Mapping Nested Loop Algorithms into Grid-Connected Systolic Arrays without Data Collisions in the Data Links[1]

PeiZong Lee

Institute of Information Science, Academia Sinica
Nankang, Taipei, Taiwan, R.O.C.

Tel: +886 (2) 788-3799, Fax: +886 (2) 782-4814
Internet: leepe@iis.sinica.edu.tw

Abstract

Systolic arrays, which are made out of simple processing elements connected by data links, have made significant improvements in speeding up computation in comparison to conventional computers. Although systolic arrays belong to distributed memory parallel devices, they adopt systolic communications instead of message passing communications for sending data between neighboring processing elements. Therefore, it is important to provide necessary and sufficient conditions to avoid data collisions in the data links for a correct algorithm design. In this paper, we present necessary and sufficient conditions for mapping the class of shift-invariant uniform-dependence algorithms structured as nested loops into grid-connected systolic arrays of arbitrary dimensions. The proposed conditions, which are based on the *ZERO-ONE-INFINITE* property of tokens' behavior that describes how many times tokens are used and generated during the computation, can allow us to generate all feasible solutions.

Index Terms— algorithm transformation, data collision, data link, grid-connected systolic array, nested-loop algorithm.

[1]This work was partially supported by the NSC under Grants NSC 80-0408-E-001-02, NSC 81-0408-E-001-505, and 82-0408-E-001-016.

1 Introduction

It is significant that many computation-intensive scientific algorithms are formulated as nested loops, which are inherently regularly structured. In this paper we study the mapping of shift-invariant uniform-dependence algorithms structured as nested loops into grid-connected systolic arrays without data collisions in the data links. Although previously there has been a significant amount of research studying the transformations of nested-loop algorithms into systolic arrays [3] [4] [8] [15]–[23] [26]–[28], the proposed methods either are not general enough for solving the mapping of nested-loop algorithms into arbitrary dimensional systolic arrays or do not consider the possibility of data collisions in data links. A complete survey of the systolic array transformations can be seen in [5] [13] [14].

In general, the transformations presented in these papers are all based on the hyperplane method [11]. This technique finds the data-dependence vectors of an algorithm first, and then finds a non-singular linear mapping which preserves the data-dependence ordering of the original algorithm. This technique is also called space-time mapping. However, previous attempts did not consider the cases when data might collide in the data links.

Unlike the above techniques, we have introduced a set of necessary and sufficient conditions for correctly transforming a p-nested-loop algorithm into a q-D systolic array, where D stands for dimensional and $1 \leq q < p$ [14]. However, in our previous model, a data link, in which a set of data tokens with a specific data-dependence vector flow through, has only one direction. Therefore, if a data token wants to flow to the diagonal PE (for example, the north-eastern processing element), there must be a data link connecting the current PE and the diagonal PE. This design will of course increase the complexity of the hardware and the fabrication cost of the target systolic array, because additional data links and their corresponding I/O pins are very expensive to be included in a chip with a limited area. Similarly, in Ganapathy and Wah's model, a data link also has only one direction [6], and they incur the same problem.

It is our goal in this paper to present a complete set of necessary and sufficient conditions for mapping nested-loop algorithms with uniform-dependence relations into grid-connected systolic arrays of arbitrary dimensions, including 1-D linear systolic arrays or 2-D mesh-connected systolic arrays or 3-D cube-connected systolic arrays or high dimensional ones. It is of independent interest to develop such necessary and sufficient conditions for a correct mapping. Because the topology of a grid-connected systolic array is regular, in addition, each PE of a q-D grid-connected systolic array has only $2q$ data links in comparison with $3^q - 1$ data links by an arbitrary q-D systolic array (in which each PE additionally has data links connecting to all diagonal PEs). This architecture is thus especially suitable for the VLSI (Very Large Scale Integration) implementations, such as the Warp computer [1] [24], which is a linear systolic array, and the iWarp computer [2], which is a mesh-connected systolic array, designed in CMU.

The rest of this paper is organized as follows. In Section 2, we relate general nested-loop algorithms with uniform-dependence relations to the grid-connected systolic arrays. We also explain data collisions with an example. In Section 3, we present a complete set of necessary and sufficient conditions for a correct mapping under the constraint that at each time step, in each data link between two neighboring PEs there is at most one data token which can flow through. In Section 4, we generalize the conditions in Section 3 by allowing rearrangement of traveling tokens. In Section 5, we compare our model with related research works. Finally, some concluding remarks are given in Section 6.

2 Relating Nested-Loop Algorithms to Systolic Arrays

A p-nested-loop algorithm Ag is comprised of three parts: (1) The *loop index set*, $I^p = \{(i_1, i_2, \ldots, i_p)^t \mid l_j \leq i_j \leq u_j,\ 1 \leq j \leq p\}$, which is also called the *problem space*, where l_j and u_j are respectively the lower bound and the upper bound for each level j of the loop. In addition, l_j and u_j are integer-valued linear expressions possibly involving $i_1, i_2, \ldots, i_{j-1}$ for $1 < j \leq p$. (2) The set of *variables* V_{Ag}, including input variables, output variables, and temporary variables. (3) The sequence of *statements* F_{Ag} in the loop body. We restrict that the loop body does not contain any *subroutine call* and any *goto* instruction that jumps to the outside of the loop body.

Given a p-nested-loop algorithm, we can find data-dependence vectors and classify them according to the *ZERO-ONE-INFINITE* property. In general, data-dependence vectors can be defined in four ways: (1) modify-modify dependency (output dependence), which is the relation that a variable is generated (modified) in different indices; (2) use-use dependency (input dependence), which is the relation that a variable is used in different indices; (3) modify-use dependency (flow dependence), which is the relation that a variable is generated in one index and will be used in the other index; and (4) use-modify dependency (anti-dependence), which is the relation that a variable is used in one index and will be generated in the other index.

First, each different variable token symbol in the loop body causes one data-dependence vector of either modify-modify dependency, if the variable token symbol is on the left-hand-side of ':=', or use-use dependency, if the variable token symbol is on the right-hand-side of ':='. If each variable token is either generated only once or used only once, the data-dependence vector is of type ZERO. If each variable token is generated or used periodically during the computation, the data-dependence vector is of type INFINITE. On the other hand, each pair of a modified token symbol (on the left-hand-side of :=) and a used token symbol (on the right-hand-side of :=) or each pair of a modified token symbol and the other modified token symbol of the same array variable causes one "type ONE" data-dependence vector of either: modify-modify dependency; or modify-use dependency; or use-modify dependency. Tokens with data-dependence vectors of type ZERO or type INFINITE are input or output variable tokens. Tokens with data-dependence vectors of type ONE are temporary variable tokens.

More precisely, in a p-nested-loop algorithm, the index of a variable token symbol $V[f_1, f_2, \ldots, f_e]$ can be represented by a linear function:

$$(f_1, f_2, \ldots, f_e)^t = R_v(i_1, i_2, \ldots, i_p)^t + \vec{g}_v,$$

where R_v is an $(e \times p)$ matrix, $\vec{g}_v$ is an $(e \times 1)$ vector, and entries of R_v and $\vec{g}_v$ are integers. First, the data-dependence vector with variable token symbol $V[f_1, f_2, \ldots, f_e]$ is the null space of R_v. In this paper, we assume that the dimension of the null space of R_v is either 0 or 1. If the dimension of the null space of R_v is 0, then the data-dependence vector with $V[f_1, f_2, \ldots, f_e]$ (is $\vec{0}_p$ and) is of type ZERO; if the dimension of the null space of R_v is 1, then the data-dependence vector with $V[f_1, f_2, \ldots, f_e]$ (is the null space of R_v and) is of type INFINITE. In the latter case, if the value of $V[f_1, f_2, \ldots, f_e]$ is not modified during the computation, then the corresponding data-dependence vector can be either positive or negative. However, if the value of $V[f_1, f_2, \ldots, f_e]$ will be modified during the computation, then in order to preserve the data-dependence relation, the corresponding data-dependence vector must be positive. The case when the dimension of the null space of R_v is greater than 1, can be dealt with by using a scalar expansion technique to introduce new auxiliary variable token symbols.

Second, the type ONE data-dependence vector $\vec{x}$ between two distinct variable token symbols $V[f_1, f_2, \ldots, f_e]$ and $V[f'_1, f'_2, \ldots, f'_e]$ can be determined as follows. Let $(f_1, f_2, \ldots, f_e)^t = R_v(i_1, i_2, \ldots, i_p)^t + \vec{g}_v$ and $(f'_1, f'_2, \ldots, f'_e)^t = R'_v(i_1, i_2, \ldots, i_p)^t + \vec{g}'_v$. Assume that $(f_1, f_2, \ldots, f_e) = (f'_1, f'_2, \ldots, f'_e)$. Then, $\vec{x}$ $(= (x_1, x_2, \ldots, x_p)^t)$ is constant if and only if $R_v = R'_v$; $\vec{x}$ satisfies $R_v\vec{x} = \vec{g}_v - \vec{g}'_v$, where $0 \leq x_i \leq u_i - l_i$ for all $1 \leq i \leq p$; in addition, $\vec{x}$ is the smallest integer vector which is larger than $\vec{0}_p$ in lexicographical order. Note that, if $R_v \neq R'_v$ then $\vec{x}$ is not a constant vector.

The ZERO-ONE-INFINITE classification can help understand whether tokens can be destroyed or not; whether tokens can be pipelined or not; and whether a PE needs additional I/O ports or not. This classification is the key to formulating the necessary and sufficient conditions for mapping nested-loop algorithms into the target systolic arrays.

Definition: The mathematical structure of a p-nested-loop algorithm Ag is a 5-tuple $M_{Ag} = (I^p,\ V_{Ag},\ F_{Ag},\ D_{Ag},\ DT_{Ag})$, where:

1. I^p is the loop index set;
2. V_{Ag} is the set of variables;
3. F_{Ag} is the sequence of statements in the loop body;
4. D_{Ag} is the sequence of data-dependence vectors;
5. DT_{Ag} is the sequence of types for each data-dependence vector.

We will say that a nested-loop algorithm is shift-invariant if the dependence relations corresponding to all loop iterations in the index space are independent of their positions [[9], page 122]. We will also say that a nested-loop algorithm is of uniform-dependence, if all of the data-dependence vectors are constant. In this paper, we only consider shift-invariant uniform-dependence algorithms.

