7 and 8.

We present two lemmata in a more general setting by extending the functions input and output so that they are of type $\mathcal{V} \to \mathbb{Z}^n \to \mathbb{Z}$. Let $I, J \in \mathbb{Z}$ and $\vartheta \in \mathbb{Z}^n$. We write $I \overset{\vartheta}{\longleftrightarrow} J$ if $I = J + m\vartheta$ for some $m \in \mathbb{Q}$. That is, we write $I \overset{\vartheta}{\longleftrightarrow} J$ to represent the fact that $I - J$ and ϑ are colinear.

Lemma 1 (1) $(\forall\ V, I : V \in \mathcal{V} \wedge I \in \mathbb{Z}^n : I \overset{\vartheta_V}{\longleftrightarrow} J \implies \mathsf{input}(V)(I) = \mathsf{input}(V)(J))$
(2) $(\forall\ V, I : V \in \mathcal{V} \wedge I \in \mathbb{Z}^n : I \overset{\vartheta_V}{\longleftrightarrow} J \implies \mathsf{output}(V)(I) = \mathsf{output}(V)(J))$

As a consequence of this lemma, we must impose the following source restriction on the algorithms input to our method. The input (output) space of a variable V cannot contain two distinct points I and J such that $I \overset{\vartheta_V}{\longleftrightarrow} J$. Otherwise the link conflict always occurs between $V(I)$ and $V(J)$ independently of the space-time mapping chosen.

This restriction reduces the class of algorithms that can be implemented in our linear array model. This is the price we pay for restricting I/O with the host at the two border processors. In fact, the same source restriction exists (although not explicitly stated in the literature) for $(n{-}1)$-dimensional arrays if the I/O with the host is only allowed at the border processors.

In reality, algorithms may exist where an output element $V(I)$ is later used to compute another output element $V(J)$ such that $I \overset{\vartheta_V}{\longleftrightarrow} J$. These algorithms must be modified so that the source restriction holds for the modified algorithms. The modified algorithms are then taken as inputs to our method. Alternatively, we can do away with the source restriction by allowing I/O to be performed at all processors of the array.

Let $p_{\min}$ and $p_{\max}$ be the coordinates of the two border processors.

Lemma 2 $(\forall\ V, I : V \in \mathcal{V} \wedge I \in \mathbb{Z}^n : \mathsf{output}(V)(I) = \mathsf{input}(V)(I) + |(p_{\max} - p_{\min})/\mathsf{flow}(V)|)$.

This lemma says that a datum of V takes $|(p_{\max} - p_{\min})/\mathsf{flow}(V)|$ to travel across the array.

7 Eliminating the Computation Constraint

Thm. 1 presents the sufficient condition under which the computation constraint can be eliminated. Let X and Y be two sets and ϑ be a vector in $\mathbb{Z}^n$. We write $X \overset{\vartheta}{\longrightarrow} Y$ if, when projected along ϑ onto a fixed hyperplane, the projection of X is a (not necessarily strict) subset of the projection of Y. That is, if $X \overset{\vartheta}{\longrightarrow} Y$, then for every I in X there is J in Y such that $I \overset{\vartheta}{\longleftrightarrow} J$. (But the converse may not be true.) Assume that a mapping satisfies the precedence constraint for variable V and that $\Phi \overset{\vartheta_V}{\longrightarrow} \mathrm{in}_V$ ($\Phi \overset{\vartheta_V}{\longrightarrow} \mathrm{out}_V$) holds. Then the absence of link conflicts for the input (output) data of V implies the absence of computational conflicts.

Theorem. 1 *Let* $\lambda\vartheta_V \neq 0$. (1) If $\Phi \overset{\vartheta_V}{\longrightarrow} \mathrm{in}_V$, then $(\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z}) \Longrightarrow (T : \Phi \hookrightarrow \mathbb{Z}^2)$
(2) If $\Phi \overset{\vartheta_V}{\longrightarrow} \mathrm{out}_V$, then $(\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z}) \Longrightarrow (T : \Phi \hookrightarrow \mathbb{Z}^2)$

Proof. We prove (1) by contradiction. The proof of (2) is similar. Let there be two distinct points $I, J \in \Phi$ such that $TI = TJ$, i.e., $\lambda I = \lambda J \wedge \sigma I = \sigma J$. Since $\lambda\vartheta_V \neq 0$, $I \xleftrightarrow{\vartheta_V} J$ does not hold. By further using $\Phi \xrightarrow{\vartheta_V} \mathrm{in}_V$, we know that in_V must contain two distinct points I' and J' such that $I \xleftrightarrow{\vartheta_V} I'$ and $J \xleftrightarrow{\vartheta_V} J'$. We show that $\mathsf{input}(V)(I') = \mathsf{input}(V)(J')$ and thus achieve the impossible. By Lemma 1, $\mathsf{input}(V)(I) = \mathsf{input}(V)(I')$ and $\mathsf{input}(V)(J) = \mathsf{input}(V)(J')$. It suffices to show that $\mathsf{input}(V)(I) = \mathsf{input}(V)(J)$. This follows from the proof hypothesis $\lambda I = \lambda J \wedge \sigma I = \sigma J$ and the definition of input. □

According to this theorem, the computation constraint can be eliminated for the class of uniform algorithms. An algorithm is said to be *uniform* if every variable is defined by one equation over the entire index space. Stated differently, an algorithm is a uniform algorithm if a dependence vector exists between every two index points whose difference is that dependence vector [35]. The dependence vector ϑ_V of a variable V exists uniformly in the dependence graph of the algorithm. $V(I - \vartheta_V)$ represents an input datum if $(I - \vartheta_V) \notin \Phi$ for every $I \in \Phi$. Thus, the input space of V is the one defined in (1). This implies $\Phi \xrightarrow{\vartheta_V} \mathrm{in}_V$. Therefore, the computation constraint can be eliminated. The method proposed by Ganapathy and Wah [28] accepts only uniform algorithms. The issue of avoiding computational conflicts was therefore not addressed.

8 Simplifying the Communication Constraint

Thms. 2 and 3 present the condition under which the communication constraint can be simplified. As mentioned previously, there are generally two communication constraints associated with a variable: one for the input space and the other for the output space. One of these two constraints can be eliminated if a certain relationship between the input and output spaces holds. Let us state this relationship in words first. Let one of the two I/O spaces of a variable V be translated by some $o \in \mathbb{Z}^n$. We then project along ϑ_V the translated I/O space and the other I/O space onto a fixed hyperplane. If one projected space is a subset of the other for some o, then the communication constraint for the containing space implies the communication constraint for the contained space. To state this relationship formally, we use the following notations. Let X be a set in $\mathbb{Z}^n$ and $a \in \mathbb{Z}^n$, the set $\{x + a \mid x \in X\}$ is called the *translation* of X by a. Let X and Y be two sets and ϑ be a vector in $\mathbb{Z}^n$. We write $X \xmapsto{\vartheta} Y$ if there exists a vector $o \in \mathbb{Z}^n$ such that $X \xrightarrow{\vartheta} (Y + o)$. If $\mathrm{out}_V \xmapsto{\vartheta_V} \mathrm{in}_V$ ($\mathrm{in}_V \xmapsto{\vartheta_V} \mathrm{out}_V$), then a mapping that is link conflict-free for the input (output) data of V is also link conflict-free for the output (input) data of V.

