Design of Space-Optimal Regular Arrays for Algorithms with Linear Schedules

Jong-Chuang Tsay and Pen-Yuang Chang

Abstract—**The problem of designing space-optimal 2D regular arrays for $N \times N \times N$ cubical mesh algorithms with linear schedule $ai + bj + ck$, $1 \le a \le b \le c$, and $N = nc$, is studied. Three novel nonlinear processor allocation methods, each of which works by combining a partitioning technique (*gcd-partition*) with different nonlinear processor allocation procedures (*traces*), are proposed to handle different cases. In cases where $a + b \le c$, which are dealt with by the first processor allocation method, space-optimal designs can always be obtained in which the number of processing elements is equal to $\frac{N^2}{c}$. For other cases where $a + b > c$ and either $a = b$ and $b = c$, two other optimal processor allocation methods are proposed. Besides, the closed form expressions for the optimal number of processing elements are derived for these cases.**

Index Items—**Algorithm mapping, data dependency, linear schedule, matrix multiplication, optimizing compiler, space-optimal, systolic array.**

I. Introduction

REGULAR arrays, or systolic arrays [1], [2], have been proposed for over a decade. They are special purpose parallel devices composed of several processing elements (PEs) whose interconnections have the properties of regularity and locality. Because of these properties, regular architectures are very suitable for VLSI implementation.

The procedure for synthesizing regular arrays from systems of recurrence equations, or nested loops, has two major steps. The first one is *regularization* [3], [4], [5], or *uniformization* [6], which includes variable full indexing [7] (defining all variables on the same index dimension only once); broadcast removing [8] (replacing broadcast vectors with pipeline vectors); reindexing [9] (re-routing pipeline vectors so that they are oriented in the same direction); and so on [10], [11], [12]. After regularization, the original system of recurrence equations is transformed into an equivalent *system of uniform recurrence equations* (SURE) [13] or a *regular iterative algorithm* (RIA) [14]. A *dependence graph* (DG) is a graphical representation of such an algorithm, in which each node corresponds to an index vector and each link represents a dependence vector between two nodes. A *dependence matrix* D is the collection of all dependence vectors in an algorithm; each column in D is a dependence vector. The second step is to find the *spacetime mapping* transformation matrix $T = \begin{bmatrix} \Pi^T \\ S^T \end{bmatrix}$ [13], [14], [15], [16] with a *valid* linear schedule vector Π^T and a *compatible* processor allocation matrix Π^T for an SURE. A schedule is valid if the precedence constraints imposed by an SURE are satisfied and a processor allocation is compatible with its schedule if two different computations are not executed on the same PE at the same time.

In the past, most researchers focused their efforts on regularization, and the first half of spacetime mapping, the time mapping [13], [14], [15], [17], [18], [19]. In particular, the problem of how to find an optimal linear schedule for an SURE has attracted special interest [20]. Only recently has its counterpart problem—that of how to design space-optimal regular arrays in which the number of PEs is minimal for an SURE executed by a given linear schedule—been studied in the literature [21], [22], [23], [24], [25], [26], [27], [28]. Studies of this latter problem fall into three categories. The first class includes [21], [22], [23], [24], [25], in which the following method is adopted: first, from the given DG and linear schedule, a set $\mathcal{V}$ of nodes is found such that all nodes in the set are scheduled to be executed at the same time and the set size $|\mathcal{V}|$ is maximal. We call such a set a *maximum concurrent set* with respect to the given linear schedule. Second, spacetime mapping is applied to assign the nodes of the DG to PEs. Any PE which has not been assigned to execute a node in $\mathcal{V}$ is piled to a PE which executes a node in $\mathcal{V}$ and has disjoined activation time intervals. This method can indeed be used to design space-optimal regular arrays. However, it has two drawbacks. The first is that finding a maximum concurrent set with respect to a linear schedule $\Pi(I) = ai + bj + ck$ is not an easy task, i.e., the nodes must be represented in a closed form expression by the parameters a, b, c, and N, where N is the problem size parameter. Thus this method designs space-optimal regular arrays case by case. Second, piling PEs results in spiral links and increases irregularity for the resulting arrays.

The second class of methods for designing space-optimal arrays [25], [26], [27] deals with this problem by grouping $\Pi^T\varphi$ PEs into a single one, where φ^T is the projection vector with respect to the space mapping matrix S^T ($S^T\varphi = 0$). Thus the resulting array has a 100% pipelining rate. The advantages of this method are that it is not necessary to find a maximum concurrent set, the resulting array does not have spiral links, and the method is applicable to all SUREs. However, this method cannot guarantee that the design is space-optimal, because a 100% pipelining rate in the array does not imply space-optimality. The regular array for matrix multiplication is a

Manuscript received Feb. 17, 1993; revised Apr. 1, 1994.

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; e-mail jctsay@csunix.csie.nctu.edu.tw.

IEEECS Log Number C95022.

good example; it has $\Pi^T\varphi = [1\ \ 1\ \ 1]\ [0\ \ 0\ \ 1]^T = 1$ but is not space-optimal.

In the third class [28], an upper bound on the length of the optimal projection vector is developed and an enumerative search procedure for finding the optimal projection vector is provided. However, this approach does not provide space-optimal designs in a general way, because only linear processor allocation is considered.

For the problem of designing space-optimal regular arrays, two interesting questions we want to investigate are: first, how many PEs are needed to design a regular array for a given $N \times N \times N$ DG with a linear schedule $\Pi(I) = ai + bj + ck$. For simplicity, the problem domain in this paper is restricted to a cubical mesh. Second, how to design a regular array which is not only space-optimal but also is locally connected, provides balanced loads, and allows for simple control. In the following, several nonlinear processor allocation procedures will be proposed to design space-optimal regular arrays for different cases. The linear schedule $\Pi(I) = ai + bj + ck$ with $1 \le a \le b \le c$ is considered. It is easy to see that an algorithm with an arbitrary linear schedule, say $\Pi'(I) = a'i + b'j + c'k$, where a', b', and c' are all non-zero integers, can be transformed to an equivalent one with $\Pi(I) = ai + bj + ck$ and $1 \le a \le b \le c$ by applying permutation transformations [29]. Therefore, without loss of generality, we assume that $1 \le a \le b \le c$. In addition, for simplicity, we also assume that $N = nc$ in the following descriptions. Furthermore, an algorithm with an arbitrary uniform affine schedule [30] $\Pi_u(I) = ai + bj + ck + u$ can also be transformed to an equivalent one with a linear schedule by *timespace mapping* or *dimension extension* [31].

In Section II, some important definitions are given. In Section III, the gcd-partition method and the first processor allocation procedure will be introduced to design space-optimal regular arrays for the case of $a + b \le c$. It is proven that in such a case $\frac{N^2}{c}$ is the minimum number of PEs required or the size of a maximum concurrent set for the given linear schedule. A space-optimal regular array for transitive closure and algebraic path problem will be given to illustrate our method. In Section VI, the other two optimal processor allocation procedures are developed to handle the cases of $a = b$ and $b = c$, respectively, when $a + b > c$, and the closed form expressions for the size of a maximum concurrent set will also be given for both cases. A new space-optimal regular array for matrix multiplication will also be given. Finally, our concluding remarks are presented in Section V.

II. Preliminaries

The variables used in this paper are all integral numbers. Each index vector in the computation domain Ψ is denoted by $I = [i\ j\ k]^T$, where $1 \le i, j, k \le N$. The linear schedule $\Pi(I) = ai + bj + ck$ with $1 \le a \le b \le c$ is a *normalized* one, i.e., $\gcd(a, b, c) = 1$.

DEFINITION 2.1. [*locally connected*]. *A regular array is said to be locally connected iff any communication link between two PEs has a displacement vector independent of the size of the problem.*

DEFINITION 2.2. [*space-optimal*]. *A regular array is said to be space-optimal with respect to a given linear schedule for an SURE iff the number of PEs used (denoted by PEused) is equal to the size of a maximum concurrent set, or the minimum number of PEs required (PEmin), for the given linear schedule.*

Clearly, for a given linear schedule, if $PE_{used} < PE_{min}$ then two different computations will be executed on the same PE at the same time.

DEFINITION 2.3. [*time-tag*]. *A time-tag $v = ai + bj + ck$ (or, in indexed notation, $u^k_{i,j}$) is a positive integral number assigned to each node or index vector $[i\ j\ k]^T$ in the computation domain Ψ of an SURE to represent the execution time of the index vector with respect to the linear schedule vector $[a\ b\ c]^T$.*

DEFINITION 2.4. [*modulo set*]. *Given a positive integer r, $0 \le r < c$, a modulo set $\phi(r)$ is a set of nodes of Ψ such that each node of $\phi(r)$ has an assigned time-tag v satisfying $mod(v, c) = r$, where $mod(a, b)$ denotes the remainder of a divided by b.*

A multiset [32] $\boldsymbol{M}$ is a collection of not necessarily distinct elements. It may be thought of as a set in which each element, say v, has an associated positive integer, its multiplicity $C_v(\boldsymbol{M})$, to represent the number of vs in $\boldsymbol{M}$. For example, $\boldsymbol{M} = \{1, 1, 2, 2, 2, 2, 3\}$ is a multiset, where $C_1(\boldsymbol{M}) = 2$, $C_2(\boldsymbol{M}) = 4$, and $C_3(\boldsymbol{M}) = 1$. We use the multiset W to denote the collection of the time-tags of all the index vectors in Ψ for an SURE.

