

LINEAR SYSTOLIC ARRAYS FOR COMPUTING TRANSITIVE CLOSURE

I. Ž. Milovanović

*Faculty of Electronic Engineering
18000 Niš, Serbia*

E. I. Milovanović

*Faculty of Electronic Engineering
18000 Niš, Serbia*

B. M. Randjelović

*Faculty of Electronic Engineering
18000 Niš, Serbia*

Abstract

This paper addresses the problem of computing transitive closure of a given directed graph on linear systolic arrays (SA). We have designed three linear SAs, including two static SAs and a bi-directional SA. Then we compare obtained arrays in term of execution time, number of processing elements and efficiency.

Keywords: transitive closure; systolic arrays.

INTRODUCTION

The problem of finding transitive closure (TC) can be defined as follows. For a given directed graph $G = (V, E)$, where V is a set of vertices and E a set of directed edges, find a graph $G^* = (V, E^*)$ which has the same vertex set V , but has a directed edge from vertex v to vertex w if and only if there is a directed path from vertex v to vertex w in G . Graph G^* is called a transitive closure of G .

The problem of computing TC of a given graph has been studied extensively in the literature because it is an abstraction of many practical problems that are used in wide variety of applications in mathematics, computer science, engineering and business. It is required, for instance, in reachability analysis of transition networks representing distributed and parallel systems and in the construction of parsing automata in compiler construction. Recently, efficient transitive closure computation has been recognized as a significant sub problem in evaluating recursive database queries, since almost all practical recursive queries are transitive.

In this paper we consider an implementation of TC problem on linear systolic arrays (SA). Our goal is to synthesize regular linear SAs with optimal number of processing elements (PE) for

computing TC. The execution time of synthesized arrays should be as minimal as possible.

SYSTOLIC ALGORITHM

A directed graph $G = (V, E)$ can be represented by the corresponding adjacency matrix $A = (a_{ij})_{n \times n}$, where $a_{ij} = 1$ if there is a direct edge from node i to node j , and $a_{ij} = 0$ otherwise. Since a transitive closure of G , denoted as G^* , is also a graph it can be represented by its adjacency matrix $A^* = (a_{ij}^*)_{n \times n}$. Therefore to compute transitive closure of G it is enough to find A^* from A (see [1]). The best known sequential algorithm for computing transitive closure is Warshall's. Given in Pascal-like notation Warshall's algorithm has the following form

Algorithm_1

for k:=1 **to** n **do**

for i:=1 **to** n **do**

for j:=1 **to** n **do**

$$a_{ij}^{(k)} := a_{ij}^{(k-1)} \vee (a_{ik}^{(k-1)} \wedge a_{kj}^{(k-1)}).$$

From geometrical point of view this algorithm is a three-dimensional. The corresponding data dependency graph of this algorithm is irregular and it is not convenient for SA synthesis. However, we can observe this algorithm as a series of two-dimensional computations where a

sequence of matrices $A_{ij}^{(0)} = (a_{ij}^{(0)})$, $A_{ij}^{(1)} = (a_{ij}^{(1)})$, ..., $A_{ij}^{(n)} = (a_{ij}^{(n)})$ is computed, where $A^{(0)} = A$ and $A^* = A^{(n)}$. Computational complexity of these two-dimensional entities is equal for each $A_{ij}^{(k)} = (a_{ij}^{(k)})$, $k=1,2,\dots,n$. Also, the computation of $A_{ij}^{(k)} = (a_{ij}^{(k)})$ depends only on $A_{ij}^{(k-1)} = (a_{ij}^{(k-1)})$. These facts indicate that it is enough to design a systolic array that computes $A^{(1)}$ according to $A^{(0)}$. The final result can be obtained by repeating the computation n times on the designed array such that result of the $(k-1)$ -step is used as an input in the k -th step. To follow this idea instead of Algorithm_1 we use the following algorithm

Algorithm_2

for $i:=1$ **to** n **do**
for $j:=1$ **to** n **do**
 $a_{ij}^{(1)} := a_{ij}^{(0)} \vee (a_{ik}^{(0)} \wedge a_{kj}^{(0)})$.

