International Journal of Computer Mathematics, 81(3/4), 2001

Space-Time Equations for Non-Unimodular Mappings

Jingling Xue
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia

Patrick Lenders
School of Mathematics and Computer Sciences
University of New England
Armidale, NSW 2351, Australia

Abstract. The class of systems of uniform recurrence equations (UREs) is closed under unimodular transformations. As a result, every systolic array described by a unimodular mapping can be specified by a system of space-time UREs, in which the time and space coordinates are made explicit. As non-unimodular mappings are frequently used in systolic designs, this paper presents a method that derives space-time equations for systolic arrays described by non-unimodular mappings. The space-time equations for non-unimodular mappings are known elsewhere as sparse UREs (SUREs) because the domains of their variables are sparse and their data dependences are uniform. Our method is compositional in that space-time SUREs can be further transformed by unimodular and non-unimodular mappings, allowing a straightforward implementation in systems like ALPHA. Specifying a systolic design by space-time equations has two advantages. First, the space-time equations exhibit all useful properties about the design, allowing the design to be formally verified. Second, depending on the application area and performance requirement, the space-time equations can be realised as custom VLSI systems, FPGAs, or programs to be run on a parallel computer.

Keywords: Systolic array, multirate array, recurrence equations, space-time mapping.

C.R. Categories: B.7.1 C.1.2

1 Introduction

The problem of synthesising systolic arrays from uniform recurrence equations (UREs) [4] is well-understood and various design methodologies have been proposed for its solution over the last two decades [11, 13, 14, 16, 20]. The main technique consists of finding a non-singular mapping to transform the original index space to a space-time domain, i.e., assigning a time and place to each point in the original index space. Such a *space-time mapping* (or *mapping* for short) must satisfy several design constraints to be valid, including the causality constraint ensuring that the original data dependences are respected [10, 12].

The class of systems of UREs is closed under unimodular mappings. As a result, every systolic array described by a unimodular mapping can be readily specified by a system of space-time UREs, in which the time or space coordinates are explicit. Section 3 recalls how such a system of space-time equations can be obtained from the original program by applying a so-called domain morphism in Crystal [3]. Applying this idea directly to a non-unimodular mapping, however, does not yield a system of well-formed equations because the domains of equations are sparse and cannot be specified by dense convex polyhedra.

This paper presents a method for deriving space-time equations for systolic arrays designed by non-unimodular mappings. The basic idea is to decompose a non-unimodular mapping into a unimodular domain morphism followed by a scaling transformation. Our method is compositional in that space-time equations can be further transformed by unimodular and non-unimodular mappings, allowing a straightforward implementation in transformational systems like ALPHA [7]. In addition, our method handles unimodular mappings as a special case.

This work is useful for three reasons. First, non-unimodular mappings are frequently used in the synthesis of systolic designs. For example, the mapping that describes Kung-Leiserson's array for matrix multiplication is non-unimodular. In addition, all multirate arrays as defined in [16] are described by non-unimodular mappings. Our method can derive space-time equations for all systolic arrays in the traditional sense [10, 12] and all multirate arrays as defined in [16]. Second, space-time equations provide a precise specification of systolic designs, allowing the designs to be realised as custom VLSI systems, as FPGAs, or as programs to be run on a general purpose parallel architecture. In particular, since wavefront arrays are the asynchronous version of multirate arrays according to S. Y. Kung [6, p. 244], the space-time equations for a multirate array can also be translated to a program to be run in a wavefront array. Third, the space-time equations for a design exhibit all useful properties about the design, allowing the design to be formally verified and further transformed. Two important properties about a non-unimodular design are the *period* of the array [16] and the *phase* in which a processor is active [8].

The objective of this work is to derive space-time equations for non-unimodular mappings. It suffices to use the familiar matrix multiplication as an example for illustrations.

The rest of the paper is organised as follows. Section 2 introduces sparse UREs and Crystal's domain morphism. Section 3 reviews how to derive space-time equations for unimodular mappings. Section 4 describes our method to deal with non-unimodular mappings. The space-time equations for Kung-Leiserson's array are given. Section 5 applies our method to derive space-time equations for two multirate array realisations of matrix multiplication. Section 6 describes the related work. Section 7 concludes the paper.

2 SUREs, UREs and Domain Morphism

For a classification of various forms of recurrence equations used in systolic designs, we refer to [8]. For the purposes of this paper, it suffices to consider a generic system of *sparse UREs* (SUREs), $\mathcal{P}_{\text{sure}}$, such that each variable X is defined by an input equation (IE), a computation equation (CE) and an output equation (OE) as follows:

$$\mathcal{P}_{\text{sure}}: \boxed{\begin{array}{rlcl} \text{IE:} & z \in D_{\text{i}} & \rightarrow & X(Mz) = x(g(z)) \\ \text{CE:} & z \in D_{\text{c}} & \rightarrow & X(Mz) = f(\cdots, Y(Mz - \vartheta), \cdots) \\ \text{OE:} & z \in D_{\text{o}} & \rightarrow & x(h(z)) = X(Mz) \end{array}} \tag{1}$$

where $z \in \mathbb{Z}^n$ is an *index point*, $M \in \mathbb{Z}^{n\times n}$ is a non-singular integer matrix, D_{i}, D_{c} and D_{i} are the *domains* of the three equations, respectively, $\vartheta \in \mathbb{Z}^n$ is a constant *dependence vector*, g and h are functions from $\mathbb{Z}^n$ to $\mathbb{Z}^m$ for some m and f is a strict, single-valued function. In the computation equation, Y is a variable not necessarily distinct from X and the dots "$\cdots$" indicate the arguments of the same syntax. The *index space* of the entire system $\mathcal{P}_{\text{sure}}$ is defined to be the union of the domains of all computation equations.

