Some New Designs of 2-D Array for Matrix Multiplication and Transitive Closure

Jong-Chuang Tsay and Pen-Yuang Chang

Abstract—**In this paper, we present some new regular iterative algorithms for matrix multiplication and transitive closure. With these algorithms, by spacetime mapping the 2-D arrays with $2N-1$ and $\lceil(3N-1)/2\rceil$ execution times for matrix multiplication can be obtained. Meanwhile, we can derive a 2-D array with $4N-2$ execution time for transitive closure based on the sequential Warshall-Floyd algorithm. All these new 2-D arrays for matrix multiplication and transitive closure have the advantages of faster and more regular than other previous designs.**

Index Terms—**Algorithm mapping, matrix multiplication, mesh array, systolic array, spherical array, transitive closure, VLSI architecture.**

I. Introduction

SYSTOLIC ARRAY [1] has been proposed over a decade. Roughly speaking, it is a special purpose parallel device composed of several processing elements (PE's) whose interconnections have the properties of regularity and locality. With these properties, systolic architecture is very suitable for VLSI technology.

In spacetime mapping design methodology, the first step is regularization [2], [3]. By regularization we mean to rewrite the sequential algorithm to a regular iterative algorithm, RIA [2] (or uniform recurrent equations, URE's [4]). Dependence graph (DG) is a graphical representation of a RIA and the length of the longest path in this graph is the minimal execution time needed for executing the RIA.

The next step of designing systolic array is to determine a proper schedule vector and its corresponding compatible processor assignment matrix to meet the constraints of data availability and processor availability. That is, a proper schedule must satisfy the precedence constraints imposed by the DG and must also ensure that no two different computations are executed at the same processor at the same time.

The well-known schedules for array processors are linear schedule, uniform affine schedule and affine schedule [2]. A linear schedule is that all variables are with the same schedule vector. In addition to the same schedule vector, the uniform affine schedule can have different translation parts for variables. The affine schedule is more general. It can have different schedule vectors and translation parts for variables. If two indexes are executed at the same time, they are on the same hyperplane. For the entire index space, we may draw several parallel iso-temporal hyperplanes. The schedule vector is in the direction normal to these hyperplanes.

We use a transformation matrix (T) to represent the spacetime mapping. The first row of this matrix is the schedule vector (Λ^T) and the remaining rows are the processor assignment matrix (S^T). The result of multiplication of transformation matrix and dependence matrix (D) is a new dependence matrix ($D' = T * D$) in which the first row represents time delay and the remaining rows are the corresponding interconnections of an array processor.

Now we turn our attention to the problem of matrix multiplication ($C = A \times B$). There are an intensive number of array processors designed for this problem in the literature. For example, the design of the linear arrays by Prasanna Kumar-Tsai [5]–[7] and Ramakrishnan-Varman [8], [9], the mesh arrays by S. Y. Kung [10] and Melkemi-Tchuente [11], the hexagonal arrays by H. T. Kung [1], Quinton [4], and Li-Wah [12], the cylindrical array by Porter-Aravena [13], and the two-layered (or the multilayered) mesh array by Kak [14], [15]. Besides, Benaini-Robert [16] and Jagadish-Kailath [17] used Winograd algorithm for matrix multiplication to design their systolic arrays.

Comparing these designs for matrix multiplication, we have: first, the mesh array has the good property of local connection, but its execution time ($3N-2$) is greater than that ($2N-1$) of the cylindrical array, where N is the size of the matrices. Second, based on the Winograd algorithm, we have $3N/2$ execution time [16], but this two-layered mesh array is not so regular.

In Section III-A, we design mesh arrays with $2N-1$ execution time. In Section III-B, we have a mesh array with $\lceil(3N-1)/2\rceil$ execution time. All of the above new designs of 2-D arrays are based on the sequential matrix multiplication algorithm of $C = A \times B$.

For the problem of transitive closure, the orthogonal array designed by S. Y. Kung *et al.* [18] has the execution time of $5N-4$, which is optimal in terms of pipelining rate, block pipelining rate, and the number of I/O connections. According to the dependence graph (DG-3 in his paper) used by S. Y. Kung, the longest path in this graph is $5N-4$. Therefore, for this DG the design is optimal in execution time. It is interested that whether there is a DG for the transitive closure problem with a shorter length of the longest path. We will show in Section IV that it is true and present a new design of 2-D array for transitive closure, which has $4N-2$ execution time. Finally, the concluding remarks are presented in Section V.

Manuscript received May 24, 1991; revised July 30, 1993 and June 30, 1994. This work was supported by the National Science Council of the R.O.C. under Contract NSC-81-0408-E-009-568.

The authors are with the Institute of Computer Science and Information Engineering, College of Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.

IEEE Log Number 9409331.

II. Preliminary Definitions

In this section, we give some preliminary definitions as a basis for following sections.

Definition 2.1: A left-shift sequence, $L(i;1,N) = (i+1, i+2, \cdots, N, 1, 2, \cdots, i) = (l_i(1), l_i(2), \cdots, l_i(N))$, is a sequence of integers resulting from shifting the sequence $(1, 2, \cdots, N)$ left cyclically i times, where $0 \leq i \leq N-1$. The jth element in L is $l_i(j) = (i+j-1)_{\text{mod } N} + 1$, where $1 \leq j \leq N$.

Definition 2.2: A right-shift sequence, $R(i;1,N) = (N-i+1, N-i+2, \cdots, N, 1, 2, \cdots, N-i) = (r_i(1), r_i(2), \cdots, r_i(N))$, is a sequence of integers resulting from shifting the sequence $(1, 2, \cdots, N)$ right cyclically i times, where $0 \leq i \leq N-1$. The jth element in R is $r_i(j) = (j-i+N-1)_{\text{mod } N} + 1$, where $1 \leq j \leq N$.

The following theorem shows the relationship between these two sequences.

Theorem 2.1: $l_i(j) = k$ *iff* $r_i(k) = j$, *where* $0 \leq i \leq N-1$ *and* $1 \leq j, k \leq N$.

Proof: (if part)

case 1: if $k > i$

$r_i(k) = (k-i+N-1)_{\text{mod } N} + 1 = k-i = j$

$\Rightarrow \quad k = i+j = (i+j-1)_{\text{mod } N} + 1 = l_i(j)$

case 2: if $k \leq i$

$r_i(k) = (k-i+N-1)_{\text{mod } N} + 1 = k-i+N = j$

$\Rightarrow \quad k = i+j-N = (i+j-1)_{\text{mod } N} + 1 = l_i(j)$

(only if part)

case 1: if $i+j < N$

$l_i(j) = (i+j-1)_{\text{mod } N} + 1 = i+j = k$

$\Rightarrow \quad j = k-i = (k-i+N-1)_{\text{mod } N} + 1 = r_i(k)$

case 2: if $i+j \geq N$

$l_i(j) = (i+j-1)_{\text{mod } N} + 1 = i+j-N = k$

$\Rightarrow \quad j = k-i+N = (k-i+N-1)_{\text{mod } N} + 1 = r_i(k)$

□

The meaning behind this theorem is that if we want to know which position (say j) the number k appears in the left-shift sequence $L(i;1,N)$, we can read j from the value of $r_i(k)$. For example, if we want to know which position the number 3 appears in the left-shift sequence $L(1;1,3)$, we have the position $j = r_1(3) = 2$.