However this is not a systolic algorithm since its data dependency graph has global dependencies. Therefore we have to systolize it. To ease the presentation we introduce the following notation

$$\begin{aligned} c(i, j, 1) &= a_{ij}^{(1)}, c(i, j, 0) = a_{ij}^{(0)}, \\ d(i, 0, 1) &= a_{i1}^{(0)}, b(0, j, 1) = a_{1j}^{(0)} \end{aligned} \quad (1)$$

for each $i=1,2,\dots,n$ and $j=1,2,\dots,n$. Having this in mind, the systolic representation of Algorithm_2 has the following form

Algorithm_3

for $i:=1$ **to** n **do**
for $j:=1$ **to** n **do**
 $d(i, j, 1) := d(i, j-1, 1)$
 $b(i, j, 1) := b(i-1, j, 1)$
 $c(i, j, 1) := c(i, j, 0) \vee (d(i, j, 1) \wedge b(i, j, 1))$

We will call this algorithm the basic systolic algorithm. The corresponding data dependency graph is $\Gamma^{(1)} = (P_{int}^{(1)}, D)$, where $P_{int}^{(1)}$ represents a set of nodes and D set of edges. Actually, a set of nodes corresponds to a set of index points where the computation of Algorithm_3 is taking place, so called inner computation space, defined as

$$P_{int}^{(1)} = \{(i, j, 1) | 1 \leq i \leq n, 1 \leq j \leq n\} \quad (2)$$

while D is data dependency matrix which consists of a set of constant dependency vectors

$$D = \begin{bmatrix} \vec{e}_b^3 & \vec{e}_d^3 & \vec{e}_c^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

each of them representing a data dependency corresponding to the one of the variables b , d and c , respectively.

The initial computation space of Algorithm_3, $P_{in}^{(1)} = P_{in}^{(1)}(d) \cup P_{in}^{(1)}(b) \cup P_{in}^{(1)}(c)$, placed at the border of space $P_{int}^{(1)}$, is defined by

$$\begin{aligned} P_{in}^{(1)}(d) &= \{(i, 0, 1) | 1 \leq i \leq n\}, \\ P_{in}^{(1)}(b) &= \{(0, j, 1) | 1 \leq j \leq n\}, \\ P_{in}^{(1)}(c) &= \{(i, j, 0) | 1 \leq i \leq n, 1 \leq j \leq n\}. \end{aligned} \quad (4)$$

An example of graph $\Gamma^{(1)} = (P_{int}^{(1)}, D)$ and $P_{in}^{(1)}$ for the case $n=3$ is presented in Fig.1.

The SA that implements Algorithm_3 is obtained by mapping graph $\Gamma^{(1)} = (P_{int}^{(1)}, D)$ into (x,y) -plane. Data schedule at the beginning of the computation on the obtained SA is obtained by the same mapping of $P_{in}^{(1)}$ into (x,y) -plane. This mapping directly depends on the chosen projection direction vectors. The next section describes how to determine all allowable projection direction vectors.

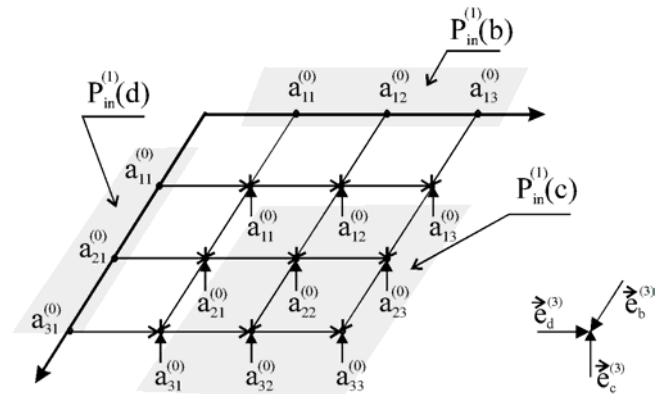


Fig. 1. Data dependency graph $\Gamma^{(1)}$ and inner computation space $P_{in}^{(1)}$ for the case $n=3$

DETERMINING PROJECTION DIRECTION VECTORS

The projection direction vectors are of the form $\vec{\mu} = [\mu_1 \ \mu_2 \ \mu_3]^T$. Since space $P_{int}^{(1)}$ is placed in the plane $k=1$ and we want to design 1D SA, μ_3 has to be zero. Further, the interconnection pattern between the processing elements (PE) in the SA must be of near-neighbor type. Therefore, $\mu_i \in \{-1, 0, 1\}$, $i=1, 2$. Finally, having in mind that projection directions vectors $\vec{\mu}$ and $-\vec{\mu}$ give the same SA, we conclude that possible projection direction vectors are $\vec{\mu} = [1 \ 0 \ 0]^T$, $\vec{\mu} = [0 \ 1 \ 0]^T$,

$\bar{\mu} = [1 \ 1 \ 0]^T$ and $\bar{\mu} = [1 \ -1 \ 0]^T$. Since the direction $\bar{\mu} = [1 \ -1 \ 0]^T$ introduces some irregularity in the SA design (see, for example [2, 6]) and that arrays are out of scope of our interest, we will consider the first three directions as allowable.