Theorem. 2 (1) If $\mathrm{out}_V \xmapsto{\vartheta_V} \mathrm{in}_V$, then $(\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z}) \Longrightarrow (\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z})$
(2) If $\mathrm{in}_V \xmapsto{\vartheta_V} \mathrm{out}_V$, then $(\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z}) \Longrightarrow (\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z})$

Proof. For reasons of symmetry, we only prove (1). Since $\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z}$, it follows from algebraic calculations that $\mathsf{input}(V) : (\mathrm{in}_V + a) \hookrightarrow \mathbb{Z}$ for any vector a in $\mathbb{Z}^n$. By Lemma 2, we obtain $\mathsf{output}(V) : (\mathrm{in}_V + a) \hookrightarrow \mathbb{Z}$. The assumption that $\mathrm{out}_V \xmapsto{\vartheta_V} \mathrm{in}_V$ means that there is an $o \in \mathbb{Z}^n$ such that $\mathrm{out}_V \xrightarrow{\vartheta_V} (\mathrm{in}_V + o)$. With the a being replaced by o, we have $\mathsf{output}(V) : (\mathrm{in}_V + o) \hookrightarrow \mathbb{Z}$. That $\mathrm{out}_V \xrightarrow{\vartheta_V} (\mathrm{in}_V + o)$ implies that, for every I in out_V, there is J in $\mathrm{in}_V + o$ such that $I \overset{\vartheta_V}{\longleftrightarrow} J$. By Lemma 1, we have $\mathsf{output}(V)(I) = \mathsf{output}(V)(J)$ and consequently $\{\mathsf{output}(V)(K) \mid K \in \mathrm{out}_V\} \subseteq \{\mathsf{output}(V)(K) \mid K \in (\mathrm{in}_V + o)\}$. Since $\mathsf{output}(V) : (\mathrm{in}_V + o) \hookrightarrow \mathbb{Z}$, we conclude that $\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z}$. $\square$

For practical algorithms, the conditions stated in this theorem often hold. It is therefore sufficient to enforce the communication constraint for one of the two I/O spaces associated with a variable. The following theorem is a straightforward generalisation of Thm. 2.

Theorem. 3 *Let V and W be two variables such that ϑ_V and ϑ_W are colinear.*

(1) If $\mathrm{in}_V \overset{\vartheta_V}{\longmapsto} \mathrm{in}_W$, then $(\mathsf{input}(W) : \mathrm{in}_W \hookrightarrow \mathbb{Z}) \implies (\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z})$

(2) If $\mathrm{in}_V \overset{\vartheta_V}{\longmapsto} \mathrm{out}_W$, then $(\mathsf{output}(W) : \mathrm{out}_W \hookrightarrow \mathbb{Z}) \implies (\mathsf{input}(V) : \mathrm{in}_V \hookrightarrow \mathbb{Z})$

(3) If $\mathrm{out}_V \overset{\vartheta_V}{\longmapsto} \mathrm{in}_W$, then $(\mathsf{input}(W) : \mathrm{in}_W \hookrightarrow \mathbb{Z}) \implies (\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z})$

(4) If $\mathrm{out}_V \overset{\vartheta_V}{\longmapsto} \mathrm{out}_W$, then $(\mathsf{output}(W) : \mathrm{out}_W \hookrightarrow \mathbb{Z}) \implies (\mathsf{output}(V) : \mathrm{out}_V \hookrightarrow \mathbb{Z})$

9 Transforming the Computation Constraint

If there exists a moving variable V such that $\Phi \overset{\vartheta_V}{\longrightarrow} \mathrm{in}_V$, then the computation constraint can be eliminated (Thm. 1). Otherwise, we replace the computation constraint by the following communication constraint. We imagine there exists a pseudo-variable, denoted Γ, such that $\Phi \overset{\vartheta_\Gamma}{\longrightarrow} \mathrm{in}_\Gamma$. The dependence vector ϑ_Γ of the pseudo-variable Γ can be chosen to be any vector in $\mathbb{Z}^n$. The associated input and output spaces can be defined as in (1), i.e.,

$$\begin{aligned} \mathrm{in}_\Gamma &= \{I - \vartheta_\Gamma \mid I \in \Phi \wedge (I - \vartheta_\Gamma) \notin \Phi\} \\ \mathrm{out}_\Gamma &= \{I \mid I \in \Phi \wedge (I + \vartheta_\Gamma) \notin \Phi\} \end{aligned}$$

By definition, $\Phi \overset{\vartheta_\Gamma}{\longrightarrow} \mathrm{in}_\Gamma$ (and $\Phi \overset{\vartheta_\Gamma}{\longrightarrow} \mathrm{out}_\Gamma$). Thus, the computation constraint can be eliminated (Thm. 1). By Thm. 2, $\mathrm{in}_\Gamma \overset{\vartheta_\Gamma}{\longmapsto} \mathrm{out}_\Gamma$ and $\mathrm{out}_\Gamma \overset{\vartheta_\Gamma}{\longmapsto} \mathrm{in}_\Gamma$. One of the communication constraints associated with the I/O spaces of Γ can be eliminated. Let us arbitrarily disregard the output space out_Γ. As for the choice of ϑ_Γ, it is preferable to choose ϑ_Γ as a dependence vector, say ϑ_V. Since $\mathrm{in}_\Gamma \overset{\vartheta_\Gamma}{\longmapsto} \mathrm{in}_V$ and $\mathrm{in}_\Gamma \overset{\vartheta_\Gamma}{\longmapsto} \mathrm{out}_V$, the communication constraint for variable V can be eliminated (Thm. 3).

10 Deriving Closed-Form Communication Constraints

As the central part of the paper, this section is concerned with the derivation of closed-form communication constraints for mapping three-dimensional algorithms into linear processor arrays. The variables in systolic algorithms represent either the matrices or the vectors in the corresponding problem specifications. These matrices are usually diagonal, rectangular or triangular matrices. Vectors can be dealt with similarly as diagonal matrices. The type of a matrix determines the shape of the input and output spaces of the corresponding variable, which consequently determines the way in which the closed-form communication constraint is derived. For reasons of symmetry, we only consider the input spaces. Often we shall speak of the communication constraint for a variable rather than for the input space of a variable. All results presented in this section apply for the output spaces.

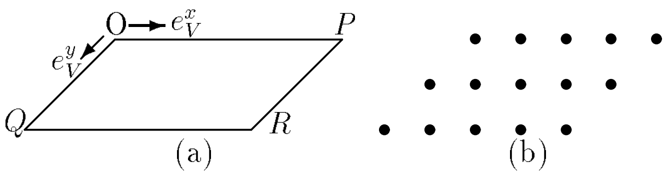

Figure 4: (a) The parallelepiped input space. (b) An instance: $\mathrm{num}_V^x = 5$ and $\mathrm{num}_V^y = 3$.

The derivation of closed-form communication constraints depends on the shapes and dimensions of input spaces. We shall divide this section into several subsections each dealing with the input spaces of a specific shape and dimension. In particular, the input spaces that are lines are associated with variables that represent diagonal matrices and vectors. The input spaces that are parallelograms are associated with variables that represent rectangular and square matrices. The input spaces that are trapezoids are associated with variables that represent triangular matrices. Trapezoids include parallelograms as a special case.

We shall use the following notations. If x and y are integers, not both zero, then their *greatest common divisor*, $\gcd(x, y)$, is the largest positive integer that divides both x and y. We define $\gcd(0, 0) = 1$. The function sign is a mapping from $\mathbb{Z}$ to the set $\{-1, 0, 1\}$: $\mathrm{sign}(x) = \textbf{if } x < 0 \rightarrow -1 \ [\!] \ x = 0 \rightarrow 0 \ [\!] \ x > 0 \rightarrow 1 \textbf{ fi}$. A set $S \subset \mathbb{Z}^n$ is *n-dimensional* if the dimension of the smallest affine set containing S is n [36]. Let P and Q be two points in $\mathbb{Z}^n$. The notation $\overline{PQ}$ stands for the line connecting the two points P and Q, inclusive. Let V be a variable. Let e_V^s be a vector in $\mathbb{Z}^n$ where s is in the set $\{x, y, z\}$. We write δ_V^s for the time difference between the input of two input data $V(I+e_V^s)$ and $V(I)$:

$$\delta_V^s \quad = \quad \mathsf{input}(V)(I+e_V^s) - \mathsf{input}(V)(I) \quad = \quad \lambda e_V^s - \sigma e_V^s (\lambda \vartheta_V / \sigma \vartheta_V)$$

10.1 Lines

The input space in_V is a *line* if it is one-dimensional. Let e_V^x be a vector parallel to the line. The closed-form communication constraint follows from the definition of δ_V^x.