DEFINITION 2.5. [*partition*]. *A possible partition $\mathcal{P}$ of W is written as $\{V_1, V_2, \cdots V_m\}$, where each V_i in $\mathcal{P}$ is a set of time-tags and the partition size $|\mathcal{P}| = m$.*

DEFINITION 2.6. [*optimal partition*]. *An optimal partition is a partition such that its partition size is minimal with respect to all possible partitions of W.*

EXAMPLE 2.1. *This example demonstrates the concept of an optimal partition: Let $W = \{1, 2, 2, 3, 3, 3, 4, 4, 5\}$. A possible partition $\mathcal{P}_1 = \{V_1, V_2, V_3, V_4\} = \{\{1, 2, 3\}, \{2, 3\}, \{4\}, \{3, 4, 5\}\}$. However, $\mathcal{P}_1$ is not optimal, because it is easy to find an optimal partition $\mathcal{P}_2 = \{V_1, V_2, V_3\} = \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}\}$ such that $|\mathcal{P}_2| < |\mathcal{P}_1|$. Of course, there may exist several optimal partitions, but at least one optimal partition always exists.*

The following lemma states a useful property of optimal partitions.

LEMMA 2.1. *A partition $\mathcal{P}$ is optimal iff there exists a time-tag $v \in V_i$ for all $V_i \in \mathcal{P}$.*

PROOF. For any partition, we have $|\mathcal{P}| \ge \max_{u \in W} C_u(W)$.

[If part] If there exists a time-tag $v \in V_i$ for all $V_i \in \mathcal{P}$, we have $|\mathcal{P}| = C_v(W) = \max_{u \in W} C_u(W)$. Then $|\mathcal{P}|$ is minimal with respect to all possible partitions of W, i.e., the partition $\mathcal{P}$ is optimal.

[Only if part] It is obvious that if $|\mathcal{P}|$ is minimal with respect to all possible partitions of W, then $|\mathcal{P}| = \max_{u \in W} C_u(W) \equiv C_v(W)$; this implies that there exists a time-tag $v \in V_i$ for all $V_i \in \mathcal{P}$. □

The following definitions are important because they are the basis for finding optimal partitions systematically.

DEFINITION 2.7 [*segment, segment domain*]. *A segment is defined as an $f \times g$ matrix*

$$G^{\gamma}_{\alpha,\beta} = \begin{bmatrix} u^{\gamma}_{(\alpha-1)f+1,(\beta-1)g+1} & \cdots & u^{\gamma}_{(\alpha-1)f+1,\beta g} \\ \vdots & \ddots & \vdots \\ u^{\gamma}_{\alpha f,(\beta-1)g+1} & \cdots & u^{\gamma}_{\alpha f,\beta g} \end{bmatrix},$$

where

$$1 \le \alpha \le \left\lceil \frac{N}{f} \right\rceil, 1 \le \beta \le \left\lceil \frac{N}{g} \right\rceil,$$

and $\gamma = k$ (see Fig. 1). The *segment domain* Θ is constructed by the set of segments.

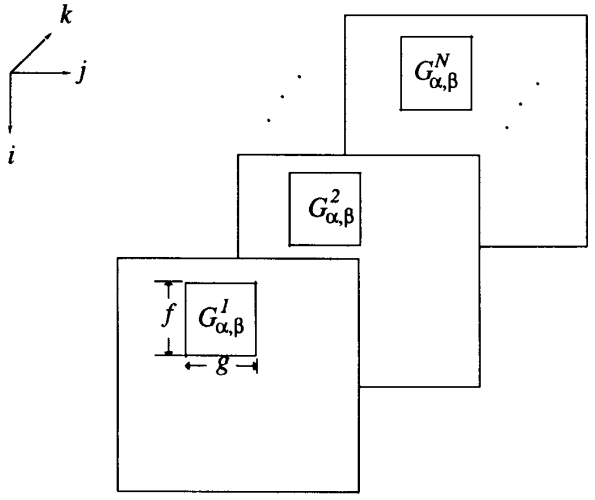

Fig. 1. The concept of a segment.

We use the notation $u^k_{i,j} \in G^{\gamma}_{\alpha,\beta}$ to represent that $u^k_{i,j}$ is an element of the segment $G^{\gamma}_{\alpha,\beta}$. The value of a pair of comma-separated integers (p, q) gives the coordinates of the location of the time-tag on the segment. The first number is the vertical coordinate, and the second number is the horizontal coordinate, measured from the top left corner of the segment. We say that two time-tags of different segments have the same (p, q) location if they are located at the same (p, q) coordinates in their respective coordinate systems.

DEFINITION 2.8 [*module, cluster*]. *A module $G_{\alpha,\beta}$ is a set of segments and a cluster G is a set of modules.*

A time-tag $u^k_{i,j}$ is said to be in module $G_{\alpha,\beta}$ (denoted by $u^k_{i,j} \in G_{\alpha,\beta}$) if $u^k_{i,j} \in G^{\gamma}_{\alpha,\beta}$ and $G^{\gamma}_{\alpha,\beta} \in G_{\alpha,\beta}$. Similarly, a time-tag $u^k_{i,j}$ is said to be in cluster G (denoted by $u^k_{i,j} \in G$) if $u^k_{i,j} \in G_{\alpha,\beta}$ and $G_{\alpha,\beta} \in G$. Various grouping methods can be used to construct modules. For example, by simply collecting all segments $G^{\gamma}_{\alpha,\beta}$ in the k-direction, the module $G_{\alpha,\beta} = \{G^1_{\alpha,\beta}, \cdots, G^N_{\alpha,\beta}\}$ is constructed. A more complex grouping method is described by the following concept:

DEFINITION 2.9 [*trace*]. *A trace $(G^{\gamma_1}_{\alpha_1,\beta_1}, \Theta)$ is a module consisting of the set of segments on a directed path that begins from $G^{\gamma_1}_{\alpha_1,\beta_1}$, passes through $G^{\gamma_2}_{\alpha_2,\beta_2}$, and reaches $G^{\gamma_n}_{\alpha_n,\beta_n}$, denoted by*

$$\text{trace}\left(G^{\gamma_1}_{\alpha_1,\beta_1}, \Theta\right) = < G^{\gamma_1}_{\alpha_1,\beta_1}, G^{\gamma_2}_{\alpha_2,\beta_2}, \cdots, G^{\gamma_n}_{\alpha_n,\beta_n} >,$$

where all segments of a trace belong to the segment domain Θ.

DEFINITION 2.10 [*size*]. *The size of a segment, module, and cluster, denoted by $|G^{\gamma}_{\alpha,\beta}|$, $|G_{\alpha,\beta}|$, and $|G|$ represent the number of time-tags, segments, and modules in them, respectively.*

DEFINITION 2.11 [*modulo-s segment*]. *A segment $G^{\gamma}_{\alpha,\beta}$ is said modulo-s iff for every time-tag $v \in G^{\gamma}_{\alpha,\beta}$ there does not exist another time-tag $v_1 \in G^{\gamma}_{\alpha,\beta}$ such that mod(v, s) = mod(v_1, s), where $s > 0$.*

DEFINITION 2.12 [*isomorphic segments*]. *Two $f \times g$ segments $G^{\gamma}_{\alpha,\beta}$, $G^{\gamma_1}_{\alpha_1,\beta_1}$ are said to be isomorphic iff for any two time-tags $v \in G^{\gamma}_{\alpha,\beta}$, $v_1 \in G^{\gamma_1}_{\alpha_1,\beta_1}$, if they have the same (p, q) location, then $(v, |G^{\gamma}_{\alpha,\beta}|) = mod(v_1, |G^{\gamma_1}_{\alpha_1,\beta_1}|)$.*

DEFINITION 2.13 [*free segment*]. *A segment is said to be free iff it has not yet been allocated to a module, and the notation free(Θ) represents the set of free segments in the segment domain Θ.*

DEFINITION 2.14 [*minimal index vector, minimal segment*]. *An index vector $I = [i\ j\ k]^T$ is said to be minimal with respect to a domain iff there does not exist another $I_1 = [i_1\ j_1\ k_1]^T$ in this domain such that $(k_1 < k) \vee ((k_1 = k) \wedge (j_1 < j)) \vee ((k_1 = k) \wedge (j_1 = j) \wedge (i_1 < i))$. A segment $G^{\gamma}_{\alpha,\beta}$ is said to be minimal with respect to a set of segments Γ, denoted by $G^{\gamma}_{\alpha,\beta} = min\{\Gamma\}$, iff there is a time-tag in $G^{\gamma}_{\alpha,\beta} \in \Gamma$ assigned to the minimal index vector $I = [i\ j\ k]^T$.*

DEFINITION 2.15 [*elementary module*]. *A module $G_{\alpha,\beta}$ is said to be elementary iff for any two time-tags $v_1, v_2 \in G_{\alpha,\beta}$, $v_1 \neq v_2$.*

With this definition, it is obvious that an elementary module is a set of time-tags. The concept of an elementary module is very important. In our processor allocation procedures, each module is allocated to one PE. An elementary module ensures that no two different computations are scheduled to be executed on the same PE at the same time, i.e., the processor allocation procedure is compatible with the given schedule.

