T-C/36/5/13944

Optimal Systolic Design for the Transitive Closure and the Shortest Path Problems

S.-Y. Kung
S.-C. Lo
P. S. Lewis

Reprinted from: IEEE TRANSACTIONS ON COMPUTERS, Vol. C-36, No. 5, MAY 1987.

Optimal Systolic Design for the Transitive Closure and the Shortest Path Problems

SUN-YUAN KUNG, SENIOR MEMBER, IEEE, SHENG-CHUN LO, AND PAUL S. LEWIS

Abstract—**Due to VLSI technological progress, algorithm-oriented array architectures, such as systolic arrays, appear to be very effective, feasible, and economic. This paper discusses how to design systolic arrays for the transitive closure and the shortest path problems. We shall focus on the Warshall algorithm for the transitive closure problem and the Floyd algorithm for the shortest path problem. These two algorithms share exactly the same structural formulation; therefore, they lead to the same systolic array design. In this paper, we first present a general method for mapping algorithms to systolic arrays. Using this methodology, two new systolic designs for the Warshall–Floyd algorithm will be derived. The first one is a *spiral* array, which is easy to derive and can be further simplified to a hexagonal array. The other is an *orthogonal* systolic array which is optimal in terms of pipelining rate, block pipelining rate, and the number of input/output connections.**

Index Terms—**Algorithm mappings, optimal algorithms, shortest path problem, systolic arrays, transitive closure problems, VLSI algorithms, VLSI architectures.**

I. Introduction

DUE to VLSI technology, hardware implementation of algorithmicly specialized parallel processors has become feasible and desirable. Implementations of regular algorithms can lead to simple and modular array processors. Recently, *systolic arrays* [10], [9] have received much attention. This is due to their simple control and timing, local interconnection between processors, and modular designs.

There have been many efforts to formalize the synthesis of systolic arrays from algorithm descriptions. For a recent review of these methods, the readers are referred to [5], [28]. In most of these methods the algorithm classes treated are usually quite restricted. More precisely, only very "regular" algorithms in the form of systems of uniform recursive equations can be successfully treated [8], [24], [2], [23], [27]. However, in practical applications there are certain classes of algorithms which are "slightly" irregular, yet can be mapped to regular arrays.

One such algorithm is the Warshall algorithm for the transitive closure problem or, equivalently, the Floyd algorithm for the shortest path problem. These two algorithms share exactly the same formulation; the only difference is that the algebraic operations in the two algorithms are different. The relation between these two algorithms is further explained in Section III. These two problems are commonly encountered in graph theory, dynamic programming, and combinatorial optimization. If the graph has N nodes, the sequential Warshall algorithm takes $O(N^3)$ time steps. To speed up the computation time to $O(N)$, a systolic array of $O(N^2)$ processing elements (PE's) can be used.

In this paper systolic arrays for solving the transitive closure/shortest path problems are investigated. Guibas *et al.* [7] first proposed a mesh-connected systolic array for the transitive closure problem. In their configuration three passes of the data matrix have to be fed into the array, which requires global wrap-around wires. Since then, several other systolic designs for the transitive closure problems have been proposed [15], [21], [31], [30], [14], [20]. This paper demonstrates a common framework for these various designs and utilizes this framework to derive two architectures. The first is a *spiral* array, which can be simplified to a hexagonal array, and the second is a new orthogonal array with superior performance.

To derive these architectures, the Warshall–Floyd algorithm is expressed in a localized, indexed, single assignment format. A *dependence graph* (DG) is then embedded in this index space, representing the data dependencies between the operations of the algorithm. These data dependencies are the constraints that have to be met in mapping the operations of the Warshall–Floyd algorithm onto a processor array. Ordinarily, systolic arrays will be derived by projections of the DG, which map multiple computations along the line of projection to a single PE. However, due to the irregularities in the Warshall–Floyd DG, certain *reindexings* of the nodes in the DG are necessary.

This paper is organized as follows: Section II outlines the concepts involved in mapping algorithms onto systolic arrays. This mapping is performed in three stages: 1) formulate the algorithm as a single assignment algorithm and construct its corresponding DG; 2) derive an abstract signal flow graph (SFG) implementation by projecting the DG onto a processor array and specifying a linear schedule for the computation; 3) retime the SFG to obtain a systolic array. Sections III–V describe how to map the Warshall–Floyd algorithm onto

Manuscript received May 31, 1985; revised December 29, 1986. This work was supported in part by the National Science Foundation under Grant ECS-82-13358, by the Semiconductor Research Corporation under USC SRC program, and by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and was administered through the Office of Naval Research under Contract N00014-85-K-0469 and N00014-85-K-0599.

S.-Y. Kung and S.-C. Lo are with the Signal and Image Processing Institute, Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089.

P. S. Lewis is with the Signal and Image Processing Institute, Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089 on leave from the Electronics Division, Los Alamos National Laboratory, Los Alamos, NM 87545.

IEEE Log Number 8713944.

0018-9340/87/0500-0603$01.00 © 1987 IEEE

systolic arrays by the above three-stage mapping. In Section III, the transitive closure and the shortest path problems are defined. The Warshall–Floyd algorithm is introduced and the DG for the Warshall–Floyd algorithm is derived. In Section IV, reindexing of the DG is used to derive a spiral SFG array. Then the global wires in the spiral array are removed by further modification of the DG. The result is a locally connected hexagonal systolic array. In Section V, a new and optimal orthogonal systolic array is derived by another projection of the revised DG. A comparison of this array with all existing systolic arrays for the transitive/shortest path problem is presented. Section VI concludes the paper. The general algebraic path problem (APP) is discussed in the Appendix.

II. Mapping Algorithms to Systolic Arrays

This section briefly outlines the techniques used in the design of the transitive closure/shortest path systolic array. This methodology is an extension of DG based mapping techniques investigated by numerous other authors [23]–[25], [27], [3], [28], [33]. These previous efforts all share as a common basis the work on uniform recurrence equations (URE's) by Karp, Miller, and Winograd [8]. The basic property of URE's is their uniformity throughout the index space. This property is very useful for theoretical development, but limits their practical application.

We have extended these basic ideas by easing the uniformity restrictions. The general details and development of this approach have been published elsewhere [14], [12], [13], [11]. Here we just summarize the basic model and mapping techniques used in the paper. Details are illustrated in the systolic array design for the transitive closure/shortest path problem.

A. Single Assignment Algorithms

The algorithms we are interested in are defined over a finite N-dimensional index space. The operation of the algorithm consists of computing the values for a set of indexed variables over this index space. Each of the indexed variables in the set is defined over a portion of the index space. (This portion may be the whole index space.) Each of these variables is similar to a multidimensional array, with a separate occurrence at each (defined) index point. Over the course of the algorithm, a single value is computed for each occurrence of each indexed variable; therefore, it is a *single assignment* algorithm.

The computation of any indexed variable in the algorithm may depend on the values of other indexed variables. If the index of one of the variables used in a computation is not defined, then it is considered to be an external input to the algorithm. The outputs of the algorithm can be any variables at the boundary of the index space.

B. Dependence Graphs (DG's)

The constraints on the ordering of the index point computations can be characterized by the *dependence graph* (DG). The DG is a directed acyclic graph embedded in the index space. Its nodes represent the computations associated with the index points and its directed arcs specify the direct data dependencies in computations. If the computation of a variable at index point B depends on the value of a variable at index point A, then B is dependent on A and there is a directed arc from A to B in the DG.

For an algorithm to be implemented in a systolic array, the dependence arcs in the associated DG of the algorithm should have a certain regularity. But they need not be entirely uniform, since there may be index dependent conditionals in the algorithm. Usually, the conditionals will partition the DG into several uniform regions. So the DG, while not uniform, will have some amount of structure and regularity.

C. Mappings

The simplest computation model to associate with this structure is that of having one processor devoted to each point in the index space. The processor at a point computes the value of all variables at that point. It does so only after all input variables used in these computations have already been computed.

If the DG was actually implemented with a separate processor for each index point, then the arcs of the DG would represent the (abstract) communication paths needed between processors. In general it is desirable to implement the DG with a smaller number of processors, using each processor to handle the computations for multiple index points. This necessitates the *mapping* of the computation onto this smaller number of processors. For the Warshall–Floyd algorithm, we utilize linear mappings from a three-dimensional index space onto a two-dimensional processor array. Linear mappings of this sort can be characterized as *projections*. These mappings are useful because they preserve the regularity properties of the DG, resulting in a regular implementation.