Theorem. 4 *Let* in_V *be a line. The communication constraint for* V *is satisfied iff* $\delta_V^x \neq 0$.

Proof. Let I and J be any two distinct points in the linear input space in_V. Then $I \overset{e_V^x}{\longleftrightarrow} J$, i.e., $I - J = m e_V^x$ ($m \in \mathbb{Q} \wedge m \neq 0$). By definition, $\delta_V^x = \mathsf{input}(V)(I') - \mathsf{input}(V)(J')$ where I' and J' satisfy $I' - J' = e_V^x$. A simple algebraic calculation shows that $\mathsf{input}(V)(I) - \mathsf{input}(V)(J) = m\delta_V^x$. Hence, $V(I)$ and $V(J)$ are input at different time steps iff $\delta_V^x \neq 0$. $\square$

10.2 Parallelograms

The input space in_V is a *parallelogram* if it satisfies the following properties (Fig. 4):

1. Four distinct index points, denoted by O, P, Q and R, are the vertices of the parallelogram.

2. Let $\mathrm{num}_V^x > 1$ and $\mathrm{num}_V^y > 1$ be the number of index points on the edges $\overline{OP}$ and $\overline{OQ}$, respectively. Let $e_V^x = (P - \mathrm{O})/(\mathrm{num}_V^x - 1)$ and $e_V^y = (Q - \mathrm{O})/(\mathrm{num}_V^y - 1)$. Then

$$\mathrm{in}_V \quad = \quad \{\mathrm{O} + (i-1)e_V^y + (j-1)e_V^x \mid 0 < i \le \mathrm{num}_V^y \wedge 0 < j \le \mathrm{num}_V^x\}$$

That is, e_V^x and e_V^y are two vectors parallel to two adjacent edges of the parallelogram.

A parallelogram is called a *rectangle* if $e_V^x e_V^y = 0$. For ease of presentation, we define a matrix, named Δ_V and called the *input matrix*, of size $\mathrm{num}_V^y \times \mathrm{num}_V^x$ as follows:

$$\Delta_V \quad = \quad \begin{bmatrix} 0 & \delta_V^x & \cdots & (\mathrm{num}_V^x - 1)\delta_V^x \\ \delta_V^y & \delta_V^y + \delta_V^x & \cdots & \delta_V^y + (\mathrm{num}_V^x - 1)\delta_V^x \\ & & \cdots & \\ (\mathrm{num}_V^y - 1)\delta_V^y & (\mathrm{num}_V^y - 1)\delta_V^y + \delta_V^x & \cdots & (\mathrm{num}_V^y - 1)\delta_V^y + (\mathrm{num}_V^x - 1)\delta_V^x \end{bmatrix}$$

The (i,j)-th element of the input matrix, denoted $\Delta_V^{i,j}$, is the time difference between the input of $V(\mathrm{O})$ and $V(\mathrm{O} + (i-1)e_V^y + (j-1)e_V^x)$, which is calculated by

$$\Delta_V^{i,j} \quad = \quad \mathsf{input}(V)(\mathrm{O} + (i-1)e_V^y + (j-1)e_V^x) - \mathsf{input}(V)(\mathrm{O}) \quad = \quad (i-1)\delta_V^y + (j-1)\delta_V^x$$

So the communication constraint for V is satisfied iff Δ_V contains no identical elements.

To ensure that all elements of Δ_V are distinct, the necessary and sufficient conditions must include $\delta_V^x \neq 0$ and $\delta_V^y \neq 0$. To derive the remaining conditions, we divide all input matrices into four groups by the case analysis: (1) $\delta_V^x > 0$ and $\delta_V^y > 0$, (2) $\delta_V^x > 0$ and $\delta_V^y < 0$, (3) $\delta_V^x < 0$ and $\delta_V^y > 0$ and (4) $\delta_V^x < 0$ and $\delta_V^y < 0$. We examine four input matrices, one from each group. The analyses for (3) and (4) are similar to (1) and (2), respectively and are therefore omitted.

$$\Delta_V \quad = \quad \begin{bmatrix} 0 & 3 & 6 & 9 & 12 \\ 1 & 4 & 7 & 10 & 13 \\ 2 & 5 & 8 & 11 & 14 \end{bmatrix}$$

(a) $\mathrm{num}_V^x = 5, \mathrm{num}_V^y = 3, \delta_V^x = 3, \delta_V^y = 1$

$$\Delta_V \quad = \quad \begin{bmatrix} 0 & 2 & 4 \\ -4 & -2 & 0 \\ -8 & -6 & -4 \end{bmatrix}$$

(b) $\mathrm{num}_V^x = 3, \mathrm{num}_V^y = 3, \delta_V^x = 2, \delta_V^y = -4$

$$\Delta_V \quad = \quad \begin{bmatrix} 0 & -3 & -6 & -9 & -12 \\ -1 & -4 & -7 & -10 & -13 \\ -2 & -5 & -8 & -11 & -14 \end{bmatrix}$$

(c) $\mathrm{num}_V^x = 5, \mathrm{num}_V^y = 3, \delta_V^x = -3, \delta_V^y = -1$

$$\Delta_V \quad = \quad \begin{bmatrix} 0 & -2 & -4 \\ 4 & 2 & 0 \\ 8 & 6 & 4 \end{bmatrix}$$

(d) $\mathrm{num}_V^x = 3, \mathrm{num}_V^y = 3, \delta_V^x = -2, \delta_V^y = 4$

The communication constraint in (a) is satisfied, since Δ_V has no identical elements. In general, since $\delta_V^x > 0$ and $\delta_V^y > 0$, all elements of the input matrix are distinct iff all elements in its first row and first column are distinct. If we scan the first row left to right and the first column top to bottom, the first pair of identical elements must be at the positions

$$(1, \frac{\delta_V^y}{\gcd(\delta_V^x, \delta_V^y)} + 1) \quad = \quad (1,2) \quad \text{and} \quad (\frac{\delta_V^x}{\gcd(\delta_V^x, \delta_V^y)} + 1, 1) \quad = \quad (4,1)$$

and have the value

$$\Delta_V^{1,1} + \mathrm{sign}(\delta_V^x) \times \frac{\delta_V^x \delta_V^y}{\gcd(\delta_V^x, \delta_V^y)} \quad = \quad 3$$

In general, the input matrix does not contain identical elements iff

$$\frac{\delta_V^y}{\gcd(\delta_V^x,\delta_V^y)} \;\geq\; \mathrm{num}_V^x \;\;\text{or}\;\; \frac{\delta_V^x}{\gcd(\delta_V^x,\delta_V^y)} \;\geq\; \mathrm{num}_V^y$$

The input matrix in (a) has no identical elements, because $3 = \frac{\delta_V^x}{\gcd(\delta_V^x,\delta_V^y)} \geq \mathrm{num}_V^y = 3$ holds.

Next, we consider the input matrix in (b). Since the input matrix contains identical elements 0 and -4, the communication constraint is violated. In general, since $\delta_V^x > 0$ and $\delta_V^y < 0$, all elements of the input matrix are distinct iff all elements in its first row and last column are distinct. If we scan the first row right to left and the last column top to bottom, the first pair of identical elements must be at the positions

$$(1, \mathrm{num}_V^x - \frac{-\delta_V^y}{\gcd(\delta_V^x,\delta_V^y)}) \;=\; (1,1) \;\;\text{and}\;\; (\frac{\delta_V^x}{\gcd(\delta_V^x,\delta_V^y)}+1, \mathrm{num}_V^x) \;=\; (2,3)$$

and have the value

$$\Delta_V^{1,\mathrm{num}_V^x} + \mathrm{sign}(\delta_V^x) \times \frac{\delta_V^x \delta_V^y}{\gcd(\delta_V^x,\delta_V^y)} \;=\; 0$$

In general, the input matrix does not contain identical elements iff

$$-\frac{\delta_V^y}{\gcd(\delta_V^x,\delta_V^y)} \;\geq\; \mathrm{num}_V^x \;\;\text{or}\;\; \frac{\delta_V^x}{\gcd(\delta_V^x,\delta_V^y)} \;\geq\; \mathrm{num}_V^y$$

The input matrix in (b) contains identical elements, because neither $2 = -\frac{\delta_V^y}{\gcd(\delta_V^x,\delta_V^y)} \geq \mathrm{num}_V^x = 3$ nor $1 = \frac{\delta_V^x}{\gcd(\delta_V^x,\delta_V^y)} \geq \mathrm{num}_V^y = 3$ holds.

