SOLUTION OF DENSE SYSTEMS OF LINEAR EQUATIONS USING A FIXED SIZE SYSTOLIC ARRAY

A. Bojańczyk

1. INTRODUCTION

We consider the problem of parallel triangularization of a dense matrix of dimension $n \times m$, $n \leq m$. Matrix triangularization is a crucial step in solving dense systems of linear equations or dense linear least squares problems. Thus being able to accomplish such a transformation is very important in practice. However algorithms for matrix triangularization are computationally expensive. Hence there is a good reason to use a special purpose (VLSI) device for speeding-up such computations. Usually these devices implement systolic algorithms (Kung [4]) derived from classical direct methods. Two of the methods are particularly well-suited for our purpose. These are Gaussian elimination and QR decomposition by Givens rotations (see Kung and Leiserson [5], Bojańczyk, Brent and Kung [2], Gentleman and Kung [3]).

This paper presents a unified treatment of a systolic implementation of both Gaussian elimination with neighbour pivoting and QR decomposition. We also derive a scheme of splitting the matrix of coefficients in case a matrix is so large that it requires more processors than a given systolic array provides. Finally, we show how to handle banded matrices with $O(\text{bandwidth})^2$ processors.

2. PARALLEL TRIANGULARIZATION

Given an $n \times m$ matrix A of rank n , we want to determine an $n \times m$

nonsingular matrix T and an upper trapezoidal matrix R such that

$$TA = R .$$

The transformation T is chosen as a product of "simpler" matrices T_{ij}, $1 \le j < i \le n$. The matrix T_{ij} applied on the left replaces the i-th row of A with some linear combination of itself and the j-th row so that the j-th element of row i becomes zero. (Simultaneously row j is replaced by a linear combination of itself and the i-th row.)

Some of the transformations T_{ij} are independent so it is possible to apply more than one at a time. A direct sum of transformations T_{ij} applied simultaneously will be referred to as parallel sweep. There are many possibilities but any algorithm triangularizing the matrix A should satisfy the condition that none of the created zeros are replaced by non-zeros in the process of triangularization. This is surely satisfied if we impose the following precedence constraint: transformation T_{ik} cannot be initiated before transformations T_{ij} are completed for all $j < k$. In the class of schemes that obey such a precedence constraint we propose a scheme that requires $N = 3n-5$ parallel sweeps. This is illustrated for $n=8$ in Figure 1 where the number indicates the sweep on which that element is zeroed (cf. Bojańczyk, Brent and Kung [2]). Each sweep T_k is a direct sum of transformations T_{ij} where $(i,j) \in L_k$ and the set of pairs of indices L_k, $k = 1,\ldots,N$ is defined by

$$(i,j) \in L_k \quad \text{iff} \quad i+2(j-1)-1 = k , \quad 1 \le j < i \le n .$$

From the definition of L_k it follows that if $(i,j) \in L_k$ then $(i+2,j-1)$ and $(i-2,j+1)$ also belong to L_k provided they are in $\{(i,j) : 1 \le j < i \le n\}$.

Before we define the transformations T_{ij} in detail we describe how these transformations handle degenerate cases, i.e. $a_{ij} = 0$ or $a_{jj} = 0$.

x							
1	x						
2	4	x					
3	5	7	x				
4	6	8	10	x			
5	7	9	11	13	x		
6	8	10	12	14	16	x	
7	9	11	13	15	17	19	x

Figure 1.

We shall assume that if the element a_{ij} is zero then the transformation T_{ij} is defined to be the identity transformation. In the case that $a_{ij} \neq 0$ but $a_{jj} = 0$ then the transformation T_{ij} is the permutation matrix that interchanges rows j and i.

Transformation T_{ij} may be chosen as a Givens rotation. More precisely

$$T_{ij} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & c_i & s_i & \\ & & & -s_i & c_i & \\ & & & & & 1 \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ j \\ i \\ \\ \\ \\ \end{matrix}$$

where the parameters of T_{ij} are defined by

$$\bar{a}_{jj} = (a_{jj}^2 + a_{ij}^2)^{\frac{1}{2}}$$

$$c_i = a_{jj}/\bar{a}_{jj}$$

$$s_i = a_{ij}/\bar{a}_{jj} .$$

Rows j and i of the product $\bar{A} = T_{ij}A$ are given by

$$\bar{a}_{jp} = c_i a_{jp} + s_i a_{ip} , \quad p = 1,\ldots,m ,$$

$$\bar{a}_{ij} = 0 ,$$

$$\bar{a}_{ip} = -s_i a_{jp} + c_i a_{ip} , \quad p \neq j .$$

The matrix $T = T_N \ldots T_1$ formed as the product of the parallel rotations T_k, is an orthogonal matrix.