The domains of the three equations are (dense) convex polyhedra of the form:

$$D_x \quad = \quad \{z \mid A_x z \leqslant b_x\}, \quad \text{where } x \in \{\text{i}, \text{c}, \text{o}\} \tag{2}$$

The *domain of variable* X, i.e., $\{Mz \mid z \in D_{\text{c}}\}$, is *sparse* when M is non-unimodular. $\mathcal{P}_{\text{sure}}$ is *sparse* when the domains of some variables are sparse. $\mathcal{P}_{\text{sure}}$ becomes a system of UREs when M is the identity matrix in all its equations. Then, we write $\mathcal{P}_{\text{sure}}$ as $\mathcal{P}_{\text{ure}}$:

$$\mathcal{P}_{\text{ure}}: \boxed{\begin{array}{lcl} z \in D_{\text{i}} & \rightarrow & X(z) = x(g(z)) \\ z \in D_{\text{c}} & \rightarrow & X(z) = f(\cdots, Y(z - \vartheta), \cdots) \\ z \in D_{\text{o}} & \rightarrow & x(h(z)) = X(z) \end{array}} \tag{3}$$

Crystal [3] is a functional language based on generalised systems of recurrence equations. One of the basic transformations in Crystal is called a domain morphism. In this paper, it is only necessary to consider the domain morphisms that are non-singular linear transformations on systems of SUREs (including UREs as a special case).

Let $T \in \mathbb{Z}^{n\times n}$ be a non-singular mapping from the original index space of $\mathcal{P}_{\text{sure}}$ to a new index space. Let $\mathsf{dom}(T, \mathcal{P}_{\text{sure}})$ be the equivalent system of SUREs obtained from applying the domain morphism T to the original program $\mathcal{P}_{\text{sure}}$. We have:

$$\mathsf{dom}(T,\mathcal{P}_{\text{sure}}): \boxed{\begin{array}{lcl} z \in T(D_{\text{i}}) & \rightarrow & X(TMT^{-1}z) = x(g(T^{-1}z)) \\ z \in T(D_{\text{c}}) & \rightarrow & X(TMT^{-1}z) = f(\cdots, Y(TMT^{-1}z - T\vartheta), \cdots) \\ z \in T(D_{\text{o}}) & \rightarrow & x(h(T^{-1}z)) = X(TMT^{-1}z) \end{array}} \tag{4}$$

where $T(D_x) = \{Tz \mid z \in D_x\}, \forall\, x \in \{\text{i}, \text{c}, \text{o}\}$

In the case of $\mathcal{P}_{\text{ure}}$, we have:

$$\text{dom}(T,\mathcal{P}_{\text{ure}}): \boxed{\begin{array}{lcl} z \in T(D_{\text{i}}) & \to & X(z) = x(g(T^{-1}z)) \\ z \in T(D_{\text{c}}) & \to & X(z) = f(\cdots, Y(z - T\vartheta), \cdots) \\ z \in T(D_{\text{o}}) & \to & x(h(T^{-1}z)) = X(z) \end{array}} \tag{5}$$

$$\text{where } T(D_x) = \{Tz \mid z \in D_x\}, \forall\, x \in \{\text{i}, \text{c}, \text{o}\}$$

3 Unimodular Mappings

Before being mapped to systolic arrays, affine recurrence equations (AREs) must be first transformed into UREs [13]. So we are only concerned with deriving space-time equations for systolic arrays synthesised from UREs.

The synthesis of a systolic array from a system of UREs consists of finding two separate functions. The *timing function* $t(z) = \lambda z$, where $\lambda \in \mathbb{Z}^n$, specifies that the index point z is computed at the time step λz. The *allocation function*, defined usually by a *projection direction* $u = (u_1, \cdots, u_n) \in \mathbb{Z}^n$ such that $\gcd(u_1, \cdots, u_n) = 1$, specifies that two index points z and z' are mapped to the same PE iff $z = z' \pm \alpha u$, where $\alpha \in \mathbb{Z}^n$.

The two functions can be collectively specified as a single space-time mapping:

$$T \quad = \quad \begin{bmatrix} \lambda \\ P \end{bmatrix} \tag{6}$$

where $P \in \mathbb{Z}^{(n-1)\times n}$ has full-row rank and satisfies $Pu = 0$. P, the *allocation matrix*, specifies that the index point z is executed at the PE Pz. Two different allocation matrices P_1 and P_2 such that $P_1 u = P_2 u = 0$ define the same array; they do not change the topology of the array but only modify the processor coordinates (i.e., labels) assigned to the PEs.

T satisfies all data dependences of the program if $\lambda\vartheta \geqslant 1$ for every dependence vector ϑ in the program. This assumes that evaluation of an index point takes one unit time. A relaxation of this assumption will allow multirate arrays to be designed, as discussed in Section 5. T must also satisfy the condition $\det(T) \neq 0$, i.e., $\lambda u \neq 0$, ensuring that two index points mapped to the same time step are not mapped to the same PE.

A unimodular mapping T transforms the original program $\mathcal{P}_{\text{ure}}$ to dom(T,$\mathcal{P}_{\text{ure}}$). The domains of equations in (5) are all convex polyhedra and can then be specified as follows:

$$T(D_x) \quad = \quad \{z \mid A_x T^{-1} z \leqslant b_x\} \tag{7}$$

Thus, the space-time equations in dom(T,$\mathcal{P}_{\text{ure}}$) are well-formed. They are the space-time equations for the systolic array described by T.