Theorem. 5 *Let* in_V *be a parallelogram. The communication constraint for* V *is satisfied iff*

$$\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}\right) \geq \gcd(\delta_V^x,\delta_V^y) \tag{2}$$

Proof. From (2) we can deduce that $\delta_V^x \neq 0$ and $\delta_V^y \neq 0$, because $\gcd(0,0) = 1$ and $\mathrm{num}_V^x > 1$ and $\mathrm{num}_V^y > 1$. The proof is conducted by the case analysis: (1) $\delta_V^x \delta_V^y > 0$ and (2) $\delta_V^x \delta_V^y < 0$.

- $\delta_V^x \delta_V^y > 0$. The input matrix Δ_V contains identical elements iff there is an element in the first row that is identical to an element in the first column. If we scan the first row left to right and the first column top to bottom, the first pair of identical elements are located at the positions $(1, \frac{|\delta_V^y|}{\gcd(\delta_V^x,\delta_V^y)}+1)$ and $(\frac{|\delta_V^x|}{\gcd(\delta_V^x,\delta_V^y)}+1, 1)$ and have the value $\Delta_V^{1,1} + \mathrm{sign}(\delta_V^x) \times \frac{\delta_V^x \delta_V^y}{\gcd(\delta_V^x,\delta_V^y)}$. Hence, the input matrix Δ_V does not contain identical elements iff either $\frac{|\delta_V^y|}{\gcd(\delta_V^x,\delta_V^y)} \geq \mathrm{num}_V^x$ or $\frac{|\delta_V^x|}{\gcd(\delta_V^x,\delta_V^y)} \geq \mathrm{num}_V^y$, which is equivalent to $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}\right) \geq \gcd(\delta_V^x,\delta_V^y)$.

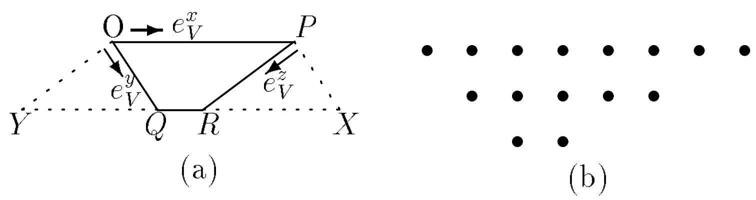

Figure 5: (a) The trapezoidal input space. (b) An instance: $\mathrm{num}_V^x = 8$ and $\mathrm{num}_V^y = \mathrm{num}_V^z = 3$.

- $\delta_V^x \delta_V^y < 0$. The input matrix Δ_V contains identical elements iff there is an element in the first row that is identical to an element in the last column. If we scan the first row right to left and the last column top to bottom, the first pair of identical elements must be located at the positions $(1, \mathrm{num}_V^x - \frac{|\delta_V^y|}{\gcd(\delta_V^x, \delta_V^y)})$ and $(\frac{|\delta_V^x|}{\gcd(\delta_V^x, \delta_V^y)} + 1, \mathrm{num}_V^x)$ and have the value $\Delta_V^{1,\mathrm{num}_V^x} + \mathrm{sign}(\delta_V^x) \times \frac{\delta_V^x \delta_V^y}{\gcd(\delta_V^x, \delta_V^y)}$. Hence, the input matrix Δ_V does not contain identical elements iff either $\frac{|\delta_V^y|}{\gcd(\delta_V^x, \delta_V^y)} \geq \mathrm{num}_V^x$ or $\frac{|\delta_V^x|}{\gcd(\delta_V^x, \delta_V^y)} \geq \mathrm{num}_V^y$, which is equivalent to $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^y)$. □

10.3 Trapezoids

The input space in_V is a *trapezoid* if it satisfies the following properties (Fig. 5):

1. Four index points, denoted O, P, Q and R, are the vertices of the trapezoid. Q and R may be identical.

2. $\overline{\mathrm{O}P}$ is parallel to and contains no less index points than $\overline{QR}$.

3. Let $\mathrm{num}_V^x > 1$, $\mathrm{num}_V^y > 1$ and $\mathrm{num}_V^z > 1$ be the number of index points on $\overline{\mathrm{O}P}$, $\overline{\mathrm{O}Q}$ and $\overline{PR}$, respectively. Then $\mathrm{num}_V^y = \mathrm{num}_V^z$.

4. Let $e_V^x = (P - \mathrm{O})/(\mathrm{num}_V^x - 1)$, $e_V^y = (Q - \mathrm{O})/(\mathrm{num}_V^y - 1)$ and $e_V^z = (R - P)/(\mathrm{num}_V^z - 1)$. (By using the conditions given so far, $e_V^y - e_V^z$ and e_V^x are co-linear.) Then

$$\mathrm{in}_V \quad = \quad \{\mathrm{O} + (i-1)e_V^y + (j-1)e_V^x \mid 0 < i \leq \mathrm{num}_V^y \wedge 0 < j \leq \mathrm{num}_V^x - (i-1)\frac{e_V^y - e_V^z}{e_V^x}\}$$

A trapezoid is a *triangle* if $Q = R$ and a parallelogram if $e_V^y = e_V^z$.

By definition, if the input space in_V is a trapezoid, $e_V^y - e_V^z$ must be an *integral* positive multiple of e_V^x. This multiple is the difference of the index point counts on any two adjacent line segments in the trapezoid that contain index points and are parallel to $\overline{\mathrm{O}P}$.

Let X be the point such that $\overline{\mathrm{O}P}$ and $\overline{QX}$ are parallel and $\overline{\mathrm{O}Q}$ and $\overline{PX}$ are parallel. Let Y be the point such that $\overline{\mathrm{O}P}$ and $\overline{YR}$ are parallel and $\overline{\mathrm{O}Y}$ and $\overline{PR}$ are parallel. Then, the trapezoidal input space in_V is the intersection of the parallelogram $OPXQ$ and the parallelogram $OPRY$. Therefore, the input matrix Δ_V for the trapezoidal input space in_V can be defined as either of the following:

(P1) Δ_V is the input matrix for the parallelogram $OPXQ$ with the last $\text{num}_V^x - (i-1)\frac{e_V^y - e_V^z}{e_V^x}$ elements in the i-th row being removed.

(P2) Δ_V is the input matrix for the parallelogram $OPRY$ with the first $\text{num}_V^x - (i-1)\frac{e_V^y - e_V^z}{e_V^x}$ elements in the i-th row are removed.

As is for the parallelepiped input space, the necessary and sufficient conditions must include $\delta_V^x \neq 0$, $\delta_V^y \neq 0$ and $\delta_V^z \neq 0$ if all elements of the input matrix are distinct. To derive the remaining conditions, we distinguish the two cases: (1) $\delta_V^x \delta_V^y > 0$ and (2) $\delta_V^x \delta_V^y < 0$. In the first case, the input matrix is regarded as being defined in (P1). The transformation of the communication constraint is carried out as if the input space is the parallelogram $OPXQ$. That is, the communication constraint is satisfied iff all elements of the first row and first column of the input matrix are distinct, i.e., iff

$$\max\left(\frac{|\delta_V^x|}{\text{num}_V^y}, \frac{|\delta_V^y|}{\text{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^y) \tag{3}$$

In the second case, the input matrix is regarded as being defined in (P2). The transformation of the communication constraint is carried out as if the input space is the parallelogram $OPRY$. That is, the communication constraint is satisfied iff all elements of the first row and last column of the input matrix are distinct, i.e., iff

$$\max\left(\frac{|\delta_V^x|}{\text{num}_V^z}, \frac{|\delta_V^z|}{\text{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^z) \tag{4}$$