DEFINITION 2.16 [*elementary cluster*]. *A cluster G is said to be elementary iff every module $G_{\alpha,\beta} \in G$ is elementary.*

DEFINITION 2.17. [*optimal cluster*]. *An elementary cluster G is said to be optimal iff $|G|$ is minimal.*

LEMMA 2.2. *The size $|G|$ of an optimal cluster is equal to the size $|\mathcal{P}|$ of an optimal partition for W.*

PROOF. Under the assumption that every module $G_{\alpha,\beta} \in G$ is elementary, we have $|G| \ge \max_{v \in G} C_v(G)$. Thus G is an optimal cluster when $|G| = \max_{v \in G} C_v(G)$. From Lemma 2.1 and the observation that the multiset W is equal to the multiset G, we have $|G| = |\mathcal{P}|$. □

Designing a space-optimal regular array is equivalent to

finding an optimal cluster. In the following sections, several procedures for finding an optimal cluster will be introduced. The central concept is to partition every *ij*-plane of the DG into several segments, to group these segments into several elementary modules, and to keep the number of modules in a cluster to a minimum.

III. Processor Allocation for $a + b \leq c$

A. Procedure

In this section, a new processor allocation procedure for algorithms with linear schedules is proposed. This procedure guarantees that the derived regular array is space-optimal for an SURE with a linear schedule $\Pi(I) = ai + bj + ck$ when $a + b \leq c$. For other situations, although a space-optimal regular array cannot always be obtained, our procedure still decreases the number PEs used from N^2 (if a 2×3 linear processor allocation matrix is used) to $\frac{N^2}{c}$.

PROCEDURE 3.1. *Given a 3D SURE with a linear schedule $\Pi(I) = ai + bj + ck$ and $a + b \leq c$, a space-optimal regular array can always be obtained by partitioning every ij-plane of the DG of the SURE into*

- $\frac{c}{g} \times g$ *segments* $G^{\gamma}_{\alpha,\beta}$, *where*

$$g = gcd(a,c), 1 \leq \alpha \leq \frac{gN}{c}, 1 \leq \beta \leq \frac{N}{g}, \text{and } 1 \leq \gamma \leq N,$$

or

- $g \times \frac{c}{g}$ *segments* $G^{\gamma}_{\alpha,\beta}$, *where*

$$g = gcd(b,c), 1 \leq \alpha \leq \frac{N}{g}, 1 \leq \beta \leq \frac{gN}{c}, \text{and } 1 \leq \gamma \leq N.$$

Then each module (PE) is constructed by collecting the set of segments in the k-direction. □

This method of partitioning is called *gcd-partitioning*. Using this method, a module is constructed by tracing the set of segments in the k-direction. This method of constructing modules is designated Trace_1 and can be defined as $G_{\alpha,\beta} = \text{Trace}_1(G^1_{\alpha,\beta}, \Theta) = < G^1_{\alpha,\beta}, G^2_{\alpha,\beta}, \cdots, G^N_{\alpha,\beta} >$. Using the same gcd-partition but different traces to construct modules results in different processor allocation procedures.

EXAMPLE 3.1 [*transitive closure and algebraic path problem*]. *From the DG of transitive closure derived by S.Y. Kung et al. [9] (DG-3 in their paper), the dependence matrix can be written as*

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

and the corresponding optimal linear schedule is $\Pi(I) = i + j + 3k$. Thus by applying Procedure 3.1, time-tags on every ij-plane can be gcd-partitioned into several 3×1 segments, as shown in Fig. 2(a). A module (PE) is constructed by collecting segments in the k-direction (Trace_1), as shown in Fig. 2(b). A space-optimal regular array with only $\frac{N^2}{3}$ PEs can be obtained as shown in Fig. 2(c).

The DG for the algebraic path problem derived by Lewis and Kung [33, Fig. 3] can be reindexed as $([i\ j\ k]^T \leftarrow$

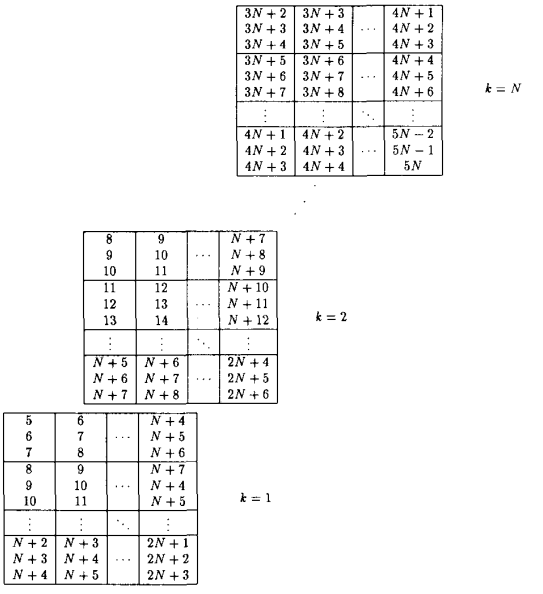

Fig. 2(a). The DG of transitive closure and algebraic path problem is gcd-partitioned into several 3×1 segments.

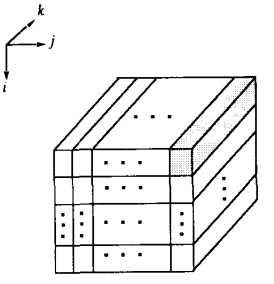

Fig. 2(b). Constructing modules by Trace_1.

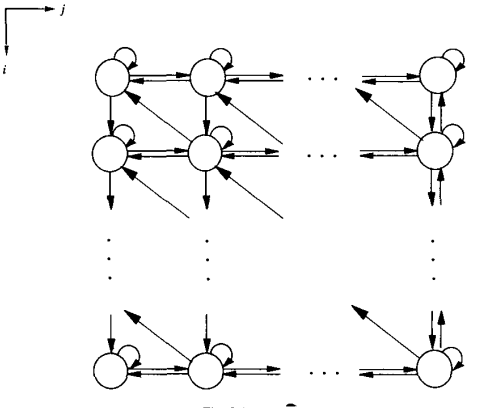

Fig. 2(c). The space-optimal regular array for transitive closure and algebraic path problem.

$[i-k+1 \quad j-k+1 \quad k]^T$) *to construct a DG similar to that for transitive closure, and the same result can be obtained.* □

B. Validity

In this section, we want to prove that Procedure 3.1 can derive a locally connected, space-optimal regular array for any SURE with a linear schedule $\Pi(I) = ai + bj + ck$ and $a + b \le c$.

LEMMA 3.1. *The regular array derived by Procedure 3.1 is locally connected.*

PROOF. Because each PE corresponds to a module constructed by collecting the segments in the k-direction, the locally connected links can always be obtained. □

THEOREM 3.1. *Procedure 3.1 is compatible with its schedule* $\Pi(I) = ai + bj + ck$.

PROOF. A processor allocation procedure is said to be compatible with the given schedule iff no two different computations are executed on the same PE at the same time, and that is the central feature of an elementary module. Thus in this proof, first, two properties of segments traced by the gcd-partition in Procedure 3.1 are derived; one is that every segment is modulo-c and the other is that any two segments in a module are isomorphic. With these properties, it can be proved that each module is elementary.

[modulo-c]

- gcd$(a, c) = g$: According to Procedure 3.1, every ij-plane should be partitioned into several segments. The size of each segment $G^{\gamma}_{\alpha,\beta}$ is $\frac{c}{g} \times g$, where (with $h \equiv \frac{c}{g}$)

$$G^{\gamma}_{\alpha,\beta} = \begin{bmatrix} v & v+b & \cdots & v+(g-1)b \\ v+a & v+a+b & \cdots & v+a+(g-1)b \\ \vdots & \vdots & \ddots & \vdots \\ v+(h-1)a & v+(h-1)a+b & \cdots & v+(h-1)a+(g-1)b \end{bmatrix}.$$

If $c = 1$ then there is only one time-tag in every segment; it is modulo-1. For $c > 1$ and any two time-tags $v_1, v_2 \in G^{\gamma}_{\alpha,\beta}$, the difference between the time-tags is $v_2 - v_1 = ia + jb$, where $1 - h \le i \le h - 1$, $1 - g \le j \le g - 1$.

By contradiction, assume that $G^{\gamma}_{\alpha,\beta}$ is not a modulo-c segment, i.e., there exist two time-tags $v_1, v_2 \in G^{\gamma}_{\alpha,\beta}$, and $v_2 = v_1 + mc$ such that $v_2 = v_1 + ia + jb = v_1 + mc$.

$$\Rightarrow ia + jb = mc \tag{1}$$

Let $a = ga'$; then we have $iga' + jb = mgh$. $\Rightarrow jb = (mh - ia')g \equiv m'g$.

$$\Rightarrow m' = \frac{jb}{g} \tag{2}$$

Because m' must be an integer, only two cases are possible:

- If $m' = 0$ then $j = 0$. Equation (1) can then be reduced to $ia = mc = q \times \text{lcm}(a, c)$, where q is an integer and lcm(a, c) denotes the least common multiplier of a and c. From this equation, we have $|ia| \ge \text{lcm}(a, c)$. Because $1 - \frac{c}{g} \le i \le \frac{c}{g} - 1$ or $|ia| \le \frac{ca}{g} - a = \text{lcm}(a, c) - a < \text{lcm}(a, c)$, a contradiction occurs.
- If $m' \ne 0$ then $j \ne 0$. We know gcd$(a, b, c) = 1$, because the linear schedule $\Pi(I) = ai + bj + ck$ is a normalized one. If gcd$(a, c) = g = 1$, then by $1 - g \le j \le g - 1$, we have $j = 0$. This is a case which we have explored already. On the other hand, if gcd$(a, c) = g \ne 1$, then gcd$(b, g) = 1$. From $g > |j|$, the right-hand side of (2), $\frac{jb}{g}$, cannot be an integer. Thus a contradiction occurs, because the left-hand side of (2), m', is an integer.

