A Family of Efficient Regular Arrays for Algebraic Path Problem

Pen-Yuang Chang and Jong-Chuang Tsay

***Abstract*—It has been shown that the method of decomposing a dependence graph into multiple phases with appropriate m-phase schedule function is useful for designing faster regular arrays for matrix multiplication and transitive closure. In this paper, we will further apply this method to design several parallel algorithms for Algebraic Path Problem and derive $N \times N$ 2-D regular arrays with execution time $\lfloor \frac{9N}{2} \rfloor - 2$ (cylindrical array and orthogonal one) and $4N - 2$ (spherical one).**

***Index Terms*—Algebraic path problem, cylindrical array, parallel algorithm design, systolic array, spherical array, transitive closure, VLSI architecture.**

I. Introduction

TWO major steps on the design of a regular array (RA) [1] from a sequential algorithm for a problem are *regularization* and *spacetime mapping*. Firstly, by regularization we can rewrite the sequential algorithm to a regular iterative algorithm (RIA) [2]–[5], which has a corresponding graphic representation called dependence graph (DG); then an RA is derived by the spacetime mapping (T) procedure which comprises the selection of a schedule function (Λ) and a compatible processor assignment function (S). Diversified RA's can be obtained by different combinations of this two steps. In [6], several new RA's for the problem of matrix multiplication and transitive closure have been derived by constructing different multi-phase RIAs according to various selections of broadcast plane together with new proposed *m-phase schedule* function and compatible processor assignment function. In this paper, we will extend its application to Algebraic Path Problem (APP) which is a generalization of problems, such as transitive closure, shortest path problem, Gauss-Jodan elimination, and so on [7].

The design of RA's for APP has been studied by several researchers in the literature [8]–[13]. The most significant RA designed by Lewis and Kung [12] with execution time $5N-2$ is optimal in terms of pipelining period, block pipelining period, and the number of I/O connections. Besides, the survey and comparison of several 2-D RA's for APP are also cited in that paper. According to the DG (Fig. 3 in their paper) used by them, the longest path in this graph is $5N - 4$. Therefore, for *this* DG, this design is also optimal in execution time to within a small additive factor. The question we want to ask here is: *Is there any DG for the APP with shorter longest path based on the same sequential algorithm?* If the answer of this question is positive, we wish to design systematically 2-D RA's with execution time the same as the longest path of this new DG.

The paper is organized as follows: Firstly, some preliminary definitions are presented in Section II. In Section III, three different new designs for APP are given. They are a cylindrical array with execution time $\lfloor \frac{9N}{2} \rfloor - 2$ in Section III-B, a spherical array with $4N - 2$ in Section III-C, and an orthogonal array with $\lfloor \frac{9N}{2} \rfloor - 2$ in Section III-D. Finally, we make conclusions in Section IV which summarizes these new designs proposed in this paper.

Manuscript received February 5, 1992; revised June 24, 1993. This workwas supported by the National Science Council of ROC 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 9401098.

II. Preliminary Definitions

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

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 $0 \le i \le N-1$. The jth element in L is $l_i(j) = (i + j - 1)_{\mathrm{mod}N} + 1$, where $1 \le j \le N$.

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 $0 \le i \le N-1$. The jth element in R is $r_i(j) = (j - i + N - 1)_{\mathrm{mod}N} + 1$, where $1 \le j \le N$.

The following theorem shows the relationship between these two sequences.

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

Proof: See [6]. □

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 4 appears in the left-shift sequence $L(2;1,5)$, we have the position $j = r_2(4) = 2$.

The *execution time* (t_e) of an RA is defined to be the time interval between the first operation executed and the last result calculated. By *computation domain* (Θ) we mean the set of finite indexes (or nodes) used by an RIA. Let I and I' be two indexes in the computation domain Θ of an RIA A. Λ^T is a linear schedule in the first row of transformation matrix T.

Assuming a unitary time increment, the execution time of an RA obtaining from the RIA A by transformation matrix T is $t_e = \max_{I,I' \in \Theta}\{\Lambda^T(I - I')\} + 1$ [14]. The actual meanings of execution time of an RIA is the number of hyperplanes sweeping the index space. Let D be a dependence matrix of an algorithm. The result of $T \times D$, denoted by D', is a new dependence matrix in which the first row represents time delays and the remaining rows are the correspondent interconnections of array processors.

In a computation domain, values of a variable may be needed to broadcast from a source node to several destination nodes. We say the source node the *broadcast point*. A line connecting several broadcast points is a *broadcast line*. By aggregating several broadcast lines, we have a *broadcast plane*. Different variables may have different broadcast planes. A *broadcast center* is located at the intersection of broadcast lines or planes. Broadcast removing is achieved by transforming the broadcast dependencies into propagation dependencies. The broadcast point, in such case, is the position where propagation variable obtains its actual value.

The general procedure of designing RA by moving the broadcast planes is as follows.

1) The broadcast planes should be shifted to some position of the computation domain in order to reduce the longest path of the original DG.
2) According to these broadcast planes, the computation domain of the original problem can be decomposed into several phases. The parallel algorithm in each phase should satisfy all properties of the RIA, e.g., all entries of dependence matrix for each phase have constant and shift-invariant values. We call this form of parallel algorithm *multiphase RIA*.
3) For each phase, the execution order of the parallel algorithm is determined by an appropriate schedule vector. This schedule is called *m-phase schedule* and will be defined later. Each phase may have not the same schedule vector and even processor assignment function. Nevertheless, the parallel algorithm of every phase should begin execution at the same time.
4) There may have dependencies between two different phases. We call them *intra-phase dependencies*. These dependencies may relate to the problem size. The interconnection delays for mapping intra-phase dependencies can be determined by the m-phase schedule.
5) The final RA is constructed by composing RA's of all phases as well as the interconnections which come from mapping intra-phase dependencies.