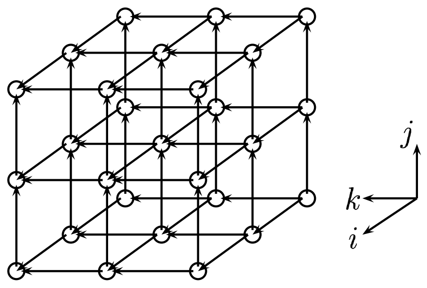


Fig. 1: The dependence graph for matrix multiplication ($N = 3$).

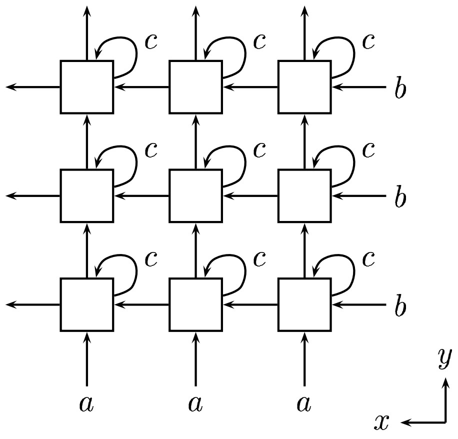


Fig. 2: S. Y. Kung's systolic array $(N = 3)$.

Let us consider matrix multiplication defined by the following UREs:

$$\begin{array}{rcl}
j = 0, 1 \leqslant i, k \leqslant N & \rightarrow & A(i,j,k) = a(i,k) \\
1 \leqslant i,j,k \leqslant N & \rightarrow & A(i,j,k) = A(i,j-1,k) \\
i = 0, 1 \leqslant j, k \leqslant N & \rightarrow & B(i,j,k) = b(k,j) \\
1 \leqslant i,j,k \leqslant N & \rightarrow & B(i,j,k) = B(i-1,j,k) \\
k = 0, 1 \leqslant i, j \leqslant N & \rightarrow & C(i,j,k) = 0 \\
1 \leqslant i,j,k \leqslant N & \rightarrow & C(i,j,k) = C(i,j,k-1) + A(i,j-1,k)B(i-1,j,k) \\
k = N, 1 \leqslant i, j \leqslant N & \rightarrow & c(i,j) = C(i,j,k)
\end{array}$$

The dependence graph of the UREs is depicted in Figure 1.

S. Y. Kung's array shown in Figure 2 is described by the unimodular mapping:

$$T = \begin{bmatrix} \lambda \\ P \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad T \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} t \\ x \\ y \end{bmatrix}$$

The processor structure is obtained by projecting the dependence graph along $u = (0, 0, 1)$.

The space-time equations for the array are obtained directly from (5) by substitutions:

$$\begin{array}{r}
y=0, 1 \leqslant x, t-x-y \leqslant N \rightarrow A(t,x,y)=a(x,t-x-y) \\
1 \leqslant x, y, t-x-y \leqslant N \rightarrow A(t,x,y)=A(t-1,x,y-1) \\
x=0, 1 \leqslant y, t-x-y \leqslant N \rightarrow B(t,x,y)=b(t-x-y,y) \\
1 \leqslant x, y, t-x-y \leqslant N \rightarrow B(t,x,y)=B(t-1,x-1,y) \\
t-x-y=0, 1 \leqslant x, y \leqslant N \rightarrow C(t,x,y)=0 \\
1 \leqslant x, y, t-x-y \leqslant N \rightarrow C(t,x,y)=C(t–1,x,y)+A(t–1,x,y–1)B(t–1,x–1,y) \\
t-x-y=N, 1 \leqslant x, y \leqslant N \rightarrow c(x,y)=C(t,x,y)
\end{array}$$

where t is the time coordinate and x and y are space coordinates.

Many useful properties about the array can be extracted from these space-time equations. For example, we find that the array consists of N^2 PEs and runs in $(3N-2)\delta_{\text{sys}}$ time, where δ_{sys} is the length of the global clock.

4 Non-Unimodular Mappings

When T is non-unimodular, the space-time equations in dom(T,$\mathcal{P}_{\text{ure}}$) from (5) are not well-formed if the domain of equations are defined as in (7). This is because $T(D_x)$ is not dense, i.e., $T(D_x)$ contains index points that do not correspond to any index points in the original index space.

Our approach to deriving space-time equations for a non-unimodular mapping is to decompose it into a unimodular domain morphism followed by a non-singular scaling transformation that scales the domains of all variables while leaving the domains of all equations unchanged.

A scaling transformation $S \in \mathbb{Z}^{n\times n}$ maps $\mathcal{P}_{\text{sure}}$ to the following program:

$$\text{scale}(S, \mathcal{P}_{\text{sure}}): \boxed{\begin{array}{lcl} z \in D_{\text{i}} & \rightarrow & X(SMz)=x(g(z)) \\ z \in D_{\text{c}} & \rightarrow & X(SMz)=f(\cdots,Y(SMz-S\vartheta)),\cdots) \\ z \in D_{\text{o}} & \rightarrow & x(h(z))=X(SMz) \end{array}}$$

In the case of $\mathcal{P}_{\text{ure}}$, we have:

$$\text{scale}(S, \mathcal{P}_{\text{ure}}): \boxed{\begin{array}{lcl} z \in D_{\text{i}} & \rightarrow & X(Sz)=x(g(z)) \\ z \in D_{\text{c}} & \rightarrow & X(Sz)=f(\cdots,Y(Sz-S\vartheta)),\cdots) \\ z \in D_{\text{o}} & \rightarrow & x(h(z))=X(Sz) \end{array}}$$

In this section, $A_{(n-1)\times n}$ denotes the bottom $(n-1)\times n$ submatrix of $A \in \mathbb{Z}^{n\times n}$.