There are many criteria (e.g., execution time, pipelining period, array size, and I/O channels) to measure performance of array processors. Since execution time is the most important criterion on designing real-time signal/imaging processing system, we pay our attention on execution time in this paper.

Definition 2.3: The execution time (t_e) of a systolic array is defined as the time interval between the time when the first operation is executed and the time when the last result is calculated.

By computation domain (Θ) we mean the set of finite indexes used by a RIA. Let I and I' be two indexes in the computation domain Θ of a RIA A. Λ^T is a linear schedule in the first row of transformation matrix T. Assuming that there is a unitary time increment, the execution time t_e of a systolic array executing the RIA A by transformation matrix T is $t_e = \max_{I,I' \in \Theta} \{\Lambda^T(I-I')\} + 1$ [19]. The actual meanings of the execution time of a RIA is the number of hyperplanes sweeping the index space.

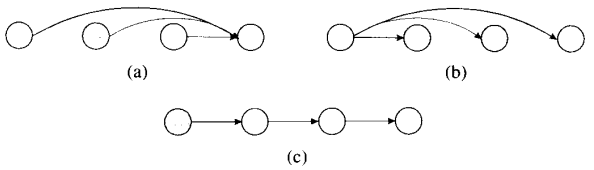

Fig. 1. (a) Multiple fan-in broadcast vector. (b) Multiple fan-out broadcast vector. (c) Propagation vector.

The multiple fan-in (Fig. 1(a)) and multiple fan-out (Fig. 1(b)) data dependence vectors are called *broadcast vectors*. All broadcast vectors can be systematically transformed into propagation vectors (e.g., Fig. 1(c)). We use the term *broadcast point* (the dark node in Fig. 1(c)) to denote the starting position for data propagation. A *broadcast line* is composed of several broadcast points. By aggregating broadcast lines, we obtain a *broadcast plane*.

III. Design of Mesh Arrays for Matrix Multiplication

The problem of matrix multiplication is to calculate matrix $C = A * B$, where A and B are both matrices. Without loss of generality, we assume that A, B, and C are all $N \times N$ matrices. The matrix multiplication can be carried out in N recursions as depicted in Algorithm 3.1.

[Algorithm 3.1]

For $i, j, k = 1$ to N

$$c_{i,j}^{k+1} = c_{i,j}^{k} + a_{i,k} * b_{k,j}$$

with $c_{i,j}^{1} = 0$

final results $c_{i,j}^{N+1}$.

□

The broadcast data $a_{i,k}$ and $b_{k,j}$ in Algorithm 3.1 can be removed by introducing propagation variables $a(i,j,k)$ and $b(i,j,k)$, respectively. Now we have Algorithm 3.2 [2]. The corresponding DG is shown in Fig. 2, where the bold line denotes the longest path in this graph.

[Algorithm 3.2]

For $i, j, k = 1$ to N

$$c(i,j,k+1) = c(i,j,k) + a(i,j,k) * b(i,j,k)$$

$$a(i,j+1,k) = a(i,j,k)$$

$$b(i+1,j,k) = b(i,j,k)$$

with $c(i,j,1) = 0$

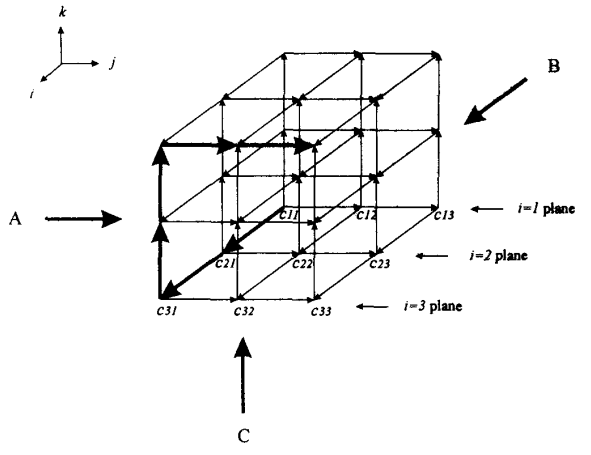

Fig. 2. Dependence graph for Algorithm 3.2.

initial values $a(i,1,k) = a_{i,k}$
$b(1,j,k) = b_{k,j}$

final results $c_{i,j} = c(i,j,N+1)$. □

The dependence matrix D of Algorithm 3.2 is

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

By selecting transformation matrix T as follows:

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

we have

$$D' = T * D = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Now we have the well-known mesh array as shown in Fig. 3 and we call it Design $m1$. Since $\Lambda^T = [1 \quad 1 \quad 1]$, the execution time of Design $m1$ is $t_e = \max_{I,I' \in \Theta} \{\Lambda^T(I - I')\} + 1 = (N + N + N) - (1 + 1 + 1) + 1 = 3N - 2$.

A. Mesh Arrays with $2N - 1$ Execution Time

The execution time of a systolic array can be decomposed into three parts. They are queuing time (t_q), waiting time (t_w), and operating time (t_o) as shown in Fig. 4. First, the queuing time (t_q) is the time from beginning execution to the position labeled by a for datum $b_{N,N}$. Second, the waiting time (t_w) is the time from position a to position b. This time is due to datum $b_{N,N}$ must wait to meet another datum at first PE. Last, the operating time (t_o) is the time from position b to position c. This time results from datum $b_{N,N}$ will operate

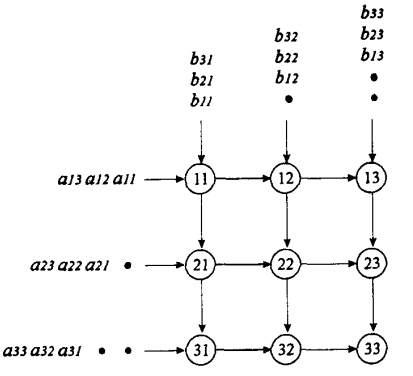

Fig. 3. Mesh array of Design $m1$.

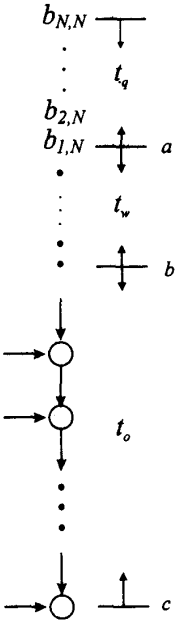

Fig. 4. $t_e = t_q + t_w + t_o$.

with N of another data. We know $t_e = t_q + t_w + t_o = (N-1) + (N-1) + (N) = 3N - 2$ for datum $b_{N,N}$ in Fig. 3, where $N = 3$.