The meaning of the recurrence equations used in this paper can be illustrated by an example, $c(i+1,j,k) = a(i,j,k) + b(i,j,k)$. If in the computation domain node $(1,2,3)$ is concerned, the above equation means that variables a and b of node $(1,2,3)$ are added on that node, then the result is sent to and saved in the variable c of node $(2,2,3)$. Note that, unless this result of computation is the final output we desired, it is meaningless and ignored if node $(i+1,j,k)$ locates outside its computation domain.

III. Design of 2-D Regular Arrays for Algebraic Path Problem

The APP [7], [12] is defined in terms of a weighted directed graph, $G = (V,E)$, with N vertices, where V is the set of vertices and E is the set of directed edges. The edge weight from vertex i to vertex j is denoted by $w_{i,j}$ which is defined in a *dioid* $(S,+,\times)$. In S, operation $+$ (addition) is a *commutative monoid* (closure, associativity, commutativity) and operation $\times$ (multiplication) is a *monoid* (closure, associativity). The elements 0 and 1 are *identity* (or *neutral*) elements of $+$ and $\times$, respectively. $\times$ is left and right distributive with respect to $+$ and 0 is an absorbing element for $\times$. The *quasi-inverse* of a, a^*, is defined by $1 + a + a \times a + a \times a \times a + \cdots$, where a is an element in S. Let C be an adjacency matrix of G, where

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

Then, the objective of APP is to compute matrix $\widetilde{C}$ in which each element, $\widetilde{c}_{i,j}$, is the sum of the weights of all distinct paths from vertex i to vertex j. The application of APP is diversified for different definitions of dioid [7]. For examples, the dioid $(\{0,1\}, \max, \min)$ is for the problem of transitive closure, $(\Re \cup \{\infty\}, \min, +)$ is for the shortest path problem, and $(\Re, +, \times)$ with $a^* = \frac{1}{1-a}$ when $a \neq 1$, where $a \in \Re$, is for Gauss-Jodan elimination. The following recurrence equations are used to compute matrix $\widetilde{C}$.

Algorithm 3.1: For all indices (i,j,k), $1 \leq i,j,k \leq N$, do

$$c_{i,j}^{k+1} = \begin{cases} c_{i,j}^k + c_{i,k}^k \times (c_{k,k}^k)^* \times c_{k,j}^k & \text{if } i \neq k \text{ and } j \neq k \\ (c_{k,k}^k)^* \times c_{k,j}^k & \text{if } i = k \text{ and } j \neq k \\ c_{i,k}^k \times (c_{k,k}^k)^* & \text{if } i \neq k \text{ and } j = k \\ (c_{k,k}^k)^* & \text{if } i = k \text{ and } j = k \end{cases}$$

initial values $c_{i,j}^1 = c_{i,j}$
final results $\widetilde{c}_{i,j} = c_{i,j}^{N+1}$ □

Removing broadcast dependencies in Algorithm 3.1 can be carried out by adding propagation variables $d(e)$ to carry $c(k,k,k)^*$ in horizontal (vertical) direction when $i = k (j = k)$. In addition, the variable a is to propagate $c(i,k,k) \times e(i,k,k)$ (that is, $c(i,k,k) \times c(k,k,k)^*$) in the j- direction except $i = k$ and the variable b is to propagate $c(k,j,k)$ in the i-direction except $j = k$. Now, we have Algorithm 3.2 and its DG (the k-plane) is shown in Fig. 1. The arc links in it represent to propagate computation results to the variable c from node (i,j,k) to node $(i,j,k+1)$ in the k-direction. The bold lines in it is to represent the broadcast lines. Broadcast planes can be constructed by aggregating these lines in the k-direction. The meaning of broadcast lines in here is that propagation variables a and b obtain their actual values in these lines and propagation variables d and e get $c(k,k,k)^*$ value from the intersection of broadcast lines for each k-plane. Note that $c(i,j,k)$ in Algorithm 3.2 corresponds to $c_{i,j}^k$ in Algorithm 3.1 and is computed at node (i,j,k) in DG shown in Fig. 1.

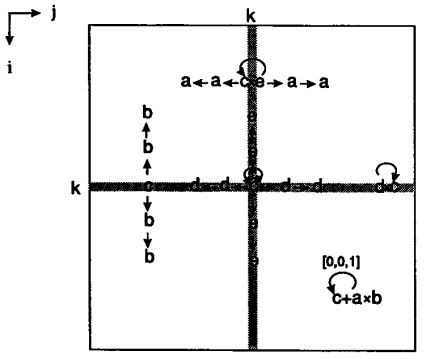


Fig. 1. Dependence graph for the original APP algorithm.