Triangularization of the matrix A may also be performed by Gaussian elimination. (For the time being we assume that pivoting is not necessary. The pivoting problem is dealt with in section 4). In this case matrices T_{ij} are defined by

$$T_{ij} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ell_{ij} & \\ & & & 1 \end{bmatrix} \begin{matrix} \\ j \\ i \\ \\ \end{matrix}$$

where

$$\ell_{ij} = -a_{ij}/a_{jj} .$$

Row j of the product $\bar{A} = T_{ij}A$ remains unchanged while row i is given by

$$\bar{a}_{ip} = a_{ip} + \ell_{ij}a_{jp} , \quad p \neq j .$$

$$\bar{a}_{ij} = 0 .$$

Now, the matrix T formed as the product of parallel sweeps T_k, namely $T = T_N \ldots T_1$, is a lower triangular matrix.

The transformation T_{ij} may change only rows j and i of the matrix A. Rows j and i are referred to as the *pivot* and *current* rows of matrix A, respectively.

3. SYSTOLIC IMPLEMENTATION

In this section we present a systolic array which efficiently implements the triangularization algorithms described in the previous section. This is done similarly in Bojańczyk, Brent and Kung [2].

Our systolic array is made up of relatively simple processing units, all of the same type, connected together by a rectangular network. The data and control flow of the algorithm is simple and regular and the algorithm extensively uses pipelining and multiprocessing.

3.1 The basic processing unit.

Two kinds of operations are required to perform the transformation T_{ij} : determination of linear combination parameters and formation of the linear combination itself. The latter operation is called a combination step.

Thus we need a processor which is able to perform both operations. We also demand that the processor be able to handle degenerate cases as well as other simple control operations. This is discussed later on. In addition to its computing capabilities the processor should have outside connections for receiving and delivering data, and several registers for storing and processing intermediate results. The main four connections are two inputs and two outputs (depicted in Figure 2 by X_{in}, Y_{in}, X_{out}, Y_{out}) .

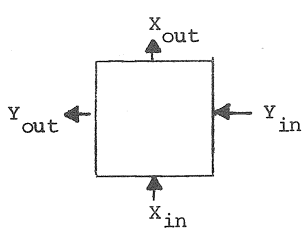


Figure 2.

A pivot row of the matrix A enters the processor through the X line while a current row enters the processor through the Y line. The computation is initiated when the processor shifts data x and y on its input lines X_{in} and Y_{in} into registers R_x and R_y . Then it computes the parameters of linear combination. The computed values of these parameters are stored in

the processor for the later use. The new value $\bar{x}$ is made available as output on the output line X_{out}. (The new value of $\bar{y}$ is not calculated since $\bar{y}$ is known to be zero. Recall that in the degenerate case when y already has value zero the parameters are chosen in such a way that the transformation is simply the identity transformation. When y is different from zero but x *is* equal to zero the transformation only interchanges rows x and y). The parameters of linear combination are determined only once and stay inside the processor during the whole computation. Every subsequent operation performed by the processor is a combination step. The processor shifts data on its input lines X_{in} and Y_{in} into registers R_x and R_y, and combines their content into new values $\bar{x}$ and $\bar{y}$ on the output lines X_{out} and Y_{out}. There is a tag bit which controls switching the processor from one kind of operation to the other.

We define a *time unit* to be the maximal time that is necessary for the processor to determine the parameters of linear combination *or* to perform a combination step together with loading and unloading its registers.

We assume that there is a synchronization mechanism which latches input and output lines. One may choose between self-timed and clocked systems so that when processors are connected together, the changing output of one processor should not interfere with input to another processor.