To generate the processor array via a projection, one first defines the *projection direction. All index points that lie on the same line in the projection direction are handled by the same processor.* The order in which these index points are handled is determined by a *linear schedule*. The linear schedule can be visualized (for a three-dimensional index space) as a series of parallel planes. All points on the same plane are computed at the same time. A linear schedule can be specified by a *schedule direction* ($\vec{s}$), defined as the vector perpendicular to the schedule planes and pointing in the direction of increasing time.

Note that the schedule planes cannot be parallel to the projection direction, otherwise multiple points in a plane will be projected to a single PE, which cannot compute these multiple points at the same time. This means that the projection and schedule directions are constrained to be nonorthogonal. In addition to this constraint, the linear schedule has to meet the data dependence constraints in the DG, that is, if there is an arc from node B to node A, then we cannot schedule A before B.

D. Signal Flow Graphs (SFG's)

The abstract array resulting from a projection and associated linear schedule can be represented by a *signal flow graph* (SFG) [17]. In an SFG each node represents a computation element. The arcs represent the communication links among

these computation elements. Communication and computation are assumed to be instantaneous, i.e., they take no time at all. Time is explicitly modeled by delays on the arcs. In representing the projection of a DG, the SFG nodes correspond to the projection of the DG nodes, and the SFG arcs correspond to the projection of the DG arcs. *The delays on an SFG arc are determined by the number of schedule planes separating the endpoints of the DG arc that it originated from.*

This SFG is not guaranteed to represent a systolic array. (An SFG is said to be systolic if all arcs have at least a single delay on them.) However, it can generally be retimed to be systolic via cut-set transformations [19], [17], [16].

E. Design Optimality Criteria

In order to find an optimal design of a systolic array, the optimality criteria must include many factors. Naturally the final choice of optimality criteria is application dependent. Some typical factors are the pipelining period, the block pipelining period, the computation time, the array size, and the number of I/O connections. Their meanings are explained below.

- *Pipelining Period:* the time interval between two successive computations for a processor, denoted by α. In other words, the processor is busy for one out of every α time intervals. Note that α is the reciprocal of the pipelining rate.
- *Computation Time:* the time interval between the start of the first computation and the end of the last computation of a problem instance by the processor array.
- *Block Pipelining Period:* the time interval between the initiations of two successive problem instances by the processor array.
- *Array Size:* the number of processors in the array. The array size determines the basic hardware cost.
- *I/O Channels:* the number of input/output lines between the processor array and the outside world (the host computer).

F. An Example: Matrix Vector Multiplication

To illustrate the mapping methodology, we present a systolic array design example for the matrix vector multiplication, $A \times b = c$, where A is an $N \times N$ matrix, and b and c are both $N \times 1$ vectors. First, we derive a *localized* DG, in which dependence arcs are "local" with respect to the dimension of the problem, N. By doing so, we can then obtain a locally connected array. A single assignment program with local data dependencies for the matrix vector multiplication is shown in the following.

For i, j from 1 to N

$$\alpha(i, j) \leftarrow a_{ij}$$

$$\beta(i, j) \leftarrow \beta(i-1, j)$$

$$x(i, j) \leftarrow x(i, j-1) + \alpha(i, j) \times \beta(i, j)$$

where the input data are a_{ij}, $\beta(0, j) \leftarrow b_j$, and $x(i, 0) \leftarrow 0$. The output vector is $c_i \leftarrow x(i, N)$.

In this single assignment algorithm, $\beta(i, j)$ is directly dependent upon $\beta(i-1, j)$, and $x(i, j)$ is directly dependent upon $x(i, j-1)$, $\alpha(i, j)$ and $\beta(i, j)$. The DG corresponding to this algorithm for the $N = 4$ case is shown in Fig. 1.

If we use the j direction as both the schedule vector and projection directions, we get the SFG shown in Fig. 1. This SFG can be retimed by the cut-set systolization [17] to derive the final systolic array shown in Fig. 2.

III. The Warshall–Floyd Algorithm

In this section, the Warshall–Floyd algorithm for solving the transitive closure and shortest path problems is presented and its DG is derived.

A. The Transitive Closure Problem

A directed graph G can be represented as a tuple $G(V, E)$, where V is the set of vertices and E is the set of edges in the graph. The graph $G^+(V, E^+)$, which has the same vertex set V as G, but has an edge from v to w if and only if there is a path (length zero or more) from v to w in G, is called the (*reflexive and*) *transitive closure of* G [1].[1]

As shown in Fig. 3, given a directed graph, its *adjacency matrix* A, whose element $a_{ij} = 1$ if there is an edge from vertex i to vertex j or $i = j$; $a_{ij} = 0$ otherwise. The transitive closure problem is to compute the *transitive closure matrix* A^+, whose element $a_{ij}^+ = 1$ if there is a path of length zero or more from vertex i to vertex j; $a_{ij}^+ = 0$ otherwise.

B. The Shortest Path Problem

The shortest path problem for a directed graph is to determine the lengths of the shortest paths between all pairs of nodes in the graph. For example, in Fig. 4, given a sample graph of six nodes, then the elements of its distance matrix A are defined as

1) a_{ij} = the distance from vertex i to vertex j if there is an edge from i to j;
2) $a_{ij} = \infty$ if there is no edge connecting i and j;
3) $a_{ij} = 0$ if $i = j$.

The shortest path problem is to compute the shortest path matrix A^+, whose element a_{ij}^+ is the shortest path length from vertex i to j.

C. Warshall–Floyd Algorithm

Given the input matrix A, the sequential Warshall–Floyd algorithm to find A^+ of the graph is quite well known and is shown below [1].

For k from 1 to N

For i from 1 to N

For j from 1 to N

$$x_{ij}^k \leftarrow x_{ij}^{k-1} + x_{ik}^{k-1} \times x_{kj}^{k-1};$$

where the initial X matrix, X^0, is equal to A; and the output matrix A^+ equals X^N.

This algorithm solves both the transitive closure and the

[1] A relation R is called *reflexive* when aRa for every a in V, and a relation R is *transitive* when it has the property: aRa, bRc implies aRc [26].

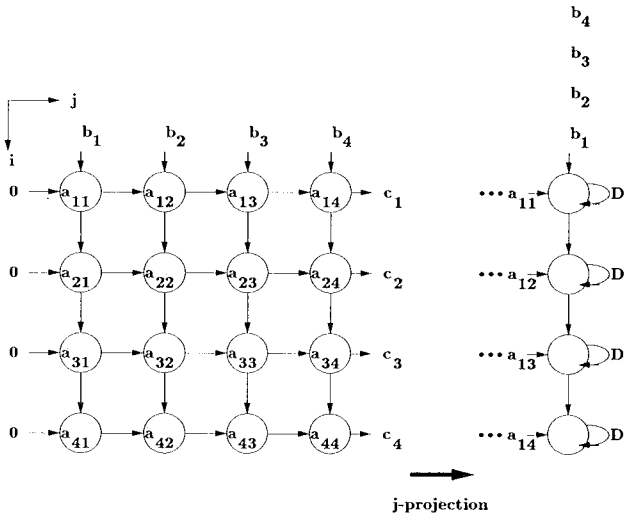


Fig. 1. DG and j projection for matrix vector multiplication.

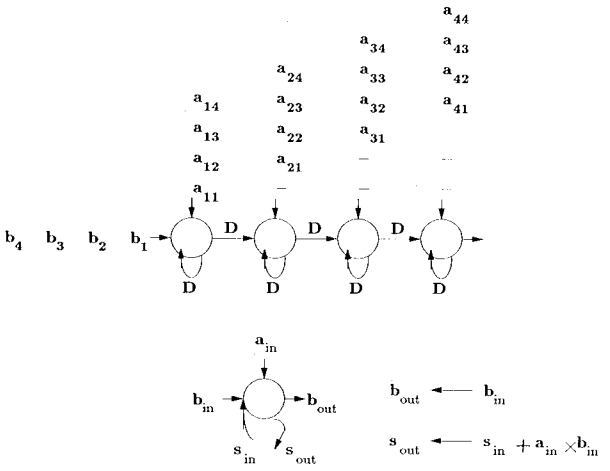
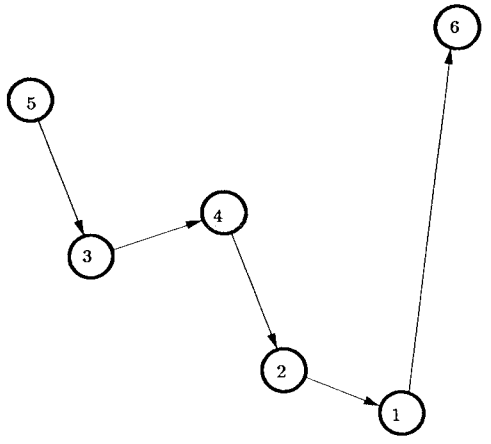


Fig. 2. A systolic array for matrix vector multiplication.