Algorithm 3.2: For all indices $(i,j,k), 1 \le i,j,k \le N$, do

$$c(i,j,k+1) = \begin{cases} c(i,j,k) + a(i,j,k) \times b(i,j,k) & \text{if } i \neq k \text{ and } j \neq k \\ d(i,j,k) \times c(i,j,k) & \text{if } i = k \text{ and } j \neq k \\ c(i,j,k) \times e(i,j,k) & \text{if } i \neq k \text{ and } j = k \\ c(i,j,k)^* & \text{if } i = k \text{ and } j = k \end{cases}$$

$$\begin{array}{ll} a(i,j,k) = c(i,j,k) \times e(i,j,k) & \text{if } i \neq k \text{ and } j = k \\ a(i,j+1,k) = a(i,j,k) & \text{if } i \neq k \text{ and } j \geq k \\ a(i,j-1,k) = a(i,j,k) & \text{if } i \neq k \text{ and } j \leq k \\ d(i,j,k) = c(i,j,k)^* & \text{if } i = k \text{ and } j = k \\ d(i,j+1,k) = d(i,j,k) & \text{if } i = k \text{ and } j \geq k \\ d(i,j-1,k) = d(i,j,k) & \text{if } i = k \text{ and } j \leq k \\ b(i,j,k) = c(i,j,k) & \text{if } i = k \text{ and } j \neq k \\ b(i+1,j,k) = b(i,j,k) & \text{if } i \geq k \text{ and } j \neq k \\ b(i-1,j,k) = b(i,j,k) & \text{if } i \leq k \text{ and } j \neq k \\ e(i,j,k) = c(i,j,k)^* & \text{if } i = k \text{ and } j = k \\ e(i+1,j,k) = e(i,j,k) & \text{if } i \geq k \text{ and } j = k \\ e(i-1,j,k) = e(i,j,k) & \text{if } i \leq k \text{ and } j = k \end{array}$$

initial values $c(i,j,1) = c_{i,j}$
final results $\tilde{c}_{i,j} = c(i,j,N+1)$ □

Although the variable c propagating in the k-direction is regular in Algorithm 3.2, the propagation vectors of variables a, d, b, and e are irregular in each k-plane since the broadcast lines are not fixed. Thus there does not exist any valid linear schedule and compatible processor assignment function for that algorithm. If we design regular arrays based on this DG, the derived arrays will become time-variant, like the design shown in [13], i.e., each PE executes different functions and propagates results to different directions at every time step. There are many drawbacks for time-variant designs, such as control is complex, array irregular, more functional unit necessary, and so on. Thus, they are not very suitable for VLSI implementation. However, this irregularity can be eliminated by reindexing (i and j) in each k-plane as S. Y. Kung *et al.* did in [12], [15]. Note that, in the following, long sequences of RIA statements will not be written since most of them are assignment statements for propagation variables. Nevertheless, we will show the DG's and readers can easily capture their essence and derive RIA's from them.

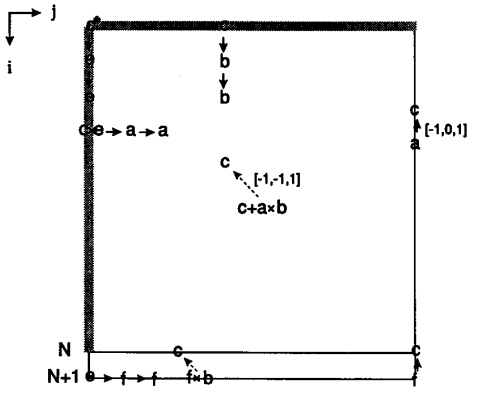


Fig. 2. Dependence graph for Design $a1$.

A. An Orthogonal Array With $t_e = 5N - 3$

In this section, we show the idea of "*reindexing*" proposed by S. Y. Kung *et al.* on their designing of orthogonal array for transitive closure and APP. This array is the optimal design so far. Note that, the orthogonal arrays for transitive closure [15] and APP [12] are the same in essence. The method used by S. Y. Kung is to reindex $c(i,j,k)$ to $c((i-k+N)_{\text{mod } N}+1, (j-k+N)_{\text{mod } N}+1, k)$ in Algorithm 3.2 for each k-plane, and the resulting DG is shown in Fig. 2. Comparing Fig. 1 with Fig. 2, we know that the *shifting of broadcast lines* has the same effect as reindexing.

By reindexing, the propagation vectors of variables a, d, b and e in Fig. 1 will be unified in every k-plane where variables a (d) and b (e) propagate in [0 1 0] and [1 0 0] direction, respectively. Consequently, the propagation vector of the variable c becomes [−1 −1 1] as well as three spiral arcs. They are $c(i-1,N,k+1) = c(i,1,k) \times e(i,1,k)$, $c(N,j-1,k+1) = d(1,j,k) \times c(1,j,k)$, and $c(N,N,k+1) = c(1,1,k)^*$. The first spiral arc can be eliminated by letting $c(i-1,N,k+1) = a(i,N,k)$. In addition, by introducing propagation variable f in [0 1 0] direction in the $(i = N+1)$-plane where $f(N+1,j+1,k) = f(N+1,j,k) = \cdots = f(N+1,1,k) = e(N+1,1,k) = e(N,1,k) = \cdots = e(1,1,k) = c(1,1,k)^*$, we can substitute $c(N,j-1,k+1) = f(N+1,j,k) \times b(N+1,j,k)$ and $c(N,N,k+1) = f(N+1,N,k)$ for the rest two spiral arcs. The reason why we let the variable f rather propagate in the $(i = N+1)$-plane than the $(i = N)$-plane is that if the former is selected then the dependence vector for $c(N,j-1,k+1) = f(N+1,j,k) \times b(N+1,j,k)$ becomes [−1 −1 1] which is the same as $c(i-1,j-1,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$. Such that, these two equations can share one interconnection in RA since they are never computed at the same time. Note that now the propagation variable d in Fig. 1 becomes useless because its duty has been replaced by the variable f in Fig. 2; thus, these arrows in the first row of DG in Fig. 2 disappear.