3.2 Network organization.

The systolic array proposed here is made up of a network of N^2 rectangularly connected processors. The position of each processor in the network is identified by its coordinates (i,k), $1 \leq i,k \leq N$, as in Figure 3. The processor (i,k) is connected to at most four neighbouring processors. Processors with identical first (second) coordinates, i.e., these connected by Y-lines (X-lines), form a Y-channel (X-channel). More precisely, processors with

coordinates (i,1) , (i,2) ,..., (i,N) and (1,j) , (2,j) ,..., (N,j) form Y(i) and X(j) channel respectively, $1 \le i,j \le N$.

The processor (i,k) , $1 \le k < i \le N$, is assigned to perform transformation T_{ik} . The task of the remaining processors, i.e., those with coordinates (i,k) , $1 \le i \le k \le N$, will become clear later on but for the time being they can be thought of as shift registers.

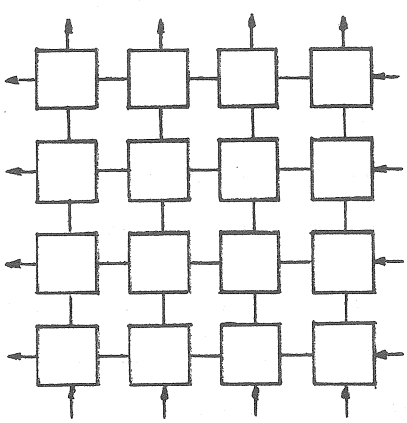


Figure 3.

3.3 Operation of the systolic array.

We consider two cases.

(i) The dimension of the matrix is equal to the size of the systolic array (n=N) .

During the computation the matrix A enters the network through the Y-channels one row after another in a skewed order. Through the X-channels zero vectors are pumped into the network, see Figure 4.

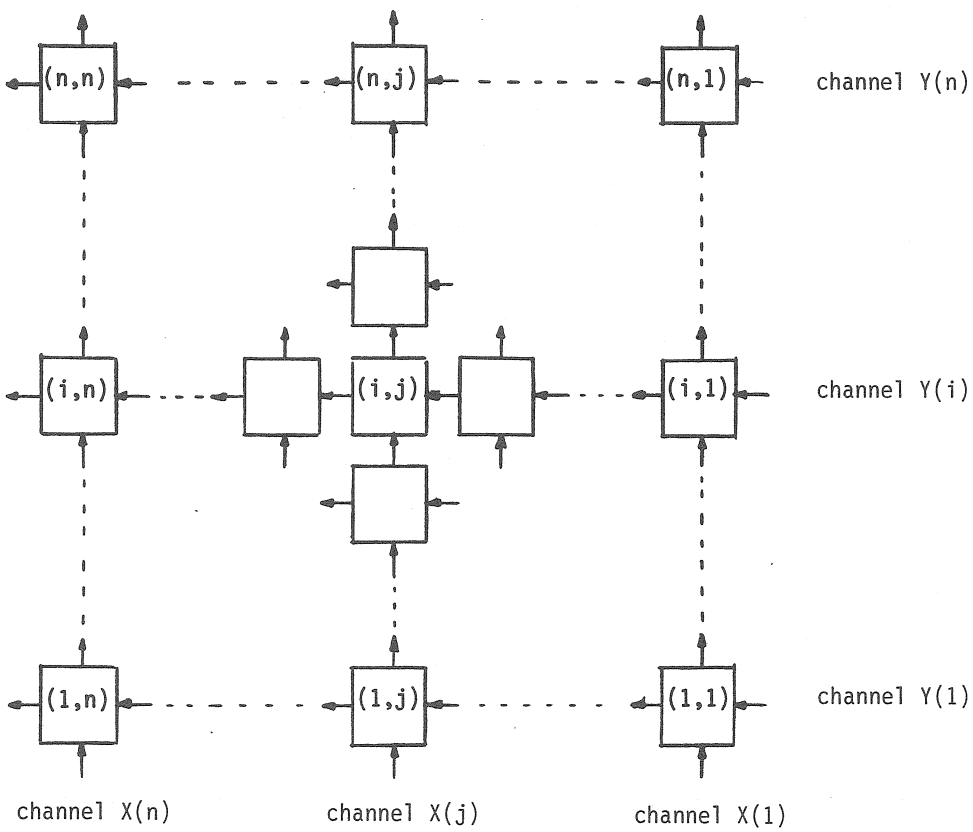


Figure 4.