(a)

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad A^+ = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) (c)

Fig. 3. An example of the transitive closure problem. (a) A directed graph. (b) Its adjacency matrix A. (c) The transitive closure matrix A^+.

shortest path problems. The difference between the two problems can be stated as follows.

1) For the transitive closure problem, the input A is the adjacency matrix. And the + operation is the Boolean OR operation; the $\times$ operation is the Boolean AND operation. The output A^+ is the transitive closure matrix.

2) For the shortest path problem, the input A is the distance matrix. And the + operation is the minimum operation; the $\times$ operation is the addition operation. The output A^+ is the shortest path matrix.

The transitive closure problem and the shortest path problem actually belong to a more general class of problems, the *algebraic path problem* (APP), which also includes the Gauss-Jordan elimination procedure for matrix inversion. The definition and examples of the APP and an algorithm to solve the APP are discussed in the Appendix.

1) Snapshots of the Warshall-Floyd Algorithm: To illustrate how the Warshall-Floyd algorithm works, let us first show the snapshots of the recursions of the X^k matrix. Initially, the input matrix $A = X^0$ is assumed to be distributed in the $N \times N$ square matrix. For the transitive closure problem, the graph and its adjacency matrix A are shown in Fig. 3. The snapshots of the Warshall algorithm for finding the transitive closure of this graph are shown in Fig. 5. The snapshots of the Floyd algorithm for the shortest path problem in Fig. 4 are shown in Fig. 6. Note that, at the kth recursion, data on the kth column and the kth row, which are marked with dashed lines in both snapshots, are used in the computation of all matrix elements.

D. Dependence Graph for the Warshall-Floyd Algorithm

In the following, we will derive systolic arrays for the Warshall-Floyd algorithm in the context of the transitive closure problem only. The readers should note that the results are directly applicable to the shortest path problem.

The sequential Warshall-Floyd algorithm can be easily converted to a single assignment form by introducing an iteration dimension k.

For i, j, k from 1 to N

$$x(i, j, k) \leftarrow x(i, j, k-1) + x(i, k, k-1) \times x(k, j, k-1).$$

The above dependencies can be localized by adding propagating variables for the row variable r and column variable c at each level k.

For i, j, k from 1 to N

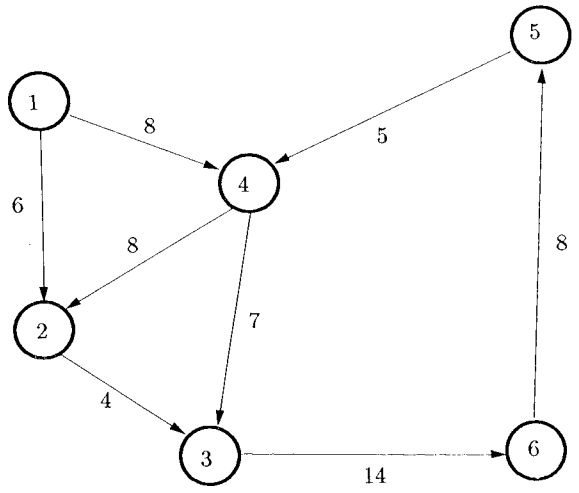
$$c(i, j, k) \leftarrow \begin{cases} x(i, j, k-1) & \text{if } j = k \text{ \#distribute column } k \\ c(i, j+1, k) & \text{if } j < k \text{ \#over row } i \\ c(i, j-1, k) & \text{if } j > k \end{cases}$$

$$r(i, j, k) \leftarrow \begin{cases} x(i, j, k-1) & \text{if } i = k \text{ \#distribute row } k \\ r(i+1, j, k) & \text{if } i < k \text{ \#over column } j \\ r(i-1, j, k) & \text{if } i > k \end{cases}$$

$$x(i, j, k) \leftarrow x(i, j, k-1) + r(i, j, k) \times c(i, j, k)$$

\#form kth connections.

where the input is $x(i, j, 0) \leftarrow a_{ij}$ and output is $a_{ij}^+ \leftarrow x(i, j, N)$.



$$A = \begin{bmatrix} 0 & 6 & \infty & 8 & \infty & \infty \\ \infty & 0 & 4 & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & 14 \\ \infty & 8 & 7 & 0 & \infty & \infty \\ \infty & \infty & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & \infty & 8 & 0 \end{bmatrix} \quad A^+ = \begin{bmatrix} 0 & 6 & 10 & 8 & 32 & 24 \\ \infty & 0 & 4 & 31 & 26 & 18 \\ \infty & 35 & 0 & 27 & 22 & 14 \\ \infty & 8 & 7 & 0 & 29 & 21 \\ \infty & 13 & 12 & 5 & 0 & 26 \\ \infty & 21 & 20 & 13 & 8 & 0 \end{bmatrix}$$

(b) (c)

Fig. 4. An example of the shortest path problem. (a) A directed graph. (b) Its distance matrix A. (c) The shortest-path matrix A^+.

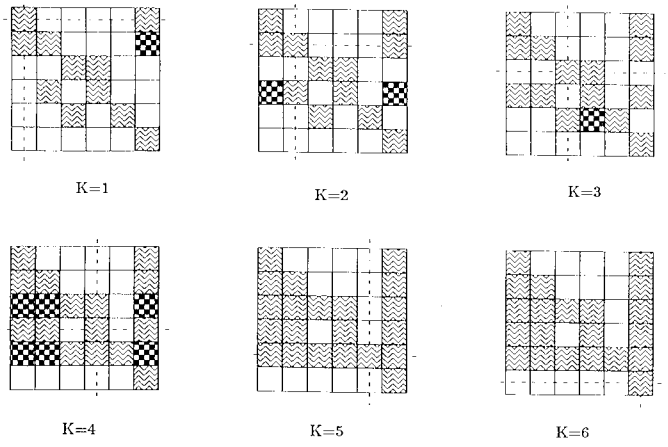


Fig. 5. Snapshots of the Warshall algorithm.

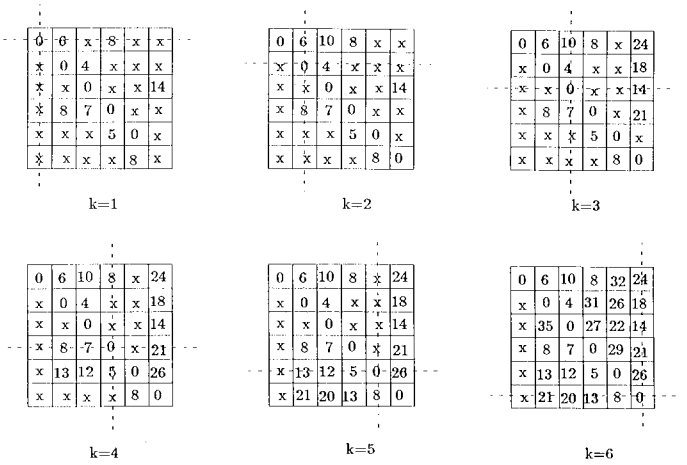


Fig. 6. Snapshots of Floyd algorithm for the shortest path problem.

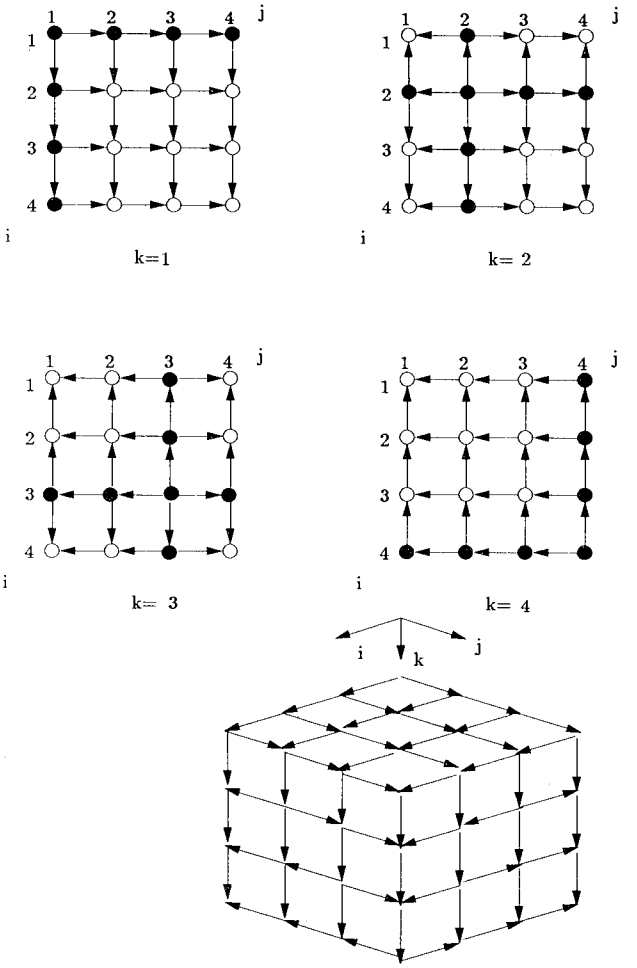


Fig. 7. Dependence graph for the $N = 4$ case.