The dependence matrix D of Fig. 2 is

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

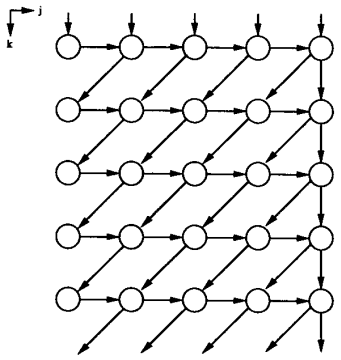


Fig. 3. Orthogonal array of Design $a1$.

By selecting transformation matrix T as follows:

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

we have

$$D' = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 \end{bmatrix}.$$

The orthogonal array for Fig. 2 by transformation matrix T is shown in Fig. 3. We call it Design $a1$. The execution time of Design $a1$ is $t_e = 5N-3$, since $t_e = \max_{I,I' \in \Theta}\{\Lambda^T(I-I')\}+1 = (N+1+N+3N)-(1+1+3)+1 = 5N-3$. One thing should be clarified is that there is a small constant factor difference between the execution time of Design $a1$ and that of Lewis and Kung's design $(5N-2)$. This is due to two factors: the first one is we add an extra row for every k-plane of the DG of Design $a1$, and the second one is definition difference for execution time, since they defined execution time as the time interval between the first operation executed and the last result outputted. Although there is minor differences between these two designs, the central concept of reindexing and moving broadcast plane are the same. That is just the idea we want to convey.

B. A Cylindrical Array With $t_e = \lfloor \frac{9N}{2} \rfloor - 2$

By inspecting the DG shown in Fig. 2, we know that the longest path is $5N-3$. Therefore, Design $a1$ is optimal for *this* DG because its execution time is also $5N-3$. The interesting question is : *Can we reindex Algorithm 3.2 in different ways to get another new DG with a shorter longest path?* The answer is, of course, YES. If we move the broadcast plane of the variable a (f) to the $\left(j = \lceil \frac{N}{2} \rceil\right)$-plane as shown in Fig. 4. The longest path in this DG is no longer $5N-3$ but $\lfloor \frac{9N}{2} \rfloor - 2$.

The criteria of designing a parallel algorithm for the DG shown in Fig. 4 are as follows.

1) By using the broadcast plane of the variable $a(f)$, we can decompose this DG into two phases. The first one is $j \le \lceil \frac{N}{2} \rceil$ and the second one is $j \ge \lceil \frac{N}{2} \rceil$.
2) It is easy to determine propagation vectors of variables $a(f)$ and $b(e)$ for each phase. In phase 1, the propagation vectors of variables a (f) and b (e) are in [0 −1 0] and [1 0 0] direction, respectively. In phase 2, the propagation vectors of variables $a(f)$ and $b(e)$ are in [0 1 0] and [1 0 0] direction, respectively.

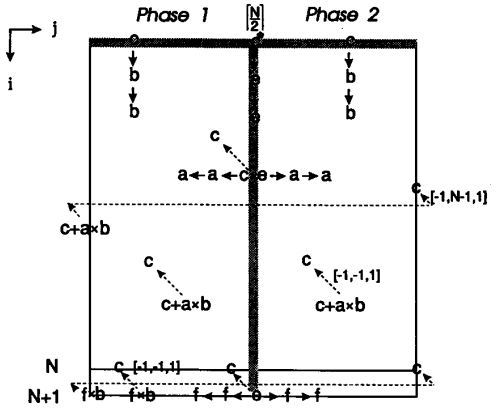


Fig. 4. Dependence graph for Design $a2$.

3) The propagation vector of the variable c is the same for both phases. It is in $[-1\ -1\ 1]$ direction as well as some spiral arcs. Note that it is meaningless if the index of a variable does not appear in its own phase. For example, $c(i-1, \lceil \frac{N}{2} \rceil - 1, k+1) = c(i, \lceil \frac{N}{2} \rceil, k) + a(i, \lceil \frac{N}{2} \rceil, k) \times b(i, \lceil \frac{N}{2} \rceil, k)$ in phase 2. We will ignore these equations.
4) There are four different types of spiral arcs for the variable c. They are $c(N, j-1, k+1) = c(1,1,k)^* \times c(1,j,k)$ if $j \neq \lceil \frac{N}{2} \rceil$, $c(N, \lceil \frac{N}{2} \rceil - 1, k+1) = c(1,1,k)^*$, $c(i-1, N, k+1) = c(i,1,k) + a(i,1,k) \times b(i,1,k)$, and $c(N,N,k+1) = f(N+1,1,k) \times b(N+1,1,k)$. The first two can be replaced by $c(N, j-1, k+1) = f(N+1, j, k) \times b(N+1, j, k)$ *if* $j \neq \lceil \frac{N}{2} \rceil$ and $c(N, \lceil \frac{N}{2} \rceil - 1, k+1) = e(N+1, \lceil \frac{N}{2} \rceil, k)$. The last two have dependencies between two phases. We call these *intra-phase dependencies* and use $\widehat{D}$ to denote them.
5) Since the $(k=1)$-plane has been reindexed, the initial values $c(i,j,1)$ are no longer equal to $c_{i,j}$. The relationship between $c(i,j,1)$ and $c_{i',j'}$ should be $i' = i$ and $j' = r_{\lfloor \frac{N-1}{2} \rfloor}(j)$ because the broadcast line of the variable a has been moved from the $(i,1,1)$-line in Fig. 1 to the $(i, \lceil \frac{N}{2} \rceil, 1)$-line in Fig. 4 and this is done by shifting right $\lfloor \frac{N-1}{2} \rfloor$ times cyclically in the j-direction for the $(k=1)$-plane.
6) The last problem needs to be solved is how to read final results $\tilde{c}_{i',j'}$ from $c(i,j,N+1)$. The same reason as above, we have $i' = i$ and $j' = r_{\lfloor \frac{N-1}{2} \rfloor}(j)$. By Theorem 2.1, we obtain $i = i'$ and $j = l_{\lfloor \frac{N-1}{2} \rfloor}(j')$.