Eliminating or lowering any part of these times can decrease the execution time of a systolic array. That is, if we do operation in each PE as soon as possible, we can obtain a systolic array with less execution time. It can be carried out by moving the broadcast planes of variables a and b to $(i = j)$-plane. By using this broadcast plane, the algorithm of matrix multiplication can be decomposed into two phases. The first phase is $i \leq j$ and the second phase is $i \geq j$. In the first phase, the propagation vectors of variables a and b are in $[0 \quad 1 \quad 0]$ and $[-1 \quad 0 \quad 0]$ directions, respectively. In the second phase, the propagation vectors of variables a and b are in $[0 \quad -1 \quad 0]$ and $[1 \quad 0 \quad 0]$ directions, respectively. Finally, we have Algorithm 3.3 and its DG is shown in Fig. 5.

[Algorithm 3.3]

[Phase 1]: $i \leq j$

For $i = 1$ to N

For $j = i$ to N

For $k = 1$ to N

$$c(i,j,k+1) = c(i,j,k) + a(i,j,k) * b(i,j,k)$$

$$a(i,j+1,k) = a(i,j,k)$$

$$b(i-1,j,k) = b(i,j,k)$$

[Phase 2]: $i \geq j$

For $i = 1$ to N

For $j = 1$ to i

For $k = 1$ to N

$$c(i,j,k+1) = c(i,j,k) + a(i,j,k) * b(i,j,k)$$

$$a(i,j-1,k) = a(i,j,k)$$

$$b(i+1,j,k) = b(i,j,k)$$

with $c(i,j,1) = 0$

initial values $a(i,i,k) = a_{i,k}$

$$b(j,j,k) = b_{k,j}$$

final results $c_{i,j} = c(i,j,N+1)$.

□

The dependence matrices $D_i, 1 \leq i \leq 2$, with respect to phase i are

$$D_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad D_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

By selecting transformation matrices $T_i, 1 \leq i \leq 2$, for D_i as follows

$$T_1 = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \qquad T_2 = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

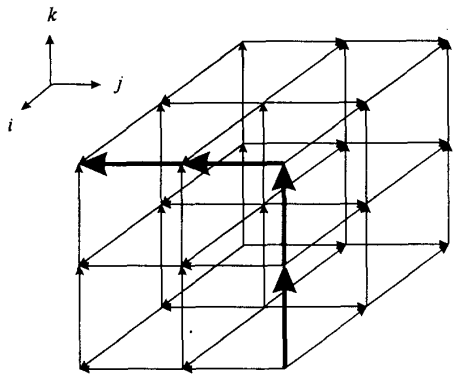

Fig. 5. Dependence graph for Algorithm 3.3.

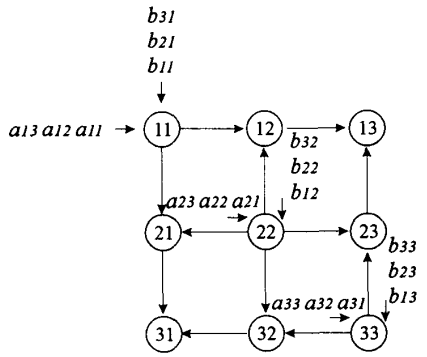

Fig. 6. Mesh array of Design $m2$.

we get

$$D_1' = T_1 * D_1 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

$$D_2' = T_2 * D_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}.$$

Hence, we have a new design of mesh array for matrix multiplication by composing these two phases in broadcast plane, as shown in Fig. 6. We call it Design $m2$. Notice that, we assume that the data can be inputted on the diagonal PE's. If the array restricts that data must be fed on the boundary PE's, then the data preloading is necessary.

This new mesh array not only has the same execution time as the cylindrical array [13] but also eliminates the spiral arcs in the cylindrical array. In this design, two phases have different schedule vectors. We call it *two-phase schedule* and this schedule can be generalized into an *m-phase schedule*. There are three different types of m-phase schedule. They are m-phase linear schedule, m-phase uniform affine schedule and m-phase affine schedule. In this paper, the first two schedules are used to design 2-D arrays for multiphase RIA's. Therefore, we give formal definitions to these two types of m-phase schedule as follows:

Definition 3.1: m-phase linear schedule: $\Pi_i(I_i) = \Lambda_i^T I_i$, where Λ_i^T is the first row of transformation matrix T_i and I_i is an index in the computation domain Θ_i of phase $i, 1 \le i \le m$.

Definition 3.2: m-phase uniform affine schedule : $\Pi_{i,x}(I_i) = \Lambda_i^T I_i + \nu_{i,x}$, where Λ_i^T is the first row of transformation matrix T_i and I_i is an index in the computation domain Θ_i of phase i. $\nu_{i,x}$ is a constant value for variable x in phase i to denote the translation part of $\Pi_{i,x}$, where $1 \le i \le m$.

If all variables have the same translation part in each phase, then ν_i is short for $\nu_{i,x}$ for an m-phase uniform affine schedule. Since the execution time of a RIA is the number of hyperplanes sweeping the index space, we have the execution time $t_e = \max_{I_i, I_i' \in \Theta_i} \{\Lambda_i^T (I_i - I_i') + 1\}$ for an m-phase linear schedule and $t_e = \max_{I_i, I_i' \in \Theta_i} \{(\Lambda_i^T I_i + \nu_{i,x}) - (\Lambda_i^T I_i' + \nu_{i,y}) + 1\}$ for an m-phase uniform affine schedule, where $1 \le i \le m$.

The two-phase linear schedule for Design $m2$ is

$$\Pi_1(I_1) = [-1 \quad 1 \quad 1] I_1$$
$$\Pi_2(I_2) = [1 \quad -1 \quad 1] I_2.$$

Notice that phase 1 and phase 2 begin execution at the same time in Design $m2$. Since nodes (i, i, k) belong to both phases, and

$$[-1 \quad 1 \quad 1] \begin{bmatrix} i \\ i \\ k \end{bmatrix} = [1 \quad -1 \quad 1] \begin{bmatrix} i \\ i \\ k \end{bmatrix}$$
$$\Rightarrow k = k.$$

The execution time of Design $m2$ is

$$t_e = \max \{(-1 + N + N) - (-i + i + 1) + 1, \text{ for phase 1}$$
$$(N - 1 + N) - (i - i + 1) + 1\} \text{ for phase 2}$$
$$= 2N - 1.$$

Recalling the execution time of a systolic array can be interpreted as $t_e = t_q + t_w + t_o$. The queuing time (t_q) can't be lowered if we input variables a and b side by side. Therefore, there are two ways to decrease the execution time. One is to reduce the waiting time (t_w) as in Design $m2$. The other is to try to decrease the operating time (t_o) and this can be accomplished by inputting variables a and b in ($j = \lceil N/2 \rceil$)-plane and ($i = \lceil N/2 \rceil$)-plane, respectively. With these two broadcast planes, the algorithm of matrix multiplication can be decomposed into four phases. The first phase is $i \le \lceil N/2 \rceil$ and $j \le \lceil N/2 \rceil$, the second phase is $i \le \lceil N/2 \rceil$ and $j \ge \lceil N/2 \rceil$, the third phase is $i \ge \lceil N/2 \rceil$ and $j \ge \lceil N/2 \rceil$, and the last phase is $i \ge \lceil N/2 \rceil$ and $j \le \lceil N/2 \rceil$. In the first phase, the propagation vectors of variables a and b are in $[0 \quad -1 \quad 0]$ and $[-1 \quad 0 \quad 0]$, respectively. In the second phase, the propagation vectors of variables a and b are in $[0 \quad 1 \quad 0]$ and $[-1 \quad 0 \quad 0]$, respectively. In the third phase, the propagation vectors of variables a and b are in $[0 \quad 1 \quad 0]$ and $[1 \quad 0 \quad 0]$, respectively. In the last phase, the propagation vectors of variables a and b are in $[0 \quad -1 \quad 0]$ and $[1 \quad 0 \quad 0]$, respectively. Finally, we have the following Algorithm 3.4 and its DG is shown in Fig. 7.