In both cases there are contradictions. This implies that $G^{\gamma}_{\alpha,\beta}$ is a modulo-c segment.

- gcd$(b, c) = g$: The argument is similar to that for gcd$(a, c) = g$.

It has now been shown that every segment derived by Procedure 3.1 (gcd-partition) is modulo-c.

[isomorphic]. Let v_1 and v_2 be two time-tags which have the same (p, q) location about two different segments, say $G^{\gamma_1}_{\alpha,\beta}$ and $G^{\gamma_2}_{\alpha,\beta}$, respectively, in a module. Let the index vector for v_1 be $[i_1 \quad j_1 \quad k_1]^T$ and that for v_2 be $[i_1 \quad j_1 \quad k_2]^T$. Then $v_1 = ai_1 + bj_1 + ck_1$ and $v_2 = ai_1 + bj_1 + ck_2$.

$$\Rightarrow v_2 - v_1 = c(k_2 - k_1). \tag{3}$$

$$\Rightarrow \text{mod}(v_1, c) = \text{mod}(v_2, c). \tag{4}$$

Because the size of every segment obtained by Procedure 3.1 is c, we have mod$(v_1, |G^{\gamma_1}_{\alpha,\beta}|) = \text{mod}(v_2, |G^{\gamma_2}_{\alpha,\beta}|)$. This shows that every segment in a module derived by Procedure 3.1 is isomorphic to all others.

Since every segment is modulo-c and is isomorphic to all others in the module, it can now be proved that each module is elementary.

[elementary]. Let v_1 and v_2 ($v_1'$and v_2') be two time-tags with the same (p, q) location about two different segments, say $G^{\gamma_1}_{\alpha,\beta}$ and $G^{\gamma_2}_{\alpha,\beta}$, respectively, in a module. Let the index vector for v_1 be $[i_1 \quad j_1 \quad k_1]^T$ and that for v_2 be $[i_1 \quad j_1 \quad k_2]^T$. Then $v_1 = ai_1 + bj_1 + ck_1$ and $v_2 = ai_1 + bj_1 + ck_2$. Similarly, let the index vector for v_1' be $[i_1' \quad j_1' \quad k_1]^T$ and that for v_2' be $[i_1' \quad j_1' \quad k_2]^T$. Then $v_1' = ai_1' + bj_1' + ck_1$ and $v_2' = ai_1' + bj_1' + ck_2$.

- $<v_1, v_2>$: If two index vectors belong to different segments in a module but have the same (p, q) location with respect to their segments, then their time-tags should be different, because (3) is not equal to zero.
- $<v_2, v_2'>$: If two index vectors belong to the same segment, then their time-tags are not the same. Since $G^{\gamma_2}_{\alpha,\beta}$ is modulo-c, we have $(v_2, c) \ne \text{mod}(v_2', c)$, $\Rightarrow v_2 \ne v_2'$.
- $<v_1, v_2'>$: If two index vectors belong to different segments and have different (p, q) locations, then their time-tags are not the same. The reason is as follows: Because $G^{\gamma_2}_{\alpha,\beta}$ is modulo-c, we have mod$(v_2, c) \ne \text{mod}(v_2', c)$, and from (4), mod$(v_1, c) = \text{mod}(v_2, c)$, $\Rightarrow$ mod$(v_1, c) \ne$ mod(v_2', c), $\Rightarrow v_1 \ne v_2'$.

- No other case.

Because any two index vectors in a module have different time-tags, the module is elementary. Hence every module constructed by Procedure 3.1 is elementary. □

THEOREM 3.2. *The regular array derived by Procedure 3.1 is space-optimal.*

PROOF. To prove that Procedure 3.1 can derive a space-optimal regular array is equivalent to proving that the cluster G derived by Procedure 3.1 is optimal. Lemmas 2.1 and 2.2 tell us that a cluster G is optimal iff every module $G_{\alpha,\beta} \in G$ is elementary and there exists a time-tag $v \in G_{\alpha,\beta}$ for all $G_{\alpha,\beta} \in G$. The former has been shown in Theorem 3.1. Now we want to prove that there is at least one time-tag $v \in G_{\alpha,\beta}$ for all $G_{\alpha,\beta} \in G$ if Procedure 3.1 is applied.

Let v_2 be the largest time-tag on the $(k = 1)$-plane with a remainder r when divided by c, i.e.,

$$v_2 = m_2 c + r \in G^1_{\frac{N}{g}, \frac{gN}{c}},$$

where

$$g = \gcd(b, c) \,(\text{or } G^1_{\frac{gN}{c}, \frac{N}{g}}, \text{ where } g = \gcd(a, c)) \ .$$

Because every segment derived by Procedure 3.1 is modulo-c, one can find a time-tag $v_1 = m_1 c + r \in G^1_{\alpha,\beta}$, $1 \le \alpha < \frac{N}{g}$, $1 \le \beta < \frac{gN}{c}$. The difference between v_1 and v_2 is $v_2 - v_1 = (m_2 - m_1)c. \Rightarrow m_2 - m_1 = \frac{v_2 - v_1}{c} \equiv m'$. In addition, the difference between any two time-tags on the $(k = 1)$-plane is equal to or less than $(a + b)(N - 1)$, because the maximum and minimum time-tags on this plane are $aN + bN + c$ (the node $[N \ N \ 1]^T$) and $a + b + c$ (the node $[1 \ 1 \ 1]^T$), respectively. Thus

$$\frac{v_2 - v_1}{c} \le \frac{(a+b)(N-1)}{c}$$

From $a + b \le c$, we have

$$\frac{(a+b)(N-1)}{c} \le \frac{(a+b)(N-1)}{a+b} = N - 1. \tag{5}$$

Then $1 \le m' \le N - 1$. Therefore, if we have the time-tag

$$v_2 = m_2 c + r \in G^1_{\frac{N}{g}, \frac{gN}{c}}$$

in the module

$$G^1_{\frac{N}{g}, \frac{gN}{c}}$$

then there exists the same time-tag $v_2 \in G_{\alpha,\beta}$ on some ij-plane, because $v_1 = m_1 c + r \in G^1_{\alpha,\beta}, \Rightarrow (m_1 + 1)c + r \in G^2_{\alpha,\beta}, \Rightarrow (m_1 + 2)c + r \in G^3_{\alpha,\beta}, \cdots, \Rightarrow (m_1 + m')c + r = m_2 c + r = v_2 \in G^{m'+1}_{\alpha,\beta}$. For the extreme case, if $m' = N - 1$ then v_2 will appear on the $(k = N)$-plane. □

THEOREM 3.3. *Procedure 3.1 can always design a locally connected, space-optimal regular array for any SURE with a linear schedule* $\Pi(I) = ai + bj + ck$ *and* $a + b \le c$.

PROOF. The theorem follows directly from Lemma 3.1, Theorem 3.1, and Theorem 3.2. □

From Procedure 3.1, because the number of time-tags in every segment is c and every module contains N segments, the number of time-tags in each and every module is Nc. Meanwhile, because there are N^3 time-tags in the computation domain, the size of the cluster or the number of PEs used is $\frac{N^2}{c}$.

On the other hand, Theorem 3.2 manifests the fact that the regular array is space-optimal, i.e., the number of PEs used (modules) by Procedure 3.1 is equal to the minimum number of PEs required. Thus $\frac{N^2}{c}$ is just the lower bound of the number of PEs required so that no two different computations are executed on the same PE at the same time. Therefore we have the following theorem.

THEOREM 3.4. *The minimum number of PEs required is* $\frac{N^2}{c}$ *for any SURE with a linear schedule* $\Pi(I) = ai + bj + ck$ *and* $a + b \le c$.

Procedure 3.1 is a simple but useful method of processor allocation for deriving a space-optimal regular array. The array derived is locally connected and regular and provides simple control and a balanced load. However, Procedure 3.1 guarantees that the optimal space is obtained only when the linear schedule $\Pi(I) = ai + bj + ck$ follows the constraint of $a + b \le c$. The case where $a + b > c$ will be discussed in the next section.

VI. PROCESSOR ALLOCATION FOR $a + b > c$

Now let us discuss the more difficult case, $a + b > c$. In this case, (5) is not always true and a time-tag v may not always be found in every module derived by Procedure 3.1. Thus a space-optimal regular array cannot always be obtained, i.e., the difference between PE_{used} and PE_{min} is a function of N (problem size parameter). Yet, Procedure 3.1 can still be used to decrease PE_{used} from N^2 to $\frac{N^2}{c}$ when $a + b > c$. For example, given a linear schedule $\Pi(I) = 2i + 3j + 4k$ and $N = 20$, the 2×2 segment can be obtained by the gcd-partition of Procedure 3.1; then by $Trace_1$, N segments can be grouped in the k-direction to construct a module (PE). Thus we have $PE_{used} = \frac{N^2}{c} = 100$, which is greater than the size of a maximum concurrent set, 96, for the given linear schedule. Nevertheless, when $a + b > c$, a space-optimal regular array can be designed for the special cases where $a = b$ and $b = c$ by adopting different processor allocation procedures (*traces*). The problem of matrix multiplication is a good example for both cases, because the optimal linear schedule for that problem is $\Pi(I) = i + j + k$ [21].