The dependence matrices D_i of phase i, $1 \le i \le 2$, in Fig. 4 are

$$D_1 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } D_2 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

By selecting transformation matrices T_i for D_i as

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

we have

$$D_1' = \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix} \text{ and } D_2' = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}.$$

In this design, two phases have different schedule vectors. We call it *two-phase schedule* and this schedule can be generalized into *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*. Since only the m-phase uniform affine schedule is used to design 2-D RA's for APP in this paper, we give its formal definition as follow:

Definition 3.1: m-phase uniform affine schedule: $\Pi_{i,x}(I_i) = \Lambda_i^{\mathrm{T}} I_i + \nu_{i,x}$, where Λ_i^{T} is the linear part of $\Pi_{i,x}$ and I_i is an index in the computation domain Θ_i of phase i. $\nu_{i,x}$ is some constant 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 one phase, then ν_i is short for $\nu_{i,x}$ for m-phase uniform affine schedule and Λ_i^{T} becomes the first row of transformation matrix of T_i. Since the execution time of an RIA is the number of hyperplanes sweeping the index space, we have execution time $t_e = \max_{I_i, I_i' \in \Theta_i}\{(\Lambda_i^{\mathrm{T}} I_i + \nu_{i,x}) - (\Lambda_i^{\mathrm{T}} I_i' + \nu_{i,y}) + 1\}$, $1 \le i \le m$, for the m-phase uniform affine schedule, if all phases begin execution at the same time.

Thus, the two-phase uniform affine schedule for Fig. 4 is $\Pi_1(I_1) = \Lambda_1^{\mathrm{T}} I_1 + \nu_1 = [1 \quad -1 \quad 3] I_1 + \nu_1$ and $\Pi_2(I_2) = \Lambda_2^{\mathrm{T}} I_2 + \nu_2 = [1 \ 1 \ 3] I_2 + \nu_2$.

Since nodes $(i, \lceil \frac{N}{2} \rceil, k)$ belong to both phases, we have

$$[1 \quad -1 \quad 3] \begin{bmatrix} i \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_1 = [1 \quad 1 \quad 3] \begin{bmatrix} i \\ \lceil \frac{N}{2} \rceil \\ k \end{bmatrix} + \nu_2,$$

or $i - \lceil \frac{N}{2} \rceil + 3k + \nu_1 = i + \lceil \frac{N}{2} \rceil + 3k + \nu_2$. Let $\nu_2 = 0$, then

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

Hence, $\Pi_1(I_1) = [1 \quad -1 \quad 3] I_1 + 2\lceil \frac{N}{2} \rceil$ and $\Pi_2(I_2) = [1 \quad 1 \quad 3] I_2$. With this two-phase uniform affine schedule, it is easy to determine the interconnection delays, say $\hat{d}_i'$, for mapping intra-phase dependencies, where $1 \le i \le l$, l is the number of intra-phase dependencies.

1)

$$c(i-1, N, k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$$
$$\text{if } i \ne N+1 \text{ and } j = 1$$
$$i - 1 + N + 3k + 3 + \nu_2 = i - 1 + 3k + \nu_1 + \hat{d}_1' \Rightarrow \hat{d}_1'$$
$$= N + 3 - \nu_1 = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd} \end{cases}$$

2)

$$c(i-1, N, k+1) = f(i,j,k) \times b(i,j,k)$$
$$\text{if } i = N+1 \text{ and } j = 1$$
$$N + N + 3k + 3 + \nu_2$$
$$= N + 1 - 1 + 3k + \nu_1 + \hat{d}_2' \Rightarrow \hat{d}_2'$$
$$= N + 3 - \nu_1 = \begin{cases} 3 & \text{if } N \text{ is even} \\ 2 & \text{if } N \text{ is odd} \end{cases}$$

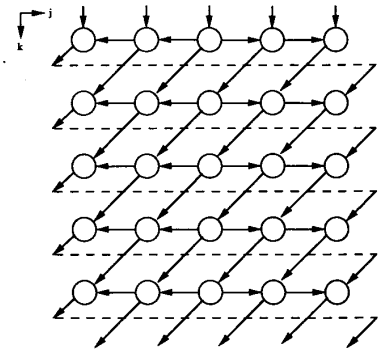


Fig. 5. Cylindrical array of Design $a2$.

From

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

we have spiral arcs for both types of intra-phase dependencies. Since these two intra-phase dependencies will never be computed at the same time, they can share one spiral arc.