[Algorithm 3.4]

[Phase 1]: $i \le \lceil N/2 \rceil, j \le \lceil N/2 \rceil$

For $i = 1$ to $\lceil N/2 \rceil$

For $j = 1$ to $\lceil N/2 \rceil$

For $k = 1$ to N

$$c(i, j, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$
$$a(i, j-1, k) = a(i, j, k)$$
$$b(i-1, j, k) = b(i, j, k)$$

[Phase 2]: $i \le \lceil N/2 \rceil, j \ge \lceil N/2 \rceil$

For $i = 1$ to $\lceil N/2 \rceil$

For $j = \lceil N/2 \rceil$ to N

For $k = 1$ to N

$$c(i, j, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$
$$a(i, j+1, k) = a(i, j, k)$$
$$b(i-1, j, k) = b(i, j, k)$$

[Phase 3]: $i \ge \lceil N/2 \rceil, j \ge \lceil N/2 \rceil$

For $i = \lceil N/2 \rceil$

to N
For $j = \lceil N/2 \rceil$

to N
For $k = 1$ to N

$$c(i, j, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$
$$a(i, j+1, k) = a(i, j, k)$$
$$b(i+1, j, k) = b(i, j, k)$$

[Phase 4]: $i \ge \lceil N/2 \rceil, j \le \lceil N/2 \rceil$

For $i = \lceil N/2 \rceil$ to N
For $j = 1$ to $\lceil N/2 \rceil$

For $k = 1$ to N

$$c(i, j, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$
$$a(i, j-1, k) = a(i, j, k)$$
$$b(i+1, j, k) = b(i, j, k)$$

with $c(i, j, 1) = 0$

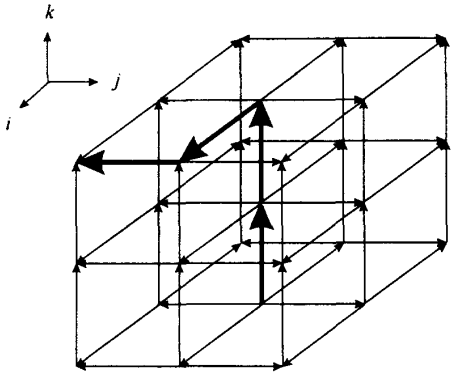

Fig. 7. Dependence graph for Algorithm 3.4.

initial values $a(i, \lceil N/2 \rceil, k) = a_{i,k}$

$$b(\lceil N/2 \rceil, j, k) = b_{k,j}$$

final results $c_{i,j} = c(i, j, N+1)$.

□

The dependence matrices $D_i, 1 \leq i \leq 4$, with respect to phase i are

$$D_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$
$$D_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

By selecting transformation matrices $T_i, 1 \leq i \leq 4$, for D_i as follows

$$T_1 = \begin{bmatrix} -1 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_2 = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$
$$T_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_4 = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

we get

$$D_1' = T_1 * D_1 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix},$$
$$D_2' = T_2 * D_2 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$
$$D_3' = T_3 * D_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$
$$D_4' = T_4 * D_4 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}.$$

Hence, we have an another new design of mesh array for matrix multiplication by composing these four phases in broadcast planes as shown in Fig. 8. We call it Design $m3$.

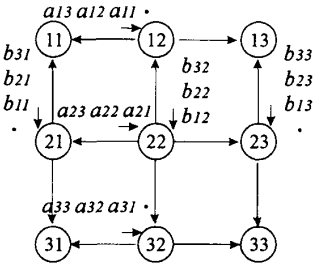

Fig. 8. Mesh array of design $m3$.

We use four-phase uniform affine schedule to let these four phases begin execution at the same time. Note that here we let every variable with the same translation part in each phase. That is, $\nu_i = \nu_{i,a} = \nu_{i,b} = \nu_{i,c}$, where $1 \leq i \leq 4$. The schedule for these four phases is

$$\Pi_1(I_1) = \Lambda_1^T I_1 + \nu_1 = [-1 \quad -1 \quad 1] I_1 + \nu_1$$
$$\Pi_2(I_2) = \Lambda_2^T I_2 + \nu_2 = [-1 \quad 1 \quad 1] I_1 + \nu_2$$
$$\Pi_3(I_3) = \Lambda_3^T I_3 + \nu_3 = [1 \quad 1 \quad 1] I_1 + \nu_3$$
$$\Pi_4(I_4) = \Lambda_4^T I_4 + \nu_4 = [1 \quad -1 \quad 1] I_1 + \nu_4$$

Since nodes $(\lceil N/2 \rceil, \lceil N/2 \rceil, k)$ belong to all four phases, we have

$$[-1 \quad -1 \quad 1] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_1 = [-1 \quad 1 \quad 1] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_2$$
$$= [1 \quad 1 \quad 1] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_3 = [1 \quad -1 \quad 1] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_4$$

$$\Rightarrow -2\left\lceil \frac{N}{2} \right\rceil + \nu_1 + k = \nu_2 + k = 2\left\lceil \frac{N}{2} \right\rceil + \nu_3 + k = \nu_4 + k.$$

Let $\nu_3 = 0$, we have

$$\nu_2 = \nu_4 = 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} N & \text{if } N \text{ is even} \\ N+1 & \text{if } N \text{ is odd} \end{cases}$$
$$\nu_1 = 4\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 2N & \text{if } N \text{ is even} \\ 2N+2 & \text{if } N \text{ is odd} \end{cases}$$

Hence

$$\Pi_1(I_1) = [-1 \quad -1 \quad 1] I_1 + 4\left\lceil \frac{N}{2} \right\rceil$$
$$\Pi_2(I_2) = [-1 \quad 1 \quad 1] I_2 + 2\left\lceil \frac{N}{2} \right\rceil$$
$$\Pi_3(I_3) = [1 \quad 1 \quad 1] I_3$$
$$\Pi_4(I_4) = [1 \quad -1 \quad 1] I_4 + 2\left\lceil \frac{N}{2} \right\rceil$$

The execution time of $m3$ is

$$t_e = \begin{cases} 2N & \text{if } N \text{ is even} \\ 2N-1 & \text{if } N \text{ is odd.} \end{cases}$$

Since (see the equation at the bottom of the next page).