For the purposes of this paper, the concept of Hermite normal form is defined as follows.

Definition 1 (**Hermite normal form**) An integer matrix of full-row rank is said to be in *Hermite normal form* if it has the form $[0, H]$, where H is an upper triangular, nonnegative square matrix, in which each row has a unique maximum entry, which is located on its main diagonal.

Theorem 1 [17, p. 45] Let A be an integer matrix of full-row rank. There always exists a unimodular matrix V that will bring A to its *Hermite normal form* $[0, H]$ such that $AV = [0, H]$.

Definition 2 (**Extended-Unimodularity [17, p. 267]**) An integer $m \times r$ matrix of full-row rank is *e-unimodular* (the "e" for extended) if the gcd of the determinants of all its $m \times m$ submatrices is 1. An e-unimodular matrix is *unimodular* if it is square.

The main result of this paper is summarised in the following theorem.

Theorem 2 *Let $T = [\begin{smallmatrix}\lambda \\ P\end{smallmatrix}]$ be a space-time mapping of the form (6) defined in Section 3.*

(a) It is always possible to decompose T such that $T = SU$, where S is a upper triangular, nonnegative matrix with $|\lambda u|$ as its top-left element and U is unimodular. In the special case when P is e-unimodular, S has the following form:

$$S = \left[\begin{array}{c|ccc} |\lambda u| & s_2 & \cdots & s_n \\ \hline 0 & 1 & & \\ \vdots & & \ddots & \mathbf{0} \\ 0 & \mathbf{0} & & 1 \end{array}\right] \tag{8}$$

(b) The space-time equations for the array are derived from $\mathcal{P}_{\text{ure}}$ according to

$$\mathsf{dom}(T, \mathcal{P}_{\text{ure}}) = \mathsf{scale}(S, \mathsf{dom}(U, \mathcal{P}_{\text{ure}})) \tag{9}$$

by first applying U as a domain morphism and then S as a scaling transformation. We obtain:

$\mathsf{scale}(S, \mathsf{dom}(U, \mathcal{P}_{\text{ure}}))$*:*

$$\boxed{\begin{array}{l} z \in U(D_{\text{i}}) \rightarrow X(Sz) = x(g(U^{-1}z)) \\ z \in U(D_{\text{c}}) \rightarrow X(Sz) = f(\cdots, Y(Sz - T\vartheta)), \cdots) \\ z \in U(D_{\text{o}}) \rightarrow x(h(U^{-1}z)) = X(Sz) \end{array}} \tag{10}$$

where $U(D_x) = \{z \mid A_x U^{-1} z \in b_x\}, \forall\, x \in \{\text{i}, \text{c}, \text{o}\}$

In the space-time equations, the first subscript function of a variable represents time and the remaining subscript functions, which are **time-invariant** *since S is upper triangular, represent processor coordinates.*

Proof. Let us prove (a). $P \in \mathbb{Z}^{(n-1)\times n}$ has full-row rank. By Theorem 1, there must exist a unimodular matrix $U^{-1} \in \mathbb{Z}^{n \times n}$ such that $PU^{-1} = [0, H]$, where $H \in \mathbb{Z}^{(n-1)\times(n-1)}$ is a non-singular upper triangular matrix. Hence, we have:

$$T = \begin{bmatrix}\lambda \\ P\end{bmatrix} = \begin{bmatrix}\lambda \\ P\end{bmatrix} U^{-1}U = \begin{bmatrix}\lambda U^{-1} \\ PU^{-1}\end{bmatrix} U = \begin{bmatrix}\lambda U^{-1} \\ 0\ H\end{bmatrix} U = SU$$

To prove that the top-left element of $S = \left[\begin{smallmatrix} \lambda U^{-1} \\ 0\ H \end{smallmatrix}\right]$ is $|\lambda u|$, it suffices to show that the first column of U^{-1} is the projection vector $\pm u$. Since $Pu = [0, H]Uu = 0$, where H is a non-singular upper triangular square matrix, we must have $Uu = (\pm 1, 0, \cdots, 0)$ and $U_{(n-1)\times n}u = 0$. That is, $u = U^{-1}(\pm 1, 0, \cdots, 0)^T$. Hence, the first column of U^{-1} is $\pm u$.

Let us prove (b). In the proof of (a), we have already shown that $U_{(n-1)\times n}u = 0$. That is, both T and U share the same projection vector u. Therefore, $\mathsf{dom}(U, \mathcal{P}_{\text{ure}})$ define exactly the same processor space as $\mathsf{dom}(T, \mathcal{P}_{\text{ure}})$. Finally, $T = SU$. By scaling $\mathsf{dom}(U, \mathcal{P}_{\text{ure}})$ to get $\mathsf{scale}(S, \mathsf{dom}(U, \mathcal{P}_{\text{ure}}))$, we ensure that the timing function is applied correctly in the final space-time equations. ■

This theorem is also correct when T is unimodular, in which case $U = T$ and S is the identity matrix. Thus, both (5) and (10) are identical.

The space-time equations derived by our method exhibit all useful properties about a design. In addition to those that are also relevant to a unimodular mapping, four properties particularly pertinent to a non-unimodular mapping are discussed below.

Let the first subscript function in the variable $X(Sz)$ be written explicitly as:

$$|\lambda u| z_1 + s_2 z_2 + \cdots + s_n z_n \tag{11}$$

where $(|\lambda u|, s_2, \cdots, s_n)$ is the top row of S. Let t_{min} and t_{max} be the time steps for the first input and last output, respectively.