After composing these two phases together, a cylindrical array for Fig. 4 is constructed as shown in Fig. 5. We call it Design $a2$. The execution time of Design $a2$ is $t_e = \lfloor \frac{9N}{2} \rfloor - 2$, since

$$t_e = \max\left\{ \left(N + 1 - 1 + 3N + 2\left\lceil \frac{N}{2} \right\rceil \right) - \left(1 - \left\lceil \frac{N}{2} \right\rceil + 3 + 2\left\lceil \frac{N}{2} \right\rceil \right) + 1, \text{ for phase 1} \right.$$
$$\left. (N + 1 + N + 3N) - (1 + \left\lceil \frac{N}{2} \right\rceil + 3) + 1, \text{ for phase 2} \right\}$$
$$= \max\left\{ 4N + \left\lceil \frac{N}{2} \right\rceil - 3,\ 5N - \left\lceil \frac{N}{2} \right\rceil - 2 \right\}$$
$$= \lfloor \frac{9N}{2} \rfloor - 2.$$

C. A Spherical Array With $t_e = 4N - 2$

In the last section, only by moving the broadcast plane of variable $a(f)$ to the center of DG in the j-direction, we could decrease the execution time from $5N - 3$ to $\lfloor \frac{9N}{2} \rfloor - 2$. In this section, we will show how we can get a spherical array with execution time of $4N - 2$ (if N is even, $4N - 3$ if N is odd) by moving both broadcast planes of variables $a(d)$ and $b(e)$ to the center of DG in the j- and i–direction, respectively. The DG is shown in Fig. 6. The general criteria of designing an RIA for this DG are similar to the last section. Besides, we have intra-phase dependencies between the seven phases shown at the bottom of the next page.

The initial values assigned and final results obtained are changed as follows.

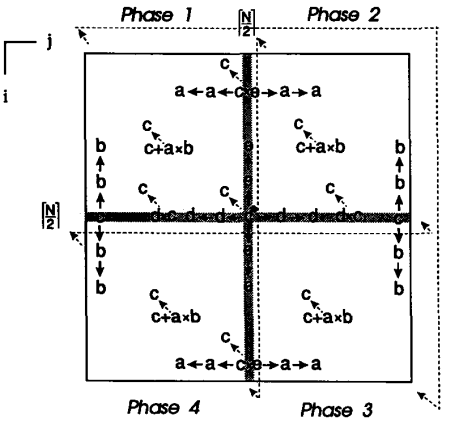


Fig. 6. Dependence graph for Design a3.

Initial values $c(i,j,1) = c_{i',j'}$ where $i' = r_{\lfloor \frac{N-1}{2} \rfloor}(i)$, $j' = r_{\lfloor \frac{N-1}{2} \rfloor}(j)$, and final results $\tilde{c}_{i',j'} = c(i,j,N+1)$ where $i = l_{\lfloor \frac{N-1}{2} \rfloor}(i')$, $j = l_{\lfloor \frac{N-1}{2} \rfloor}(j')$.

The dependence matrices D_i of phase i, $1 \le i \le 4$, in Fig. 6 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 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

$$D_1' = \begin{bmatrix} 1 & 1 & 5 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}, \quad D_2' = \begin{bmatrix} 1 & 1 & 3 \\ -1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix},$$
$$D_3' = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}, \quad D_4' = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix}.$$

We use a 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 $(\lceil \frac{N}{2} \rceil, \lceil \frac{N}{2} \rceil, k)$ 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 $\nu_3 = 0$, then

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

---

1) phase 2, 1: $c(i-1,N,k+1) = \begin{cases} c(i,j,k) + a(i,j,k) \times b(i,j,k) & \text{if } i \ne \lceil \frac{N}{2} \rceil \text{ and } j = 1 \\ d(i,j,k) \times c(i,j,k) & \text{if } i = \lceil \frac{N}{2} \rceil \text{ and } j = 1 \end{cases}$,

2) phase 3, 4: $c(i-1,N,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ if $j = 1$,

3) phase 2, 4: $c(i-1,N,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ if $i = \lceil \frac{N}{2} \rceil + 1$ and $j = 1$,

4) phase 4, 1 $c(N,j-1,k+1) = \begin{cases} c(i,j,k) + a(i,j,k) \times b(i,j,k) & \text{if } i = 1 \text{ and } j \ne \lceil \frac{N}{2} \rceil \\ c(i,j,k) \times e(i,j,k) & \text{if } i = 1 \text{ and } j = \lceil \frac{N}{2} \rceil \end{cases}$,

5) phase 3, 2: $c(N,j-1,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ if $i = 1$,

6) phase 4, 2: $c(N,j-1,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ if $i = 1$ and $j = \lceil \frac{N}{2} \rceil + 1$,

7) phase 3, 1: $c(N,N,k+1) = c(i,j,k) + a(i,j,k) \times b(i,j,k)$ if $i = 1$ and $j = 1$.

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, the interconnection delays for mapping intra-phase dependencies can be calculated by similar ways as Design $a2$. They are

$$(\hat{d}'_1, \cdots, \hat{d}'_7) = \begin{cases} (5,3,3,5,3,3,5) & \text{if } N \text{ is even} \\ (4,2,2,4,2,2,3) & \text{if } N \text{ is odd} \end{cases}.$$

The interconnections of RA for intra-phase dependencies can be gotten from the equation found at the bottom of the page. They represent spiral arcs as shown in Fig. 7. We call it Design a3 and its execution time is

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

$$\begin{aligned} t_e = \max\Bigg\{ & \left(-1-1+3N+4\left\lceil \frac{N}{2} \right\rceil\right) \\ & -\left(-\left\lceil \frac{N}{2} \right\rceil - \left\lceil \frac{N}{2} \right\rceil + 3 + 4\left\lceil \frac{N}{2} \right\rceil\right) + 1, \text{ for phase 1} \\ & \left(-1+N+3N+2\left\lceil \frac{N}{2} \right\rceil\right) \\ & -\left(-\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil + 3 + 2\left\lceil \frac{N}{2} \right\rceil\right) + 1, \text{ for phase 2} \\ & (N+N+3N) - \left(\left\lceil \frac{N}{2} \right\rceil + \left\lceil \frac{N}{2} \right\rceil + 3\right) + 1, \text{ for phase 3} \\ & \left(N-1+3N+2\left\lceil \frac{N}{2} \right\rceil\right) \\ & -\left(\left\lceil \frac{N}{2} \right\rceil - \left\lceil \frac{N}{2} \right\rceil + 3 + 2\left\lceil \frac{N}{2} \right\rceil\right) + 1, \text{ for phase 4} \Bigg\} \end{aligned}$$

$$\begin{aligned} &= \max\left\{3N + 2\left\lceil \frac{N}{2} \right\rceil - 4,\ 4N-3,\right. \\ &\quad \left. 5N - 2\left\lceil \frac{N}{2} \right\rceil - 2,\ 4N-3\right\} \\ &= \begin{cases} 4N-2 & \text{if } N \text{ is even} \\ 4N-3 & \text{if } N \text{ is odd} \end{cases}. \end{aligned}$$