The cubic DG is shown for the $N = 4$ case in Fig. 7, with each i-j plane drawn separately. What should have been included but not drawn are the additional dependence lines in the k direction that run from points (i, j, k) to $(i, j, k + 1)$. In each plane the points which serve as the "source" of the row and column values are marked as dark nodes. A notion of *transmittent data* is useful for the following discussion. A transmittent data is a data stream which does not change its value along the direction of its propagation; otherwise, it is *nontransmittent*. The transmittent data simulates a broadcasting activity.

In the following, we will project the DG in several directions to find the optimal systolic array.

IV. Spiral and Hexagonal Systolic Arrays

A. Reindexing the DG

The DG in Fig. 7 is not totally regular. The "source" nodes in the DG move from row (column) k to row (column) $k + 1$ from the kth i-j plane to the next one, and the transmittent data streams in the i-j planes propagate in two opposite directions. These irregularities will cause problems in projecting onto an SFG. One way to get around these problems is by reindexing the nodes in the DG; thereby transforming the DG into a more regular one. We would like to reindex the nodes in the DG such that all "source" nodes are located in the first

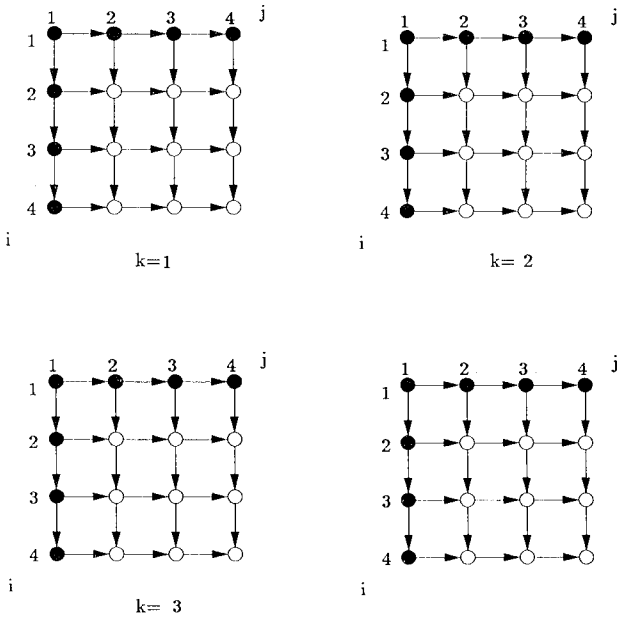


Fig. 8. Dependence arcs in i–j planes in DG-2 for the $N = 4$ case.

row or the first column in each i–j plane and the transmittent data propagate in only one direction. This can be achieved by the following reindexing:

$$\text{node } (i, j, k) \rightarrow \text{node } ((i-k)_{\text{mod } N}+1,\ (j-k)_{\text{mod } N}+1,\ k).$$

Let us call the original DG as DG-1, and the reindexed DG as DG-2. In DG-2, the above reindexing rearranges nodes in any i–j plane such that the "source" nodes are always on the first column or the first row. *Since the data propagating in the i–j planes are transmitted data,* i.e., *they do not change their values, the dependence arcs in the i–j planes may still remain as local arcs.* Therefore, all four i–j planes should look the same in DG-2 (cf. Fig. 8). However, the dependence arcs between the i–j planes will become somewhat complicated. In the original DG-1, the dependence arcs from the kth i–j plane to the $(k+1)$st i–j plane are represented by the vector $[0\ 0\ 1]^T$. In DG-2, these dependence vectors are transformed to

$$[(i-k-1)_{\text{mod } N}-(i-k)_{\text{mod } N},\ (j-k-1)_{\text{mod } N}-(j-k)_{\text{mod } N},\ 1]^T.$$

The modulo operation will lead to two types of dependence arcs.

1) One is represented by the dependence vector $[-1, -1, 1]^T$, when $(i-k)_{\text{mod } N} \neq 0$ and $(j-k)_{\text{mod } N} \neq 0$. The dependence arcs of this type are local.

2) Another is represented by the vectors $[(N-1), -1, 1]^T$ or $[-1, (N-1), 1]^T$ or $[(N-1), (N-1), 1]^T$, when $(i-k)_{\text{mod } N} = 0$, or $(j-k)_{\text{mod } N} = 0$. Note that dependence arcs of this type are global. They are termed as *spiral* arcs.

Now, we are able to draw a more complete DG-2. The dependence arcs in the i–j planes are shown in Fig. 8. The two types of dependence arcs between any two adjacent i–j planes (in this case, the first two planes) in DG-2 are shown in Fig. 9.

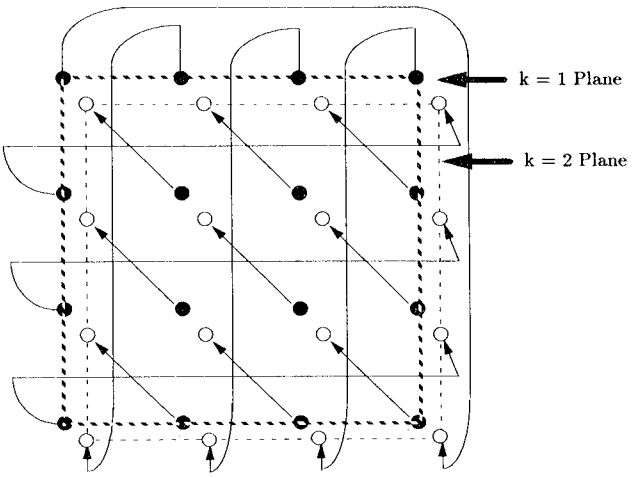


Fig. 9. Dependence arcs between the first two i–j planes in DG-2.

B. The Reindexed Single Assignment Algorithm

We have shown the effect of the modulo reindexing on the DG. The corresponding effect on the single assignment algorithm can be similarly derived. The reindexed single assignment algorithm is shown in the following.

For i, j, k from 1 to N

$$c(i, j, k) \leftarrow \begin{cases} x(i_{\text{mod } N}+1,\ j_{\text{mod } N}+1,\ k-1) \\ \quad \text{if } j=1 \\ c(i, j-1, k) \\ \quad \text{if } j \neq k \end{cases}$$

$$r(i, j, k) \leftarrow \begin{cases} x(i_{\text{mod } N}+1,\ j_{\text{mod } N}+1,\ k-1) \\ \quad \text{if } i=1 \\ r(i-1, j, k) \\ \quad \text{if } i \neq k \end{cases}$$

$$x(i, j, k) \leftarrow x(i_{\text{mod } N}+1,\ j_{\text{mod } N}+1,\ k-1) + r(i, j, k) \times c(i, j, k)$$

where the input and output have to be adjusted accordingly.

C. Spiral SFG Array: Projection in the k Direction

Note that the dependence arcs are invariant from one i–j plane to another in DG-2, i.e., it is uniform in the k direction. Thus, if we project DG-2 in the k direction (the $[0\ 0\ 1]^T$ direction) and use a linear schedule vector in the k direction, (i.e., all the nodes in an i–j plane are computed at the same time), then we can derive a regularly structured SFG. According to the projection, the input matrix A should reside inside the array, i.e., a_{ij} should be at PE (i, j) to be ready for the execution. Practically, one has to load (input) the matrix A from the outside (i.e., the host) to the processor array in the

loading phase. In our case, we choose to input the matrix A from the lower right corner of the SFG array along the diagonal (north-west) direction, as shown in Fig. 10. The *loading phase* takes N time steps so that all the input a_{ij} arrives at PE (i, j). The SFG array is then ready for the computation, i.e., the *execution phase*, which also takes N time steps. After the computation is done, the output matrix A^+ is sent out from the left and upper side of the array (i.e., the *unloading phase*).