1. The *period* of the array is $|\lambda u|$, i.e., the coefficient of z_1. That is, a PE is active every $|\lambda u|$ clock cycles.

2. A PE at the location $S_{(n-1)\times n}z$ is active in the following time steps:

$$\{|\lambda u| t + s_2 z_2 + \cdots + s_n z_n \mid \exists\, t \in \mathbb{Z} : t_{\text{min}} \leqslant |\lambda u| t + s_2 z_2 + \cdots + s_n z_n \leqslant t_{\text{max}}\}$$

3. The PEs in the array can be divided into $|\lambda u|$ groups such that all PEs in the same group can be simultaneously active. The k-th group G_k, where $0 \leqslant k < |\lambda u|$, contains the following PEs:

$$G_k \;=\; \{S_{(n-1)\times n}z \mid (s_2 z_2 + \cdots + s_n z_n) \,\%\, |\lambda u| = k\}$$

These groups are active periodically according to the order:

$$\begin{tikzcd} G_0 \arrow[r] & G_1 \arrow[r] & \cdots \arrow[r] & G_{k-1} \arrow[lll, bend right] \end{tikzcd}$$

We say that the array has $|\lambda u|$ *phases* and the PEs in G_k are active in phase k.

4. The equivalence of all mappings that have the same λ and u is obvious. Let a systolic array be designed using $\lambda = (2, 1, 0)$ and $u = (1, 0, 0)$. Consider three equivalent mappings and their decompositions:

$$T_1 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = S_1U_1 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = S_2U_2 = \begin{bmatrix} 2 & 1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} = S_3U_3 = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}$$

The same array is viewed differently from the perspective of each mapping. Let us use the processor structure described by T_1 as a reference. T_2 changes its shape (but not its topology) by applying a skewing transformation $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, i.e., the bottom-left $(n-1) \times (n-1)$ submatrix of U_2 to its processor space. T_3 also modifies the shape of the array by applying $\begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, i.e., the bottom-left $(n-1) \times (n-1)$ submatrix of U_3. This consists of applying the same skewing transformation as in T_2 followed by interchanging the two processor axes and then reversing the first (new) processor axis. The lower $(n-1) \times (n-1)$ submatrix of S_3, i.e., $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ corresponds to a relabeling of the processor coordinates.

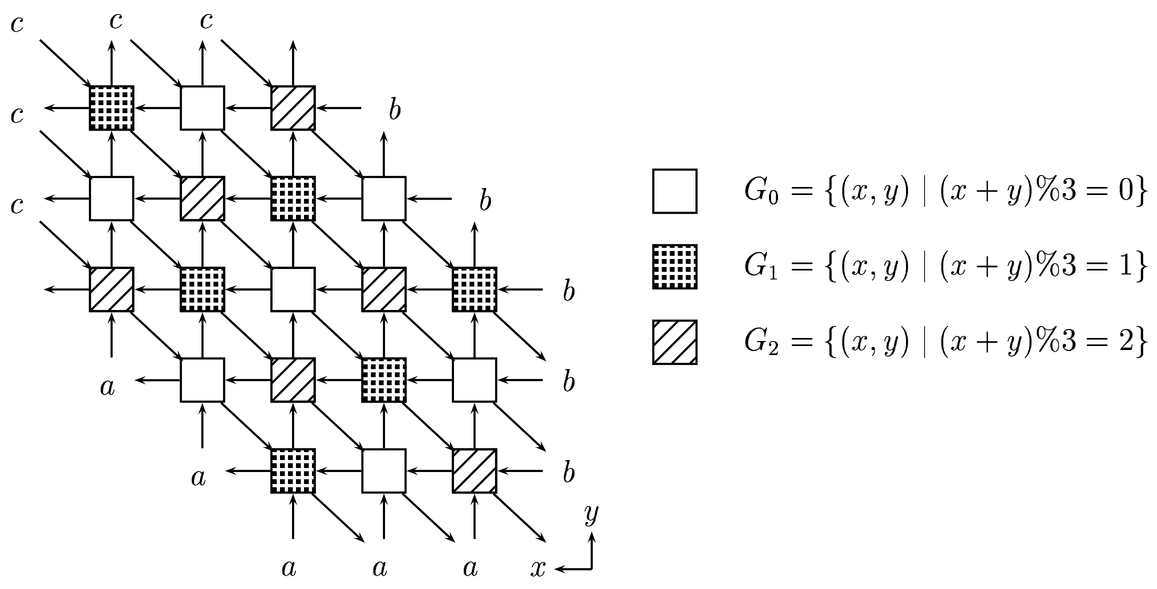
Let us construct the space-time equations for Kung-Leiserson's hexagonally mesh-connected array [5], depicted in Figure 3. Unlike S. Y. Kung's array, this array is designed using the following non-unimodular mapping:

$$T = \begin{bmatrix} \lambda \\ P \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

The processor structure is obtained by projecting the dependence graph along $u = (1, 1, 1)$.

Applying Theorem 2, we decompose T as follows:

$$T = SU = \begin{bmatrix} 3 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$



c
c
c
c
c
b
b
b
b
b
a
a
a
a
a
y
x
$G_0 = \{(x, y) \mid (x + y)\%3 = 0\}$
$G_1 = \{(x, y) \mid (x + y)\%3 = 1\}$
$G_2 = \{(x, y) \mid (x + y)\%3 = 2\}$

Fig. 3: Kung-Leiserson's systolic array for matrix multiplication $(N = 3)$.