Example 1: The 3-nested-loop matrix multiplication algorithm

for $i,\ j,\ k := 0$ to n **do**

$C[i,j] := C[i,j] + A[i,k] * B[k,j];$

has three different variable token symbols, which cause three data-dependence vectors and all of the three data-dependence vectors are of type INFINITE as shown in Table 1. □

variable token	data-dependence vector	vector type	token type	dependency relation
$A[i,k]$	$d_1 = (0,1,0)^t$	INFINITE	input	use-use
$B[k,j]$	$d_2 = (1,0,0)^t$	INFINITE	input	use-use
$C[i,j]$	$d_3 = (0,0,1)^t$	INFINITE	output	modify-modify

Table 1: All three data-dependence vectors in the matrix multiplication algorithm are of type INFINITE.

Example 2: The 3-nested-loop algorithm

for $i,\ j,\ k := 0$ to n **do**

$A[i,j,k] := A[i,j-1,k];$

$B[i,j,k] := B[i-1,j,k];$

$C[i,j,k] := C[i,j,k-1] + A[i,j,k] * B[i,j,k];$

has six different variable token symbols, which cause nine data-dependence vectors. Among them, three are of type ONE and six are of type ZERO as shown in Table 2.

Note that, $A[i,j,k]$, $B[i,j,k]$, and $C[i,j,k]$ are with data dependence vectors of type ZERO, and they are generated only once. We probably should call their dependency relations "modify"; however, for consistency with the dependency relations of data dependence vectors of type INFINITE and of type ONE, we call their

variable token	data-dependence vector	vector type	token type	dependency relation
$< A[i,j,k], A[i,j-1,k] >$	$d_1 = (0,1,0)^t$	ONE	temporary	modify-use
$< B[i,j,k], B[i-1,j,k] >$	$d_2 = (1,0,0)^t$	ONE	temporary	modify-use
$< C[i,j,k], C[i,j,k-1] >$	$d_3 = (0,0,1)^t$	ONE	temporary	modify-use
$A[i,j,k]$	$d_4 = (0,0,0)^t$	ZERO	output	modify-modify
$B[i,j,k]$	$d_5 = (0,0,0)^t$	ZERO	output	modify-modify
$C[i,j,k]$	$d_6 = (0,0,0)^t$	ZERO	output	modify-modify
$A[i,j-1,k]$	$d_7 = (0,0,0)^t$	ZERO	input	use-use
$B[i-1,j,k]$	$d_8 = (0,0,0)^t$	ZERO	input	use-use
$C[i,j,k-1]$	$d_9 = (0,0,0)^t$	ZERO	input	use-use

Table 2: In Example 2, six different variable token symbols cause nine data-dependence vectors.

dependency relations "modify-modify". Similarly, $A[i,j-1,k]$, $B[i-1,j,k]$, and $C[i,j,k-1]$ are with data dependence vectors of type ZERO, and they are used only once. Also, we call their dependency relations "use-use" instead of "use" for consistency with the dependency relations of data dependence vectors of type INFINITE and of type ONE. $\square$

We now briefly state our q-D grid-connected systolic array model. A PE in the q-D grid-connected systolic array is represented by the tuple $(pi_1, pi_2, \ldots, pi_q)$, where $pl_j \leq pi_j \leq pu_j$ for $1 \leq j \leq q$, pl_j and pu_j are respectively the lower bound index and the upper bound index for each dimension j of the systolic array. In addition, it has only two physical data links connected to the neighboring PEs in each dimension. Therefore, there are in total $2q$ physical data links in each PE, or say, in each PE there are in total $2q$ interconnection primitives connected to $2q$ neighboring PEs. For convenience, we will say that P_i is the ith interconnection primitive, and $P_1 = [1,0,\ldots,0]^t_{1\times q}$; $P_2 = [-1,0,\ldots,0]^t_{1\times q}$; $P_{2i-1} = [0,\ldots,0,1,0,\ldots,0]^t_{1\times q}$, where the only non-zero entry in position i is 1; $P_{2i} = [0,\ldots,0,-1,0,\ldots,0]^t_{1\times q}$, where the only non-zero entry in position i is -1; $P_{2q-1} = [0,\ldots,0,1]^t_{1\times q}$; and $P_{2q} = [0,\ldots,0,-1]^t_{1\times q}$. We also make a reasonable assumption that only boundary PEs have I/O ports connected to the host computer.

We now explain how to relate nested-loop algorithms to systolic arrays. After finding and classifying all of the data-dependence vectors, we associate a single dedicated virtual data link with each data-dependence vector, which is with the corresponding set of data tokens. Note that, in the sequential algorithm there is only one copy of data tokens in the shared memory. However, in the systolic array algorithm, since dependent indices may be executed in different PEs, multiple copies of data tokens may be flowing through the systolic array. In general, if there are w data-dependence vectors with a specific array variable, then w copies of array variable tokens will be "traveling" through the systolic array, and each in a dedicated virtual data link. If a data-dependence vector has the condition $d_i = \vec{0}_p$, the corresponding virtual data link will be fixed in PEs, and therefore, the corresponding array variable tokens will be fixed in PEs. The purpose of classifying data-dependence vectors of type ZERO is to specify the requirements of local registers (or local memory) for those tokens with these zero data-dependence vectors.

A virtual data link may consist of several physical data links (interconnection primitives), therefore, a virtual data link may change its direction although each physical data link has only one direction. Next, in order to give a formal proof of avoiding data collisions in the data links, we add the following restriction: if a data token flows from $PE_{(pi_1,pi_2,\ldots,pi_q)}$ to $PE_{(pi_1+j_1,pi_2+j_2,\ldots,pi_q+j_q)}$, it must start to flow along the first dimension j_1 PEs, then it flows along the second dimension j_2 PEs, and so on. The cases when a data token flows along a given specific order of dimensions can be dealt in a similar way. However, the cases when data tokens flow along a random order of dimensions, and which will complicate providing a formal proof of avoiding data collisions in the data links, are beyond the scope of this paper.

The number of virtual data links is usually more than the number of physical data links. In this case, the time-slicing method is used to assign different data token streams using different time slots flowing through the physical data links. As we will associate a single dedicated virtual data link with each data-dependence vector, we will say for convenience that data-dependence vector d_i will correspond to virtual data link i and data token stream i. A correct systolic array algorithm must manage the data token streams so that the data tokens' values are the same as those in the sequential algorithm.

To simplify the discussion, we assume throughout this paper that for each data-dependence vector $d_i = (d_{i1}, d_{i2}, \ldots, d_{ip})^t$, $\gcd(d_{i1}, d_{i2}, \ldots, d_{ip}) = 1$. The discussion for the general case when $\gcd(d_{i1}, d_{i2}, \ldots, d_{ip}) \neq 1$, which requires additional virtual data links in order to avoid data collisions in the data links, is the same as that of [14].

The main effort of mapping a p-nested-loop algorithm into a q-D systolic array is to find a feasible space and time mapping, which maps an index iteration to be executed in a specific PE at a specific time. At the end of this section, we present a linear systolic array algorithm which we claimed incurs data collisions in the data link [13]. Let $\mathbf{H} \in Z^{1\times p}$ be the time mapping vector and $\mathbf{S} \in Z^{1\times p}$ be the space mapping vector for mapping p-nested loop algorithms into linear systolic arrays. Then the computation indexed by $\bar{I}$ is executed at time $\mathbf{H}\bar{I}$ in $PE_{\mathbf{S}\bar{I}}$.

Example 3: Consider the 3-nested-loop matrix multiplication algorithm in Example 1. The linear systolic array implementation, where the time mapping $\mathbf{H} = (2, 1, 2)$ and the space mapping $\mathbf{S} = (1, 1, -2)$ for the problem size $n = 3$ will generate data collisions in data link 3 as mentioned in [13]. This is because $C[0, 3]$ collides with $C[2, 0]$ and $C[1, 3]$ collides with $C[3, 0]$ in data link 3 during the computation. For instance, in Figure 1, there is only one shift register in data link 3. It is true that $C[0, 3]$ is first used and generated in index $(0, 3, 0)^t$ in PE_3 at time step 3, and $C[0, 3]$ will be used and generated next time in index $(0, 3, 1)^t$

in PE_1 at time step 5. Therefore, at time step 4, $C[0,3]$ will be in PE_2. However, at time step 4, $C[2,0]$ also reaches PE_2 along data link 3. It is a data collision. □

3 Synthesizing Grid-Connected Systolic array Algorithms under a Restricted Model

In this section, we show conditions for mapping nested loop algorithms into grid-connected systolic arrays with the constraint that at each time step, in each virtual data link between two neighboring PEs there is at most one data token which can flow through. Before introducing any conditions, we show an example of data collisions in a mesh-connected systolic array.

Example 4: The 3-nested-loop algorithm

for $i := 0$ to 15, $j := 0$ to 15, $k := 0$ to 13 **do**

$A[i,j,k] := F_1(A[i,j-4,k-3],\ C[-3j+2k,i]);$

$C[-3j+2k,i] := F_2(A[i-1,j,k-2],\ B[3i-j+k,3i-j]);$

has five different variable token symbols, which cause seven data-dependence vectors as shown in Table 3. Note that F_1 and F_2 are two functions.

variable token	data-dependence vector	type	token type	dependency relation
$< A[i,j,k], A[i,j-4,k-3] >$	$d_1 = (0,4,3)^t$	ONE	temporary	modify-use
$< A[i,j,k], A[i-1,j,k-2] >$	$d_2 = (1,0,2)^t$	ONE	temporary	modify-use
$B[3i-j+k,3i-j]$	$d_3 = (1,3,0)^t$	INFINITE	input	use-use
$C[-3j+2k,i]$	$d_4 = (0,2,3)^t$	INFINITE	output	modify-modify
$A[i,j,k]$	$d_5 = (0,0,0)^t$	ZERO	output	modify-modify
$A[i,j-4,k-3]$	$d_6 = (0,0,0)^t$	ZERO	input	use-use
$A[i-1,j,k-2]$	$d_7 = (0,0,0)^t$	ZERO	input	use-use

Table 3: In Example 4, there are seven data-dependence vectors.