A. Procedure for $b = c$

PROCEDURE 4.1. *Given a 3D SURE with a linear schedule* $\Pi(I) = ai + bj + ck$, *where* $a + b > c$ *and* $b = c$, *a space-optimal regular array can always be obtained by gcd-partitioning every ij-plane of the DG of the SURE into several* $c \times 1$ *segments. Each module (PE) is constructed by*

using Trace$_2$ as follows to collect the set of segments of the module (Fig. 3):

$$\begin{aligned}\text{Trace}_2(G^{\gamma}_{\alpha,\beta}, free(\Theta)) = < \; & G^{\gamma}_{\alpha,\beta}, G^{\gamma}_{\alpha,\beta+1}, \cdots, G^{\gamma}_{\alpha,\beta+a-1}, \\ & G^{\gamma}_{\alpha+1,\beta}, G^{\gamma}_{\alpha+1,\beta+1}, \cdots, G^{\gamma}_{\alpha+1,\beta+a-1}, \\ & \vdots \\ & G^{\gamma}_{n_1,\beta}, G^{\gamma}_{n_1,\beta+1}, \cdots, G^{\gamma}_{n_1,\beta+a-1}, \\ & G^{\gamma}_{n_1,\beta+a}, G^{\gamma}_{n_1,\beta+a+1}, \cdots, G^{\gamma}_{n_1,n_2}, \\ & G^{\gamma+1}_{n_1,n_2}, G^{\gamma+2}_{n_1,n_2}, \cdots, G^{N}_{n_1,n_2} >,\end{aligned}$$

where n_1 is the maximum row index of segments on the γ-plane in the current free(Θ) and n_2 is the maximum column index of segments on the γ-plane in the current free(Θ) when $\alpha = n_1$. The processor allocation procedure is greedy, such that n_1, n_2 can be determined by this greedy procedure:

Step 1: Let $m = 1$.

Step 2: Find a free segment which is minimal, $G^{\gamma}_{\alpha,\beta} = min\{free(\Theta)\}$.

Step 3: Construct the module $G_m = Trace_2(G^{\gamma}_{\alpha,\beta}, free(\Theta))$.

Step 4: If free(Θ) $\neq \varnothing$ then $m = m + 1$ goto Step 2 else stop. □

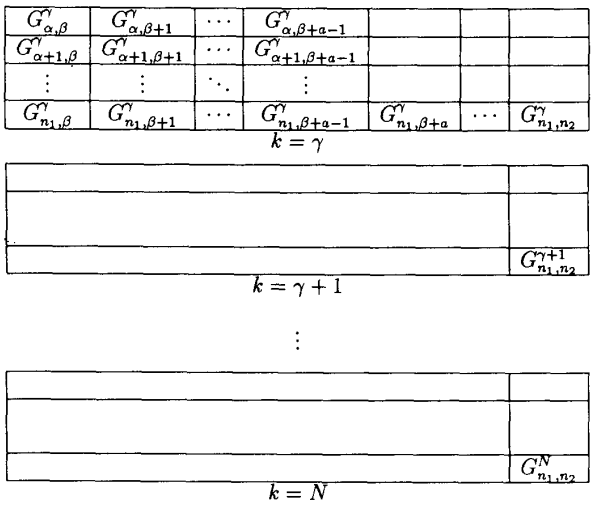

Fig. 3. Trace$_2$.

THEOREM 4.1. *The processor allocation Procedure 4.1 is compatible with its schedule $\Pi(I) = ai + bj + ck$, $a + b > c$, and $b = c$.*

PROOF.

[modulo-c]. Segments derived by gcd-partitioning must be modulo-c.

[isomorphic]. Let $v_1 = ai_1 + bj_1 + ck_1$ and $v_2 = ai_2 + bj_2 + ck_2$ be two time-tags whose index vectors are on the same (p, q) locations about their segments $G^{\gamma_1}_{\alpha_1,\beta_1}$, $G^{\gamma_2}_{\alpha_2,\beta_2}$, respectively. From the fact that every segment is a $c \times 1$ matrix, we have $v_2 = a(i_1 + i'c) + bj_2 + ck_2$. From $b = c$, we have $v_2 - v_1 = ai'c + c(j_2 - j_1) + c(k_2 - k_1) = (ai' + j_2 - j_1 + k_2 - k_1)c$. Thus, $\text{mod}(v_1, |G^{\gamma_1}_{\alpha_1,\beta_1}|) = \text{mod}(v_2, |G^{\gamma_2}_{\alpha_2,\beta_2}|)$, where $|G^{\gamma_1}_{\alpha_1,\beta_1}| = |G^{\gamma_2}_{\alpha_2,\beta_2}| = c$.

Now we can say that every segment derived by Procedure 4.1 is isomorphic to all others.

[elementary]. Because every segment is isomorphic to all others, if two time-tags v_1 and v_2 are not on the same (p, q) location, then $v_1 \neq v_2$. We now want to prove that no two time-tags in a module with the same (p, q) location are equal. From the module constructed by Trace$_2$, as shown in Fig. 3, let $v = mc + r \in G^{\gamma}_{\alpha,\beta}$.

$$\text{Let } v_1 = m_1 c + r \in G^{\gamma}_{\alpha,\beta+1}, \text{ then } v_1 = (m+1)c + r.$$

$$\text{Let } v_2 = m_2 c + r \in G^{\gamma}_{\alpha,\beta+a-1} \text{ then } v_2 = (m+a-1)c + r.$$

$$\text{Let } v_3 = m_3 c + r \in G^{\gamma}_{\alpha+1,\beta} \text{ then } v_3 = (m+a)c + r.$$

$$\text{Let } v_4 = m_4 c + r \in G^{\gamma}_{n_1,\beta} \text{ then } v_4 = \left(m + m'_1\, a\right)c + r,$$

where $m'_1 > 0$.

All other formulae can be derived similarly. The quotients of dividing the time-tags v'_i s by c with remainder r are shown in Fig. 4, from which we can see that all time-tags with the same (p, q) location are not equal, because they have different quotients. Hence every module derived by Trace$_2$ of Procedure 4.1 is elementary, i.e., the processor allocation Procedure 4.1 is compatible with its schedule $\Pi(I) = ai + bj + ck$. □

THEOREM 4.2. *The minimum number of PEs required for the schedule $\Pi(I) = ai + bj + ck$, $a + b > c$, and $b = c$ is*

$$\text{PE}_{min} = \frac{N^2}{c} - \left\lfloor \frac{N}{2c} \right\rfloor \left\lceil \frac{N}{2c} \right\rceil a.$$

PROOF. Fig. 5 is the $(k = 1)$-plane of a DG. The slanted lines represent a time hyperplane with the normal vector $[a \;\; c \;\; c]^T$ projected on the $(k = 1)$-plane. They pass through the nodes (represented by black nodes in Fig. 5) belonging to the modulo set $\phi(r)$. From left to right, we have the following observations:

- there are a lines each of which passes through only one node;
- there are a lines each of which passes through two nodes;

  ⋮
- there are a lines each of which passes through $\frac{N}{c} - 1$ nodes;
- there are $N - a\left(\frac{N}{c} - 1\right)$ lines each of which passes through $\frac{N}{c}$ nodes;
- there are a lines each of which passes through $\frac{N}{c} - 1$ nodes;

  ⋮
- there are a lines each of which passes through only one node.

Because all the nodes on a time hyperplane are executed at the same time, they are assigned the same time-tag. These nodes with the same time-tag must be allocated to different PEs. Therefore, in order to find PE$_{min}$, we need to find the hyperplane, say $\mathcal{H}$, which contains the maximum number of nodes. We project the nodes of $\mathcal{H}$ in the k-direction onto the $(k = 1)$-plane. These nodes should be projected onto the

m	$m+1$	$\cdots$	$m+a-1$			
$m+a$	$m+a+1$	$\cdots$	$m+2a-1$			
$\vdots$	$\vdots$	$\ddots$	$\vdots$			
$m+m_1'a$	$m+m_1'a+1$	$\cdots$	$m+(m_1'+1)a-1$	$m+(m_1'+1)a$	$\cdots$	$m+(m_1'+1)a+m_2'$

$k=\gamma$

	$m+(m_1'+1)a+m_2'+1$

$k=\gamma+1$

$\vdots$

	$m+(m_1'+1)a+m_2'+m_3'$

$k=N$

Fig. 4. The quotients obtained by dividing the time-tags in all segments of a module by c for the case of $b=c$.

slanted lines of Fig. 5. Because the total number of slanted lines is greater than N and there are only N k-planes, it is

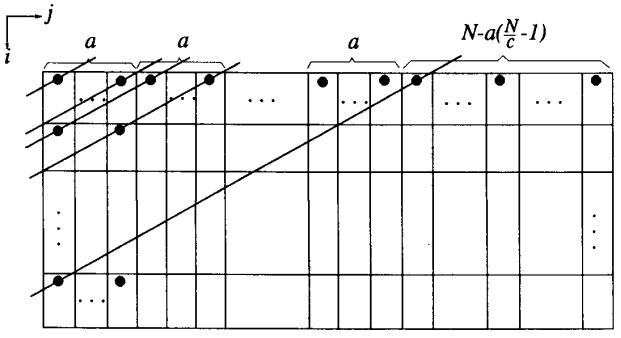

Fig. 5. The $(k=1)$-plane of a DG with schedule $ai+bj+ck$, $a+b>c$, and $b=c$.

necessary to select N slanted lines for which the total number of black nodes passed through is maximal. From the above observations, the selection is as follows: If $\frac{N}{c}$ is an odd number then there are

- $(N-a(\frac{N}{c}-1))$ lines each of which has $\frac{N}{c}$ nodes,
- $2a$ lines each of which has $\frac{N}{c}-1$ nodes,
- $2a$ lines each of which has $\frac{N}{c}-2$ nodes,

:

- $2a$ lines each of which has $\frac{N}{c}-\lfloor\frac{N}{2c}\rfloor$ nodes.