Denote by SA1, SA2 and SA3 systolic arrays obtained according to directions $\bar{\mu} = [1 \ 0 \ 0]^T$, $\bar{\mu} = [0 \ 1 \ 0]^T$ and $\bar{\mu} = [1 \ 1 \ 0]^T$, respectively. In the next section we will describe the synthesis of these arrays.

SA SYNTHESIS

The array SA1

Mapping T that maps graph $\Gamma^{(1)} = (P_{\text{int}}^{(1)}, D)$ into SA is called a valid transformation matrix. It is determined for each allowable projection direction separately. Matrix T is not uniquely defined for a given projection direction. More about the criteria for determining valid transformation matrices one can find in [7,8]. One of the possible transformation matrices for the direction $\bar{\mu} = [1 \ 0 \ 0]^T$ is

$$T_1 = \begin{bmatrix} \bar{\Pi} \\ S_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The (x,y)-coordinates of the PEs in the projection plane are obtained according to mapping

$$S_1 : P_{\text{int}}^{(1)} \rightarrow \hat{P}_{\text{int}}^{(1)}, \hat{P}_{\text{int}}^{(1)} = \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \right\}, \text{ i.e.}$$

$$PE \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = S_1 \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} = \begin{bmatrix} j \\ 1 \end{bmatrix}, \quad j = 1, 2, \dots, n.$$

The communication links between the PEs are implemented along the projection of data dependency vectors obtained according to mapping $S_1 : D \rightarrow \Delta$, i.e.

$$\Delta = S_1 \cdot D = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To obtain data schedule at the beginning of the computation in the SA that preserves the correctness of the computation, the initial computation space, $P_{\text{in}}^{(1)}$, has to be rearranged before mapping S_1 is applied. The rearrangement is performed by the timing function $t(\bar{p})$, $\bar{p} \in P_{\text{in}}^{(1)} \cup P_{\text{int}}^{(1)}$ which is defined as

$$t(\bar{p}) = \bar{\Pi}^T \bar{p} + \alpha,$$

where α is a constant determined from the condition that the first computation is performed

in the point $\bar{p} = [1 \ 1 \ 1]^T$, $\bar{p} \in P_{\text{int}}^{(1)}$. The rearrangement of $P_{\text{in}}^{(1)}$ is performed according to the following equality

$$\bar{p}_\chi := \bar{p}_\chi - (t(\bar{p}_\chi) + 1)\bar{e}_\chi^3,$$

for each $\chi \in \{d, b, c\}$. In our case we have that $t(\bar{p}_d) = i - 2$, $t(\bar{p}_b) = j - 2$ and $t(\bar{p}_c) = i + j - 3$. Consequently, the rearranged initial computation space

$$\hat{P}_{\text{in}}^{(1)} = \hat{P}_{\text{in}}^{(1)}(d) \cup \hat{P}_{\text{in}}^{(1)}(b) \cup \hat{P}_{\text{in}}^{(1)}(c)$$

is defined as

$$\hat{P}_{\text{in}}^{(1)}(d) = \{(i, 1 - i, 1) \mid 1 \leq i \leq n\},$$

$$\hat{P}_{\text{in}}^{(1)}(b) = \{(1 - j, j, 1) \mid 1 \leq j \leq n\},$$

$$\hat{P}_{\text{in}}^{(1)}(c) = \{(i, j, 2 - i - j) \mid 1 \leq i \leq n, 1 \leq j \leq n\}.$$

Finally, initial data schedule at the beginning of the computation in SA1 is determined according to the following formulas

$$d(i, 0, 1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_d = \begin{bmatrix} 1 - i \\ 1 \end{bmatrix} + r_1 n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$b(0, j, 1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} j \\ 1 \end{bmatrix} + r_1 n \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$c(i, j, 0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} j \\ 2 - i - j \end{bmatrix} + r_1 n \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where r_1 is determined for all pairs (i, j) as greater of the integers from the set $\{0, 1\}$ such that the following is valid

$$2 - i - j + r_1 n \leq 0 \quad (5)$$

for each $i=1, 2, \dots, n$ and $j=1, 2, \dots, n$. The parameter r_1 is used to minimize the execution time (see [6,8,9]).

An example of the array SA1 for the case $n=3$ as well as functional properties of the PE are given in Fig.2.