To illustrate our method, we derive the space-time equations in two steps. In the first step, we apply the following domain morphism:

$$U \;=\; \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}, \quad U \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} t \\ x \\ y \end{bmatrix}$$

to obtain the following equations:

$$\begin{aligned}
t + y = 0, 1 \leqslant t + x, t \leqslant N &\rightarrow A(t, x, y) = a(t + x, t) \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow A(t, x, y) = A(t, x, y - 1) \\
t + x = 0, 1 \leqslant t + y, t \leqslant N &\rightarrow B(t, x, y) = b(t, t + y) \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow B(t, x, y) = B(t, x - 1, y) \\
t = 0, 1 \leqslant t + x, t + x \leqslant N &\rightarrow C(t, x, y) = 0 \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow C(t, x, y) = C(t-1, x+1, y+1) + A(t, x, y-1)B(t, x-1, y) \\
t = N, 1 \leqslant t + x, t + y \leqslant N &\rightarrow c(t + x, t + y) = C(t, x, y)
\end{aligned}$$

In the second step, we apply S to scale the domains of three variables A, B and C:

$$\begin{aligned}
t + y = 0, 1 \leqslant t + x, t \leqslant N &\rightarrow A(3t + x + y, x, y) = a(t + x, t) \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow A(3t + x + y, x, y) = A(3t + x + y - 1, x, y - 1) \\
t + x = 0, 1 \leqslant t + y, t \leqslant N &\rightarrow B(3t + x + y, x, y) = b(t, t + y) \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow B(3t + x + y, x, y) = B(3t + x + y - 1, x - 1, y) \\
t = 0, 1 \leqslant t + x, t + y \leqslant N &\rightarrow C(3t + x + y, x, y) = 0 \\
1 \leqslant t + x, t + y, t \leqslant N &\rightarrow C(3t + x + y, x, y) = C(3t + x + y - 1, x + 1, y + 1) \\
&\qquad + A(3t + x + y - 1, x, y - 1)B(3t + x + y - 1, x - 1, y) \\
t = N, 1 \leqslant t + x, t + y \leqslant N &\rightarrow c(t + x, t + y) = C(3t + x + y, x, y)
\end{aligned}$$

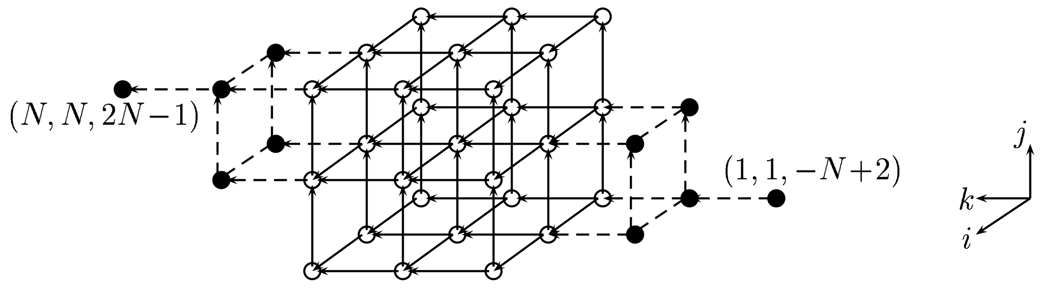


Fig. 4: The extended index space for Kung-Leiserson's array ($N = 3$).

From these equations, we can see clearly that the period of Kung-Leiserson's array is $|\lambda u| = 3$. This means that the array operates in 3 phases. The PEs that can be simultaneously active are shaded identically in Figure 3. The array consists of $3N^2 - 3N + 1$ PEs. If we assume that all I/O are performed at border PEs, the latency of the array can be calculated as follows. Following [16], the index space is extended to find out the point(s) mapped to the first time step and the point(s) mapped to the last time step. By extending the index space as shown in Figure 4, we find that $(1, 1, -N + 2)$ and $(N, N, 2N - 1)$ are mapped to the first and last time steps, respectively. Thus,

$$\begin{aligned} t_{\text{min}} &= (1,1,1)(1,1,-N+2) = -N+4 \\ t_{\text{max}} &= (1,1,1)(N,N,2N-1) = 4N-1 \end{aligned}$$

The latency of the array is $(5N - 4)\delta_{\text{sys}}$. PE (x, y) is active in the time steps:

$$\{3t + x + y \mid \exists\, t \in \mathbb{Z} : -N + 4 \leqslant 3t + x + y \leqslant 4N - 1\}$$

Our method for deriving space-time equations is compositional in the sense that the space-time equations for a mapping can be further transformed by applying unimodular and non-unimodular mappings.

The following theorem describes how a non-singular mapping can be applied to the space-time equations given in (10).

Theorem 3 *Let T_1 and T_2 be two non-singular mappings such that their decompositions are $T_1 = S_1U_1$ and $T_2 = S_2U_2$. Let T_2T_2 be decomposed into $T_2T_1 = SU$. Let $S_2U_2S_1$ be decomposed such that $S_2U_2S_1 = S_3U_3$. Then,*

$$\mathsf{scale}(S, \mathsf{dom}(U, \mathcal{P}_{\text{ure}}) = \mathsf{scale}(S_3U_3S_1^{-1}U_3^{-1}, \mathsf{dom}(U_3, \mathsf{scale}(S_1, \mathsf{dom}(U_1, \mathcal{P}_{\text{ure}}))))$$

Proof. From $S_2U_2S_1 = S_3U_3$, we have $S_2U_2S_1U_1 = S_3U_3U_1$. By using further the fact that $T_2T_1 = S_2U_2S_1U_1 = SU$, we establish that $S_3U_3U_1 = SU$. Because S and S_3 are in Hermite normal form as defined in Definition 1 and U_1, U_3 and U are all unimodular, we must have $S = S_3$ and $U = U_3U_1$. (Note that a diagonal element in S_3 (and S) is the largest of the row containing that element.) This leads to the equality of both sides of the equation. ∎

By this theorem, the space-time equations obtained by applying T_2T_1 as one compound mapping are the same as those obtained by applying T_1 and T_2 individually in that order.