The execution time of this design can be interpreted as

$$t_e = t_q + t_w + t_o$$
$$= \begin{cases} (N-1) + \left(\frac{N}{2}\right) + \left(\frac{N}{2}+1\right) = 2N & \text{if } N \text{ is even} \\ (N-1) + \left(\left\lceil\frac{N}{2}\right\rceil - 1\right) + 1\left(\left\lceil\frac{N}{2}\right\rceil\right) = 2N-1 & \text{if } N \text{ is odd} \end{cases}$$

Comparing with the previous Design $m2$, Design $m3$ results in the waiting time increasing to $\lceil(N-1)/2\rceil$, though we halve the operating time. Therefore, it has the same execution time as Design $m2$. The interesting problem is how we can decrease the operating time without increasing the waiting time. This problem will be tackled by adding delays to some PE's.

B. Mesh Array with $t_e = \lceil(3N-1)/2\rceil$ Execution Time

In this section, we will show how to eliminate the waiting time in a mesh array by adding delays to some PE's. We know that the relative data must meet on the same place at the same time in a systolic array. For example, in Fig. 8, $a_{2,1}$ and $b_{1,3}$ must meet at the same time at PE_{23}, so $b_{1,3}$ should wait one time step to meet $a_{2,1}$. Nevertheless, adding one delay to PE_{23} for variable b will eliminate this waiting time. In other words, $b_{1,3}$ goes into self loop in PE_{23} and in $[-1 \quad 0]$ direction to PE_{13} at the same time, then $b_{1,3}$ can operate simultaneously with $a_{2,1}$ at PE_{23} and with $a_{1,1}$ at PE_{13} at the next time step. Using this method, we can eliminate the waiting time in Design $m3$ and get an another new design of mesh array for matrix multiplication with execution time of $\lceil(3N-1)/2\rceil$. We call this mesh array Design $m4$ as shown in Fig. 9. It can be verified that the PE_{ij} should add delays $d = ||i - \lceil N/2\rceil| - |j - \lceil N/2\rceil||$.

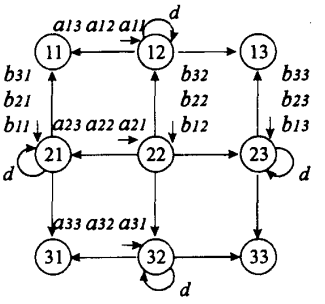

Fig. 9. Mesh array of Design $m4$.

IV. Design of a 2-D Array for Transitive Closure

In this section, we consider the transitive closure problem based on the sequential Warshall-Floyd algorithm. In this problem a directed graph, $G = (V, E)$, with N vertices is given. Let C be an adjacency matrix for G, where

$$c_{i,j} = \begin{cases} 1 & \text{if there is an edge from vertex } i \text{ to vertex } j \\ & \quad \text{or } i = j \\ 0 & \text{otherwise} \end{cases}$$

The objective is to compute the transitive closure matrix C^+, where

$$c^+_{i,j} = \begin{cases} 1 & \text{if there is a path of length } \geq 0 \text{ from vertex } i \\ & \quad \text{to vertex } j \\ 0 & \text{otherwise} \end{cases}$$

The well-known sequential Warshall-Floyd algorithm can be written as Algorithm 4.1.

[Algorithm 4.1]

For $i, j, k = 1$ to N

$$c^{k+1}_{i,j} = c^k_{i,j} + c^k_{i,k} * c^k_{k,j}$$

initial values $c^1_{i,j} = c_{i,j}$

final results $c^+_{i,j} = c^{N+1}_{i,j}$.

□

In the case of transitive closure problem, the operator $+$ performs boolean OR operation and $*$ performs boolean AND operation. The same algorithm can be used to solve the all pairs shortest path problem if matrix C is the cost adjacency matrix with $c_{i,i} = 0, 1 \leq i \leq N, +$ performs minimum operation,

$$t_e = \max \left\{ \begin{array}{ll} \left(-1-1+N+4\left\lceil\frac{N}{2}\right\rceil\right) - \left(-\left\lceil\frac{N}{2}\right\rceil - \left\lceil\frac{N}{2}\right\rceil + 1 + 4\left\lceil\frac{N}{2}\right\rceil\right) + 1, & \text{for phase 1} \\ \left(-1+N+N+2\left\lceil\frac{N}{2}\right\rceil\right) - \left(-\left\lceil\frac{N}{2}\right\rceil + \left\lceil\frac{N}{2}\right\rceil + 1 + 2\left\lceil\frac{N}{2}\right\rceil\right) + 1, & \text{for phase 2} \\ (N+N+N) - \left(\left\lceil\frac{N}{2}\right\rceil + \left\lceil\frac{N}{2}\right\rceil + 1\right) + 1, & \text{for phase 3} \\ \left(N-1+N+2\left\lceil\frac{N}{2}\right\rceil\right) - \left(\left\lceil\frac{N}{2}\right\rceil - \left\lceil\frac{N}{2}\right\rceil + 1 + 2\left\lceil\frac{N}{2}\right\rceil\right) + 1 & \text{for phase 4} \end{array} \right\}$$
$$= \max \left\{ N + 2\left\lceil\frac{N}{2}\right\rceil - 2, 2N-1, 3N-2\left\lceil\frac{N}{2}\right\rceil, 2N-1 \right\}$$
$$= \begin{cases} 2N & \text{if } N \text{ is even} \\ 2N-1 & \text{if } N \text{ is odd.} \end{cases}$$

$*$ performs addition operation, and output matrix C^+ is the shortest path matrix.

Since dependencies are not localized in Algorithm 4.1, we can add propagation variables a and b to remove broadcast dependencies, then we have Algorithm 4.2. The corresponding DG is shown in Fig. 10 for the $N = 5$ case with each constant k-plane drawn separately. The 3-D DG can be drawn by adding lines from $c(i, j, k)$ to $c(i, j, k+1)$ in the k-direction. The bold lines in the DG of transitive closure problem is to represent the *broadcast lines*. We can aggregate these lines in the k-direction to construct *broadcast planes*. The actual meaning of broadcast planes here is that the propagation variables a and b get their values of c in these planes. The vertical broadcast line and the horizontal broadcast line are for variables a and b, respectively. Comparing Algorithm 4.2 with Algorithm 4.1, we know that $c_{i,j}^k$ in Algorithm 4.1 corresponds to $c(i, j, k)$ in Algorithm 4.2 and it is computed at node (i, j, k) in Fig. 10.

[Algorithm 4.2]

For $i, j, k = 1$ to N

$$c(i, j, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$

$$a(i, j, k) = \begin{cases} c(i, k, k) & \text{if } j = k \\ a(i, j-1, k) & \text{if } j > k \\ a(i, j+1, k) & \text{if } j < k \end{cases}$$

$$b(i, j, k) = \begin{cases} c(k, j, k) & \text{if } i = k \\ b(i-1, j, k) & \text{if } i > k \\ b(i+1, j, k) & \text{if } i < k \end{cases}$$

initial values $c(i, j, 1) = c_{i,j}$

final results $c_{i,j}^+ = c(i, j, N+1)$.