Thus the total number of nodes of $\mathcal{H}$ is

$$\left(N-a\left(\tfrac{N}{c}-1\right)\right)\tfrac{N}{c}+2a\left(\left(\tfrac{N}{c}-1\right)+\left(\tfrac{N}{c}-2\right)+\cdots+\left(\tfrac{N}{c}-\left\lfloor\tfrac{N}{2c}\right\rfloor\right)\right)$$
$$=\tfrac{N^2}{c}-a\left\lfloor\tfrac{N}{2c}\right\rfloor\left\lceil\tfrac{N}{2c}\right\rceil.$$

Similarly, if $\frac{N}{c}$ is an even number then there are

- $(N-a(\frac{N}{c}-1))$ lines each of which has $\frac{N}{c}$ nodes,
- $2a$ lines each of which has $\frac{N}{c}-1$ nodes,
- $2a$ lines each of which has $\frac{N}{c}-2$ nodes,

:

- $2a$ lines each of which has $\frac{N}{c}-\frac{N}{2c}+1$ nodes,
- a lines each of which has $\frac{N}{c}-\frac{N}{2c}$ nodes.

Thus the total number of nodes of $\mathcal{H}$ is

$$\left(N-a\left(\tfrac{N}{c}-1\right)\right)\tfrac{N}{c}+2a\left(\left(\tfrac{N}{c}-1\right)+\left(\tfrac{N}{c}-2\right)+\cdots+\left(\tfrac{N}{c}-\tfrac{N}{2c}+1\right)\right)$$
$$+a\left(\tfrac{N}{c}-\tfrac{N}{2c}\right)=\tfrac{N^2}{c}-a\left\lfloor\tfrac{N}{2c}\right\rfloor\left\lceil\tfrac{N}{2c}\right\rceil.$$

□

THEOREM 4.3. *Procedure 4.1 can always design a locally connected, space-optimal regular array for any SURE with a linear schedule* $\Pi(I)=ai+bj+ck$, $a+b>c$, *and* $b=c$.

PROOF: From Procedure 4.1, we know that nodes on every plane, from the $(k=1)$-plane to the $(k=N-(a\frac{N}{c}-1))$-plane, can be allocated to $\frac{N}{c}$ PEs. However, nodes on the $(k=N-a(\frac{N}{c}-1)+1)$-plane can be allocated only to $(\frac{N}{c}-1)$ PEs, because this plane has only $a(\frac{N}{c}-1)$ columns of index vectors which are free; the others have already been allocated. Similarly, the nodes on the next $2a-1$ k-planes can be allocated to $(\frac{N}{c}-1)$ PEs, and then there are $2a$ k-planes which can be allocated to $(\frac{N}{c}-2)$ PEs, and so on, until all N k-planes are allocated. If $\frac{N}{c}$ is an odd number then the number of PEs used is:

$$\mathrm{PE}_{used}$$
$$=\left(N-a\left(\tfrac{N}{c}-1\right)\right)\tfrac{N}{c}+2a\left(\left(\tfrac{N}{c}-1\right)+\left(\tfrac{N}{c}-2\right)+\cdots+\left(\tfrac{N}{c}-\left\lfloor\tfrac{N}{2c}\right\rfloor\right)\right)$$
$$=\tfrac{N^2}{c}-a\left\lfloor\tfrac{N}{2c}\right\rfloor\left\lceil\tfrac{N}{2c}\right\rceil.$$

Similarly, if $\frac{N}{c}$ is an even number, then the number of PEs used is:

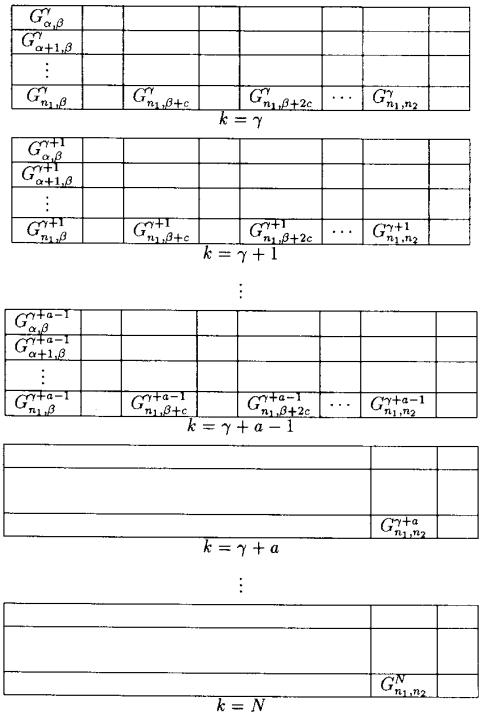

Fig. 6. Trace_3.

$$\begin{aligned}&\text{PE}_{used}\\&=\left(N-a\left(\tfrac{N}{c}-1\right)\right)\tfrac{N}{c}+2a\left(\left(\tfrac{N}{c}-1\right)+\left(\tfrac{N}{c}-2\right)+\cdots+\left(\tfrac{N}{c}-\tfrac{N}{2c}+1\right)\right)\\&\quad+a\left(\tfrac{N}{c}-\tfrac{N}{2c}\right)=\tfrac{N^2}{c}-a\left\lfloor\tfrac{N}{2c}\right\rfloor\left\lceil\tfrac{N}{2c}\right\rceil.\end{aligned}$$

□

B. Procedure for $a = b$

PROCEDURE 4.2. *Given a 3D SURE with a linear schedule $\Pi(I) = ai + bj + ck$, where $a + b > c$ and $a = b$, a space-optimal regular array can always be obtained by partitioning every ij-plane of the DG of the SURE into several $c \times 1$ segments. Each module is constructed by using Trace_3 as follows to collect the set of segments of the module (Fig. 6):*

$$\begin{aligned}&\text{Trace}_3\left(G^{\gamma}_{\alpha,\beta},\text{free}(\Theta)\right)\\&=<G^{\gamma}_{\alpha,\beta},G^{\gamma}_{\alpha+1,\beta},\cdots,G^{\gamma}_{n_1,\beta},G^{\gamma}_{n_1,\beta+c},G^{\gamma}_{n_1,\beta+2c},\cdots,G^{\gamma}_{n_1,n_2},\\&\quad G^{\gamma+1}_{\alpha,\beta},G^{\gamma+1}_{\alpha+1,\beta},\cdots,G^{\gamma+1}_{n_1,\beta},G^{\gamma+1}_{n_1,\beta+c},G^{\gamma+1}_{n_1,\beta+2c},\cdots,G^{\gamma+1}_{n_1,n_2},\\&\quad\vdots\\&\quad G^{\gamma+a-1}_{\alpha,\beta},G^{\gamma+a-1}_{\alpha+1,\beta},\cdots,G^{\gamma+a-1}_{n_1,\beta},G^{\gamma+a-1}_{n_1,\beta+c},G^{\gamma+a-1}_{n_1,\beta+2c},\cdots,G^{\gamma+a-1}_{n_1,n_2},\\&\quad G^{\gamma+a}_{n_1,n_2},G^{\gamma+a+1}_{n_1,n_2},\cdots,G^{N}_{n_1,n_2}>,\end{aligned}$$

where n_1 is the maximum row index of segments on the γ-plane in free(Θ) and n_2 is the maximum column index of segments on the γ-plane in free(Θ) when $\alpha = n_1$ and is equal to $\beta + mc$. The processor allocation procedure is greedy such that n_1, n_2 can be determined by this greedy procedure:

Step 1: Let $m = 1$.

Step 2: Find a free segment which is minimal, $G^{\gamma}_{\alpha,\beta} = \min\{\text{free}(\Theta)\}$.

Step 3: Construct the module $G_m = \text{Trace}_3(G^{\gamma}_{\alpha,\beta}, \text{free}(\Theta))$.

Step 4: If free(Θ) $\neq \varnothing$ then $m = m + 1$ goto Step 2 else stop.