The following four input matrices are those in Sect. 10.2 adapted for the trapezoidal input space. (Due to the space limit, num_V^x, num_V^x and num_V^z are not shown.) Neither of them contains identical elements. Hence, the communication constraint is satisfied in each case. The conclusion can be reached by simple algebraic calculations following (3) and (4).

$$\Delta_V = \begin{bmatrix} 0 & 3 & 6 & 9 & 12 \\ 1 & 4 & 7 & 10 & \\ 2 & 5 & 8 & & \end{bmatrix}$$

(a) $\delta_V^x = 3, \delta_V^y = 1, \delta_V^z = -2$

$$\Delta_V = \begin{bmatrix} 0 & 2 & 4 \\ & -4 & -2 \\ & & -8 \end{bmatrix}$$

(b) $\delta_V^x = 2, \delta_V^y = -4, \delta_V^z = -6$

$$\Delta_V = \begin{bmatrix} 0 & -3 & -6 & -9 & -12 \\ -1 & -4 & -7 & -10 & \\ -2 & -5 & -8 & & \end{bmatrix}$$

(c) $\delta_V^x = -3, \delta_V^y = -1, \delta_V^z = 2$

$$\Delta_V = \begin{bmatrix} 0 & -2 & -4 \\ & 4 & 2 \\ & & 8 \end{bmatrix}$$

(d) $\delta_V^x = -2, \delta_V^y = 4, \delta_V^z = 6$

Theorem. 6 *Let* in_V *be a trapezoid. The communication constraint for* V *is satisfied iff*

$$\begin{aligned} \delta_V^x \delta_V^y > 0 &\implies \max\left(\frac{|\delta_V^x|}{\text{num}_V^y}, \frac{|\delta_V^y|}{\text{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^y) \\ \text{and} \quad \delta_V^x \delta_V^y < 0 &\implies \max\left(\frac{|\delta_V^x|}{\text{num}_V^z}, \frac{|\delta_V^z|}{\text{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^z) \end{aligned} \tag{5}$$

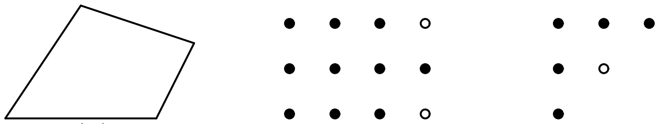

Figure 6: (a) The arbitrary two-dimensional input space. (b) Two instances depicted by solid dots.

Proof. From (5) we can deduce that $\delta_V^x \neq 0$, $\delta_V^y \neq 0$ and $\delta_V^z \neq 0$, because $\gcd(0,0) = 1$, $\mathrm{num}_V^x > 1$, $\mathrm{num}_V^y > 1$ and $\mathrm{num}_V^z > 1$. The proof is conducted by the case analysis (Fig. 5): (1) $\delta_V^x \delta_V^y > 0$ and (2) $\delta_V^x \delta_V^y < 0$. When $\delta_V^x \delta_V^y > 0$, the communication constraint for the trapezoidal input space is the same as for the parallelepiped input space $OPXQ$. When $\delta_V^x \delta_V^y < 0$, the communication constraint for the trapezoidal input space is the same as for the parallelepiped input space $OPRY$. The rest of the proof follows from the proof for the parallelepiped input space (Thm. 5). □

The communication constraint can be put in a simpler form (in the sense that δ_V^z is eliminated) if the trapezoidal input space satisfies the condition $e_V^x + e_V^z = e_V^y$.

Theorem. 7 *Let* in_V *is a trapezoid such that* $e_V^x + e_V^z = e_V^y$. *The communication constraint for* V *is satisfied iff*

$$\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}, \frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x}\right) \geq \gcd(\delta_V^x, \delta_V^y) \tag{6}$$

Proof. From (6) we can deduce that $\delta_V^x \neq 0$, $\delta_V^y \neq 0$ (and $\delta_V^x \neq \delta_V^y$), because $\gcd(0,0) = 1$, $\mathrm{num}_V^x > 1$ and $\mathrm{num}_V^y > 1$. The proof is conducted by the case analysis: (1) $\delta_V^x \delta_V^y > 0$ and (2) $\delta_V^x \delta_V^y < 0$.

- $\delta_V^x \delta_V^y > 0$. In this case, $\frac{|\delta_V^y|}{\mathrm{num}_V^x} > \frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x}$. This implies that $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}, \frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x}\right) = \max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}\right)$. Hence, $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x},\right) \geq \gcd(\delta_V^x, \delta_V^y)$.

- $\delta_V^x \delta_V^y < 0$. In this case, $\frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x} > \frac{|\delta_V^y|}{\mathrm{num}_V^x}$. This implies that $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y|}{\mathrm{num}_V^x}, \frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x}\right) = \max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^y}, \frac{|\delta_V^y - \delta_V^x|}{\mathrm{num}_V^x}\right)$. We are given that $e_V^x + e_V^z = e_V^y$, which implies that $\delta_V^x + \delta_V^z = \delta_V^y$. By further using the facts that $\gcd(\delta_V^x, \delta_V^y) = \gcd(\delta_V^x, \delta_V^y - \delta_V^x)$ and that $\mathrm{num}_V^y = \mathrm{num}_V^z$, we obtain $\max\left(\frac{|\delta_V^x|}{\mathrm{num}_V^z}, \frac{|\delta_V^z|}{\mathrm{num}_V^x},\right) \geq \gcd(\delta_V^x, \delta_V^z)$.

The proof is completed by combining the two cases and applying Thm. 6. □

10.4 Two-Dimensional Sets

The input space in_V is a two-dimensional set (Fig. 6). If it is a parallelogram or a trapezoid, the respective method in Sect. 10.2 or 10.3 applies. Otherwise, we first extend the input space

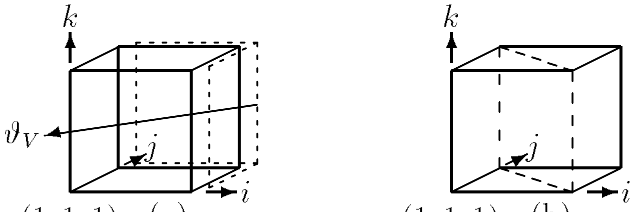

Figure 7: (a) The arbitrary three-dimensional input space. The index space is the cube. The input space in_V is the union of the two dashed rectangles. The arrow shows the direction of the dependence vector ϑ_V. (b) The projected index space is the the long-dashed rectangle.

by adding points to the input space in such a way that the *extended input space* becomes either a parallelogram or a trapezoid. We then apply the respective method in Sect. 10.2 or 10.3. An enclosing parallelogram of a convex polyhedron can be calculated using, for example, Feautrier's PIP method [37]. Refer to Fig. 6. By adding the points represented by circles, the two input spaces become a rectangle and a triangle, respectively.

The communication constraint stated over the extended input space may be stronger than the original communication constraint. As a result, the space of correct space-time mappings may be reduced. There is a danger that some optimal mappings (in a certain sense) are excluded from the search space. We offer two general comments here. Firstly, the reduced search space may still contain all optimal mappings. Take any of the two input spaces in Fig. 6(b) for example. If the optimal mappings satisfy the conditions $\delta_V^x > 0$ and $\delta_V^y > 0$, then all the optimal mappings are contained in the reduced search space. This is because the communication constraints for the original and extended input spaces are equivalent. Secondly, the input spaces of arbitrary shapes are inherently difficult to deal with. The problem of finding optimal mappings has to be studied in a case-by-case basis.

10.5 Three-Dimensional Sets

The input space in_V is a three-dimensional set. We proceed as follows:

1. Project the input space along ϑ_V onto an arbitrarily chosen hyperplane. The projected input space is two-dimensional.

2. Scale the projected input space such that the *scaled input space* is integral.

3. Extend the projected input space such that the *extended input space* is either a parallelogram or a trapezoid, following Sect. 10.4.

4. Derive the closed-form communication constraint for the extended index space, following Sect. 10.2 or 10.3.

It makes no difference onto which hyperplane an input space is projected.