Let $\mathbf{H} = (1,1,1)$ and $\mathbf{S} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Then $\mathbf{H}$ preserves data-dependence relations; in addition, $\mathbf{H}$ and $\mathbf{S}$ satisfy the non-singularity condition. However, as shown in Table 4 and Figure 2: tokens $A[0,5,5]$ (which is generated in index $(0,5,5)^t$ in $PE_{5,5}$ at time 10 and will be used in index $(0,9,8)^t$ in $PE_{9,8}$ at time 17), $A[0,6,5]$, $A[0,7,5]$, and $A[0,8,5]$, are with the data-dependence vector $d_1 = (0,4,3)^t$ and will pass through virtual data link 1 between $PE_{8,5}$ and $PE_{9,5}$ at time 13 simultaneously. Clearly, they cause data collisions in virtual data link 1. □

In the following, we show conditions for a correct mapping. Let $\mathbf{H} \in Z^{1\times p}$ be the time mapping vector

data token	generated index	$PE_{\mathbf{S}\bar{I}}$	$\mathbf{H}\bar{I}$	used index	$PE_{\mathbf{S}\bar{I}'}$	$\mathbf{H}\bar{I}'$
$A[0,5,5]$	$(0,5,5)^t$	$PE_{5,5}$	10	$(0,9,8)^t$	$PE_{9,8}$	17
$A[0,6,5]$	$(0,6,5)^t$	$PE_{6,5}$	11	$(0,10,8)^t$	$PE_{10,8}$	18
$A[0,7,5]$	$(0,7,5)^t$	$PE_{7,5}$	12	$(0,11,8)^t$	$PE_{11,8}$	19
$A[0,8,5]$	$(0,8,5)^t$	$PE_{8,5}$	13	$(0,12,8)^t$	$PE_{12,8}$	20

data	time step										
token	10	11	12	13	14	15	16	17	18	19	20
$A[0,5,5]$	$PE_{5,5}$	$PE_{6,5}$	$PE_{7,5}$	$\mathbf{PE_{8,5}}$	$\mathbf{PE_{9,5}}$	$PE_{9,6}$	$PE_{9,7}$	$PE_{9,8}$			
$A[0,6,5]$		$PE_{6,5}$	$PE_{7,5}$	$\mathbf{PE_{8,5}}$	$\mathbf{PE_{9,5}}$	$PE_{10,5}$	$PE_{10,6}$	$PE_{10,7}$	$PE_{10,8}$		
$A[0,7,5]$			$PE_{7,5}$	$\mathbf{PE_{8,5}}$	$\mathbf{PE_{9,5}}$	$PE_{10,5}$	$PE_{11,5}$	$PE_{11,6}$	$PE_{11,7}$	$PE_{11,8}$	
$A[0,8,5]$				$\mathbf{PE_{8,5}}$	$\mathbf{PE_{9,5}}$	$PE_{10,5}$	$PE_{11,5}$	$PE_{12,5}$	$PE_{12,6}$	$PE_{12,7}$	$PE_{12,8}$

Table 4: Tokens $A[0,5,5]$, $A[0,6,5]$, $A[0,7,5]$, and $A[0,8,5]$ are with the data-dependence vector $d_1 = (0,4,3)^t$ and will pass through virtual data link 1 between $PE_{8,5}$ and $PE_{9,5}$ at time 13 simultaneously. They cause data collisions in virtual data link 1.

and $\mathbf{S} \in Z^{q\times p}$ be the space mapping matrix, where $1 \le q < p$. Then the computation indexed by $\bar{I}$ is executed at time $\mathbf{H}\bar{I}$ in $PE_{\mathbf{S}\bar{I}}$. A correct q-D systolic array algorithm $(\mathbf{H}, \mathbf{S})$ that maps a p-nested-loop algorithm into a q-D grid-connected systolic array must preserve data-dependence relations, and the right tokens must be in the right place at the right time, and in addition, data tokens must not collide in the data links. We define $sign(\delta) = 1$, -1, or 0, if $\delta > 0$, $\delta < 0$, or $\delta = 0$, respectively.

Theorem 1 : *Suppose that at each time step, in each virtual data link between two neighboring PEs there is at most one data token which can flow through. Then, a q-D systolic array algorithm $(\mathbf{H}, \mathbf{S})$ maps a p-nested-loop algorithm correctly into a q-D grid-connected systolic array if and only if it satisfies the following four conditions.*

1. $\mathbf{H}d_i > 0$ *for each non-zero data-dependence vector* d_i.
2. $\forall\ \bar{I}_1,\ \bar{I}_2 \in I^p$, *if* $\bar{I}_1 \ne \bar{I}_2$, *then either* $\mathbf{H}\bar{I}_1 \ne \mathbf{H}\bar{I}_2$ *or* $\mathbf{S}\bar{I}_1 \ne \mathbf{S}\bar{I}_2$.
3. *If* $\mathbf{S}d_i = (sd_{i1},\, sd_{i2},\, \ldots,\, sd_{iq})^t \ne \vec{0}_q$, *then* $\mathbf{H}d_i = b_i \sum_{j=1}^{q} |sd_{ij}|$, *where* b_i *is a positive integer.*
4. *Three cases must be true to avoid data collisions in the data links.*

   Case 1: /* for type ONE data-dependence vector d_i. */

   If (1) *the data-dependence vector* d_i *is of type ONE,*

   (2) $\mathbf{S}d_i = (sd_{i1},\, sd_{i2},\, \ldots,\, sd_{iq})^t \ne \vec{0}_q$,

   (3) $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, *and*

   (4) $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \alpha\, sign(sd_{ij})\, P_{2j-1} \ne \vec{0}_q$, *where* α *is a positive integer and* $0 < \alpha < |sd_{ij}|$,

   then $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \ne b_i\alpha$.

Case 2: /* for type INFINITE data-dependence vector d_i and $\mathbf{S}d_i = sd_{ij}P_{2j-1} \neq \vec{0}_q$. */

If (1) *the data-dependence vector* d_i *is of type INFINITE,*

(2) $\mathbf{S}d_i = sd_{ij}P_{2j-1} \neq \vec{0}_q$,

(3) $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ *for all integers* γ,

(4) $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, *and*

(5) $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \alpha\, sign(sd_{ij})\, P_{2j-1} \neq \vec{0}_q$, *where* α *is a positive integer,*

then $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq b_i\alpha$.

Case 3: /* for type INFINITE data-dependence vector d_i and $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t \neq \vec{0}_q$. */

If (1) *the data-dependence vector* d_i *is of type INFINITE,*

(2) $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t \neq \vec{0}_q$,

(3) $\mathbf{S}d_i \neq sd_{ij}P_{2j-1}$, *for every* j,

(4) $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ *for all integers* γ,

(5) $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, *and*

(6) $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta\mathbf{S}d_i + \alpha\, sign(sd_{ij})\, P_{2j-1} \neq \vec{0}_q$,

where β *is a non-negative integer and* $-|sd_{ij}| < \alpha < |sd_{ij}|$,

then $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq b_i(\beta \sum_{j=1}^{q} |sd_{ij}| + \alpha) = \beta\mathbf{H}d_i + b_i\alpha$. □

Note that, formally, the PEs in the q-D systolic array should be numbered from $(1, 1, \ldots, 1)$ to $(pu_1, pu_2, \ldots, pu_q)$, and $\mathbf{S}$ should map the indices to $\{PE_{(1,1,\ldots,1)}, \ldots, PE_{(pu_1,pu_2,\ldots,pu_q)}\}$. However, it will be convenient to number the PE's from $\min\{\mathbf{S}\bar{I}_1 | \bar{I}_1 \in I^p\}$ to $\max\{\mathbf{S}\bar{I}_2 | \bar{I}_2 \in I^p\}$ in this presentation.

In the following, we describe the theorem instead of giving a formal proof. Condition 1 ensures that data-dependence relations are preserved according to the original sequential algorithm. Condition 2 insists that $(\mathbf{H}, \mathbf{S})$ is a non-singular mapping. That is, no two indices can be mapped to the same PE at the same time. Condition 3 specifies that tokens flow through data links at a constant speed. A benefit of this condition is that we can assign a constant amount of shift registers or data buffers for each of the virtual data links in each PE so that data streams can flow at a constant speed. Therefore, data congestion can be avoided. This condition also guarantees that data tokens will flow to the right place at the right time.

Suppose that a variable with the data-dependence vector d_i is generated in index $\bar{I}$ and will be used next time in index $\bar{I} + d_i$. Index $\bar{I}$ is executed in $PE_{\mathbf{S}\bar{I}}$ at time $\mathbf{H}\bar{I}$, and the index $\bar{I} + d_i$ will be executed in $PE_{\mathbf{S}(\bar{I}+d_i)}$ at time $\mathbf{H}(\bar{I} + d_i)$. Suppose that $\mathbf{S}d_i \neq \vec{0}_q$. Then the Manhattan distance between $PE_{\mathbf{S}\bar{I}}$ and $PE_{\mathbf{S}(\bar{I}+d_i)}$ is equal to $\sum_{j=1}^{q} |sd_{ij}|$, because $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t$. Therefore, each token in virtual data link i is delayed by $b_i = \mathbf{H}d_i / \sum_{j=1}^{q} |sd_{ij}|$ units of time in each PE, where b_i must be a positive integer. Thus, if there are r_i non-zero entries of sd_{ij}, where $1 \leq j \leq q$, we require in total $b_i r_i$ shift registers for virtual data link i in each PE.

The purpose of Condition 4 is to avoid data collisions in the data links. If $\mathbf{S}d_i = \vec{0}_q$, the virtual data link i is fixed in each PE and tokens with the data-dependence vector d_i are stored in local registers (or local memory). Therefore, in this case, tokens will not collide with each other in virtual data link i. If $\mathbf{S}d_i \neq \vec{0}_q$, there are three cases for data stream i, depending on the type of the data-dependence vector d_i and the value of $\mathbf{S}d_i$.

Case 1: d_i is of type ONE. The active life of each data token, which is with the data-dependence vector d_i, has only $\mathbf{H}d_i$ units of time, because a token is generated only once and will be used only once. Therefore, after that token is used, it can be destroyed. Suppose that a token is generated in index $\bar{I}_1$ at time $\mathbf{H}\bar{I}_1$ in $PE_{\mathbf{S}\bar{I}_1}$ and is used in index $\bar{I}_1 + d_i$ at time $\mathbf{H}(\bar{I}_1 + d_i)$ in $PE_{\mathbf{S}(\bar{I}_1+d_i)}$. As $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t$, the token starts to flow along the first dimension sd_{i1} PEs in P_1 direction if $sd_{i1} > 0$, or $-sd_{i1}$ PEs in P_2 direction if $sd_{i1} < 0$; then, it flows along the second dimension sd_{i2} PEs in P_3 direction if $sd_{i2} > 0$, or $-sd_{i2}$ PEs in P_4 direction if $sd_{i2} < 0$; and so on. Data collisions occur only if there are other tokens using part of the path when the token flows through the path.