Processor (i,k) begins its activity at time $t=2(k-1) + (i-1)$. From this formula it follows that if processor (i,k) becomes active at time t then processor (i+1,k) becomes active at time t+1 while processor (i,k+1) becomes active at time t+2 .

From consideration of the degenerate case (section 2) it follows that processors (i,k) , $1 \le i \le k \le N$, play the role of shift registers. They simply delay data transmission so that the system works correctly. Moreover, because of the special choice of transformation in the degenerate case, we see that if row i has its first elements equal to zero it cannot leave the network through channels X(1) , X(2) ,..., X(n) .

It follows from the schedule of the output processors, i.e., processors (n,1) ,..., (n,n) , that transformed coefficients $a_{ij}^{(i-1)}$ (the superscript of a matrix coefficient indicates the number of transformations T_{ik} in which this coefficient is involved), $1 \le i \le j \le m$, leave the network through the channel X(i) at time $t=2(i-1) + (n-1) + (j-i+1)$ where m is the length of rows. Output processors (j,n) , j=1,...,n , produce only zeros as their output (they play an important role in the case when $n > N$) .

We summarize this subsection in the following lemma (compare with Bojańczyk, Brent and Kung [2] and Gentelman and Kung [3]).

LEMMA: *Transformation of an* $n \times m$ *matrix , into an upper trapezoidal form requires*

$$t = 2n + m - 2$$

units of time using a network of n^2 *rectangularly connected processors.*

■

(ii) The dimension of the matrix A is greater than the size of the systolic array $(n > N)$.

Let us partition the matrix A into strips of width N as shown in Figure 5.

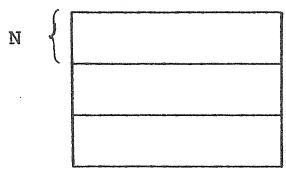
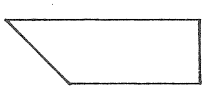


Figure 5.

Triangularization is achieved in $\lceil n/N \rceil$ cycles. Each cycle consists of triangularization of "pivot" strip and annihilation of block column beneath the "pivot". As all cycles are analogous we describe only the first one.

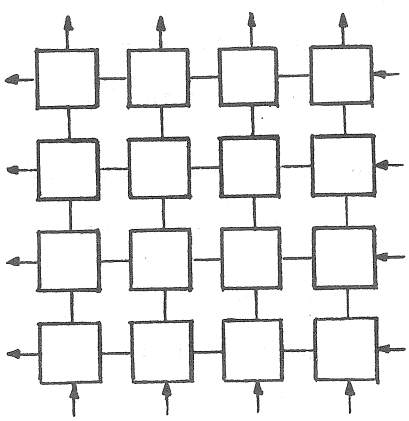
During a cycle strips are transformed by means of a systolic array. Every transition of a strip through the systolic is referred to as a step.

In the first step we feed the first strip into the systolic array through the Y-channels. After transformation the first strip has the following upper trapezoidal form



and is considered as a "pivot" strip. In the second and subsequent steps the "pivot" strip is used to introduce zeroes in the first block column. More precisely, after each step the systolic array is switched to perform a new transformation having as data rows of the "pivot" strip and "current" strip (i.e., the next not transformed strip). Now, rows of the "pivot" strip enter the systolic through X-channels while rows of the "current" strip enter the systolic array through the Y-channels (see Figure 6.) In the phase of the transformation the whole array of

processors performs useful work annihilating coefficients in the first block of the "current" strip. (Timing specification is the same as before, i.e., processor (i,k) begins its activity at time t=2(k-1) + (i-1)). "Pivot" and "current" strips are leaving the network through X and Y-channels respectively.



	t=0	t=1	t=2	t=3	t=4
	.	.	.	a_{81}	a_{82}
	.	.	a_{71}	a_{72}	a_{73}
	.	a_{61}	a_{62}	a_{63}	a_{64}
	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

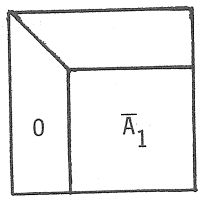
"current" strip

t=0	.	.	.	$\bar{a}_{11}$
t=1	.	.	.	$\bar{a}_{12}$
t=2	.	.	$\bar{a}_{22}$	$\bar{a}_{13}$
t=3	.	.	$\bar{a}_{23}$	$\bar{a}_{14}$
t=4	.	$\bar{a}_{33}$	$\bar{a}_{24}$	$\bar{a}_{15}$
t=5	.	$\bar{a}_{34}$	$\bar{a}_{25}$	$\bar{a}_{16}$
t=6	$\bar{a}_{44}$	$\bar{a}_{35}$	$\bar{a}_{26}$	$\bar{a}_{17}$

"pivot" strip

Figure 6.