D. Hexagonal Systolic Array: Elimination of the Spiral Arcs

1) Elimination of the Spiral Arcs: Due to the spiral arcs, the SFG in Fig. 10 is termed a *spiral* SFG array. The spiral interconnections imply potentially expensive global wiring in VLSI technology. Fortunately, there is a simple way to circumvent this problem based on the following observation: *the x variables in the kth row or kth column of the kth i–j plane in the original DG-1*[2] *do not change their values.*

For example, let us examine the computation of the first two *i–j* planes in DG-2. From the original Warshall–Floyd algorithm, we observe that

$$x(i, 1, 1) = x(i, 1, 0) + x(i, 1, 0) \times x(1, 1, 0)$$

since nodes are connected to themselves, therefore,

$$x(1, 1, 0) = 1.$$

Thus,

$$x(i, 1, 1) = x(i, 1, 0).$$

This shows that no change is effected for the x variables in the first column of the $k = 1$ *i–j* plane. Similarly, no change is effected for the x variables in the first row of the $k = 1$ *i–j* plane. Furthermore, notice that $c(i, 1, 1) = x(i, 1, 0)$ are the transmittent data in the $k = 1$ plane. Then

$$x(i, 1, 0) = c(i, 1, 1) = c(i, N, 1) = x(i, 1, 1). \quad (1)$$

Therefore, the original global arc in DG-2 from node $(i, 1, 1)$ to node $(i - 1, N, 2)$, carrying data $x(i, 1, 1)$, is now replaced by a local arc from node $(i, N, 1)$ to node $(i - 1, N, 2)$, carrying data $c(i, N, 1)$ [cf. (1)]. Similarly, the original global arc from node $(1, j, 1)$ to node $(N, j - 1, 2)$ carrying data $x(1, j, 1)$, is now replaced by a local arc from node $(N, j, 1)$ to node $(N, j - 1, 2)$, carrying data $r(N, j, 1)$ [note that $r(N, j, 1) = r(1, j, 1) = x(1, j, 0) = x(1, j, 1)$]. We further note that the data $x(1, 1, 1)$ is a constant 1 in the computation. Therefore, it is not necessary to send the constant 1 from node $(1, 1, 1)$ to node $(N, N, 2)$ by the long global arc. Instead, it can be locally generated by node $(N, N, 2)$ and therefore this global arc can be removed. These local dependence arcs between the first two *i–j* planes are shown in Fig. 11.

The above observation is true for other *i–j* planes, i.e., the x variables in the first row or first column of the kth *i–j* plane in DG-2 (or the kth row or kth column of the kth *i–j* plane in DG-1) do not change their values. This can be seen from the original Warshall–Floyd algorithm.

$$x(i, k, k) = x(i, k, k-1) + x(i, k, k-1) \times x(k, k, k-1) = x(i, k, k-1)$$

$$x(k, j, k) = x(k, j, k-1) + x(k, k, k-1) \times x(k, j, k-1) = x(k, j, k-1)$$

since $x(k, k, k - 1) = 1$.

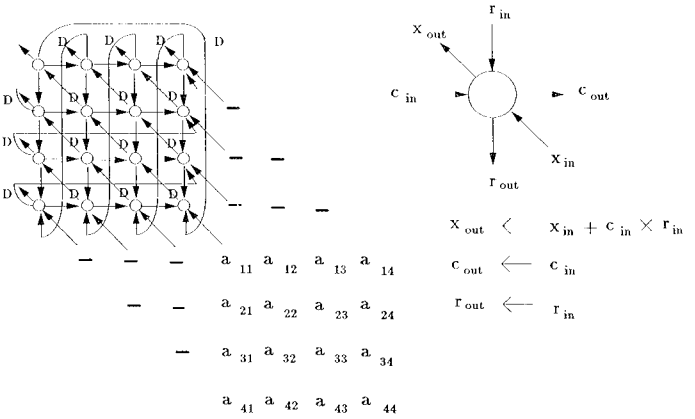


Fig. 10. The spiral SFG array derived by projection in the k direction of DG-2.

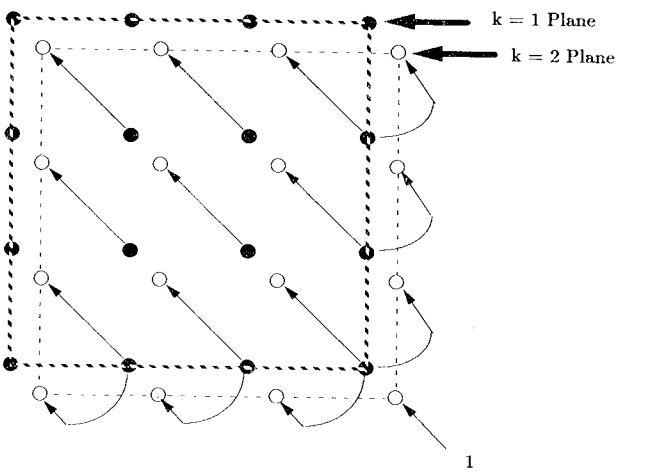


Fig. 11. Removal of spiral arcs between the first two *i–j* planes of DG-2.

2) Local and Regular Dependence Graph (DG-3): By the same argument, all global arcs in DG-2 can be replaced by local arcs for all *i–j* plane computations. This localized DG is denoted as DG-3 and is shown in Fig. 12. Note that DG-3 is also uniform in the k direction.

3) Hexagonal SFG Array: Now if we project DG-3 again in the k direction, the result will be the locally connected hexagonal SFG array shown in Fig. 13. Note that the delays associated with the original spiral arcs in Fig. 10 are the same as the delays on the corresponding local arcs in Fig. 13 to ensure the correct scheduling.[3]

[2] or, equivalently, the x varibles in the first column or first row in all *i–j* planes in DG-2.

[3] We can further simplify this hexagonal SFG array by deleting the first row and first column PE's, since they do not perform any computation. This leads to an array of size $(N - 1)^2$.

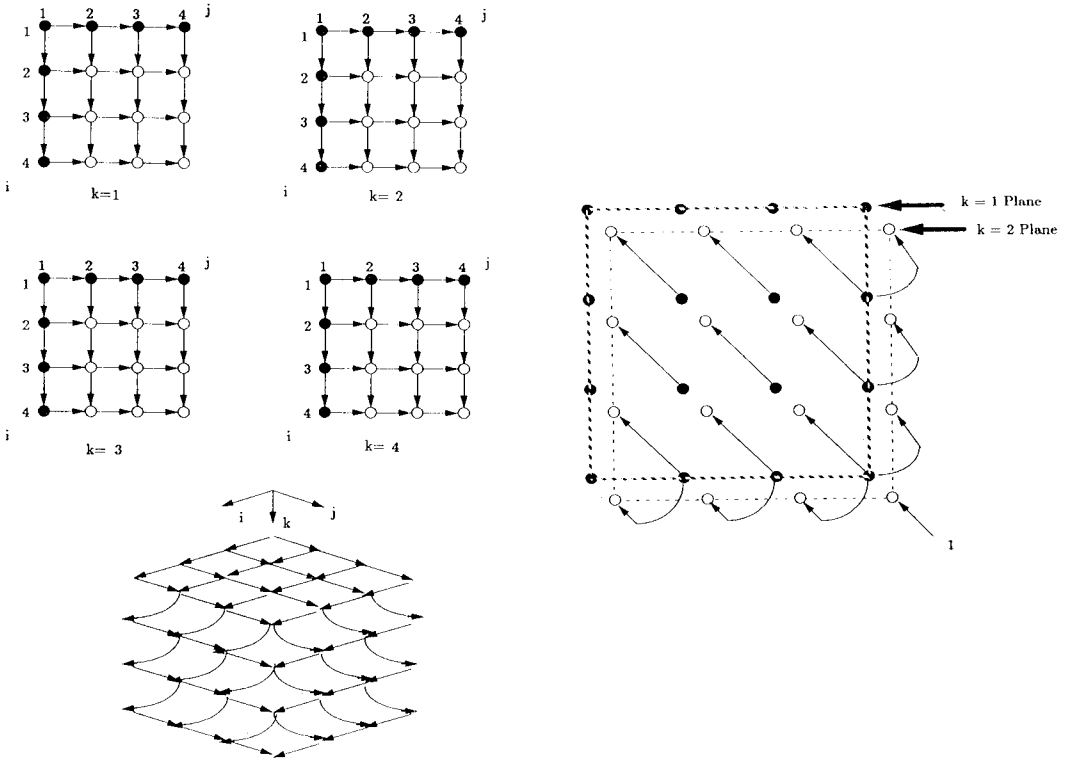


Fig. 12. The localized DG-3.

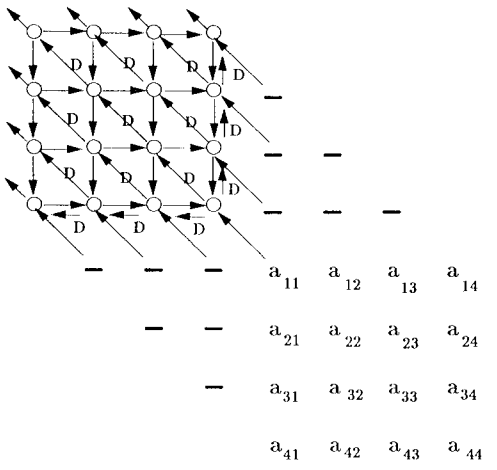


Fig. 13. Locally connected hexagonal SFG.