Theorem. 8 *The communication constraint for the input space of V is equivalent to the communication constraint for the scaled input space of V.*

Proof. We prove this theorem in a more general setting by assuming that the input and output spaces of variable V are $\mathbb{Z}^n$. Let I and J be two distinct points in the input space in_V. Let I' and J' be their projected points in the projected input space. Let α be the scaling factor in the generation of the scaled input space. The scaled points of I' and J' are $\alpha I'$ and $\alpha J'$. The proof is to show that $\mathsf{input}(V)(I)$ and $\mathsf{input}(V)(J)$ are distinct iff $\mathsf{input}(V)(\alpha I')$ and $\mathsf{input}(V)(\alpha J')$ are distinct. I' and J' are the projections of I and J along ϑ_V. We have $I \overset{\vartheta_V}{\longleftrightarrow} I'$ and $J \overset{\vartheta_V}{\longleftrightarrow} J'$. By Lemma 2, we conclude that $\mathsf{input}(V)(I) = \mathsf{input}(V)(I')$ and $\mathsf{input}(V)(J) = \mathsf{input}(V)(J')$. The input time difference between $V(I')$ and $V(J')$ is $\mathsf{input}(V)(I') - \mathsf{input}(V)(J')$, while the input time difference between $V(\alpha I')$ and $V(\alpha J')$ is $\mathsf{input}(V)(\alpha I') - \mathsf{input}(V)(\alpha J')$, which simplifies to $\alpha(\mathsf{input}(V)(I') - \mathsf{input}(V)(J'))$. Hence the former differs iff the latter does. Hence, $\mathsf{input}(V)(I)$ and $\mathsf{input}(V)(J)$ are distinct iff $\mathsf{input}(V)(\alpha I')$ and $\mathsf{input}(V)(\alpha J')$ are distinct. □

Consider an example input space depicted in Fig. 7(a). Assume that the index space is a cube of size m: $\Phi = \{(i,j,k) \mid 0 < i,j,k \le m\}$. The input space in_V is the union of the two dashed rectangles: $\text{in}_V = \{(i{+}1, m{+}1, k) \mid 0 < i,k \le m\} \cup \{(m{+}1, j{+}1, k) \mid 0 < j,k \le m\}$. Assume that $\vartheta_V = (-1,-1,0)$. Choosing the projection hyperplane $\{(i,j,k) \mid i{+}j = m{+}1\}$, we obtain the projected input space: $\{(i/2, m - i/2 + 1, k) \mid 2 \le i \le 2m \wedge 0 < k \le m\}$, which is depicted in Fig. 7(b) as the long-dashed rectangle. The projected input space contains non-integral points. Scaling it by a factor of two gives rise to $\{(i, 2m - i + 2, 2k) \mid 2 \le i \le 2m \wedge 0 < k \le m\}$. The scaled input space is a rectangle. The method for the parallelepiped input space applies.

11 Finding Optimal and Conflict-Free Mappings

Let us summarise the process for the derivation of closed-form computation and communication constraints. Let $IOspace$ be the set of all I/O spaces.

1. Transform all spaces in $IOspace$ into parallelograms or trapezoids (Sects. 10.4 and 10.5).

2. Transform the computation constraint (Sect. 7). Eliminate the computation constraint if possible, or replace it by the communication constraint for an imaginary variable Γ by adding an enclosing parallelogram or trapezoid of in_Γ to $IOspace$.

3. Eliminate all redundant spaces in $IOspace$ (Thms. 2 and 3).

4. Transform the communication constraint. Derive closed-form communication constraints for the spaces in $IOspace$ (Thms. 4 – 7).

To find optimal and conflict-free mappings for linear processor arrays, we solve the following optimisation problem:

Minimise	$\text{cost}(T)$
Subject to	The precedence constraint
	The delay constraint
	The transformed computation constraint
	The transformed communication constraint

where $\text{cost}(T)$ denotes some cost function for a space-time mapping. The cost function can take many forms. We may choose the cost function as a weighted combination of several objectives such as the minimisation of the latency and the processor count. We may also choose the cost function as a hierarchy of objectives to be satisfied in sequence. That is, minimising the latency is the primitive objective, while minimising the processor count is the secondary objective, and so on.

The constraints of this optimisation problem depend on the space-time mapping and the problem size parameters only. Their verification takes constant time that is independent of the problem size. As a result, it takes a constant time to check the correctness of a given space-time mapping. On the contrary, the computation and communication constraints in Def. 1 do not have closed-form expressions. The time spent on their verification depends on the problem size. Clearly, with the closed-form constraints presented in this paper, the time needed for finding optimal mappings is significantly reduced. Therefore, problems of larger problem sizes can be solved in practice.

Since the close-form communication constraints are non-linear, this optimisation problem cannot be solved by integer programming technique, in general. An efficient enumeration method was proposed elsewhere [28]. It was reported that optimal arrays for any objective functions expressed in terms of the problem size and some design parameters can be found over a polynomial search space.

Note that both the delay and communication constraints apply for moving variables only. Before applying a synthesis procedure to an algorithm, we must declare explicitly which variables are moving and which variables are stationary. For the moving variables, the corresponding delay and communication constraints are used. For the stationary variables V, the constraints $\mathsf{flow}(V) = 0$, i.e., $\sigma\vartheta_V = 0$ are used. To find optimal mappings for an algorithm, the synthesis procedure simply tries all possible combinations in a systematic manner. This approach is feasible since a systolic algorithm usually contains a few variables.

12 Two Examples

In this section, we show how to derive closed-form computation and communication constraints for two examples: matrix product and LU decomposition. The derivation of closed-form communication constraints is conducted for all variables. If a variable is chosen to be stationary, then no communication constraint needs to be enforced for that variable.

12.1 Matrix Product

The specification of matrix product was given in Sect. 2. Since $\Phi \xrightarrow{\vartheta_V} \text{in}_V$ ($V \in \{A, B, C\}$), the computation constraint can be eliminated (Thm. 1). Since $\text{in}_C \xmapsto{\vartheta_C} \text{out}_C$ and $\text{out}_C \xmapsto{\vartheta_C} \text{in}_C$, one of the communication constraints for variable C can be eliminated (Thm. 2). Let us disregard the communication constraint for the output space out_C.

The input spaces in_A, in_B and in_C are rectangles: $e^x_A = (1, 0, 0)$, $e^y_A = (0, 0, 1)$ and $\text{num}^x_A = \text{num}^y_A = m$; $e^x_B = (0, 1, 0)$, $e^y_B = (0, 0, 1)$ and $\text{num}^x_B = \text{num}^y_B = m$; $e^x_C = (1, 0, 0)$, $e^y_C = (0, 1, 0)$ and

$\mathrm{num}_C^x = \mathrm{num}_C^y = m$. We obtain

$$\begin{array}{lll} \delta_A^x = \lambda_1 - \sigma_1 \frac{\lambda_2}{\sigma_2} & \delta_B^x = \lambda_2 - \sigma_2 \frac{\lambda_1}{\sigma_1} & \delta_C^x = \lambda_1 - \sigma_1 \frac{\lambda_3}{\sigma_3} \\ \delta_A^y = \lambda_3 - \sigma_3 \frac{\lambda_2}{\sigma_2} & \delta_B^y = \lambda_3 - \sigma_3 \frac{\lambda_1}{\sigma_1} & \delta_C^y = \lambda_2 - \sigma_2 \frac{\lambda_3}{\sigma_3} \end{array}$$

The derivation of closed-form communication constraints for the input spaces in_A, in_B and in_C follows from Thm. 5. The problem of finding optimal and conflict-free mappings can be formulated as follows ($m > 1$):

$$\begin{array}{rll} \textbf{Minimise} & \mathrm{cost}(T) & \\ \textbf{Subject to} & \lambda_1 > 0,\ \lambda_2 > 0,\ \lambda_3 > 0 & \text{(Precedence Constraint)} \\ & \sigma_1 | \lambda_1,\ \sigma_2 | \lambda_2,\ \sigma_3 | \lambda_3 & \text{(Delay Constraint)} \\ & \max(|\delta_A^x|, |\delta_A^y|) \geq m \gcd(\delta_A^x, \delta_A^y) & \\ & \max(|\delta_B^x|, |\delta_B^y|) \geq m \gcd(\delta_B^x, \delta_B^y) & \\ & \max(|\delta_C^x|, |\delta_C^y|) \geq m \gcd(\delta_C^x, \delta_C^y) & \text{(Communication Constraint)} \end{array}$$