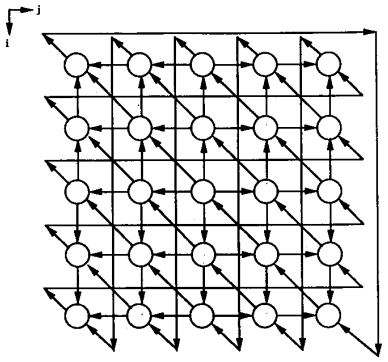


Fig. 7. Spherical array of Design a3.

D. An Orthogonal Array With $t_e = \lfloor \frac{9N}{2} \rfloor - 2$

In this section, we propose an orthogonal array which eliminates the spiral arcs in Design $a2$ but remains the same execution time of $\lfloor \frac{9N}{2} \rfloor - 2$.

We explain our design criteria as follows.

1) In Design $a2$, the spiral arcs result from intra-phase dependencies. Removing intra-phase dependencies can be done by letting the variable c in the $(k+1)$-plane get their k-plane value from different directions for two phases and using propagation vectors of the variable a to replace the intra-phase dependencies. In addition, it is necessary to divide DG into two parts in the k-direction.

   Part 1 ($k \le \lceil \frac{N-1}{2} \rceil$): The propagation vectors are in [−1 0 1] and [−1 −1 1] for the variable c in phase 1 and phase 2, respectively. Now, we can substitute $c(i-1, N, k+1) = a(i, N, k)$ for the intra-phase dependencies $c(i-1, N, k+1) = c(i, 1, k) + a(i, 1, k) \times b(i, 1, k)$ in Fig. 4. The other intra-phase dependencies, $c(N, N, k+1) = f(N+1, 1, k) \times b(N+1, 1, k)$, can be replaced by $c(N, N, k+1) = f(N+1, N, k)$. All dependencies in part 1 are shown in Fig. 8(a).

   Part 2 ($k > \lceil \frac{N-1}{2} \rceil$): The similar method in part 1 is used here again except the propagation vectors of the variable c in phase 1 and phase 2 of part 2 become [−1 1 1] and [−1 0 1], respectively. Besides, we use $c(i-1, 1, k+1) = a(i, 1, k)$ and $c(N, 1, k+1) = f(N+1, 1, k)$ to replace intra-phase dependencies as described above. All dependencies in part 2 are shown in Fig. 8(b).

2) Notice that the initial values assigned and final results obtained are also changed as follows: Initial values

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

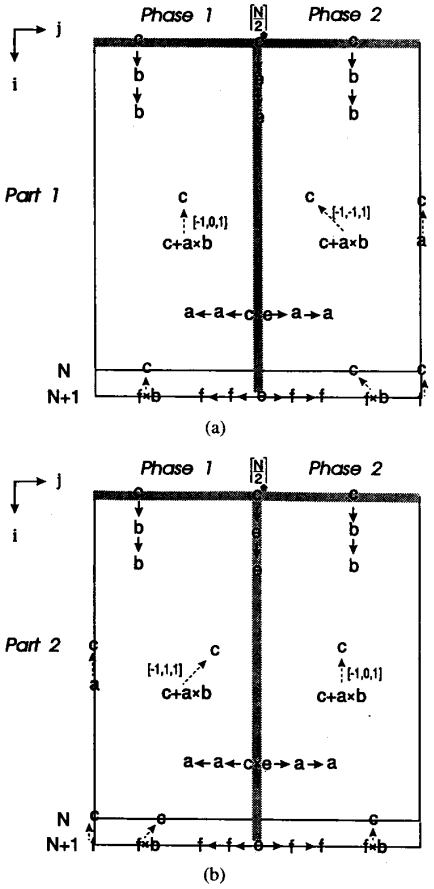


Fig. 8. (a) Part 1. (b) Part 2 of dependence graph for Design a4.

$$c(i,j,1) = c_{i',j'} \text{ where } i' = i,$$

$$j' = \begin{cases} N-j+1 & \text{if } j < \lceil \frac{N}{2} \rceil \\ r_{\lfloor \frac{N-1}{2} \rfloor}(j) & \text{if } j \geq \lceil \frac{N}{2} \rceil \end{cases},$$

and final results $\tilde{c}_{i',j'} = c(i,j,N+1)$ where $i = i'$,

$$j = \begin{cases} l_{\lceil \frac{N}{2} \rceil}(j') & \text{if } j' < \lceil \frac{N+1}{2} \rceil \\ N-j'+1 & \text{if } j' \geq \lceil \frac{N+1}{2} \rceil \end{cases}.$$

The dependence matrices D_{ij} for phase j of part i, $1 \leq i,j \leq 2$, are

$$D_{11} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_{12} = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

$$D_{21} = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad D_{22} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

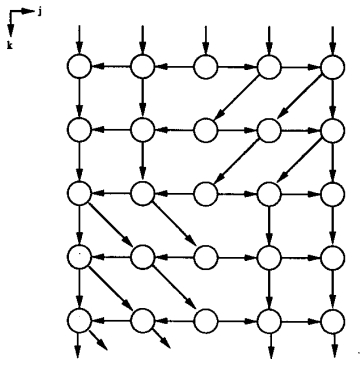


Fig. 9. Orthogonal array of Design $a4$.