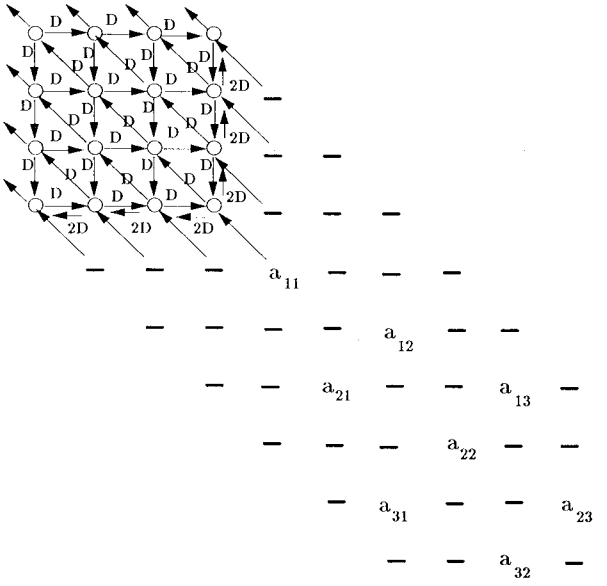


Fig. 14. The hexagonal systolic array.

4) Hexagonal Systolic Array Design: The hexagonal systolic array is derived by applying the cut-set systolization [17] to the hexagonal SFG array in Fig. 13. The result is shown in Fig. 14. Note that the *pipelining period* $\alpha = 3$ for this systolic array. As is shown in Fig. 14, the input data are loaded from the southeast end of the array. Once the data are loaded into their respective PE positions, execution of the Warshall–Floyd algorithm can proceed.

A short historical note is in order here. The fact that the spiral connections can be avoided was pointed out by Lin and Wu [21]. However, their scheme actually includes an extra PE column (on the right-hand side) and an extra PE row (at the bottom), which are not needed in the design specified above. Rote [31] also developed the hexagonal systolic array independently.

V. Optimal Orthogonal Systolic Array

Although the hexagonal systolic array has only local connections, the pipelining period $\alpha = 3$ and the number of I/O pins of this array is $4N$. This is not very desirable. To obtain an SFG with $\alpha = 1$ and with fewer I/O pins, we have to opt for an alternative projection direction. After a close

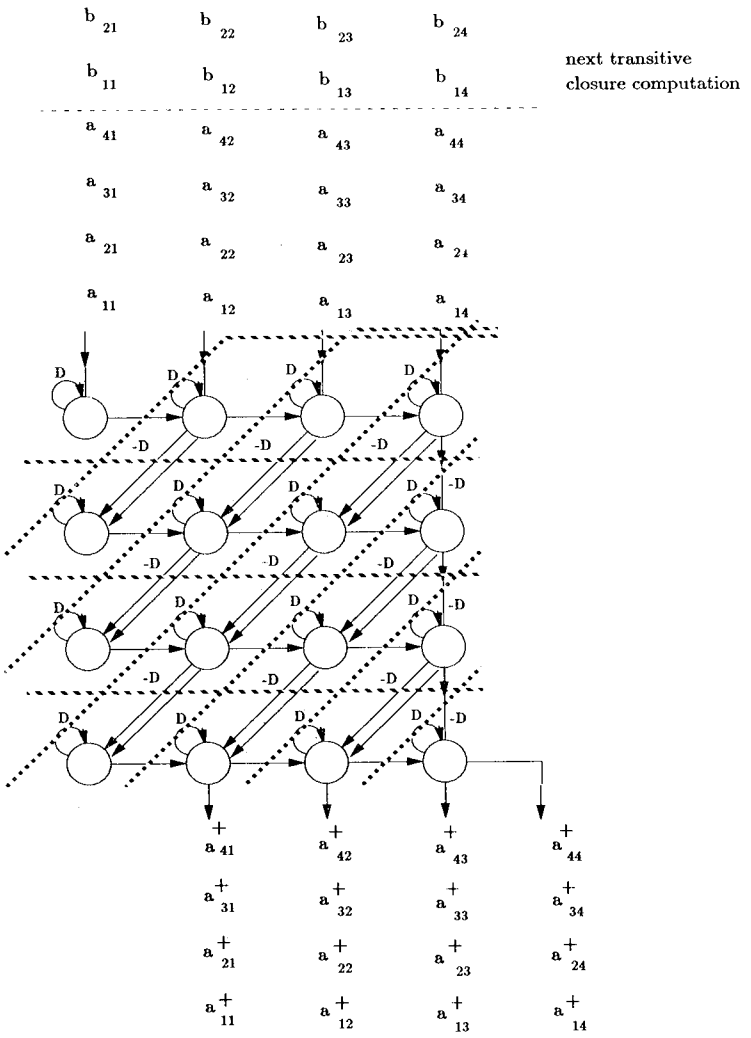


Fig. 15. An orthogonal SFG array derived by projection in i-direction of DG-3.

investigation, we find that projection in the either i or j direction will serve the purpose. Note that projections in both the i direction and j direction are similar due to the symmetry in DG-3. Consider, now, projecting DG-3 along the i direction.

A. Orthogonal SFG Array

Now, we project DG-3 in the i direction. The schedule vector we choose is also in the i direction, that is, all nodes in the k–j plane are computed at the same time, and the time step increases in the positive i direction. The resulting SFG array is shown in Fig. 15.[4] Note that in order to ensure the proper computation of the SFG, certain control signals are needed, which are not explicitly shown in the figure. For detailed SFG operations, the readers are referred to [20]. Note that this SFG has pipelining period $\alpha = 1$ and a block pipelining period of N. Moreover, the number of I/O pins is only $2N$.

[4] Note that now there are negative delays in the SFG. This is because we have used the schedule vector $\vec{s} = [1\ 0\ 0]^T$. (This problem can be circumvented by properly choosing another schedule vector.) However, the negative delays in the SFG can be transformed to be positive ones by the cut-set systolization [11].

B. Orthogonal Systolic Array

The above orthogonal SFG array in Fig. 15 can be systolized by the cut-set retiming transformations [17], [11] on the cut-sets (denoted by dashed lines) in the same figure. Note that one delay is transferred along the diagonal cuts and two delays are transferred along the horizontal cuts. The resulting systolic array is shown in Fig. 16.[5]

C. Optimality of the Orthogonal Systolic Design and Comparisons to Other Arrays

This orthogonal systolic array is an optimal one for the transitive closure problem. The array uses N^2 processors and $2N$ I/O ports. It supports a noninterleaved block pipeline period of N, so successive computations can be repeated every N steps. (Note that in Fig. 16, $\{b_{ij}\}$ data block follows immediately after the $\{a_{ij}\}$ data block.) Due to the systolization process, the total computation time is $5N - 4$. We can see from the original "parallelized" Warshall–Floyd algorithm dependence graph (DG-1) that this is indeed optimal.

[5] This systolic array can be also obtained directly by adopting a schedule vector $\vec{s} = [1\ 1\ 3]^T$.

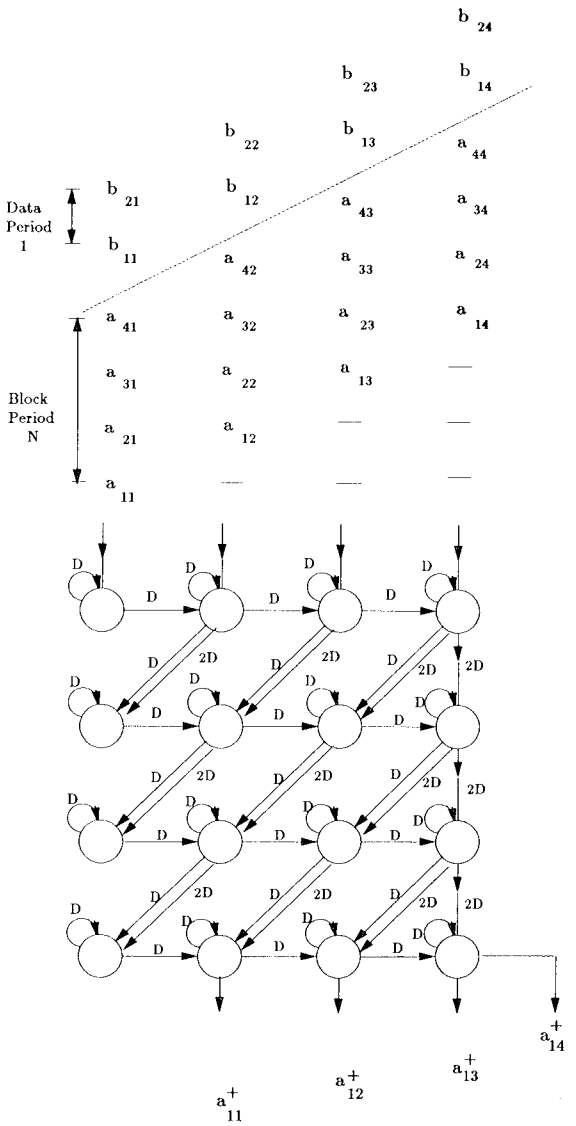


Fig. 16. The optimal orthogonal systolic array.