□

Although the variable c propagating in the k-direction is regular in Algorithm 4.2, the propagation vectors of variables a and b are irregular in each k-plane. This irregularity can be eliminated by reindexing (i and j) in every k-plane as S. Y. Kung *et al.* did in [18].

Now we will show how to get a spherical array with execution time of $4N - 2$ (if N is even, $4N - 3$ if N is odd) by moving respectively the broadcast planes of variables a and b to the center of the DG in the j- and i-directions. The DG is shown in Fig. 11. And we have the following algorithm.

[Algorithm 4.3]

[Phase 1]: $i \le \left\lceil \frac{N}{2} \right\rceil, j \le \left\lceil \frac{N}{2} \right\rceil$

For $k = 1$ to N

For $i = 1$ to $\left\lceil \frac{N}{2} \right\rceil$

For $j = 1$ to $\left\lceil \frac{N}{2} \right\rceil$

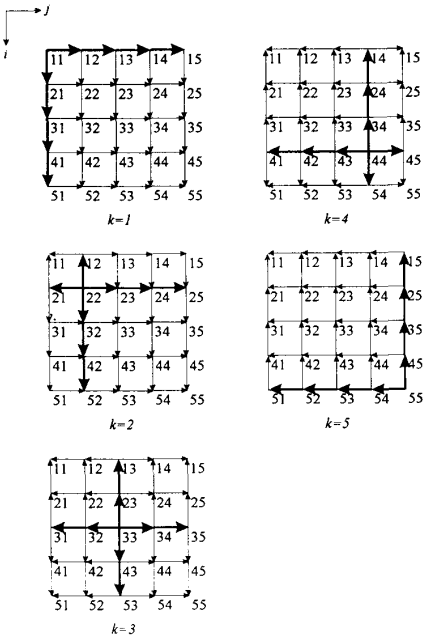

Fig. 10. Dependence graph for Algorithm 4.2.

$$c(i-1, j-1, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$

$$a(i, j-1, k) = a(i, j, k)$$

$$b(i-1, j, k) = b(i, j, k)$$

[Phase 2]: $i \le \left\lceil \frac{N}{2} \right\rceil, j \ge \left\lceil \frac{N}{2} \right\rceil$

For $k = 1$ to N

For $i = 1$ to $\left\lceil \frac{N}{2} \right\rceil$

For $j = \left\lceil \frac{N}{2} \right\rceil$ to N

$$c(i-1, j-1, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$

$$a(i, j+1, k) = a(i, j, k)$$

$$b(i-1, j, k) = b(i, j, k)$$

[Phase 3]: $i \ge \left\lceil \frac{N}{2} \right\rceil, j \ge \left\lceil \frac{N}{2} \right\rceil$

For $k = 1$ to N

For $i = \left\lceil \frac{N}{2} \right\rceil$ to N

For $j = \left\lceil \frac{N}{2} \right\rceil$ to N

$$c(i-1, j-1, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$

$$a(i, j+1, k) = a(i, j, k)$$

$$b(i+1, j, k) = b(i, j, k)$$

[Phase 4]: $i \geq \left\lceil \frac{N}{2} \right\rceil, j \leq \left\lceil \frac{N}{2} \right\rceil$

For $k = 1$ to N

For $i = \left\lceil \frac{N}{2} \right\rceil$ to N

For $j = 1$ to $\left\lceil \frac{N}{2} \right\rceil$

$$c(i-1, j-1, k+1) = c(i, j, k) + a(i, j, k) * b(i, j, k)$$

$$a(i, j-1, k) = a(i, j, k)$$

$$b(i+1, j, k) = b(i, j, k)$$

In broadcast plane $a\left(i, \left\lceil \frac{N}{2} \right\rceil, k\right) = c\left(i, \left\lceil \frac{N}{2} \right\rceil, k\right)$

$$b\left(\left\lceil \frac{N}{2} \right\rceil, j, k\right) = c\left(\left\lceil \frac{N}{2} \right\rceil, j, k\right)$$

Intraphase dependencies between

1) phase 2 and phase 1
$$c(i-1, N, k+1) = c(i, 1, k) + a(i, 1, k) * b(i, 1, k)$$

2) phase 3 and phase 4
$$c\left(\left\lceil \frac{N}{2} \right\rceil, N, k+1\right) = c\left(\left\lceil \frac{N}{2} \right\rceil + 1, 1, k\right)$$

3) phase 2 and phase 4
$$c(i-1, N, k+1) = c(i, 1, k) + a(i, 1, k) * b(i, 1, k) + a\left(\left\lceil \frac{N}{2} \right\rceil + 1, 1, k\right) * b\left(\left\lceil \frac{N}{2} \right\rceil + 1, 1, k\right)$$

4) phase 4 and phase 1
$$c(N, j-1, k+1) = c(1, j, k) + a(1, j, k) * b(1, j, k)$$

5) phase 3 and phase 2
$$c(N, j-1, k+1) = c(1, j, k) + a(1, j, k) * b(1, j, k$$

6) phase 4 and phase 2
$$c\left(N, \left\lceil \frac{N}{2} \right\rceil, k+1\right) = c\left(1, \left\lceil \frac{N}{2} \right\rceil + 1, k\right) + a\left(1, \left\lceil \frac{N}{2} \right\rceil + 1, k\right) * b\left(1, \left\lceil \frac{N}{2} \right\rceil + 1, k\right)$$

initial values $c(i, j, 1) = c_{i', j'}$

where $i' = r_{\lfloor (N-1)/2 \rfloor}(i)$

$j' = r_{\lfloor (N-1)/2 \rfloor}(j)$

final results $c^{+}_{i', j'} = c(i, j, N+1)$

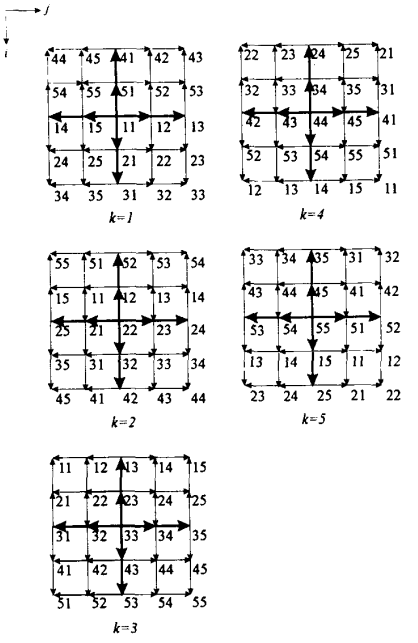

Fig. 11. Dependence graph for Algorithm 4.3.

where $i = l_{\lfloor (N-1)/2 \rfloor}(i')$

$j = l_{\lfloor (N-1)/2 \rfloor}(j')$.