For instance, in Figure 2 of Example 4, token $A[0, 9, 5]$, which is with the data-dependence vector $d_1 = (0, 4, 3)^t$ of type ONE, is generated in index $(0, 9, 5)^t$ in $PE_{9,5}$, and will be used in index $(0, 13, 8)^t$ in $PE_{13,8}$. It may collide with tokens generated in $PE_{6,5}$, $PE_{7,5}$, $PE_{8,5}$, $PE_{10,5}$, $PE_{11,5}$, $PE_{12,5}$, $PE_{9,3}$, $PE_{9,4}$, $PE_{9,6}$, and $PE_{9,7}$ which are marked by outlined circles. This is because all of these tokens will flow through part of the path from $PE_{9,5}$ to $PE_{13,8}$. Similarly, token $A[0, 15, 10]$, which is with the data-dependence vector $d_2 = (1, 0, 2)^t$ of type ONE, is generated in index $(0, 15, 10)^t$ in $PE_{15,10}$, and will be used in index $(1, 15, 12)^t$ in $PE_{15,12}$. It may collide with tokens generated in $PE_{15,9}$ and $PE_{15,11}$ which are marked by dotted circles.

The constraint in Case 1 of Condition 4 allows us to avoid data collisions in a <u>grid-connected</u> systolic array. That is, if two distinct indices are executed in the same <u>horizontal</u> PEs, or in the same <u>vertical</u> PEs, or in the same <u>perpendicular</u> PEs within a certain distance, and the direction of these two PEs ($\mathbf{S}(\bar{I}_2 - \bar{I}_1)$) is consistent with one of the basic directions $sign(sd_{ij})P_{2j-1}$ of the data path, then the difference of the execution time steps between these two indices cannot be equal to the time that a token takes to flow from one PE to the other PE along the interconnection primitive $sign(sd_{ij})P_{2j-1}$ at speed $1/b_i$.

Case 2: d_i is of type INFINITE and $\mathbf{S}d_i = sd_{ij}P_{2j-1}$. If a token is with a data-dependence vector of type INFINITE, this token will be generated or used periodically during the computation. Therefore, throughout the computation, we cannot destroy this token. The data path in which tokens flow through has only one direction, $sign(sd_{ij})P_{2j-1}$. For instance, in Figure 2, $B[0, 0]$, which is with the data-dependence vector $d_3 = (1, 3, 0)^t$ of type INFINITE, is generated in index $(0, 0, 0)^t$ in $PE_{0,0}$, and will be used again in all indices $(\gamma, 3\gamma, 0)^t$ in $PE_{3\gamma,0}$ for integers $\gamma > 0$. It may collide with other tokens generated in $PE_{i,0}$ for all $i > 0$ which are marked by netted circles. Note that if $\bar{I}_2 - \bar{I}_1 = \gamma d_i$, where γ is an integer, the same token is generated or used, and therefore, it does not cause any data collision.

We conclude the condition in Case 2 (which is slightly different from the condition in Case 1): if two distinct indices are executed in the same horizontal PEs, or in the same vertical PEs, or in the same perpendicular PEs without any constraint of the distance between these two execution PEs, and the direction of these two PEs is consistent with the direction $sign(sd_{ij})P_{2j-1}$ of the data path, then the difference of the execution time steps between these two indices cannot be equal to the time that a token flows from one PE to the other PE along the interconnection primitive $sign(sd_{ij})P_{2j-1}$ at speed $1/b_i$.

Case 3: d_i is of type INFINITE and $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t \neq sd_{ij}P_{2j-1}$ for every j. Similar to Case 2, if a token is with such a data-dependence vector of type INFINITE, then this token cannot be destroyed during the computation. However, unlike Case 2, the data path in which the token flows through zigzags. For instance, in Figure 2, $C[8,0]$, which is with the data-dependence vector $d_4 = (0,2,3)^t$ of type INFINITE, is generated in index $(0,0,4)^t$ in $PE_{0,4}$, and will be used and generated again in indices $(0,0,4)^t + \beta d_4$ in $PE_{\mathbf{S}((0,0,4)^t+\beta d_4)}$ for $\beta > 0$. The data path which token $C[8,0]$ flows through has a zigzag pattern. In addition, all tokens generated in the PEs marked by black circles will use part of the path, and they may cause data collisions.

The constraint in Case 3 of Condition 4 states that only the tokens, (which are generated or used in index $\bar{I}_2$ in $PE_{\mathbf{S}\bar{I}_2}$ that are located horizontally, vertically, or perpendicularly within a certain distance to $PE_{\mathbf{S}(\bar{I}_1+\beta d_i)}$, for $\beta \geq 0$, and the direction of $PE_{\mathbf{S}\bar{I}_2}$ and $PE_{\mathbf{S}(\bar{I}_1+\beta d_i)}$ is consistent with one of the basic directions $sign(sd_{ij})P_{2j-1}$ of the data path), may collide with the token generated or used in index $\bar{I}_1$ in $PE_{\mathbf{S}\bar{I}_1}$ in virtual data link i. Therefore, the difference of the execution time steps between indices $\bar{I}_2$ and $\bar{I}_1$ cannot be equal to the time that a token flows from $PE_{\mathbf{S}\bar{I}_1}$ to $PE_{\mathbf{S}\bar{I}_2}$ along the composite data path "$\sum_{h=1}^{q} sd_{ih}P_{2h-1}$" β times and then along the interconnection primitive P_{2j-1} or P_{2j} at speed $1/b_i$ depending on the location of $PE_{\mathbf{S}\bar{I}_2}$.

Corollary 2 : *If the systolic array algorithm* $(\mathbf{H}, \mathbf{S})$ *satisfies Condition* 1 *through Condition* 3, *and the data-dependence vector* d_i *is of type ONE, and* $|sd_{ij}| \leq 1$ *in Condition* 3, *then there is no data collision along the direction* $sign(sd_{ij})P_{2j-1}$ *in virtual data link* i.

Proof : If $|sd_{ij}| \leq 1$ in Condition 3, then there does not exist any positive integer α such that $0 < \alpha < |sd_{ij}| \leq 1$ in Case 1 of Condition 4. Therefore, Condition 4 is satisfied automatically. □

4 Synthesizing Grid-Connected Systolic Array Algorithms under a General Model

Condition 4 in the last section is over-restricted. It eliminates some solutions which allow more than one data token flowing through a specific data link and can be handled by employing a cyclic shuffle of traveling tokens. For instance, in Example 4, the mapping $\mathbf{H} = (1, 1, 1)$ and $\mathbf{S} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ violates Condition 4 of Theorem 1, this is because at each time step three tokens will reach each PE via virtual data link 1 from the south and four tokens will reach each PE via virtual data link 1 from the west (left). However, these seven tokens in effect are sent to seven distinct PEs. The first token from the southern neighboring PE will be used to perform the iteration loop body; the second and the third tokens from the southern neighboring PE will be sent to the northern neighboring PE; the fourth token from the western neighboring PE will be sent to the northern neighboring PE; and the remaining three tokens from the western neighboring PE will be sent to the eastern neighboring PE all via virtual data link 1. Therefore, a feasible sequence of microcodes in $PE_{(j,k)}$ at time step $i + j + k$ for handling tokens traveling through virtual data link 1 can be as follows.

1. Get a token $A[i, j-4, k-3]$ from the southern neighboring PE;
   /* $A[i, j-4, k-3]$ is used to compute the iteration loop body. */
2. compute the iteration loop body;
   /* Step 2 can be performed concurrently with Step 3 to Step 8 in order to overlap
   the computation time and the communication time. */
3. get a token $A[i, j-4, k-2]$ from the southern neighboring PE and then send to the northern neighboring PE;
4. get a token $A[i, j-4, k-1]$ from the southern neighboring PE and then send to the northern neighboring PE;
5. get a token $A[i, j-4, k]$ from the western neighboring PE and then send to the northern neighboring PE;
6. get a token $A[i, j-3, k]$ from the western neighboring PE and then send to the eastern neighboring PE;
7. get a token $A[i, j-2, k]$ from the western neighboring PE and then send to the eastern neighboring PE;
8. get a token $A[i, j-1, k]$ from the western neighboring PE and then send to the eastern neighboring PE;
9. send a generated token $A[i, j, k]$ which is with data dependence vector d_1 to the eastern neighboring PE. □

In general, for an arbitrary data stream i which is with data dependence vector d_i, the following algorithm can be used to handle the cyclic shuffle of traveling tokens in virtual data link i.

Algorithm for shuffling traveling tokens in virtual data link i :

Suppose that $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t \neq \vec{0}_q$. Then the generated microcodes in $PE_{\mathbf{S}\bar{I}}$ at time step $\mathbf{H}\bar{I}$ for handling virtual data link i can be as follows.

Step 0: Starting from index $j = q$ of sd_{ij}, where $q \geq j \geq 1$, and using a decreasing order,

Step 1: /* obtain a token which is used to compute the iteration loop body */

search the last non-zero entry of sd_{ij} and get a token from $PE_{\mathbf{S}\bar{I}-sign(sd_{ij})P_{2j-1}}$;

compute the iteration loop body;

Step 2: /* transfer other tokens from $PE_{\mathbf{S}\bar{I}-sign(sd_{ij})P_{2j-1}}$ to $PE_{\mathbf{S}\bar{I}+sign(sd_{ij})P_{2j-1}}$ */

for $h = 2$ to $|sd_{ij}|$ **do**

get a token from $PE_{\mathbf{S}\bar{I}-sign(sd_{ij})P_{2j-1}}$ and then send to $PE_{\mathbf{S}\bar{I}+sign(sd_{ij})P_{2j-1}}$;

end_for

Step 3: /* transfer remaining tokens between neighboring PEs */

let $last = j$ and let $j = j - 1$;

while $(j \geq 1)$ **do**

if $(sd_{ij} \neq 0)$ **then**

get a token from $PE_{\mathbf{S}\bar{I}-sign(sd_{ij})P_{2j-1}}$ and then send to $PE_{\mathbf{S}\bar{I}+sign(sd_{i\,last})P_{2last-1}}$;

for $h = 2$ to $|sd_{ij}|$ **do**

get a token from $PE_{\mathbf{S}\bar{I}-sign(sd_{ij})P_{2j-1}}$ and then send to $PE_{\mathbf{S}\bar{I}+sign(sd_{ij})P_{2j-1}}$;

end_for

let $last = j$;

end_if

let $j = j - 1$;

end_while

Step 4: after computing the iteration loop body, send a generated token with data-dependence vector d_i to $PE_{\mathbf{S}\bar{I}+sign(sd_{i\,last})P_{2last-1}}$. □

Using the technique of shuffling traveling tokens, we can generalize Theorem 1 as the following theorem.