□

THEOREM 4.4. *The processor allocation Procedure 4.2 is compatible with its schedule $\Pi(I) = ai + bj + ck$, $a + b > c$, and $a = b$.*

PROOF: The proof is similar to that for Theorem 4.2. □

THEOREM 4.5: *The minimum number of PEs required for the schedule $\Pi(I) = ai + bj + ck$, $a + b > c$, and $a = b$ is*

$$\text{PE}_{\min}=\tfrac{N^2}{c}-\left\lceil\tfrac{l}{2}\right\rceil\left\lceil\tfrac{l+1}{2}\right\rceil c,$$

where

$$l=\begin{cases}\left\{\tfrac{2N-c}{c}-\left\lceil\tfrac{N}{a}\right\rceil\right., & \text{if } \tfrac{2N-c}{c}>\left\lceil\tfrac{N}{a}\right\rceil\\ 0, & \text{otherwise}\end{cases}.$$

PROOF. Fig. 7 is the (k = 1)-plane of a DG. The slanted lines represent a time hyperplane with the normal vector $[a\ a\ c]^T$ projected onto the (k = 1)-plane. These slanted lines pass through black nodes which belong to modulo set $\phi(r)$. Assume that $\mathcal{H}$ is the time hyperplane which contains the maximum number of nodes. These nodes when projected onto the (k = 1)-plane should be on the positions of black nodes. The number of slanted lines which cover these projected nodes is $\lceil\frac{N}{a}\rceil$ because the time-tags' difference between two adjacent slanted lines is ac and the time tags' difference between two adjacent k-planes is c. However, it can be observed from Fig. 7 that the total number of slanted lines is $\frac{2N-c}{c}$ Therefore, the total number of nodes on $\mathcal{H}$ can be calculated by selecting $\lceil\frac{N}{a}\rceil$ slanted lines for which the total number of black nodes passed through is maximal. Let

$$l=\frac{2N-c}{c}-\left\lceil\tfrac{N}{a}\right\rceil.$$

If l is an even number, then we have

$$\text{PE}_{\min}=\tfrac{N^2}{c}-c\left(2\left(1+2+\cdots+\tfrac{l}{2}\right)\right)=\tfrac{N^2}{c}-c\left\lceil\tfrac{l}{2}\right\rceil\left\lceil\tfrac{l+1}{2}\right\rceil;$$

otherwise, we have

$$\begin{aligned}\text{PE}_{\min}&=\tfrac{N^2}{c}-c\left(2\left(1+2+\cdots+\tfrac{l-1}{2}\right)+\tfrac{l+1}{2}\right)\\&=\tfrac{N^2}{c}-c\left\lceil\tfrac{l}{2}\right\rceil\left\lceil\tfrac{l+1}{2}\right\rceil.\end{aligned}$$

u

THEOREM 4.6. *Procedure 4.2 can always design a locally connected, space-optimal regular array for any SURE with a linear schedule $\Pi(I) = ai + bj + ck$, $a + b > c$ and $a = b$.*

PROOF. By Procedure 4.2, the DG of the SURE can be divided into c regions, e.g., the shaded segments in Fig. 7 and those extended in the k-direction form one of these regions. Every

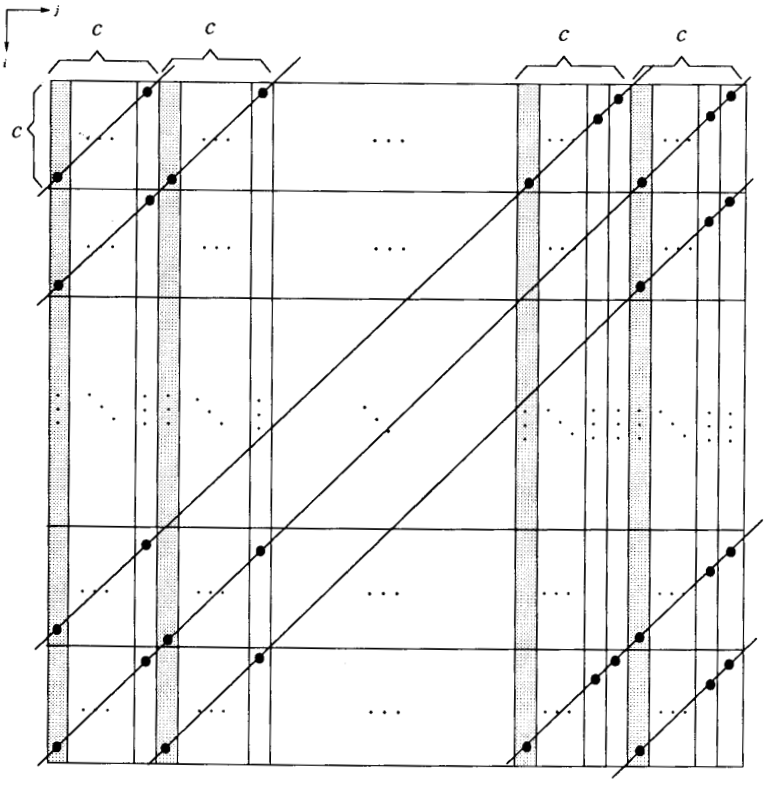

Fig. 7. The ($k = 1$)-plane of a DG with schedule $ai + bj + ck$, $a + b > c$, and $a = b$.

PROOF. By Procedure 4.2, the DG of the SURE can be divided into c regions, e.g., the shaded segments in Fig. 7 and those extended in the k-direction form one of these regions. Every region is formed entirely of isomorphic segments, can be allocated independently, and will have the same number of PEs. Let us consider any one region. If $\frac{N}{a}$ is an even number, $\frac{N}{c}$ is the number of PEs for all the nodes in the region between the ($k = 1$)-plane and the ($k = a$)-plane, $2(\frac{N}{c}-1)$ is the number of PEs for all the nodes in the region between the ($k = a + 1$)-plane to those of the ($k = 3a$)-plane, and so on. Thus the number of PEs used by each region is

$$\frac{N}{c}+2\left(\frac{N}{c}-1\right)+\cdots+2\left(\frac{N}{c}-\left(\frac{\left\lceil\frac{N}{a}\right\rceil}{2}-1\right)\right)$$
$$+\left(\frac{N}{c}-\frac{\left\lceil\frac{N}{a}\right\rceil}{2}\right)=\frac{N^2}{c^2}-\left\lceil\frac{l}{2}\right\rceil\left\lceil\frac{l+1}{2}\right\rceil,$$

where

$$l=\frac{2N-c}{c}-\left\lceil\frac{N}{a}\right\rceil$$

On the other hand, if $\lfloor\frac{N}{a}\rfloor$ is an odd number, then the number of PEs used by each region is

$$\frac{N}{c}+2\left(\frac{N}{c}-1\right)+\cdots+2\left(\frac{N}{c}-\left(\frac{\left\lceil\frac{N}{a}\right\rceil}{2}-1\right)\right)$$
$$=\frac{N^2}{c^2}-\left\lceil\frac{l}{2}\right\rceil\left\lceil\frac{l+1}{2}\right\rceil.$$

Because there are c regions in the DG, the number of PEs used by Trace$_3$ is $\mathrm{PE}_{used}=\frac{N^2}{c}-c\left\lceil\frac{l}{2}\right\rceil\left\lceil\frac{l+1}{2}\right\rceil$. □

EXAMPLE 4.1. **[matrix multiplication]** The dependence matrix D for matrix multiplication [21] can be written as

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

and its DG is shown in Fig. 8(a) ($N = 6$). Its corresponding optimal linear schedule is $\Pi(I) = i + j + k$. Thus by applying Procedure 4.1 (or Procedure 4.2), the time-tags of all index vectors on every ij-plane can be gcd-partitioned into several 1×1 segments. The module is then constructed by Trace$_2$ of Procedure 4.1 (or Trace$_3$ of Procedure 4.2), as shown in Fig. 8(b). By mapping each module onto one PE, a space-optimal regular array can be constructed, as shown in Fig. 8(c). If we adopt the linear space mapping, then $N^2 = 36$ PEs is necessary. But Procedure 4.1 (or Procedure 4.2) can reduce the PE_{used} to

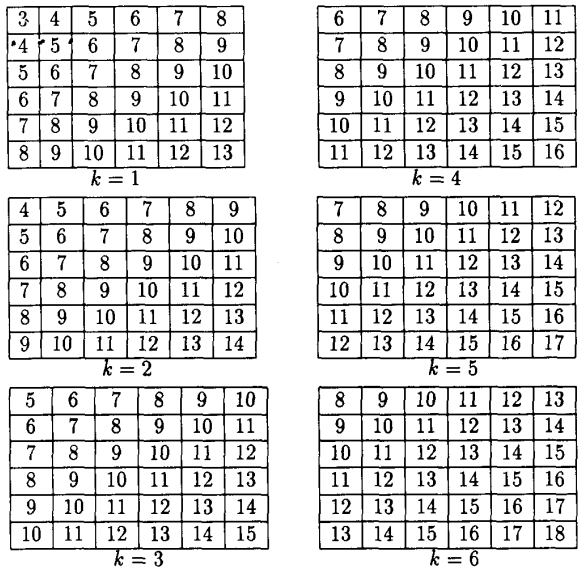

Fig. 8(a). The DG for matrix multiplication ($N = 6$).