The array SA2

Since the design procedure is the same as for the array SA1 we will immediately give the formulas for the synthesis of SA2. The (x,y) positions of the PEs in the SA2 and initial data schedule at the beginning of the computation are determined by the following formulas:

$$PE \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} i \\ 1 \end{bmatrix}$$

$$d(i, 0, 1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_d = \begin{bmatrix} i \\ 1 \end{bmatrix} + r_1 n \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$b(0, j, 1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 1 - j \\ 1 \end{bmatrix} + r_1 n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$c(i, j, 0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} i \\ 2 - i - j \end{bmatrix} + r_1 n \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

for each $i=1,2,\dots,n$ and $j=1,2,\dots,n$. An example of the array SA2 for the case $n=3$ and functional properties of the PE are given in Fig.3.

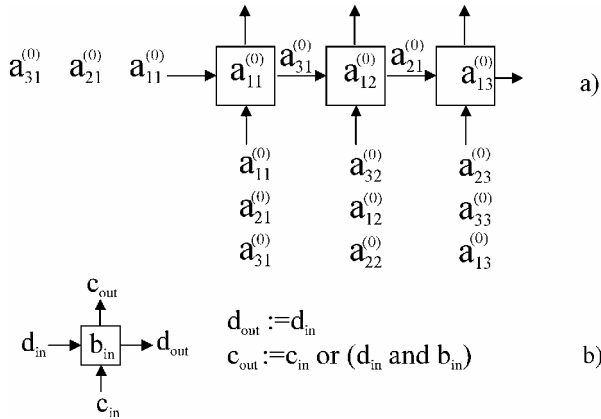


Fig. 2. a) The array SA1 for $n=3$. b) Functional property of PE

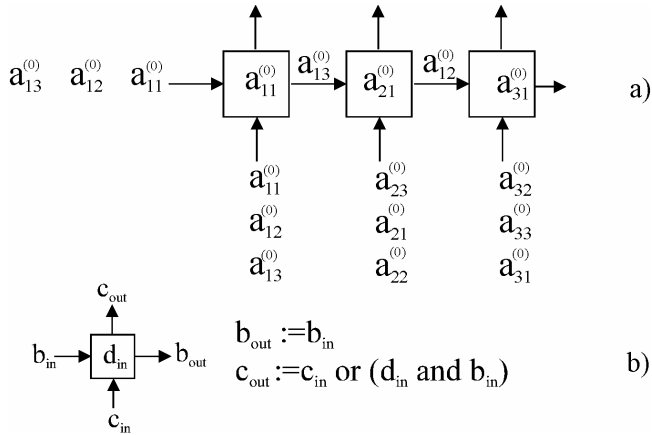


Fig. 3. a) The array SA2 for $n=3$. b) Functional property of PE

The array SA3

The (x,y) positions of the PEs in the SA3 and initial data schedule at the beginning of the computation are determined by the following formulas:

$$PE \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} j-1 \\ 1 \end{bmatrix}$$

$$d(i,0,1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_d = \begin{bmatrix} 1-2i \\ 1 \end{bmatrix} + (r_2 + r_3)\bar{n} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$b(0,i+j-1,1) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_b = \begin{bmatrix} 2i+2j-3 \\ 1 \end{bmatrix} + (r_2 + r_3)\bar{n} \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

$$c(i,i+j-1,0) \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}_c = \begin{bmatrix} j-1 \\ 3-2i-j \end{bmatrix} + (r_2 + r_3)\bar{n} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

for each $i=1,2,\dots,n$ and $j=1,2,\dots,n$. Here we assume the following periodicity of data elements:

$$b(0,j+n,1) \equiv b(0,j,1), \quad c(i,j+n,0) \equiv c(i,j,0)$$

and. The parameters \bar{n}, r_2 and r_3 are used to minimize the execution time of Algorithm_3 on SA3. The constant \bar{n} is determined as

$$\bar{n} = \begin{cases} n, & \text{if } n \text{ is odd} \\ n+1, & \text{if } n \text{ is even} \end{cases}$$

The parameters r_2 and r_3 are determined for all pairs (i, j) as greater of the integers from the set $\{0,1\}$ such that the following is satisfied

$$\begin{aligned} -2(i-1) + r_2\bar{n} &< 0, \quad i=1 \Rightarrow r_2 = 0 \\ -2(i-1) - (j-1) + (r_2 + r_3)\bar{n} &\leq 0 \end{aligned}$$

Parameter r_2 is determined first. An example of the array SA3 for the case $n=3$ and functional properties of the PE are given in Fig.4.