Theorem 3 : *A q-D systolic array algorithm* $(\mathbf{H}, \mathbf{S})$ *maps a p-nested-loop algorithm correctly into a q-D grid-connected systolic array if and only if it satisfies the following four conditions.*

1. $\mathbf{H}d_i > 0$ *for each non-zero data-dependence vector* d_i.

2. $\forall\ \bar{I}_1,\ \bar{I}_2 \in I^p$, *if* $\bar{I}_1 \neq \bar{I}_2$, *then either* $\mathbf{H}\bar{I}_1 \neq \mathbf{H}\bar{I}_2$ *or* $\mathbf{S}\bar{I}_1 \neq \mathbf{S}\bar{I}_2$.

3. *If* $\mathbf{S}d_i = (sd_{i1},\ sd_{i2},\ \ldots,\ sd_{iq})^t \neq \vec{0}_q$, *then* $\mathbf{H}d_i = b_i \sum_{j=1}^{q} |sd_{ij}|$, *where* b_i *is a positive integer.*

4. /* for type INFINITE data-dependence vector d_i. */
   If (1) *the data-dependence vector* d_i *is of type INFINITE,*
   (2) $\mathbf{S}d_i = (sd_{i1},\ sd_{i2},\ \ldots,\ sd_{iq})^t \neq \vec{0}_q$,
   (3) $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ *for all integers* γ,
   (4) $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, *and*
   (5) $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta\mathbf{S}d_i$, *where* β *is a positive integer,*
   then $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq \beta\mathbf{H}d_i$.

Proof: in Appendix. □

Corollary 4 : *If the systolic array algorithm* $(\mathbf{H}, \mathbf{S})$ *satisfies Condition* 1 *through Condition* 3*, and the data-dependence vector* d_i *is of type ONE, and the algorithm for shuffling traveling tokens in virtual data link* i *is used, then there is no data collision along virtual data link* i*.*

Proof: Because the data-dependence vector d_i is of type ONE, Condition 4 in Theorem 3 is satisfied automatically. □

A systolic array algorithm based on Theorem 3, which employs the technique of shuffling traveling tokens, is more general than a systolic array algorithm based on Theorem 1, which restricts at most one data token flowing from one PE to a neighboring PE in each virtual data link at each time step. It is easy to check that $\mathbf{H} = (2, 1, 2)$ and $\mathbf{S} = (1, 1, -2)$ in Example 3 and $\mathbf{H} = (1, 1, 1)$ and $\mathbf{S} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ in Example 4 are infeasible systolic array algorithms under Theorem 1; however, they are feasible systolic array algorithms under Theorem 3.

Although systolic array algorithms based on Theorem 3 are more general than systolic array algorithms based on Theorem 1, the former systolic array algorithms require more shift registers than the latter ones. Suppose that $\mathbf{S}d_i = (sd_{i1}, sd_{i2}, \ldots, sd_{iq})^t \neq \vec{0}_q$. A systolic array algorithm based on Theorem 3 requires $b_i(\sum_{j=1}^{q} |sd_{ij}|)$ shift registers in virtual data link i in each PE. However, a systolic array algorithm based on Theorem 1 only requires $b_i r_i$ shift registers in virtual data link i in each PE, where r_i is the number of non-zero entries of sd_{ij}.

Therefore, when evaluating whether $(\mathbf{H}, \mathbf{S})$ is a feasible systolic array algorithm, we first check whether data dependence vector d_i satisfies Theorem 1; if it does not, then we check whether d_i satisfies Theorem 3.

Algorithm for evaluating whether (H, S) is a feasible systolic array implementation :

Step 1: Check Conditions 1 through 3 in Theorem 1;

Step 2: **for** each data dependence vector d_i **do** Steps 3 through 5:

Step 3: **if** d_i satisfies Condition 4 of Theorem 1, **then** as at each time step there is at most one data token flowing from one PE to a neighboring PE via virtual data link i, thus, virtual data link i only requires $b_i r_i$ shift registers in each PE, where r_i is the number of non-zero entries of sd_{ij};

Step 4: **else if** d_i satisfies Condition 4 of Theorem 3, **then** because the technique of shuffling traveling tokens is used, virtual data link i requires $b_i(\sum_{j=1}^{q} |sd_{ij}|)$ shift registers in each PE;

Step 5: **else** data collisions occur in virtual data link i. □

4.1 Other Properties for Evaluating Feasible Systolic Array Implementations

In general, the four necessary and sufficient conditions in Theorem 3 are exclusive. That is, none of them can be derived from other conditions. However, under certain constraints, either Condition 2 or Condition 4 would be redundant. In the following, we derive two theorems which allow us to reduce the computation complexity of evaluating feasible systolic array implementations for some special cases. First, for the special case when $q = p - 1$, Condition 4 can be derived from Condition 2.

Theorem 5 : *Let $q = p-1$. A q-D systolic array algorithm $(\mathbf{H}, \mathbf{S})$ from a p-nested-loop algorithm Ag that satisfies Conditions 1 through 3 maps Ag correctly into a q-D grid-connected systolic array.*

Proof : We show that Condition 4 can be derived from Condition 1 through Condition 3. First, from Corollary 4, there is no data collision along virtual data link i if the data dependence vector d_i is of type ONE. In the following we only consider the case when d_i is a type INFINITE data dependence vector. We want to show that if $\mathbf{S}d_i \neq \vec{0}_q$, and $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ for all integers γ, and $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, and $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i$ for some positive integer β, then $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq \beta \mathbf{H} d_i$.

Assume by contradiction that $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{H} d_i > 0$. Then, from the assumption, $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i$, thus, we have $\mathbf{H}((\bar{I}_2 - \bar{I}_1) - \beta d_i) = 0$ and $\mathbf{S}((\bar{I}_2 - \bar{I}_1) - \beta d_i) = \vec{0}_{p-1}$. That is, $\begin{pmatrix} \mathbf{H} \\ \mathbf{S} \end{pmatrix} ((\bar{I}_2 - \bar{I}_1) - \beta d_i) = \vec{0}_p$.

However, because $\mathbf{H}$ is of rank 1 and $\mathbf{S}$ is of rank $p - 1$ (because $q = p - 1$), from Condition 2, $\begin{pmatrix} \mathbf{H} \\ \mathbf{S} \end{pmatrix}$ is non-singular. Therefore, $\begin{pmatrix} \mathbf{H} \\ \mathbf{S} \end{pmatrix}$ is a basis. As a basis will not map a non-zero vector to a zero vector, $(\bar{I}_2 - \bar{I}_1) - \beta d_i = \vec{0}_p$ must be true. Therefore, $\bar{I}_2 - \bar{I}_1$ is equal to βd_i. From the assumption that $\gcd(d_{i1}, d_{i2}, \ldots, d_{ip}) = 1$, where $d_i = (d_{i1}, d_{i2}, \ldots, d_{ip})^t$, it follows that β is an integer. However, this contradicts our assumption that $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ for all integers γ. □

Second, if there exists a type INFINITE data dependence vector d_i such that $\mathbf{S}d_i \neq \vec{0}_q$, then Condition 2 is implicitly satisfied by Condition 1, Condition 3, and Condition 4.

Lemma 6 : *Suppose that $(\mathbf{H}, \mathbf{S})$ is a feasible systolic array algorithm which satisfies Theorem 3. Then, for any type INFINITE data dependence vector d_i, if $\mathbf{S}d_i \neq \vec{0}_q$, and $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ for all integers γ, and $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i$ for some non-zero integer β, we have $\mathbf{H}(\bar{I}_2 - \bar{I}_1)\mathbf{S}d_i \neq \mathbf{H}d_i\mathbf{S}(\bar{I}_2 - \bar{I}_1)$.*

Proof : Assume by contradiction that $\mathbf{H}(\bar{I}_2 - \bar{I}_1)\mathbf{S}d_i = \mathbf{H}d_i\mathbf{S}(\bar{I}_2 - \bar{I}_1)$. $\mathbf{H}(\bar{I}_2 - \bar{I}_1)$ and $\mathbf{H}d_i$ are integers, $\mathbf{S}(\bar{I}_2 - \bar{I}_1)$ and $\mathbf{S}d_i$ are q-tuple vectors. Because $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i$, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{H} d_i$. Consider the following two cases depending on the non-zero integer β:

1. $\beta > 0$.

   Because $\mathbf{H}d_i > 0$, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$. However, this contradicts Condition 4 that $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq \beta\mathbf{H}d_i$.

2. $\beta < 0$.

   We exchange the roles of $\bar{I}_1$ and $\bar{I}_2$. We have $\mathbf{S}(\bar{I}_1 - \bar{I}_2) = (-\beta)\mathbf{S}d_i$, where $(-\beta)$ is a positive integer; $\mathbf{H}(\bar{I}_1 - \bar{I}_2) = (-\beta)\mathbf{H}d_i > 0$; in addition, $\mathbf{H}(\bar{I}_1 - \bar{I}_2)\mathbf{S}d_i = \mathbf{H}d_i\mathbf{S}(\bar{I}_1 - \bar{I}_2)$. However, this also contradicts Condition 4. $\square$

Note that, Theorem 3 implies Lemma 6; however, Lemma 6 is not Theorem 3. For example, if $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = (2,2)^t$, $\mathbf{S}d_i = (5,5)^t$, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = 2$, and $\mathbf{H}d_i = 5$, then $\mathbf{H}(\bar{I}_2 - \bar{I}_1)\mathbf{S}d_i = \mathbf{H}d_i\mathbf{S}(\bar{I}_2 - \bar{I}_1)$. However, tokens generated or used in $\bar{I}_1$ will not collide with tokens generated or used in $\bar{I}_2$. This is because these two groups of tokens flow along different data paths, although all of them travel through virtual data link i.

Theorem 7 : *Suppose that* $(\mathbf{H}, \mathbf{S})$ *is a systolic array algorithm. If there exists at least one type INFINITE data-dependence vector* d_i *such that* $\mathbf{S}d_i \neq \vec{0}_q$, *then Condition* 2 *is implicitly satisfied by Condition* 1, *Condition* 3, *and Condition* 4 *in Theorem* 3.