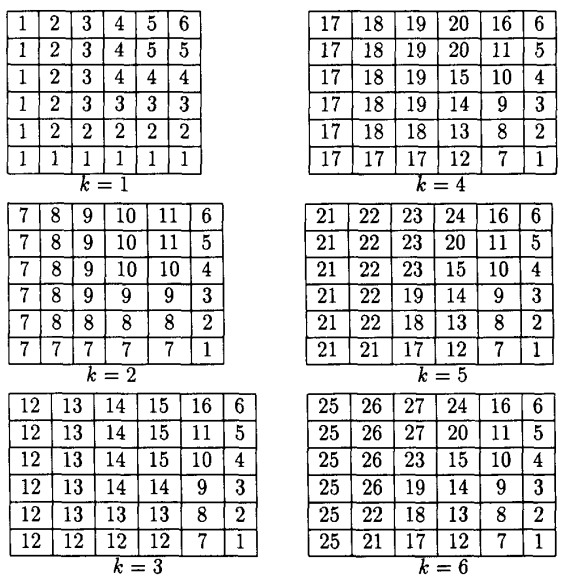

Fig. 8(b). The processor allocation for matrix multiplication by Procedure 4.1 (or Procedure 4.2).

advance a maximum concurrent set for a given linear schedule in order to design space-optimal regular arrays. The proposed processor allocation procedures ensure that no two nodes scheduled at the same time are mapped onto the same PE (Theorem 3.1) and that all PEs are active simultaneously at some one time instance (Theorem 3.2). Second, for a given linear schedule $\Pi(I) = ai + bj + ck$, $1 \le a \le b \le c$, for an SURE, two cases were studied: $a + b \le c$ and $a + b > c$. In the former case, a space-optimal design can always be obtained by Procedure 3.1; the number of PEs used is $\frac{N^2}{c}$. The resulting array has the advantages of local connection, load balance,

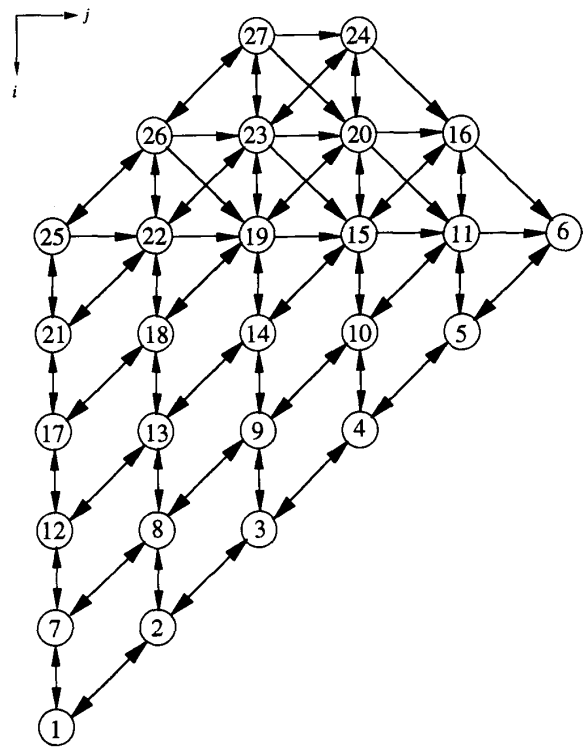

Fig. 8(c). The space-optimal regular array for matrix multiplication.

simple control, and space optimality. For the latter case, $\frac{N^2}{c}$ becomes the upper bound of PE_{min}. We also discussed two special cases of $a + b > c$, $a = b$ and $b = c$. By Procedures 4.1 ($b = c$) and 4.2 ($a = b$), space-optimal regular arrays can also be obtained for these cases. The closed form expressions for PE_{min} are also given for the cases of $b = c$ and $a = b$ in Theorems 4.2 and 4.5, respectively. Although only three dimensional algorithms with linear schedules are discussed here, the method proposed in this paper can easily be extended to higher dimensional algorithms. More research on the topic of space-optimal design should be pursued; one important project would be to solve the problem of space-optimality for linear schedule $\Pi(I) = ai + bj + ck$ with $a + b > c$ and its closed form expressions for PE_{min}.

ACKNOWLEDGMENTS

We would like to thank the referees for their constructive and helpful comments. This research was supported by the National Science Council of the Republic of China under contract NSC-83-0408-E-009-044.

REFERENCES

[1] H.T. Kung and C.E. Leiserson, "Systolic arrays for VLSI," *Proc. 1978 Soc. for Industrial and Applied Math.*, pp. 256–282, 1979.

[2] J.A.B. Fortes and B.W. Wah, "Systolic arrays} From concept to implementation," *Computer*, pp. 12–17, July 1987.

[3] J.C. Tsay and P.Y. Chang, "Some new designs of 2D array for matrix multiplication and transitive closure," *IEEE Trans. Parallel and Dis-*

ACKNOWLEDGMENTS

We would like to thank the referees for their constructive and helpful comments. This research was supported by the National Science Council of the Republic of China under contract NSC-83-0408-E-009-044.

REFERENCES

[1] H.T. Kung and C.E. Leiserson, "Systolic arrays for VLSI," *Proc. 1978 Soc. for Industrial and Applied Math.*, pp. 256–282, 1979.

[2] J.A.B. Fortes and B.W. Wah, "Systolic arrays} From concept to implementation," *Computer*, pp. 12–17, July 1987.

[3] J.C. Tsay and P.Y. Chang, "Some new designs of 2D array for matrix multiplication and transitive closure," *IEEE Trans. Parallel and Distributed Systems*. Submitted for publication.

[4] P.Y. Chang and J.C. Tsay, "A family of efficient regular arrays for algebraic path problem," *IEEE Trans. Computers*, vol. 43, no. 7, pp. 769-777, July 1994.

[5] J.C. Tsay and P.Y. Chang, "Design of efficient regular arrays for matrix multiplication by two step regularization," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 2, pp. 215-222, Feb. 1995.

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

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

[8] Y.W. Wong and J.M. Delosme, "Broadcast removal in systolic algorithms," *Proc. Int'l Conf. Systolic Arrays*, pp. 403–412, 1988.

[9] S.Y. Kung, S.C. Lo, and P.S. Lewis, "Optimal systolic design for the transitive closure and the shortest path problems," *IEEE Trans. Computers*, vol. 36, pp. 603–614, May 1987.

[10] C. Choffrut and K. Culik II, "Folding of the plane and the design of systolic arrays," *Information Processing Letters*, vol. 17, pp. 149–153, 1983.

[11] J.M. Delosme and I.C.F. Ipsen, "Efficient systolic arrays for the solution of toeplitz systems: An illustration of a methodology for the construction of systolic architectures in VLSI," *Proc. Int'l Workshop on Systolic Arrays*, pp. 37–45, 1986.

[12] J.H. Moreno and T. Lang, "Graph-based partitioning of matrix algorithms for systolic arrays: application to transitive closure," *Proc. Int'l Conf. Parallel Processing*, pp. 28–31, 1988.

[13] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," *Proc. Int'l Symp. Computer Architecture*, pp. 208–214, 1984.

[14] S.K. Rao, "Regular iterative algorithms and their implementations on processor arrays," PhD thesis, Stanford Univ., 1985.

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

[16] W.L. Miranker and A. Winkler, "Spacetime representations of computational structures," *Computing*, vol. 32, pp. 92–114, 1984.

[17] P.Z. Lee and Z.M. Kedem, "Synthesizing linear array algorithms from nested for loop algorithms," *IEEE Trans. Computers*, vol. 37, pp. 1,578–1,598, Dec. 1988.

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

[19] V. Van Dongen, "Quasi-regular arrays: Definition and design methodology," *Proc. Int'l Conf. on Systolic Arrays*, pp. 126–135, 1989.

[20] W. Shang and J.A.B. Fortes, "Time optimal linear schedules for algorithms with uniform dependencies," *IEEE Trans. Computers*, vol. 40, pp. 723–742, June 1991.

[21] P. Cappello, "A processor-time-minimal systolic array for cubical mesh algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 4–13, Jan. 1992.

[22] C.J. Scheiman and R.P. Cappello, "A processor-time-minimal systolic array for transitive closure," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 257–269, May 1992.

[23] A. Benaini and Y. Robert, "Spacetime-minimal systolic arrays for gaussian elimination and the algebraic path problem," *Parallel Computing*, vol. 15, pp. 211–225, 1990.

[24] P. Clauss, C. Mongenet, and G.R. Perrin, "Synthesis of size-optimal toroidal arrays for the algebraic path problem: A new contribution," *Parallel Computing*, vol. 18, pp. 185–194, 1992.

[25] P. Clauss, C. Mongenet, and G. Perrin, "Calculus of space-optimal mappings of systolic algorithms on processor arrays," *J. VLSI Signal Processing*, vol. 4, pp. 27–36, 1992.

[26] J. Bu and E.F. Deprettere, "Processor clustering for the design of optimal fixed-size systolic arrays," *Proc. Int'l Conf. on Application Specific Array Processors*, pp. 402–413, Sept. 1991.

[27] A. Darte, T. Risset, and Y. Robert, "Synthesizing systolic arrays: some recent developments," *Proc. Int'l Conf. on Application Specific Array Processors*, pp. 372–386, Sept. 1991.

[28] Y. Wong and J. M. Delosme, "Space-optimal linear processor allocation for systolic arrays synthesis," *Proc. Sixth Int'l Parallel Processing Symp.*, pp. 275–282, Mar. 1992.

[29] Y.C. Hou and J.C. Tsay, "Equivalent transformations on systolic design represented by generating functions," *J. Information Science and Eng.*, vol. 5, pp. 229–250, 1989.

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

[31] P.Y. Chang and J.C. Tsay, "Timespace mapping for regular arrays," *Parallel Algorithms and Applications*. Submitted for publication.

[32] E.M. Reingold, J. Nieverglt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, Englewood Cliffs, N.J., 1977.

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

Jong-Chuang Tsay received the MS and PhD degrees in computer science from the National Chiao-Tung University in the Republic of China in 1968 and 1975, respectively. He has been on the faculty of the Department of Computer Engineering at the National Chiao-Tung University since 1968 and is currently a professor in the Department of Computer Science and Information Engineering. His research interests include systolic arrays, parallel computation, and computer-aided typesetting.

Pen-Yang Chang received the MS degree in computer science from the National Chiao-Tung University in the Republic of China in 1986. He has been an assistant researcher in the Telecommuncation Laboratories of the Ministry of Communication in the R.O.C. since 1987 and is a PhD candidate at the Institute of Computer Science and Information Engineering at the National Chiao-Tung University. His research interests include systolic arrays and multimedia information systems.