□

The dependence matrices $D_i, 1 \leq i \leq 4$, of phase i in Algorithm 4.3 are

$$D_1 = \begin{bmatrix} -1 & 0 & -1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_2 = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix},$$

$$D_3 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_4 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

By selecting transformation matrices $T_i, 1 \leq i \leq 4$, for D_i as follows

$$T_1 = \begin{bmatrix} -1 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_2 = \begin{bmatrix} -1 & 1 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

$$T_3 = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_4 = \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

we have

$$D'_1 = T_1 * D_1 = \begin{bmatrix} 1 & 1 & 5 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix},$$

$$D'_2 = T_2 * D_2 = \begin{bmatrix} 1 & 1 & 3 \\ -1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

$$D'_3 = T_3 * D_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix},$$
$$D'_4 = T_4 * D_4 = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix}.$$

We use four-phase uniform affine schedule to let these four phases begin execution at the same time. The schedule for these four phases is

$$\Pi_1(I_1) = \Lambda_1^T I_1 + \nu_1 = [-1 \quad -1 \quad 3] I_1 + \nu_1$$
$$\Pi_2(I_2) = \Lambda_2^T I_2 + \nu_2 = [-1 \quad 1 \quad 3] I_1 + \nu_2$$
$$\Pi_3(I_3) = \Lambda_3^T I_3 + \nu_3 = [1 \quad 1 \quad 3] I_1 + \nu_3$$
$$\Pi_4(I_4) = \Lambda_4^T I_4 + \nu_4 = [1 \quad -1 \quad 3] I_1 + \nu_4$$

Since nodes $(\lceil N/2 \rceil, \lceil N/2 \rceil, k)$ belong to all four phases, we have

$$[-1 \quad -1 \quad 3] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_1 = [-1 \quad 1 \quad 3] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_2$$
$$= [1 \quad 1 \quad 3] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_3 = [1 \quad -1 \quad 3] \begin{bmatrix} \lceil \frac{N}{2} \rceil \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_4$$
$$\Rightarrow -2\left\lceil \frac{N}{2} \right\rceil + \nu_1 = \nu_2 = 2\left\lceil \frac{N}{2} \right\rceil + \nu_3 = \nu_4.$$

Let $\nu_3 = 0$, we have

$$\nu_2 = \nu_4 = 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} N & \text{if } N \text{ is even} \\ N+1 & \text{if } N \text{ is odd} \end{cases}$$
$$\nu_1 = 4\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 2N & \text{if } N \text{ is even} \\ 2N+2 & \text{if } N \text{ is odd} \end{cases}$$

Hence

$$\Pi_1(I_1) = [-1 \quad -1 \quad 3] I_1 + 4\left\lceil \frac{N}{2} \right\rceil$$
$$\Pi_2(I_2) = [-1 \quad 1 \quad 3] I_2 + 2\left\lceil \frac{N}{2} \right\rceil$$
$$\Pi_3(I_3) = [1 \quad 1 \quad 3] I_3$$
$$\Pi_4(I_4) = [1 \quad -1 \quad 3] I_4 + 2\left\lceil \frac{N}{2} \right\rceil.$$

With this four-phase uniform affine schedule, it is easy to determine the interconnection delays for mapping intraphase dependencies.

1) $c(i-1, N, k+1) = c(i, 1, k) + a(i, 1, k) * b(i, 1, k)$

$$-i + 1 + N + 3k + 3 + 2\left\lceil \frac{N}{2} \right\rceil = -i - 1 + 3k + 4\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_1$$
$$\Rightarrow \hat{d}'_1 = N + 5 - 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 5 & \text{if } N \text{ is even} \\ 4 & \text{if } N \text{ is odd.} \end{cases}$$

2) $c(i-1, N, k+1) = c(i, 1, k) + a(i, 1, k) * b(i, 1, k)$

$$i - 1 + N + 3k + 3 = i - 1 + 3k + 2\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_2$$
$$\Rightarrow \hat{d}'_2 = N + 3 - 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd.} \end{cases}$$

3) $c(\lceil N/2 \rceil, N, k+1) = c(\lceil N/2 \rceil + 1, 1, k) + a(\lceil N/2 \rceil + 1, 1, k) * b(\lceil N/2 \rceil + 1, 1, k)$

$$-\left\lceil \frac{N}{2} \right\rceil + N + 3k + 3 + 2\left\lceil \frac{N}{2} \right\rceil = \left\lceil \frac{N}{2} \right\rceil + 1 - 1 + 3k + 2\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_3$$
$$\Rightarrow \hat{d}'_3 = N + 3 - 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd.} \end{cases}$$

4) $c(N, j-1, k+1) = c(1, j, k) + a(1, j, k) * b(1, j, k)$

$$N - j + 1 + 3k + 3 + 2\left\lceil \frac{N}{2} \right\rceil = -1 - j + 3k + 4\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_4$$
$$\Rightarrow \hat{d}'_4 = N + 5 - 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 5 & \text{if } N \text{ is even} \\ 4 & \text{if } N \text{ is odd.} \end{cases}$$

5) $c(N, j-1, k+1) = c(1, j, k) + a(1, j, k) * b(1, j, k)$

$$N + j - 1 + 3k - 3 = -1 + j + 3k + 2\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_5$$
$$\Rightarrow \hat{d}'_5 = N + 3 - 2\left\lceil \frac{N}{2} \right\rceil = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd.} \end{cases}$$

6) $c(N, \lceil N/2 \rceil, k+1) = c(1, \lceil N/2 \rceil + 1, k) + a(1, \lceil N/2 \rceil + 1, k) * b(1, \lceil N/2 \rceil + 1, k)$

$$\begin{aligned} & N - \left\lceil \frac{N}{2} \right\rceil + 3k + 3 + 2\left\lceil \frac{N}{2} \right\rceil \\ & = -1 + \left\lceil \frac{N}{2} \right\rceil + 1 + 3k + 2\left\lceil \frac{N}{2} \right\rceil + \hat{d}'_6 \\ & \Rightarrow \hat{d}'_6 = N + 3 - 2\left\lceil \frac{N}{2} \right\rceil \\ & = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd.} \end{cases} \end{aligned}$$

The interconnections of array processor for intraphase dependencies can be gotten from the first equation shown at the bottom of the page.

These interconnections represent spiral arcs as shown in Fig. 12. We call it Design t.

The reason why we pick out $S^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ rather than $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ is if the latter is selected, we will face the problem of executing two multiply-add operations in a PE simultaneously. Since $S^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, we assume adjacency matrix, $c_{i',j'}$, in each $\mathrm{PE}_{i,j}$ initially.

The execution time of Design t is

$$t_e = \begin{cases} 4N-2 & \text{if } N \text{ is even} \\ 4N-3 & \text{if } N \text{ is odd.} \end{cases}$$

Since (see the second equation at the bottom of the page).

V. Conclusions

In this paper, we have presented several new 2-D arrays for the problems of matrix multiplication and transitive closure. Besides these new designs, we have also proposed two types of the m-phase schedule to help us on mapping multiphase algorithms onto array processors. They are the m-phase linear schedule and the m-phase uniform affine schedule.