Proof : We want to show that for any two distinct indices $\bar{I}_1$ and $\bar{I}_2$ either $\mathbf{H}\bar{I}_1 \neq \mathbf{H}\bar{I}_2$ or $\mathbf{S}\bar{I}_1 \neq \mathbf{S}\bar{I}_2$. Consider two cases:

1. $\mathbf{H}\bar{I}_1 = \mathbf{H}\bar{I}_2$.

   If $(\bar{I}_2 - \bar{I}_1) = \gamma d_i$ for some integer γ, then $\gamma\mathbf{H}d_i = \mathbf{H}\bar{I}_2 - \mathbf{H}\bar{I}_1 = 0$ which contradicts Condition 1.

   If $(\bar{I}_2 - \bar{I}_1) \neq \gamma d_i$ for all integers γ, then because $\mathbf{S}d_i \neq \vec{0}_q$ and $\mathbf{H}d_i \neq 0$, from Lemma 6, $\vec{0}_q = \mathbf{H}(\bar{I}_2 - \bar{I}_1)\mathbf{S}d_i \neq \mathbf{H}d_i\mathbf{S}(\bar{I}_2 - \bar{I}_1)$ which implies $\mathbf{S}\bar{I}_1 \neq \mathbf{S}\bar{I}_2$.

2. $\mathbf{S}\bar{I}_1 = \mathbf{S}\bar{I}_2$.

   If $(\bar{I}_2 - \bar{I}_1) = \gamma d_i$ for some integer γ, then $\mathbf{H}\bar{I}_2 - \mathbf{H}\bar{I}_1 = \gamma\mathbf{H}d_i \neq 0$ by Condition 1.

   If $(\bar{I}_2 - \bar{I}_1) \neq \gamma d_i$ for all integers γ, then because $\mathbf{S}d_i \neq \vec{0}_q$ and $\mathbf{H}d_i \neq 0$, from Lemma 6, $\mathbf{H}(\bar{I}_2 - \bar{I}_1)\mathbf{S}d_i \neq \mathbf{H}d_i\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \vec{0}_q$ which implies $\mathbf{H}\bar{I}_1 \neq \mathbf{H}\bar{I}_2$. $\square$

5 Comparisons with Related Works

As mentioned in Section 1, the problem of finding feasible or optimal mappings for systolic array implementations has a rich history with a large number of transformations proposed by various researchers. However, most of them only considered the case when mapping p-nested-loop algorithms into $(p-1)$-dimensional systolic arrays. In addition, they did not consider the cases when data might collide in the data links [4] [8]

[16]–[20] [23] [26]. In effect, as shown in Theorem 5, when $q = p - 1$, if the non-singularity condition holds, the condition for avoiding data collision also holds when the algorithm for shuffling traveling tokens is used. However, the condition for avoiding data collision is still necessary, if we restrict at most one data token flowing from one PE to a neighboring PE in each virtual data link at each time step, such as the case in Example 4.

In the following, we compare our method with three related works published in the relevant literature. First, we compare our (new) method with our previous work. Earlier, we first proposed a set of necessary and sufficient conditions for mapping nested-loop algorithms into linear systolic arrays [13]. We then generalized the method for mapping nested-loop algorithms into systolic arrays of arbitrary dimensions [14]. However, our previous model restricts each virtual data link to have only one direction. Therefore, the systolic arrays based on our previous model may require more physical data links than our new model. Certainly, that design may increase the complexity of the hardware and the fabrication cost of the target systolic arrays. Next, our previous model only allows at most one data token flowing from one PE to a neighboring PE in each virtual data link at each time step. Therefore, some interesting systolic array algorithms which can be handled by shuffling traveling tokens are eliminated by our old constraints, such as the systolic array algorithm in Example 3.

Recently, Ganapathy and Wah [6] [7] and Shang and Fortes [25] also proposed conditions for transforming p-nested-loop algorithms into a 1-D time and q-D space systolic array implementations, where $1 \leq q \leq p-1$. We found that both of them only considered part of our model. Ganapathy and Wah only dealt with problems whose data-dependence vectors are all of type INFINITE; Shang and Fortes, on the other hand, reduced type-INFINITE data-dependence vectors to type-ONE and type-ZERO data-dependence vectors. In the following, we use two examples to explain the difference of two solution spaces of two algorithms, which have the same set of non-zero data-dependence vectors, but data-dependence vectors of one algorithm are all of type INFINITE and data-dependence vectors of another algorithm are all of type ONE and of type ZERO. The application studied here is a 4×4 matrix multiplication — $C_{4\times4} = A_{4\times4}B_{4\times4}$, which is implemented by a linear systolic array with the time mapping $\mathbf{H} = (1, 2, 2)$ and the space mapping $\mathbf{S} = (1, 1, -1)$ derived by Shang and Fortes [25].

Example 5: Consider the 3-nested-loop matrix multiplication algorithm in Example 1 in Section 2. The linear systolic array implementation, where the time mapping $\mathbf{H} = (1, 2, 2)$ and the space mapping $\mathbf{S} = (1, 1, -1)$ for the problem size $n = 3$, will generate data collisions in data link 2. This is because $\mathbf{H}$ and $\mathbf{S}$ do not satisfy Condition 4 in either Theorem 1 or Theorem 3.

In Figure 3, $B[k, j]$, which are with the data-dependence vector $d_2 = (1, 0, 0)^t$ of type INFINITE, are input variables for $0 \leq j,\ k \leq 3$, and therefore, $B[k,j]$ cannot be destroyed during the computation. We now examine the behavior of two tokens $B[0,3]$ and $B[1,0]$. $B[0,3]$ is first used in index $\bar{I}_2 = (0,3,0)^t$ in PE_3 at time step 6; $B[1,0]$ is first used in index $\bar{I}_1 = (0,0,1)^t$ in PE_{-1} at time step 2. It is true that $\bar{I}_2 - \bar{I}_1 = (0,3,-1)^t \neq \gamma d_2$ for all integers γ, $\mathbf{S}d_2 = 1$, $b_2 = 1$, and $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = 4 = \alpha$; however, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = 4 = b_2\alpha$ violates the constraint at Case 2 of Condition 4 in Theorem 1, or the constraint of Condition 4 in Theorem 3. Therefore, $B[0,3]$ must collide with $B[1,0]$ in virtual data link 2.

This is because $B[0,3]$ is an input token, and $B[0,3]$ enters into the linear array from PE_{-3}. As $\mathbf{S}d_2 = 1$ and $b_2 = \mathbf{H}d_2/\mathbf{S}d_2 = 1$, there is only one shift register in virtual data link 2 in each PE. In addition, $B[0,3]$ is delayed by only one unit of time in each PE. Therefore, if $B[0,3]$ is used in PE_3 at time step 6, it is in PE_2 at time step 5, in PE_1 at time step 4, in PE_0 at time step 3, and in PE_{-1} at time step 2. However, at these times, $B[1,0]$ also passes through virtual data link 2, and therefore, $B[0,3]$ collides with $B[1,0]$. Similarly, $B[1,3]$ collides with $B[2,0]$, and $B[2,3]$ collides with $B[3,0]$ in virtual data link 2. □

Example 6: Consider the 3-nested-loop algorithm in Example 2 in Section 2. Then, the linear array implementation $\mathbf{H} = (1,2,2)$ and $\mathbf{S} = (1,1,-1)$ for the problem size $n = 3$ is feasible. This is because $\mathbf{H}$ and $\mathbf{S}$ satisfy Conditions 1 through 3 of Theorem 1; all of the non-zero data-dependence vectors d_1, d_2, and d_3 are of type ONE; and in addition, $\mathbf{S}d_1 = 1$, $\mathbf{S}d_2 = 1$, and $\mathbf{S}d_3 = -1$. Thus, from Corollary 2 or from Corollary 4, there is no data collision in virtual data links 1, 2, and 3. □

Example 5 and Example 6 illustrate the difference between our model and Shang and Fortes' model. It is true that the semantic meaning of the 3-nested-loop matrix multiplication algorithm in Example 1 is different from that of the 3-nested-loop algorithm in Example 2. In the former case, all data dependence vectors are of type INFINITE. Therefore, if $(\mathbf{H},\mathbf{S})$ is a feasible systolic array implementation and $\mathbf{S}d_i \neq \vec{0}_q$, input data tokens can be pipelined entering into the systolic arrays, output data tokens can be pipelined extracted from the systolic arrays, the I/O time can be overlaid by the computation time, and in addition, each processing element can contain only a constant number of shift registers for data link i. On the other hand, in the latter case, all data dependence vectors are of type ZERO and of type ONE. Therefore, input data tokens with data dependence vectors of type ZERO must be loaded into the systolic arrays before the computation, and output data tokens with data dependence vectors of type ZERO must be unloaded from the systolic array after the computation. The I/O time and the computation time cannot be overlaid; moreover, each processing element must contain a sufficiently large local memory for storing data tokens with data dependence vectors of type ZERO. Although some may argue that if each PE contains one I/O port which directly connects to the host computer, then a sufficiently large local memory is not necessary. However, this hardware model may be very costly. On the contrary, in our systolic array model, only boundary PEs have I/O ports.

In effect, Shang and Fortes' communication model is different from ours. Therefore, it may be difficult to compare these two models. Shang and Fortes divided the whole execution process into two phases. An active phase: when data tokens go through a chain of computations; and a passive phase: when data tokens move from input peripheral processors before the active phase or to output peripheral processors after the active phase. Therefore, they considered collisions in data links only between data tokens in the active phase. Using our wording and methodology, they reduced type-INFINITE data-dependence vectors to type-ONE and type-ZERO data-dependence vectors. The analysis of hardware requirements of these two cases has been discussed in the last paragraph. Consequently, their necessary and sufficient conditions for a correct mapping only include the first three conditions in Theorem 3. Thus, under their model, $\mathbf{H} = (1, 2, 2)$ and $\mathbf{S} = (1, 1, -1)$ in Example 5 is a feasible systolic array implementation. This is because for example, in Figure 3, $B[1, 0]$ is using virtual data link 2 from time step 2 to 5, then no computations will need $B[1, 0]$ again. Hence, after that, one can use virtual data link 2 to pipeline $B[0, 3]$ assuming that $B[0, 3]$ can be input at processing element PE_3 (Note that PE_3 is not a boundary PE), or $B[0, 3]$ can be loaded to PE_3 before any computation if each PE has a sufficiently large local memory for storing these pre-loaded data. Similarly, $B[2, 0]$ and $B[1, 3]$ are active at different time steps; and $B[3, 0]$ and $B[2, 3]$ are active at different time steps.