5 Space-Time Equations for Multirate Arrays

In the synthesis of systolic arrays, the evaluation of every equation (i.e., operation) is assumed to take one unit time. As a result, the clock cycle has to be the maximum of these operation times. A *multirate array* is a generalised systolic array, allowing different equations to take different time units to complete by making use of a finer clock cycle. Some discussions about multirate arrays can be found in [6, 16]. S. Y. Kung demonstrated by an example how to design a multirate array using the retiming theorem [6, pp. 243–246]. Rao used integer programming to search for multirate arrays with the maximal efficiency as defined below [16, pp.156–166]:

$$\text{Efficiency} = \frac{\text{the maximum of the evaluation times for all computation equations}}{\lambda u}$$

In this section, we apply our method to derive space-time equations for a multirate array once the corresponding mapping is known.

In the case of matrix multiplication, we assume that all PEs are implemented as serial multiply-accumulators. We further assume that the computation equations for A and B take one unit each and the computation equation for C takes 16 time units. Then Rao formulated the problem of finding efficiency-maximal multirate arrays as follows [16]:

$$\begin{array}{rl} \text{Minimise} & \lambda u \\ \text{Subject to} & \lambda(1,0,0)^T \geqslant 1 \\ & \lambda(0,1,0)^T \geqslant 1 \\ & \lambda(0,0,1)^T \geqslant 16 \\ & \lambda u \geqslant 16 \end{array} \tag{12}$$

There are three dependence vectors in the program. The first three constraints ensure that all three dependence vectors $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ are respected. The last constraint ensures that the time interval for computing two consecutive index points in the same PE is at least 16 time units.

In general, one of the constraints in designing multirate arrays is:

$$\lambda u \geqslant \tau_{\max}$$

where $\tau_{\max}$ is the maximum of the times for evaluating all operations. Thus, all mappings for multirate arrays are non-unimodular.

Let us consider two multirate arrays, one designed using the projection vector for S. Y. Kung's array and one using the projection vector for Kung-Leiserson's array.

5.1 Projection Vector $u = (0, 0, 1)$

In this case, $\lambda = (1, 1, 16)$ is a solution to (12) that achieves 100% efficiency. The array can be described by the mapping:

$$T = \begin{bmatrix} 1 & 1 & 16 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

An application of Theorem 2 will decompose T to:

$$T = SU = \begin{bmatrix} 16 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The space-time equations can be derived as:

$$\begin{array}{rcl}
y = 0, 1 \leqslant x, t \leqslant N & \to & A(16t + x + y, x, y) = a(x, t) \\
1 \leqslant x, y, t \leqslant N & \to & A(t, x, y) = A(16t + x + y - 1, x, y - 1) \\
x = 0, 1 \leqslant y, t \leqslant N & \to & B(16t + x + y, x, y) = b(t, y) \\
1 \leqslant x, y, t \leqslant N & \to & B(16t + x + y, x, y) = B(16t + x + y - 1, x - 1, y) \\
t = 0, 1 \leqslant x, y \leqslant N & \to & C(16t + x + y, x, y) = 0 \\
1 \leqslant x, y, t \leqslant N & \to & C(16t + x + y, x, y) = C(16t + x + y - 16, x, y) \\
 & & +A(16t+x+y-1, x, y-1)B(16t+x+y-1, x-1, y) \\
t = N, 1 \leqslant x, y \leqslant N & \to & c(x, y) = C(16t + x + y, x, y)
\end{array}$$

By extending the index space, we find that the latency of the array is $((1, 1, 16)((N, N, N) - (1, 1, 1)) + 16)\delta_{\text{mul}} = (18N - 2)\delta_{\text{mul}}$, where δ_{mul} is the length of the array's clock. This array is much faster than S. Y. Kung's array since $\delta_{\text{sys}} = 16\delta_{\text{mul}}$. This is because, by allowing different operations to consume different time units, the computations of all elements of C in the multirate array can begin as soon as the respective elements of A and B are available.

Due to the projection $(0, 0, 1)$ used, a PE is responsible for executing one vertical line of the points in the dependence graph (Figure 1). The dependence vector for C in the space-time equations is $(16, 0, 0)$. Since the computation equation for C takes $16\delta_{\text{mul}}$ time units to evaluate, a PE is active in every clock cycle from the time when it begins to compute the first point. Hence, the array has 100% efficiency.

5.2 Projection Vector $u = (1,1,1)$

In this case, there are no multirate arrays achieving 100% efficiency. One solution is to choose the same timing function $\lambda = (1,1,16)$ as before. This gives rise to the following equivalent space-time mapping:

$$T = \begin{bmatrix} 1 & 1 & 16 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