By selecting transformation matrices T_{ij}, $1 \leq i,j \leq 2$, for D_{ij} as follows

$$T_{11} = \begin{bmatrix} 1 & -1 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_{12} = \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$T_{21} = \begin{bmatrix} 1 & -1 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad T_{22} = \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

we have

$$D'_{11} = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \quad D'_{12} = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 \end{bmatrix},$$

$$D'_{21} = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & -1 & 1 & 0 \end{bmatrix}, \quad D'_{22} = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

After composing the designs of part 1 and part 2, an orthogonal array for Fig. 8 is constructed as shown in Fig. 9. We call it Design $a4$.

A two-phase uniform affine schedule same as Π_1 and Π_2 in Section III-B can be derived for the Design $a4$, where Π_1 is for phase 1 of part 1 and part 2, and Π_2 is for phase 2 of part 1 and part 2. The execution time of Design $a4$ is the same as Design $a2$, that is, $t_e = \lfloor \frac{9N}{2} \rfloor - 2$.

IV. Conclusion

In this paper, by decomposing a dependence graph into multiple phases with appropriate m-phase schedule functions, three novel more efficient regular arrays for APP have been derived systematically. They are summarized at Table I. Since the linear schedule vector is adopted by every phase for all designs, the derived arrays have the properties of simple-control and time-invariance, i.e., each PE executes fixed computation and passes results to fixed PE's for every time step. Moreover, every design is time optimal for its DG since its execution time is the same as the longest path of its DG.

TABLE I
COMPARISON OF DIFFERENT DESIGNS FOR ALGEBRAIC PATH PROBLEM

Design	*Dependence graph*	*Array type*	*Execution time*	*Broadcast plane of*	
				variable a	*variable b*
a1	Fig. 3.2	orthogonal	$5N-3$	$j=1$	$i=1$
*a2**	Fig. 3.4	cylindrical	$\lceil\frac{9N}{2}\rceil-2$	$j=\lceil\frac{N}{2}\rceil$	$i=1$
a3‡	Fig. 3.6	spherical	$4N-2$†	$j=\lceil\frac{N}{2}\rceil$	$i=\lceil\frac{N}{2}\rceil$
*a4**	Fig. 3.8	orthogonal	$\lceil\frac{9N}{2}\rceil-2$	$j=\lceil\frac{N}{2}\rceil$	$i=1$

* new design for algebraic path problem

† $4N-2$ if N is even, $4N-3$ if N is odd

‡ pre-loading initial adjacency matrix is necessary

REFERENCES

[1] H. T. Kung and C. E. Leiserson, "Systolic arrays for VLSI," in *Proc. 1978 Society for Indust. Appl. Math.*, 1979, pp. 256–282.

[2] S. K. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor arrays," *Proc. IEEE*, vol. 76, pp. 259–269, Mar. 1988.

[3] ——, "What is a systolic algorithm,"? in *Proc. SPIE Highly Parallel Signal Processing Architectures*, 1986, pp. 34–48.

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

[5] V. P. Roychowdhury and T. Kailath, "Subspace scheduling and parallel implementation of non-systolic regular iterative algorithms," *J. VLSI Signal Processing*, vol. 1, pp. 127–142, 1989.

[6] J. C. Tsay and P. Y. Chang, "Some new designs of 2-D array for matrix multiplication and transitive closure," *IEEE Trans. Parallel Distrib. Syst.*, June 1991. submitted.

[7] M. Gondran and M. Minoux, *Graphs and Algorithms*. Chichester, UK: Wiley, 1984.

[8] G. Rote, "A systolic array algorithm for the algebraic path problem," *Computing*, vol. 34, pp. 191–219, 1985.

[9] Y. Robert and D. Trystram, "Systolic solution of the algebraic path problem," in *Proc. Int. Workshop on Systolic Arrays*, 1986, pp. 171–180.

[10] F. J. Nunez and M. Valero, "A block algorithm for the algebraic path problem and its execution on a systolic array," in *Proc. Int. Conf. on Systolic Arrays*, 1988, pp. 265–274.

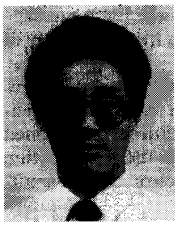
[11] A. Benaini and Y. Robert, "Spacetime-minimal systolic architectures for gaussian elimination and the algebraic path problem," in *Proc. Int. Conf. on Applicat. Specific Array Processors*, 1990, pp. 747–757.

[12] P. S. Lewis and S. Y. Kung, "An optimal systolic array for the algebraic path problem," *IEEE Trans. Comput.*, vol. 40, pp. 100–105, Jan. 1991.

[13] S. Rajopadhye, "An improved systolic algorithm for the algebraic path problem," *Integration, the VLSI J.*, vol. 14, pp. 279–296, Feb. 1993.

[14] 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.

[15] 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. 36, pp. 603–614, May 1987.



Pen-Yuang Chang was born in Kaohsiung, R.O.C. in 1962. He received the M.S. degree in computer science from the National 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 include systolic arrays and multimedia information systems.

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 the National Chiao-Tung University in 1968 and 1975, respectively.

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