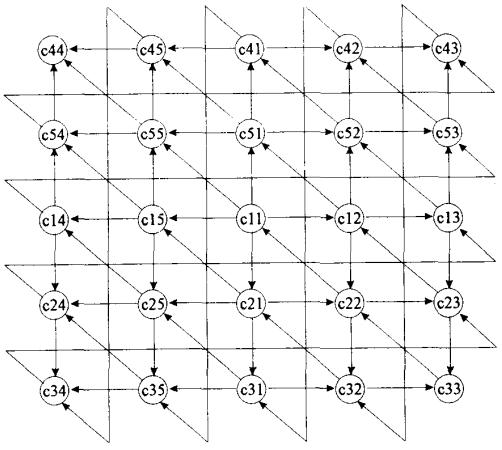

Fig. 12. Spherical array of Design t.

Furthermore, it is easily to extend the definitions of these two types of m-phase schedule to the m-phase affine schedule and may apply it to map nonsystolic RIA's onto array processors. These new m-phase schedules can actually map an algorithm that fails by other schedules and design 2-D arrays that can not be obtained by other methodologies we know.

The idea of moving broadcast planes and decomposing algorithms by the broadcast planes can help us on designing a DG with a shorter length of the longest path. Its penalty is that the control complexity will be increased somewhat. We can apply the method proposed in this paper to many other problems to design some even faster VLSI array processors to cater for the requirements of real-time applications.

$$\begin{aligned} S^T \hat{D} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 & N-1 & N-1 & N-1 \\ N-1 & N-1 & N-1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -1 & -1 & -1 & N-1 & N-1 & N-1 \\ N-1 & N-1 & N-1 & -1 & -1 & -1 \end{bmatrix}. \end{aligned}$$

$$\begin{aligned} t_e &= \max \left\{ \begin{array}{ll} \left(-1-1+3N+4\left\lceil \frac{N}{2} \right\rceil\right) - \left(-\left\lceil \frac{N}{2} \right\rceil - \left\lceil \frac{N}{2} \right\rceil + 3 + 4\left\lceil \frac{N}{2} \right\rceil\right) + 1, & \text{for phase 1} \\ \left(-1+N+3N+2\left\lceil \frac{N}{2} \right\rceil\right) - \left(-\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil + 3 + 2\left\lceil \frac{N}{2} \right\rceil\right) + 1, & \text{for phase 2} \\ (N+N+3N) - \left(\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil + 3\right) + 1, & \text{for phase 3} \\ \left(N-1+3N+2\left\lceil \frac{N}{2} \right\rceil\right) - \left(\left\lceil \frac{N}{2} \right\rceil - \left\lceil \frac{N}{2} \right\rceil + 3 + 2\left\lceil \frac{N}{2} \right\rceil\right) + 1 & \text{for phase 4} \end{array} \right\} \\ &= \max \left\{ 3N + 2\left\lceil \frac{N}{2} \right\rceil - 4, 4N-3, 5N-2\left\lceil \frac{N}{2} \right\rceil - 2, 4N-3 \right\} \\ &= \begin{cases} 4N-2 & \text{if } N \text{ is even} \\ 4N-3 & \text{if } N \text{ is odd.} \end{cases} \end{aligned}$$

REFERENCES

[1] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Proc. Sparse Matrix Symp.*, Soc. Indust. Appli. Math., 1978, pp. 256–282.

[2] S. K. Rao, "Regular iterative algorithms and their implementations on processor arrays," Ph.D. dissertation, Standford Univ., 1985.

[3] V. Van Dongen and P. Quinton, "Uniformization of linear recurrence equations: A step towards the automatic synthesis of systolic arrays," in *Proc. Int. Conf. Systolic Array*, 1988, pp. 473–482.

[4] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," in *Proc. Int. Symp. Comput. Architecture*, 1984, pp. 208–214.

[5] V. K. Prasanna Kumar and Y. C. Tsai, "Designing linear systolic arrays," *J. Parallel Distribut. Comput.*, vol. 7, pp. 441–463, 1989.

[6] ——, "Mapping two dimensional systolic arrays to one dimensional arrays and applications," in *Proc. Int. Conf. Parallel Processing*, 1988, pp. 39–46.

[7] ——, "Synthesizing optimal family of linear systolic arrays for matrix computations," in *Proc. Int. Conf. Systolic Array*, 1988, pp. 51–60.

[8] P. J. Varman and I. V. Ramakrishnan, "Synthesis of an optimal family of matrix multiplication algorithms on linear arrays," *IEEE Trans. Comput.*, vol. C-35, pp. 989–996, Nov. 1986.

[9] ——, "Modular matrix multiplication on a linear array," *IEEE Trans. Comput.*, vol. C-33, pp. 952–958, Nov. 1984.

[10] S. Y. Kung, *VLSI Array Processor*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[11] L. Melkemi and M. Tchuente, "Complex of matrix product on a class of orthogonally connected systolic arrays," *IEEE Trans. Comput.*, vol. C-36, pp. 615–619, May 1987.

[12] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 66–77, Jan. 1985.

[13] W. A. Porter and J. L. Aravena, "Cylindrical arrays for matrix multiplication," in *Proc. 24th Annu. Allerton Conf. Commun., Control Comput.*, Mar. 1988, pp. 595–602.

[14] S. C. Kak, "Multilayered array computing," in *Proc. 1985 Annu. Conf. Inform. Sci. Syst.*, Mar. 1984, pp. 4.36–4.41.

[15] ——, "A two-layered mesh array for matrix multiplication," *Parallel Comput.*, vol. 6, pp. 383–385, 1988.

[16] A. Benaini and Y. Robert, "An even faster systolic array for matrix multiplication," *Parallel Comput.*, vol. 12, pp. 249–254, 1989.

[17] H. V. Jagadish and T. Kailath, "A family of new efficient arrays for matrix multiplication," *IEEE Trans. Comput.*, vol. C-38, pp. 149–155, Jan. 1989.

[18] 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.*, vol. C-36, pp. 603–614, May 1987.

[19] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Trans. Comput.*, vol. C-35, pp. 1–12, Jan. 1986.

Jong-Chuang Tsay was born in Taipei, R.O.C., in 1943. He received the M.S. and Ph.D. degrees in computer science from National Chiao-Tung University in 1968 and 1975, respectively.

Since 1968, he has been with the Faculty of the Department of Computer Engineering, Chiao-Tung University. Currently, he is a Professor of the Department of Computer Science and Information Engineering. His current research interests include systolic algorithm design, parallel computations, and computer-aided typesetting.

Pen-Yuang Chang was born in Kaohsiung, R.O.C., in 1962. He received the M.S. degree from the Department of Computer Engineering, Chiao-Tung University in 1986.

Since 1987, he has been an Associate Researcher in Telecommunication Laboratories, Ministry of Communications. Currently, he is a Ph.D. candidate with the Institute of Computer Science and Information Engineering, National Chiao-Tung University. His research interests are in design of systolic algorithm and computer network.