Ganapathy and Wah, who based their work on Li and Wah's method [15], used parameter methods to synthesize systolic array implementations. As mentioned in Section 1, in their model, tokens flowing through systolic arrays cannot change directions, and it only allows at most one data token flowing from one PE to a neighboring PE in each virtual data link at each time step, both as the cases in [14]. In addition, their model does not classify the tokens' behavior. And finally, their method can only deal with algorithms whose data dependence vectors are all of type INFINITE; they do not consider algorithms whose data dependence vectors are all of type ZERO and of type ONE such as the 3-nested-loop algorithm in Example 2. However, because they only dealt with problems whose data-dependence vectors are all of type INFINITE, they could design fast polynomial-time algorithms for finding optimal space and time mappings. In addition, they provided certain conditions for dealing with shift-varying algorithms such as the reindexed transitive closure algorithm [10].

6 Conclusions

We have presented in this paper two sets of necessary and sufficient conditions for mapping nested-loop algorithms with shift-invariant uniform-dependence relations into grid-connected systolic arrays of arbitrary dimensions without data collisions in the data links. The first set of conditions is under the constraint that at each time step, in each virtual data link between two neighboring PEs there is at most one data token which can pass through. The second set of conditions is under a general model in which a cyclic shuffling algorithm is used to handle traveling tokens, and therefore, more than one data token is allowed to flow from one PE to a neighboring PE at each time step. These two sets of conditions, which are based on the ZERO-ONE-INFINITE classification of tokens' behavior, can be used to generate all feasible grid-connected systolic array implementations. In addition, when the size of the problem to be solved grows and the size of the provided systolic array is less than the size which a space mapping needs, then the partition method presented by Moldovan and Fortes can be used to handle this case [18]. However, although systolic array algorithms based on the second set of conditions are more general than systolic array algorithms based on the first set of conditions, the former systolic array algorithms require more shift registers than the latter ones.

Acknowledgements

Part of this paper was presented at the Sixth ISCA International Conf. on Parallel and Distributed Computing Systems. The author would like to thank Cheng-Fong Chen for his valuable discussions on avoiding data collisions and for preparing three figures. The author also would like to thank Prof. B. W. Wah, Prof. H. A. Sholl, and the referees of the ISCA ICPADCS and of this journal for their helpful comments that improved the quality and the readability of this paper.

Appendix

Theorem 8 : *A q-D systolic array algorithm* $(\mathbf{H}, \mathbf{S})$ *that maps correctly a p-nested-loop algorithm into a q-D grid-connected systolic array must satisfy Conditions* 1 *through* 4.

Proof: Condition 1 follows from Lamport's result [11]. Condition 2 is related to the non-singularity condition of our previous paper [14], Moldovan and Fortes [18], and others. We now show that Condition 3 is necessary. From the restriction of systolic arrays, tokens must flow through data links at a constant speed. Suppose that a variable with the data-dependence vector d_i is generated in index $\bar{I}$ and will be used next time in index $\bar{I} + d_i$. Index $\bar{I}$ is executed in $PE_{\mathbf{S}\bar{I}}$ at time $\mathbf{H}\bar{I}$, and the index $\bar{I} + d_i$ will be executed in $PE_{\mathbf{S}(\bar{I}+d_i)}$ at time $\mathbf{H}(\bar{I} + d_i)$. Suppose that $\mathbf{S}d_i \neq \vec{0}_q$. Then the Manhattan distance from $PE_{\mathbf{S}\bar{I}}$ to $PE_{\mathbf{S}(\bar{I}+d_i)}$ is equal to $\sum_{j=1}^{q} |sd_{ij}|$, because $\mathbf{S}d_i = (sd_{i1},\, sd_{i2},\, \ldots,\, sd_{iq})^t$. Therefore, each token in virtual data link i is delayed by $b_i = \mathbf{H}d_i / \sum_{j=1}^{q} |sd_{ij}|$ units of time in each PE, where b_i must be a positive integer.

We now show that Condition 4 is necessary. Let x_1 and x_2 be two tokens of the variable X corresponding to d_i that is of type INFINITE, such that x_1 is the token generated in $\bar{I}_1$ and x_2 is the token generated in $\bar{I}_2$. Then, x_1 is in $PE_{\mathbf{S}\bar{I}_1}$ at time $\mathbf{H}\bar{I}_1$, x_2 is in $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$, and they cannot be destroyed during the computation. Since $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ for all integers γ, from Lemma 7 in [13], $x_1 \neq x_2$.[2] Since $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i \neq \vec{0}_q$, $\beta \sum_{j=1}^{q} |sd_{ij}|$ is the Manhattan distance from $PE_{\mathbf{S}\bar{I}_1}$ to $PE_{\mathbf{S}\bar{I}_2}$ along the composite interconnection primitives "$\sum_{j=1}^{q} sd_{ij} P_{2j-1}$" β times.

Since $\mathbf{H}(\bar{I}_2 - \bar{I}_1) > 0$, $\mathbf{H}\bar{I}_1 < \mathbf{H}\bar{I}_2$. Let c be the index such that x_1 is in PE_c at time $\mathbf{H}\bar{I}_2$. Then, from Condition 3, and the assumption $\mathbf{S} d_i = \sum_{j=1}^{q} sd_{ij} P_{2j-1} \neq \vec{0}_q$, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = b_i \times$ (the Manhattan distance from $PE_{\mathbf{S}\bar{I}_1}$ to PE_c along the composite interconnection primitives $\sum_{j=1}^{q} sd_{ij} P_{2j-1}$).

Assume by contradiction that $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{H} d_i = b_i(\beta \sum_{j=1}^{q} |sd_{ij}|)$. Then, the Manhattan distance from $PE_{\mathbf{S}\bar{I}_1}$ to PE_c along the composite interconnection primitives $\sum_{j=1}^{q} sd_{ij} P_{2j-1}$ is equal to $\beta \sum_{j=1}^{q} |sd_{ij}|$, which is the Manhattan distance from $PE_{\mathbf{S}\bar{I}_1}$ to $PE_{\mathbf{S}\bar{I}_2}$ along the composite interconnection primitives $\sum_{j=1}^{q} sd_{ij} P_{2j-1}$, because β is a positive integer. We thus have $c = \mathbf{S}\bar{I}_2$. Hence, the tokens x_1 and x_2 both use virtual data link i, and appear in $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$. Therefore, even if we apply the algorithm for shuffling traveling tokens in virtual data link i, tokens x_1 and x_2 both must be handled by the first step of the algorithm. This is a "data collision", which is not allowed. Therefore, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) \neq \beta \mathbf{H} d_i$. □

Theorem 9 : *A q-D systolic array algorithm* $(\mathbf{H}, \mathbf{S})$ *from a p-nested-loop algorithm Ag that satisfies Conditions* 1 *through* 4 *maps Ag correctly into a q-D grid-connected systolic array.*

Proof: First, from Condition 1, $\mathbf{H}$ preserves the data-dependence ordering.

Second, we will show that the "needed" tokens will flow to the right place at the right time. Since all the statements in F_{Ag} can be handled in the same way, in the following we only consider a single representative statement. Such statement can be rewritten so that it has m variables and each variable corresponds to a data-dependence vector, this is because the p-nested-loop algorithm has m data-dependence vectors. Let the single statement be $\hat{F}_{Ag}(v_1, v_2, \ldots, v_m)$, where $\hat{F}_{Ag}$ is a statement in F_{Ag} and v_i, which is with the data-dependence vector d_i and was generated in $\bar{I}_i$ ($\bar{I}_i \in I^p$), is an entry of variable V_i. For simplicity, we assume that only one time unit is needed to execute the loop body. We now show by induction on $\mathbf{H}\bar{I}$ that all tokens v_i arriving at $PE_{\mathbf{S}\bar{I}}$ at time $\mathbf{H}\bar{I}$ will have correct values.

Basic step: $\mathbf{H}\bar{I} = \min\{\mathbf{H}\bar{I}_1 | \bar{I}_1 \in I^p\}$. All tokens v_i are initial input/output tokens (variables). From Condition 3 and the assumption that every variable enters into the systolic array at a correct time, v_i,

[2] Lemma 7 in [13]: Let x_1 and x_2 be tokens of variable X in the data stream corresponding to d_i that is of type INFINITE. Let x_1 be used in some index $\bar{I}_1$ and let x_2 be used in some index $\bar{I}_2$. Then, $x_1 \neq x_2$ if and only if $\bar{I}_2 - \bar{I}_1$ is not an integer multiple of d_i.

which flows through virtual data link i, will reach $PE_{\mathbf{S}\bar{I}}$ at time $\mathbf{H}\bar{I}$. Thus, the systolic array algorithm performs the same function as $\hat{F}_{Ag}$ in $\bar{I}$.

Induction step: $\mathbf{H}\bar{I} > \min\{\mathbf{H}\bar{I}_1 | \bar{I}_1 \in I^p\}$. From the definition of the data-dependence vectors we have

$$\bar{I} = \bar{I}_1 + d_1 = \bar{I}_2 + d_2 = \ldots = \bar{I}_m + d_m, \tag{1}$$

for appropriate $\bar{I}_1, \bar{I}_2, \ldots, \bar{I}_m$. By induction, v_i was regenerated with the correct value in $PE_{\mathbf{S}\bar{I}_i}$ at time $\mathbf{H}\bar{I}_i$. However, v_i was not used (and was not modified) during the time instances $\mathbf{H}\bar{I}_i + 1$, $\mathbf{H}\bar{I}_i + 2$, $\ldots$, $\mathbf{H}(\bar{I}_i + d_i) - 1$. Then from Condition 3, v_i, which flows through virtual data link i, will reach $PE_{\mathbf{S}\bar{I}_i + \mathbf{S}d_i}$ at time $\mathbf{H}\bar{I}_i + \mathbf{H}d_i$. But from (1), $\mathbf{S}\bar{I} = \mathbf{S}(\bar{I}_i + d_i)$ and $\mathbf{H}\bar{I} = \mathbf{H}(\bar{I}_i + d_i)$. Therefore, all of the tokens $v_1, v_2, \ldots, v_m$ arriving at $PE_{\mathbf{S}\bar{I}}$ at time $\mathbf{H}\bar{I}$ will have correct values. The systolic array algorithm will thus perform the same function as $\hat{F}_{Ag}$ in $\bar{I}$.