12.2 LU Decomposition

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

Specification: $(\forall\ i,j : 0 < i,j \leq m : (\sum\ k : 0 < k \leq m : \ell_{i,k} u_{k,j} = a_{i,j}))$

$$\begin{array}{rl} \textbf{INPUT}: & \mathrm{in}_A = \{(i,j,0) \mid 0 < i,j \leq m\} \\ \textbf{OUTPUT}: & \mathrm{out}_L = \{(i,m,k) \mid k < i \leq m \wedge 0 < k \leq m\} \\ & \mathrm{out}_U = \{(m,j,k) \mid k \leq j \leq m \wedge 0 < k \leq m\} \\ \textbf{LOOPS}: & \textbf{for } i \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & \textbf{for } j \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & \textbf{for } k \textbf{ from } 0 \textbf{ to } m \textbf{ do} \\ & L(i,j,k) = \left\{ \begin{array}{l} \textbf{if } k < i \leq m \wedge k = j \wedge 0 < k \leq m \rightarrow A(i,j,k-1) U(i-1,j,k) \\ [\,]\ k < i \leq m \wedge k < j \leq m \wedge 0 < k \leq m \rightarrow L(i,j-1,k) \\ \textbf{fi} \end{array} \right. \\ & U(i,j,k) = \left\{ \begin{array}{l} \textbf{if } k = i \wedge k < j \leq m \wedge 0 < k \leq m \rightarrow A(i,j,k-1) \\ [\,]\ k = i \wedge k = j \wedge 0 < k \leq m \rightarrow A(i,j,k-1)^{-1} \\ [\,]\ k < i \leq m \wedge k \leq j \leq m \wedge 0 < k \leq m \rightarrow U(i-1,j,k) \\ \textbf{fi} \end{array} \right. \\ & A(i,j,k) = A(i,j,k-1) - L(i-1,j,k) U(i,j-1,k) \end{array}$$

where $A(i,j,0) = a_{i,j}$, $L(i,m,k) = \ell_{i,k}$ and $U(m,j,k) = u_{k,j}$.

The corresponding dependence graph is depicted in Fig. 8. The index space is:

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

The dependence vectors associated with variables A, L and U are:

$$\vartheta_A = (0,0,1), \quad \vartheta_L = (0,1,0), \quad \vartheta_U = (1,0,0)$$

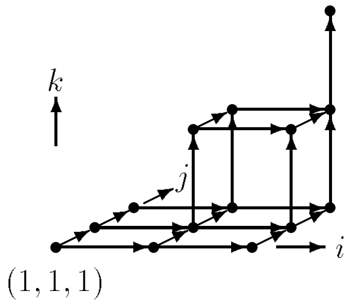

Figure 8: The dependence graph for LU decomposition ($m=3$).

Since $\Phi \xrightarrow{\vartheta_A} \mathrm{in}_A$, the computation constraint can be eliminated (Thm. 1). The input space in_A is a rectangle: $e^x_A=(1,0,0)$, $e^y_A=(0,1,0)$ and $\mathrm{num}^x_A=\mathrm{num}^y_A=m$. The output spaces out_L and out_U are triangles: $e^x_L=(-1,0,0)$, $e^y_L=(0,0,1)$, $e^z_L=(1,0,1)$ and $\mathrm{num}^x_L=\mathrm{num}^y_L=\mathrm{num}^z_L=m-1$; $e^x_U=(0,-1,0)$, $e^y_U=(0,0,1)$, $e^z_U=(0,1,1)$ and $\mathrm{num}^x_U=\mathrm{num}^y_U=\mathrm{num}^z_U=m$. We obtain

$$\begin{array}{lll}
\delta^x_A = \lambda_1-\sigma_1\frac{\lambda_3}{\sigma_3} & \delta^x_L = \sigma_1\frac{\lambda_2}{\sigma_2}-\lambda_1 & \delta^x_U = \sigma_2\frac{\lambda_1}{\sigma_1}-\lambda_2 \\
\delta^y_A = \lambda_2-\sigma_2\frac{\lambda_3}{\sigma_3} & \delta^y_L = \lambda_3-\sigma_3\frac{\lambda_2}{\sigma_2} & \delta^y_U = \lambda_3-\sigma_3\frac{\lambda_1}{\sigma_1} \\
 & \delta^z_L = (\lambda_1+\lambda_3)-(\sigma_1+\sigma_3)\frac{\lambda_2}{\sigma_2} & \delta^z_U = (\lambda_2+\lambda_3)-(\sigma_2+\sigma_3)\frac{\lambda_1}{\sigma_1}
\end{array}$$

The derivation of the closed-form communication constraint for the input space in_A follows from Thm. 5. Since $e^x_L+e^z_L=e^y_L$ and $e^x_U+e^z_U=e^y_U$, we obtain the closed-form communication constraints for out_L and out_U by Thm. 7. Therefore, δ^z_L and δ^z_U are not used. The problem of finding optimal and conflict-free mappings can be formulated as follows ($m>1$):

$$\begin{array}{lll}
\textbf{Minimise} & \mathrm{cost}(T) & \\
\textbf{Subject to} & \lambda_1>0,\ \lambda_2>0,\ \lambda_3>0 & \text{(Precedence Constraint)} \\
 & \sigma_1|\lambda_1,\ \sigma_2|\lambda_2,\ \sigma_3|\lambda_3 & \text{(Delay Constraint)} \\
 & \max(|\delta^x_A|,|\delta^y_A|)\geq m\gcd(\delta^x_A,\delta^y_A) & \\
 & \max(|\delta^x_L|,|\delta^y_L|,|\delta^x_L-\delta^y_L|)\geq (m-1)\gcd(\delta^x_L,\delta^y_L) & \\
 & \max(|\delta^x_U|,|\delta^y_U|,|\delta^x_U-\delta^y_U|)\geq m\gcd(\delta^x_U,\delta^y_U) & \text{(Communication Constraint)}
\end{array}$$

13 Conclusions

In this paper, we presented a constructive method for finding necessary and sufficient closed-form computation and computation constraints for mapping three-dimensional algorithms to linear processor arrays. We provided conditions to check whether the computation constraint is implied by the communication constraint. If the computation constraint is not redundant, we proposed to replace the computation constraint by the communication constraint for an imaginary variable. To derive closed-form communication constraints, we distinguish I/O spaces by their shapes and dimensions. We presented closed-form communication constraints for the I/O spaces that are lines, parallelograms and trapezoids. For arbitrary two- and three-dimensional I/O spaces, we presented methods that first transform them into parallelograms or trapezoids and then deliver closed-form communication constraints for the transformed I/O spaces.

The results presented in Sects. 6, 7 and 8 are correct for n-dimensional algorithms. One result that needs to be emphasised here is Thm. 1. This theorem provides the sufficient condition for the elimination of the computation constraint. In addition, it also provides the basis for dealing with the computation constraint in the general case where n-dimensional algorithms are mapped to r-dimensional arrays $(0 < r < n-1)$. If the sufficient condition is not met, we simply transform the computation constraint into a communication constraint.

The derivation of closed-form communication constraints depends on the shapes and dimensions of the I/O spaces. The method as presented here only works for three-dimensional algorithms. For two-dimensional algorithms, the communication constraint is not needed. In this case, the algorithm is two-dimensional and the target array is one-dimensional. The method for the synthesis of (n–1)-dimensional arrays applies. Finding necessary and sufficient closed-form communication constraints for mapping n-dimensional to r-dimensional arrays in the general case is still open $(0 < r < n-1)$; this paper provided a solution to the special case where $n = 3$ and $r = 1$. As pointed out in [24], many bit-level algorithms are four- or five-dimensional, such as matrix product, convolution, LU decomposition, etc. Finding a solution to the open problem is of practical significance.