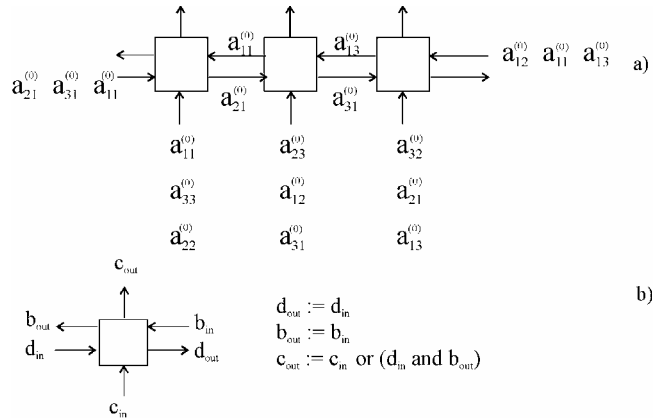


Fig. 4. a) The array SA3 for $n=3$. b) Functional property of PE

PERFORMANCE ANALYSIS

Various performance parameters can be used to measure the features of the synthesized SAs. Here we will use: number of processing elements, Ω , initialization time, t_{in} , execution time, t_{exe} , output time, t_{out} , during the computation of $A^{(k)} = (a_{ij}^{(k)})$ for some fixed k , total execution time to compute transitive closure of a given graph, T_{tot} , and the efficiency, E . The obtained results are summarized in Table 1.

SA	Ω	t_{in}	t_{exe}	t_{out}	T_{tot}	E
SA1	n	$n-1$	n	$n-1$	$2n^2 - 1$	$\approx 1/2$
SA2	n	$n-1$	n	$n-1$	$2n^2 - 1$	$\approx 1/2$
SA3	n	$n-1$	\bar{n}	$n-1$	$n(n+\bar{n}) - 1$	$\approx 1/2$

Table 1. Performance of the designed arrays

As can be seen from Tab. 1 performances of the designed arrays are almost identical. All arrays have optimal number of PEs for a given problem size n . However the arrays SA1 and SA2 are static arrays in the sense that some data

elements are resident in the array, while SA3 is fully pipelined. Another important property of SA3 is its suitability for fault-tolerant computations. Namely with low hardware overhead this array can detect and correct single permanent or transient errors during the computation.

CONCLUSION

We have discussed the problem of computing transitive closure of a given directed graph on linear systolic arrays. We have designed three linear regular arrays with optimal number of PEs for a given problem size. The execution time of these arrays is minimal possible for a given number of PEs.

REFERENCES

- [1] S. Warshall, A Theorem on Boolean Matrices, J. ACM, 9 (1962), 11-12.
- [2] I. Ž. Milovanović, E. I. Milovanović, A problem of selecting mathematical method for systolic processing, Proc.: International Scientific Conference UNITECH'03, Tehn. Univ. Gabrovo, Gabrovo, Bulgaria, 2003, 277-279.
- [3] S. G. Sedukhin, The designing and analysis of systolic algorithms and structures, Programming, 2 (1990), 20-40. (In Russian)
- [4] H. T. Kung, Why systolic architectures, Computer, 15 (1982), 37-46.
- [5] K. T. Johnson, A. R. Hurson, B. Shirazi, General-purpose systolic arrays, Computer, 11 (1993), 20-31.
- [6] I. Ž. Milovanović, E. I. Milovanović, I. Z. Milentijević, M. K. Stojčev, Designing of processor-time optimal systolic arrays for band matrix-vector multiplication, Comput. Math. Appl., Vol 32, 2 (1996), 21-31.
- [7] M. Bekakos, E. Milovanović, N. Stojanović, T. Tokić, I. Milovanović, I. Milentijević, Transformation matrices for systolic array synthesis, J. Electrotechn. Math., Vol 7, 1 (2002), 9-15.
- [8] T. I. Tokić, I. Ž. Milovanović, D. M. Randjelović, E. I. Milovanović, Determining VLSI array size for one class of nested loop algorithms, In: Advances in Computer and Information Sciences (U. Gudukbay, T. Dagar, A. Gursay, E. Gelembek, eds.) IDS Press, Amsterdam, 1998, 389-396.
- [9] M. P. Bekakos, I. Ž. Milovanović, E. I. Milovanović, T. I. Tokić, M. K. Stojčev, Hexagonal systolic arrays for matrix multiplication, In: Highly Parallel Computations: Algorithms and Applications, (M. P. Bekakos, ed.) WITpress, Southampton-Boston, 2001, 175-209.
- [10] E. Milovanović, B. Randjelović, I. Milovanović, N. Novaković, Systolic array synthesis under predefined constraints, J. Electrotechn. Math., Vol 8, 1 (2003), 31-38.