Third, we will show that no two tokens (variables) collide in any data link. If $\mathbf{S}d_i = \vec{0}_q$, the virtual data link i is fixed in each PE and tokens with the data-dependence vector d_i are stored in local registers (or local memory). Therefore, in this case, tokens will not collide with each other in virtual data link i. In the following we consider the case when $\mathbf{S}d_i \neq \vec{0}_q$. Assume by contradiction that virtual data link i has two distinct tokens x_1 and x_2 of the variable X that collide in PE_a during the execution. (Thus, of course, d_i is with X.) Suppose that x_1 was just generated in index $\bar{I}_1$ in $PE_{\mathbf{S}\bar{I}_1}$ at time $\mathbf{H}\bar{I}_1$ and x_2 was just generated in index $\bar{I}_2$ in $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$. (If x_1 or x_2 was not generated before, then x_1 is then immediately used in index $\bar{I}_1$ or x_2 is then immediately used in index $\bar{I}_2$.) Consider two cases:

1. $\mathbf{H}\bar{I}_1 = \mathbf{H}\bar{I}_2$.

Our assumption is that at time instance $\mathbf{H}\bar{I}_1 = \mathbf{H}\bar{I}_2$, x_1 is in $PE_{\mathbf{S}\bar{I}_1}$ and x_2 is in $PE_{\mathbf{S}\bar{I}_2}$, and then at some time instance x_1 and x_2 collide in PE_a. However, since all tokens of data stream i flow through virtual data link i at the same speed, we have $PE_{\mathbf{S}\bar{I}_1} = PE_{\mathbf{S}\bar{I}_2}$ which contradicts Condition 2.

2. $\mathbf{H}\bar{I}_1 \neq \mathbf{H}\bar{I}_2$.

Without loss of generality, $\mathbf{H}\bar{I}_1 < \mathbf{H}\bar{I}_2$. Since x_1 and x_2 flow at the same speed, x_1 flowing from $PE_{\mathbf{S}\bar{I}_1}$ at time $\mathbf{H}\bar{I}_1$ will reach PE_a at some time instance t and x_2 flowing from $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$ will reach PE_a at the time instance t. Therefore, x_1 will reach $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$. As all tokens flow with non-zero velocity, $\mathbf{S}\bar{I}_1 \neq \mathbf{S}\bar{I}_2$. Consider two subcases:

2(a). $\bar{I}_2 - \bar{I}_1 = \gamma d_i$ for some integer γ.

Consider two cases:

i. d_i is a type ONE data-dependence vector.

Since x_1 is generated in $\bar{I}_1$, x_1 is only used in $\bar{I}_1 + d_i$. However, since x_1 is generated in $PE_{\mathbf{S}\bar{I}_1}$

and $\bar{I}_2 \neq \bar{I}_1$ and $\bar{I}_2 - \bar{I}_1 = \gamma d_i$ for some integer γ, we have $\bar{I}_2 = \bar{I}_1 + d_i$ and $PE_a = PE_{\mathbf{S}\bar{I}_2}$. From the definition of type ONE data-dependence vector, x_1 will not be used again after it is used in $\bar{I}_2$. Therefore, when x_2 is generated in $PE_{\mathbf{S}(\bar{I}_1+d_i)} = PE_{\mathbf{S}\bar{I}_2}$, x_1 is destroyed. Thus, no collisions will occur.

ii. d_i is a type INFINITE data-dependence vector.

From Lemma 7 in [13], $x_1 = x_2$. However, this contradicts our assumption that $x_1 \neq x_2$.

2(b). $\bar{I}_2 - \bar{I}_1 \neq \gamma d_i$ for all integers γ.

Consider two cases:

i. $\mathbf{S}(\bar{I}_2 - \bar{I}_1) = \beta \mathbf{S} d_i$ for some integer β.

Since $\mathbf{S}\bar{I}_1 \neq \mathbf{S}\bar{I}_2$ and $\mathbf{S}d_i \neq \vec{0}_q$, β is non-zero. Suppose that β is a negative integer. Because our assumption is that $\mathbf{H}\bar{I}_1 < \mathbf{H}\bar{I}_2$, thus at time instance $\mathbf{H}\bar{I}_2$, the Manhattan distance between tokens x_1 and x_2 is at least $\beta \sum_{j=1}^{q} |sd_{ij}|$. However, this contradicts our derivation result that x_1 will reach $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$.

Suppose that β is a positive integer. Consider two subcases:

i(a). d_i is a type ONE data-dependence vector.

Since d_i is a type ONE data-dependence vector, token x_1, which is generated in $PE_{\mathbf{S}\bar{I}_1}$, can be destroyed after it is used in $PE_{\mathbf{S}(\bar{I}_1+d_i)}$. On the other hand, token x_2 is generated in $PE_{\mathbf{S}\bar{I}_2} = PE_{\mathbf{S}(\bar{I}_1+\beta d_i)}$ and is used in $PE_{\mathbf{S}(\bar{I}_1+(\beta+1)d_i)}$, where $\beta \geq 1$. Therefore, tokens x_1 and x_2, which flow through different PEs, will not collide with each other in virtual data link i.

i(b). d_i is a type INFINITE data-dependence vector.

Because d_i is a type INFINITE data-dependence vector, tokens x_1 and x_2 cannot be destroyed during the execution. Since x_1 is in $PE_{\mathbf{S}\bar{I}_1}$ at time $\mathbf{H}\bar{I}_1$ and is in $PE_{\mathbf{S}\bar{I}_2}$ at time $\mathbf{H}\bar{I}_2$, and therefore, $\mathbf{H}(\bar{I}_2 - \bar{I}_1) = b_i(\beta \sum_{j=1}^{q} |sd_{ij}|) = \beta \mathbf{H} d_i$. However, this contradicts Condition 4.

ii. $\mathbf{S}(\bar{I}_2 - \bar{I}_1) \neq \beta \mathbf{S} d_i$ for all integers β.

Although both x_1 and x_2 may be in the same PE at the same time, we can apply the algorithm for shuffling traveling tokens. Then, tokens x_1 and x_2 are handled at different steps of the algorithm. Thus, tokens x_1 and x_2 will not collide with each other in virtual data link i.

We conclude that $(\mathbf{H}, \mathbf{S})$ can be performed in parallel. $\square$

References

[1] M. A. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu, and J. A. Webb. The Warp computer: Architecture, implementation, and performance. *IEEE Trans. Comput.*, C-36:1523–1538, December 1987.

[2] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, et al. iWarp: An integrated solution to high-speed parallel computing. In *Proc. Supercomputing '88*, pages 330–339, Orlando, Florida, November 1988.

[3] P. R. Cappello and K. Steiglitz. *Advances in Computing Research*, volume 2: VLSI Theory, chapter Unifying VLSI Array Design with Linear Transformations of Space-Time, pages 23–65. JAI Press, Greenwich, CT, 1984.

[4] M. C. Chen. The generation of a class of multipliers: Synthesizing highly parallel algorithms in VLSI. *IEEE Trans. Comput.*, C-37:329–338, March 1988.

[5] J. A. B. Fortes, K. S. Fu, and B. W. Wah. Systematic approaches to the design of algorithmic specified systolic array. In *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pages 300–303, Tampa, FL, March 1985.

[6] K. N. Ganapathy and B. W. Wah. Synthesizing optimal lower dimensional processor arrays. In *Proc. International Conf. on Parallel Processing*, pages III–96–103, August 1992.

[7] K. N. Ganapathy and B. W. Wah. Optimal synthesis of algorithm-specific lower-dimensional processor arrays. Technical Report UILU-ENG-93-2246, Center for Reliable and High-Performance Computing, Univ. of Illinois at Urbana-Champaign, 1993.

[8] S. Y. Kung. On supercomputing with systolic/wavefront array processors. *Proc. of the IEEE*, 72:867–884, July 1984.

[9] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, N.J., 1988.

[10] S. Y. Kung, S. C. Lo, and P. S. Lewis. Optimal systolic design for the transitive closure and the shortest path problems. *IEEE Trans. Comput.*, C-36:603–614, May 1987.

[11] L. Lamport. The parallel execution of Do loops. *Communications of the ACM*, pages 83–93, Feb 1974.

[12] P.-Z. Lee and C. F. Chen. Mapping for-loop algorithms into grid-connected systolic arrays. In *Proc. The Sixth ISCA International Conference on Parallel and Distributed Computing Systems*, pages 381–386, Louisville, Kentucky, October 1993. ISCA.

[13] P.-Z. Lee and Z. M. Kedem. Synthesizing linear-array algorithms from nested For loop algorithms. *IEEE Trans. Comput.*, C-37:1578–1598, December 1988.

[14] P.-Z. Lee and Z. M. Kedem. Mapping nested loop algorithms into multi-dimensional systolic arrays. *IEEE Trans. Parallel Distributed Syst.*, 1:64–76, January 1990.

[15] G. Li and B. W. Wah. The design of optimal systolic arrays. *IEEE Trans. Comput.*, pages 66–77, January 1985.

[16] W. L. Miranker and A. Winkler. Spacetime representations of computational structures. *Computing*, 32:93–114, 1984.

[17] D. I. Moldovan. ADVIS: A software package for the design of systolic arrays. *IEEE Trans. CAD*, CAD-6:33–40, January 1987.

[18] D. I. Moldovan and J. A. B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Trans. Comput.*, C-35:1–12, January 1986.

[19] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrence equations. In *11th Annu. Symp. Comput. Architecture*, pages 208–214, 1984.

[20] P. Quinton. Mapping recurrences on parallel architectures. In *Proc. The Third International Conference on Supercomputing*, Boston, Massachussets, May 15–20 1988.

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

[22] I. V. Ramakrishnan, D. Fussell, and A. Silberschatz. Mapping homogeneous graphs on linear arrays. *IEEE Trans. Comput.*, C-35(3):198–209, March 1986.

[23] S. K. Rao and T. Kailath. Regular iterative algorithms and their implementation on processor arrays. *Proc. of the IEEE*, 76(3):259–269, March 1988.

[24] H. B. Ribas. *Automatic Generation of Systolic Programs from Nested Loops*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, June 1990.

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

[26] Y. Wong and J. Delosme. Optimal systolic implementations of n-dimensional recurrences. In *Proc. International Conf. on Computer Design*, pages 618–621. IEEE, 1985.

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

[28] Y. Yaacoby and P. R. Cappello. Scheduling a system of nonsingular affine recurrence equations onto a processor array. *Journal of VLSI Signal Processing*, 1:115–125, 1989.