The efficiency of the array is $\frac{16}{\lambda u}\% = 89\%$. Applying Theorem 2 decomposes T to:

$$T = SU = \begin{bmatrix} 18 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Due to the similarities between this mapping and the one for Kung-Leiserson's array, we obtain the following space-time equations, which are the same as those for Kung-Leiserson's array except that the coefficient of t is 18 instead of 3:

$$\begin{aligned}
t+y=0, 1 \leqslant t+x, t \leqslant N &\rightarrow A(18t+x+y, x, y) = a(t+x, t) \\
1 \leqslant t+x, t+y, t \leqslant N &\rightarrow A(18t+x+y, x, y) = A(18t+x+y-1, x, y-1) \\
t+x=0, 1 \leqslant t+y, t \leqslant N &\rightarrow B(18t+x+y, x, y) = b(t, t+y) \\
1 \leqslant t+x, t+y, t \leqslant N &\rightarrow B(18t+x+y, x, y) = B(18t+x+y-1, x-1, y) \\
t=0, 1 \leqslant t+x, t+y \leqslant N &\rightarrow C(18t+x+y, x, y) = 0 \\
1 \leqslant t+x, t+y, t \leqslant N &\rightarrow C(18t+x+y, x, y) = C(18t+x+y–16, x+1, y+1) \\
&\qquad +A(18t+x+y–1, x, y–1)B(18t+x+y-1, x–1, y) \\
t=N, 1 \leqslant t+x, t+y \leqslant N &\rightarrow c(t+x, t+y) = C(18t+x+y, x, y)
\end{aligned}$$

Again by extending the index space, we find that the latency of this multirate array is $((1,1,16)((N,N,2N-1)-(1,1,-N+2))+16)\delta_{\text{mul}} = (48N-34)\delta_{\text{mul}}$, which is faster than Kung-Leiserson's array since $\delta_{\text{sys}} = 16\delta_{\text{mul}}$.

However, it is well-known that we can maximise the throughput of Kung-Leiserson's array by interleaving the execution of multiple instances of matrix multiplication. The time for executing three instances can be calculated to be $(5N-2)\delta_{\text{sys}}$.

The pipelined execution of three instances in the multirate array given above will take three times longer than the execution of a single instance.

Therefore, the multirate version has a lower latency but Kung-Leiserson's array can achieve better throughput by interleaving the execution of multiple instances in the array.

Note that the first subscript function of the rhs C in the computation equation is $18t+x+y-16$. This means that a PE wastes two cycles when evaluating the computation equation for C for two consecutive points allocated to the PE. Hence, the efficiency of the array is $\frac{16}{18}\% = 89\%$.

6 Related Work

Loop transformation and systolic design are two closely related fields. In both fields, a matrix transformation is sought that specifies precisely the parallel code to be generated or the systolic array to be designed. The major focuses in loop transformations are on increasing parallelism, improving data locality and reducing communication overheads. The emphases in systolic design are on minimising the latency, throughput, and processor count of a design.

This work is related to the code generation problem arising in loop transformation, which consists of producing the new loop code to execute the iterations in the original loop code according to the order specified by a given loop transformation. If the loop transformation is unimodular, the new index space is convex since the original index space is convex. The code generation is simple. The new loop code can be generated as described in [1, 18]. If the loop transformation is non-unimodular, the new index space is *not* convex. In this case, the generation of the new loop code is solved in several papers [9, 15, 19]. The basic idea is to obtain the Hermite normal form from the loop transformation matrix and derive from it the new loop bounds and step sizes. The non-unity step sizes serve to skip the *holes* in the new non-convex index space.

In systolic design, many researchers have focused on finding the scheduling vector λ and the projection vector u to describe a systolic array [12, 13, 14, 16]. Although both λ and u can be collectively specified using a single non-singular matrix as in (6), the generation of new space-time equations has been discussed only for unimodular mappings [2, 7, 12, 16]. To the best of our knowledge, no systematic methods for deriving space-time equations under non-unimodular mappings have been reported in the literature. This paper provides a systematic method for solving this problem. Theorem 2 shows that the space-time equations can be obtained from the original equations by first applying a unimodular domain morphism and then a non-unimodular scaling transformation. Theorem 3 shows that our method is compositional so that the space-time equations can be further transformed by unimodular and non-unimodular mappings.

7 Conclusion

In this paper, we have presented a method for deriving space-time equations for systolic arrays described by non-unimodular space-time mappings. Our method allows both unimodular and non-unimodular mappings to be treated in a unified manner. The space-time equations provide a precise specification of systolic designs, allowing them to be formally manipulated. Depending on the application area and performance requirement, these space-time equations can be realised in a variety of ways, as VLSI custom systems, as FPGAs and as programs to be run on general purpose parallel computers.

In presenting our method, we have assumed all variables are scheduled by the same linear timing function. In the general case, every variable V is scheduled by an affine timing function of the form $t_V(z) = \lambda z + a_V$. Since the effect of the affine constant a_V is to perform a translation on the domain of variable V, our method applies in this case as well. For a similar reason, our method also works in the case when the allocation matrix is affine.

References

[1] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundation*. Kluwer Academic Publishers, 1993.

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

[3] Marina C. Chen. A parallel language and its compilation to multiprocessor machines or VLSI. In *ACM Symposium on Principles of Programming Languages*, pages 131–139, St. Petersburg Beach, Fla., Jan 1986.

[4] 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.

[5] 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.

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

[7] H. Le Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *J. VLSI Signal Processing*, 3:173–182, 1991.

[8] P. Lenders and S. Rajopadhye. Multirate VLSI arrays and their synthesis. *IEEE Trans. on Computers*, C-40(5):515–529, May 1997.

[9] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matric es. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages*

and Compilers for Parallel Computing, Lecture Notes in Computer Science 757, pages 391–405. Springer-Verlag, 1992.

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

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

[12] 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.

[13] 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.

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

[15] J. Ramanujam. Non-unimodular transformations of nested loops. In *Supercomputing '92*, pages 214–223. IEEE Computer SocietyPress, 1992.

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

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

[18] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452–471, Oct. 1991.

[19] J. Xue. Automating non-unimodular loop transformations for massive parallelism. *Parallel Computing*, 20(5):711–728, 1994.

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