Since N^3 computations must be performed with N^2 processors, N is the minimal period. This minimal period insures 100 percent processor utilization for pipelined series of computations. In DG-1 we find that there exists a directed path of length $5N - 4$ along the path $(1, 1, 1) \rightarrow (N, 1, 1) \rightarrow (N, N, 1) \rightarrow (N, N, N) \rightarrow (1, N, N) \rightarrow (1, 1, N)$. This path is shown in Fig. 17 by boldfaced arcs. Hence, any systolic implementation, in which all inputs must be calculated at least one step before they are used, will have a computation time of at least $5N - 4$. Finally, we note that for a period of N we need to input N data and output N data at each step, and, hence, $2N$ is the minimal number of I/O connections required.

A number of other authors have proposed systolic arrays for the transitive closure/shortest path problems or for the general APP. All use approximately N^2 processors to perform the computations in $O(N)$ time, and all require some sort of control signals to temporally phase the processing elements.

The first systolic array for transitive closure was a three-pass mesh-connected array described by Guibas, Kung, and Thompson [7]. This structure is relatively slow and requires $2N$ global interconnections. Hexagonal transitive closure

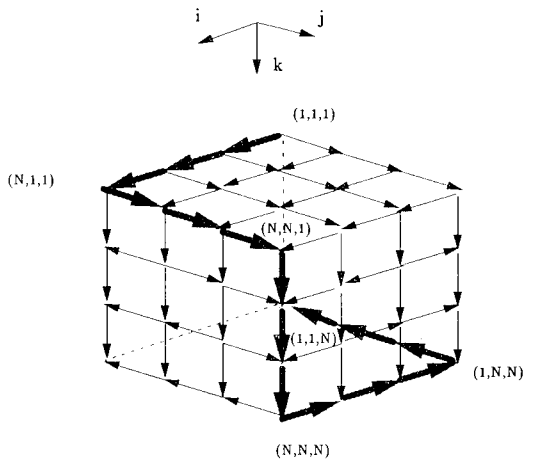


Fig. 17. The longest path in DG-1 of the Warshall–Floyd algorithm.

arrays were proposed by Kung and Lo [15] and Lin and Wu [21]. Lin and Wu's configuration is similar to our previous hexagonal array but requires an additional row and column of processing elements.

Rote [31] described the first arrays for the general APP. His basic array is a hexagonal configuration similar to those mentioned above. Like those above, this array has a pipelining period of three, and must utilize pipeline interleaving to achieve maximum efficiency. Even with pipeline interleaving, the minimum block pipelining period the array can support is $2N$. Rote proposed a method of overcoming this shortcoming by duplicating all of the communication paths in the array to allow overlap of the loading and computation phases. A configuration of this nature would support a rate of N, but at the price of significantly increased control complexity and a doubling of the internal interconnections.

In the process of preparing the final manuscript, the authors have learned that Robert and Trystram [29] have independently derived a similar orthogonal array. However, their version is only half as efficient as the one proposed here. It can only support a block pipelining period of $2N$, while our array supports a block pipelining period of N.

Table I provides a comparison of some of the key parameters of the arrays discussed above. Based on the comparison, the array proposed here is superior to the other currently existing designs.

VI. Conclusion

In this paper, we have presented two systolic arrays for the transitive closure and shortest path problems, which are derived by a systematic design methodology. One has a spiral (or hexagonal) structure and the other an orthogonal structure. The orthogonal systolic array is optimal in terms of pipelining period, block pipelining period, and I/O pins. The transitive closure and shortest path problems belong to a more general algebraic path problem (APP). Our systolic designs can be applied to the APP with minor modification [20].

Appendix

The Algebraic Path Problem

The basic structures developed for the transitive closure problem can be applied to a wide range of other problems.

TABLE I
COMPARISON OF TRANSITIVE CLOSURE SYSTOLIC ARRAYS

Array	Comp Time	Data Period	Block Period	Ext I/O	Int I/O	Global Lines
Ours	5N	1	N	2N	$2N^2$	No
Guibas et al.	5N	1	3N	4N	$2N^2$	Yes
Kung, Lo	7N	3	2N	4N	$3N^2$	No
Lin, Wu	7N	3	2N	4N	$3N^2$	No
Rote-basic	7N	3	2N	4N	$3N^2$	No
Rote-cmplx	7N	3	N	4N	$6N^2$	No
Robert Trystram	5N	1	2N	2N	$2N^2$	No

This is because mathematically transitive closure is a special case of a general problem known as the *algebraic path problem* (APP). Other problems in this class are the *shortest-path* problem from graph theory, the *matrix inversion* problem from linear algebra, and the generation of *regular languages* from automata theory [6]. The basic structures developed here can be applied to any instance of the APP.

In this section we briefly summarize the APP and a set of recursive equations to solve it, along with several examples. In depth development of arrays for the general APP can be found in [20], [31], and [30]. Thorough derivations of the mathematics associated with the APP can be found in [18], [22], and [6].

A. Problem Definition

We are given a weighted directed graph $G = (V, E, w(e))$, with V the set of nodes, E the set of directed arcs, and $w(e)$ a weighting of the arcs. The arc weights, $w(e)$, are drawn from a semiring (or dioid) $S = \{H, +, \times, 0, 1\}$. In S "addition" ($+$) and "multiplication" ($\times$) are closed binary associative operations over the set of elements H, with 0 and 1 the respective identity (or neutral) elements. Addition is commutative, multiplication distributes over addition, and 0 is absorptive with respect to multiplication.[6] In addition, a unary operation of closure, denoted a^*, is defined.[7] The weight of a directed path $w(p)$ is defined as the product of the arc weights in the path p.

$$w(p) \equiv \prod_{\forall e \in p} w(e).$$

If we number the graph nodes from 1 to N, then the APP is to find, for each pair of nodes i and $j (1 \le i, j \le N)$, the sum of the weights of all distinct paths from i to j. Denoting this sum as d_{ij} we have

$$d_{ij} \equiv \sum_{\forall p} w(p) \qquad \text{for } p\text{: path from } i \text{ to } j.$$

The algorithm to find the d_{ij}'s is defined in terms of a matrix formulation of the problem. Here we define $A = \{a_{ij}\}$ as the *adjacency matrix* of the graph where

$$a_{ij} = \begin{cases} w(e) & \text{if } e \text{ is the arc from } i \text{ to } j \\ 0 & \text{if there is no arc from } i \text{ to } j. \end{cases}$$

We can then define the result of the algebraic path calculation as a matrix $D = \{d_{ij}\}$. For a graph with N nodes, the following recurrence equations in C^k allow us to calculate $D = C^N$, starting with $C^0 = A$ [6].

$$c_{ij}^k = c_{ij}^{k-1} + \{c_{ik}^{k-1} \times (c_{kk}^{k-1})^* \times c_{kj}^{k-1}\} \qquad \text{for } i \neq k,\ j \neq k$$

$$c_{ij}^i = (c_{ii}^{i-1})^* \times c_{ij}^{i-1} \qquad \text{for } i \neq j$$

$$c_{ij}^j = c_{ij}^{j-1} \times (c_{jj}^{j-1})^* \qquad \text{for } i \neq j$$

$$c_{ii}^i = (c_{ii}^{i-1})^*.$$

B. Examples of the APP

1) Transitive Closure: Finding the transitive closure $G^+ = (V, E^+)$ of a directed graph $G = (V, E)$ is a special case of the APP where the arc weights of the graph belong to the semiring $S1 = \{H1, \vee, \wedge, 0, 1\}$, where $H1 = \{0, 1\}$. For each arc in G, $w(e) = 1$, so the adjacency matrix A contains just zeros and ones. In this case, the unary operation $*$ is equivalent to the constant operation 1.[8] The recurrence equations above, along with a serial ordering, simplify in this case to Warshall algorithm for transitive closure [32].