In our methodology, we assume that as many data links are available as required. We dedicate as many data links to a variable as there are dependence vectors associated with that variable (Sect. 4.2). Essentially, every dependence vector of a variable is mapped to a distinct data link. Our methodology can be generalised straightforwardly in the case where more than one link, say K links, are dedicated to the same dependence vector. Several such arrays for matrix product were presented in [27]. To ensure that the mapping is link conflict-free, we partition the I/O spaces of the variable associated with that dependence vector into K disjoint subspaces. We then enforce the communication constraint for each of these subspaces. The K links carry data elements from among these K disjoint subspaces, respectively. Clearly, the more data links there are for a dependence vector, the lower the array's latency will potentially be.

We mentioned in Sect. 11 that the problem of finding optimal arrays as formulated in this paper can be solved efficiently by a systematic enumeration of a polynomial search space. The efficiency of the enumeration is attributed to the fact that the verification of closed-form computation and communication constraints can be done in constant time that is independent of the problem size. On the contrary, the time spent on verifying the computation and communication constraints as proposed in Def. 1 depends on the problem size. Hence, closed-form constraints are desirable if we want to find optimal arrays in a reasonable time complexity. Sometimes, the user may add more design constraints to search for processor arrays with certain predefined characteristics. Below we mention two such design constraints. It is interesting to investigate how closed-form conditions for enforcing these two constraints can be derived.

Modular Array Constraint: A linear array is *modular* if the number of buffers on channels and the sizes of local processor memories are independent of the problem size

[27, 38, 39]. The modular array constraint for variable V can take the following form:

$$\begin{array}{lll} \textbf{if} & \mathsf{flow}(V)=0 & \rightarrow \quad (\forall\, p : p_{\min} \leq p \leq p_{\max} : |\{I \mid I \in \Phi \wedge \sigma I = p\}| \leq \mathrm{mem}_V) \\ [] & \mathsf{flow}(V) \neq 0 & \rightarrow \quad lb_V \leq \mathrm{buf}(V) \leq ub_V \\ \textbf{fi} & & \end{array}$$

where mem_V, lb_V and ub_V are constants independent of the problem size. If V is stationary, mem_V specifies the size of local processor memories for variable V. If V is moving, lb_V and ub_V specify the range on the number of buffers for a channel associated with V. The constraint for moving variables is already in the closed-form. But the constraint for stationary variables is not.

Control-Free Constraint: A linear array is *control-free* if its execution does not require control signals. For example, not every linear array for matrix product is control-free. A linear array is control-free iff whenever elements $a_{i1,k1}$, $b_{k2,j2}$ and $c_{i3,j3}$ meet at a processor, $i1=i3 \wedge j2=j3 \wedge k1=k2$ holds. In this case, 0 is injected on links A and B whenever no elements of A and B are injected. Otherwise, control signals are needed to tell the processor to propagate rather than accumulate the c_{i_3,j_3}. The derivation of a closed-form constraint on control-free mappings is an interesting problem.

Acknowledgements

Thanks to the referees for their comments and corrections and to Professor J. A. B. Fortes of Purdue University for many helpful discussions on the subject of this paper.

References

[1] H. T. Kung and C. E. Leiserson. Algorithms for VLSI processor arrays. In C. Mead and L. Conway, editors, *Introduction to VLSI Systems*, chapter 8.3. Addison-Wesley, 1980.

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

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

[4] B. Lisper. *Synthesizing Synchronous Systems by Static Scheduling in Space-Time.* Lecture Notes in Computer Science 362. Springer-Verlag, 1989.

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

[6] 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.

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

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

[9] 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.

[10] W. Shang and J. A. B. Fortes. On the optimality of linear schedules. *J. VLSI Signal Processing*, 1(2):209–220, Oct. 1989.

[11] W. Shang and W. A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependences. *IEEE Trans. on Computers*, C-40(6):723–742, June 1991.

[12] Y. Wong and J. M. Delosme. Optimization of computation time for systolic arrays. Technical Report YALEU/DCS/RR-651, Department of Computer Science, Yale University, May 1989.

[13] Y. Wong and J. M. Delosme. Optimization of processor count for systolic arrays. Technical Report YALEU/DCS/RR-697, Department of Computer Science, Yale University, May 1989.

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

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

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

[17] J. Teigh and L. Thiele. Partitioning of processor arrays: a piecewise regular approach. *Integration*, 14(3):297–332, Feb. 1993.

[18] X. Zhong and S. Rajopadhye. Deriving fully efficient systolic arrays by quasi-linear allocation functions. In E. H. L. Aarts, J. van Leeuwen, and M. Rem, editors, *Parallel Architectures and Languages Europe (PARLE '91), Vol. I: Parallel Architectures and Algorithms*, Lecture Notes in Computer Science 505, pages 219–235. Springer-Verlag, 1991.

[19] A. Benaini and Y. Robert. Spacetime-minimal systolic architectures for Gaussian elimination and the algebraic path problem. *Parallel Computing*, 15(1):211–226, 1990.

[20] P. R. Cappello. A processor-time-minimal systolic array for cubical mesh algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 3(1):4–13, Jan. 1992.

[21] P. Clauss, C. Mongenet, and G. R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. In S. Y. Kung and E. E. Swartzlander, editors, *Application Specific Array Processors*, pages 5–18. IEEE Computer Society Press, 1990.

[22] J. Teich and L. Thiele. Control generation in the design of processor arrays. *J. VLSI Signal Processing*, 3(1/2):77–92, June 1991.

[23] J. Xue and C. Lengauer. The synthesis of control signals for one-dimensional systolic arrays. *Integration*, 14(1):1–32, Nov. 1992.

[24] W. Shang and W. A. B. Fortes. On time mapping of uniform dependence algorithms into lower dimensional processor arrays. *IEEE Trans. on Parallel and Distributed Systems*, 3(3):350–363, May 1992.

[25] D. I. Moldovan. ADVIS: A software package for the design of systolic arrays. *IEEE Trans. on Computer-Aided Design*, CAD-6(1):33–40, Jan. 1987.

[26] P. Lee and Z. Kedem. Mapping nested loop algorithms into multidimensional systolic arrays. *IEEE Trans. on Parallel and Distributed Systems*, 1:64–76, Jan. 1990.

[27] A. Benaini and M.Tchuente. Matrix product on modular linear systolic array. In M. Cosnard et al., editor, *Parallel & Distributed Algorithms*, pages 79–88. North-Holland, 1989.

[28] K. Ganapathy and B. W. Wah. Synthesizing optimal lower dimensional processor arrays. In *Int. Conf. on Parallel Processing*, pages 96–103, Aug. 1992.

[29] 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.

[30] I. Ramakrishnan, D. Fussell, and A. Silberschatz. Mapping homogeneous graphs on linear arrays. *IEEE Trans. on Computers*, C-35(3):189–209, Mar. 1986.

[31] V.P. Roychowdhury and T. Kailath. Subspace scheduling and parallel implementation of non-systolic regular iterative algorithms. *J. VLSI Signal Processing*, 1(2):127–142, Oct. 1989.

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

[33] J. Xue. On the loading, recovery and access of stationary data in systolic arrays. In L. Bouge, M. Cosnard, Y. Robert, and D. Trystram, editors, *Parallel Processing: CONPAR92–VAPP V*, Lecture Notes in Computer Science 634, pages 259–264. Elsevier (North-Holland), Sept. 1992.

[34] E. W. Dijkstra. *Predicate Calculus and Programming Semantics*. Series in Automatic Computation. Prentice-Hall, 1990.

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

[36] A. Schrijver. *Theory of Linear and Integer Programming.* Series in Discrete Mathematics. John Wiley & Sons, 1986.

[37] P. Feautrier. Parametric integer programming. *Operations Research*, 22:243–268, 1988.

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

[39] J.F. Myoupo. Synthesizing linear systolic arrays for dynamic programming problems. *Parallel Processing Letters*, 2(1):97–110, 1992.