2) Shortest Path: Determining the lengths of the shortest paths between all pairs of nodes in a directed graph is another instance of the APP. In this case the arc weightings, corresponding to distances, belong to the semiring $S2 = \{H2, \min, +, \infty, 0\}$, where $H2$ is the set of nonnegative real numbers augmented with ∞, ($H2 = \{[0, \infty), \infty\}$). In this case the unary $*$ operation is equivalent to the constant operation 0.[9] The above recurrences, along with a serial ordering, simplify to the shortest path algorithm of Floyd [4].

3) Gauss–Jordan Elimination (Matrix Inversion): Finding the inverse of a real nonsingular matrix can also be cast as a special case of the APP. The matrix to be inverted is related to the adjacency matrix A. Its elements, corresponding to the arc weights, belong to the semiring $S3 = \{H3, +, \times, 0, 1\}$, where $H3$ is the set of real numbers and $+$ and $\times$ correspond to ordinary addition and multiplication. An extended closure is defined as

$$c^* \equiv \begin{cases} 1/(1-c) & \text{for } c \neq 1 \\ \text{undefined} & \text{for } c = 1. \end{cases}$$

Solving the APP yields $(I - A)^{-1}$. Minor changes in the algorithm allow A^{-1} to be computed directly. With these changes, the recurrence equations, along with a serial ordering, correspond to the Gauss–Jordan algorithm (without pivoting) of linear algebra [31].

C. Systolic Array Design for the APP

The two systolic arrays presented in this paper can be directly transformed to the APP by taking care of the extra closure computation. For details, the readers are referred to [20].

[6] $a \times 0 = 0 \times a = 0$.
[7] $a^* \equiv 1 + a + (a \times a) + (a \times a \times a) + \cdots$.
[8] $a^* \equiv 1 \vee (a \wedge a) \vee (a \wedge a \wedge a) \vee \cdots = 1$.
[9] $a^* \equiv \min(0, a, a + a, a + a + a, \cdots) = 0$.

REFERENCES

[1] A. V. Aho, J. E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] P. R. Cappello and K. Steiglitz, "Unifying VLSI array designs with geometric transformations," in *Proc. Int. Conf. Parallel Processing*, 1983.

[3] P. R. Cappello *et al.*, *VLSI Signal Processing*. New York: IEEE Press, 1984.

[4] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, June 1962.

[5] J. A. B. Fortes, K. S. Fu, and B. W. Wah, "Systematic approaches to the design of algorithmic specified systolic arrays," in *Proc. IEEE ICASSP*, Tampa, FL, Mar. 26–29, 1985.

[6] M. Gondran, M. Minoux, and S. Vajda, *Graphs and Algorithms*. New York: Wiley, 1984.

[7] L. J. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI implementation of combinatorial algorithms," in *Proc. Caltech Conf. VLSI*, Los Angeles, CA, 1979.

[8] R. M. Karp, R. E. Miller, and S. Winograd, "The organization of computations for uniform recurrence equations," *J. Ass. Comput. Mach.*, vol. 14, pp. 563–590, July 1967.

[9] H. T. Kung, "Why systolic architectures?" *IEEE Computer*, vol. 15, Jan. 1982.

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

[11] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[12] ——, "VLSI array processors," *SIAM Newsletter*, Jan. 1987.

[13] S. Y. Kung, P. S. Lewis, and S. N. Jean, "Canonic and generalized mapping from algorithms to arrays—A graph based methodology," in *Proc. Hawaii Int. Conf. Syst. Sci.*, Jan. 1987.

[14] S. Y. Kung, P. S. Lewis, and S. C. Lo, "On optimally mapping algorithms to systolic arrays with application to the transitive closure problem," in *Proc. 1986 IEEE Int. Symp. Circuits Syst.*, pp. 1316–1322.

[15] S. Y. Kung and S. C. Lo, "A spiral systolic architecture/algorithm for transitive closure problems," in *Proc. IEEE Int. Conf. Comput. Design*, 1985.

[16] S. Y. Kung, S. C. Lo, and J. Annevelink, "Temporal localization and systolization of signal flow graph (SFG) computing networks," in *SPIE*. San Diego, CA: McGregor and Werner, Aug. 1984.

[17] S. Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, July 1984.

[18] D. H. Lehmann, "Algebraic structures for transitive closure," *Theoret. Comput. Sci.*, vol. 4, pp. 59–76, 1977.

[19] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. Caltech VLSI Conf.*, Pasadena, CA, 1983.

[20] P. S. Lewis and S. Y. Kung, "Dependence graph based design of systolic arrays for the algebraic path problem," in *Proc. 12th Ann. Asilomar Conf. Signals, Syst., Comput.*, Nov. 1986.

[21] F. C. Lin and Wu, "Systolic arrays for transitive closure algorithms," in *Proc. Int. Symp. VLSI Syst. Designs*, Taipei, May 1985.

[22] B. Mahr, "Iteration and summability in semirings," *Ann. Discrete Math.*, vol. 19, Dec. 1984.

[23] W. L. Miranker, "Space-time representations of computational structures," *Computing*, 1984.

[24] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays," *Proc. IEEE*, vol. 71, Jan. 1983.

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

[26] O. Ore, *Theory of Graphs*. Providence, RI: Amer. Math. Soc., 1962.

[27] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," in *Proc. 11th Annu. Symp. Comput. Architecture*, 1984, pp. 208–214.

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

[29] Y. Robert and D. Trystram, "An orthogonal systolic array for the algebraic path problem," *Res.* Report 553, TIM3/IMAG Grenoble, France, 1985.

[30] ——, "An orthogonal systolic array for the algebraic path problem," presented at *Int. Workshop Systolic Arrays*, 1986. *Arrays*, 1986.

[31] G. Rote, "A systolic array algorithm for the algebraic path problems (shortest paths, matrix inversion)," *Computing*, vol. 34, pp. 192–219, 1985.

[32] S. Warshall, "A theorem on Boolean matrices," *J. Ass. Comput. Mach.*, vol. 9, Jan. 1962.

[33] Y. Wong and J-M Delosme, "Optimal systolic implementation of *n*-dimensional recurrences," in *Proc. ICCD*, 1985, pp. 618–621.

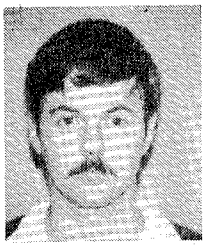
Sun-Yuan Kung (S'74–M'78–SM'84) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1977.

He was associated with the Amdahl Corporation, Sunnyvale, CA as an Associate Engineer in LSI design and simulation in 1974. He was a Visiting Professor at the Delft University of Technology, and a Visiting Professor at Stanford University in 1984. Since July 1977, he has been on the faculty of the University of Southern California, Los Angeles, where he is presently a Professor of Electrical Engineering. His research interests include linear systems approximation, modern spectrum analysis, and special-purpose signal processing applications, VLSI array processors, and design of special-purpose supercomputers.

From 1984 to 1985 he was the Associate Editor for the VLSI area in the IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING (ASSP) and currently serves on the VLSI Technical Committee. He served as the General Chairman of the IEEE Workshops on VLSI Signal Processing in 1982 and in 1986. He was a U.S. delegate to the U.S.–Japan Joint Seminar on Mathematical Systems Theory in 1983. He was the Keynote Speaker for the First International Workshop on Systolic Arrays, Oxford, July 1986. He is co-editor of *VLSI and Modern Signal Processing*, (Englewood Cliffs, NJ: *Prentice-Hall*, 1985), and co-editor of *VLSI Signal Processing, II* (IEEE Press, 1987). He is the author of a textbook *VLSI Array Processors* (Englewood Cliffs, NJ: Prentice-Hall, 1987).

Sheng-Chun Lo was born in Taiwan, on March 13, 1959. He received the B.S. degree from the National Taiwan University, Taiwan, in 1980 and the M.S. degree from the University of Southern California, Los Angeles, in 1984, both in electrical engineering.

Currently, he is a Research Assistant pursuing the Ph.D. degree in electrical engineering at U.S.C. His research interests are in parallel computer architectures, digital signal processing, and parallel algorithm design.



Paul S. Lewis was born in Albany, NY, on January 22, 1956. He received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1978, and the M.S. degree in electrical engineering from the University of New Mexico, Albuquerque, in 1984.

In 1978 he joined the Technical Staff at the Los Alamos National Laboratory, and is currently a member of the Mechanical and Electronic Engineering Division. He is currently pursuing a Ph.D. degree in electrical engineering at the University of Southern California, Los Angeles, under the sponsorship of the Los Alamos National Laboratory Advanced Study Program. His research interests are in signal processing and computer architectures.