After the first cycle is finished the transformed matrix $\bar{A}$ has the following form

$\bar{A}$ = 

Now, in the second cycle, computation is applied to matrix $\bar{A}_1$ and the scheme is repeated in subsequent cycles until the matrix $\bar{A}$ is in upper trapezoidal form.

It is easy to check that when applied to an $n \times m$ matrix, $n \leq m$, our algorithm requires order

$$\frac{(n+N)n}{N} \quad (1 + \frac{m}{2N})$$

units of time.

Note that our design is modular and extensible. A larger device can be obtained by combining several smaller ones provided they form a rectangle. This is very important since a systolic array typically receives data and delivers results through an attached host. Here the host can mean another special purpose device, a memory, general purpose computer etc. Thus an I/O interface can greatly influence overall performance. Hence, it is necessary to be able to easily adjust the size of the systolic array to balance the available I/O bandwidth with the host. This is surely possible with our design.

4. NEIGHBOUR PIVOTING

In section 2 we assumed that Gaussian elimination can be performed for the matrix A without pivoting. In general, partial or complete pivoting is necessary. But classical forms of pivoting are very inefficient on rectangularly connected processors. Another pivoting strategy, called neighbour pivoting, often referred to as the Bartels-Golub scheme for LU factorization, see

Bartels and Golub [1], can be easily incorporated into the systolic array considered in section 3 (this is done as in Gentelman and Kung [3]. Basically this technique interchanges "pivot" and "current" rows as they enter the processor so as to prevent the multipliers from exceeding unity. The "pivot" and "current" rows are neighbouring rows in the sense that they are processed by the same processor. Such a pivot selection does not require global communication and hence is well-suited for systolic computation. On the other hand since the multipliers are fractional then the growth of elements is not excessive. An upper bound on element growth with neighbour pivoting is 2^n and can be attained only for very artificial cases (see Wilkinson [6]). Practical experiments show that it is extremely rare for elements to grow faster than two orders of magnitude.

In order to implement the neighbour pivoting strategy we require that in addition to calculating multipliers the processors should be able to recognize the necessity of row interchange. We also would like to have the possibility of performing Gaussian elimination without *any* form of pivoting. This is helpful when the matrix we deal with is symmetric positive definite, irreducible, or diagonally dominant matrix: for such classes of matrices we can safely perform Gaussian elimination without pivoting. The choice between pivoting and non-pivoting strategies may be signalled by a tag bit which is sent from one processor to another simultaneously with the data vector.

5. APPLICATIONS

Systolic triangularization arrays are very helpful for solving linear systems

$$Ax = b \ .$$

The solution of a linear system is obtained in two steps. First, the linear system is transformed to the triangular one

$$Rx = \bar{b} \ ,$$

by applying a triangularization scheme to the augmented matrix $\hat{A} = [A,b]$. Second, a back substitution process is performed by a linear systolic array introduced by Kung and Leiserson [5]. These two systolic arrays together form a powerful system capable of solving any nonsingular system of linear equations.

The QR decomposition is a successful way for solving the linear least squares problem. In the linear least squares problem we deal with a rectangular $n \times m$ matrix A , $m \leq n$. By decomposing the matrix A into blocks of smaller dimension we obtain subproblems that can be treated efficiently by our systolic array.

In many applications we want to solve a banded linear system. Typically, the bandwidth p is much smaller than the dimension of the matrix. Thus it would be desirable to have a systolic array which handles band matrices efficiently, i.e., whose size depends on the bandwidth of the matrix rather than on the dimension of the matrix. It turns out that triangularization of a band matrix can be easily achieved through a minor modification of computing organization leaving unmodified the systolic array itself.

The general idea is to represent a band matrix as a block tridiagonal matrix,

$$A = \begin{bmatrix} A_{11} & A_{12} & & \\ A_{21} & A_{22} & A_{23} & \\ & \ddots & \ddots & \ddots \end{bmatrix} .$$

Every block row constitutes a strip of width equal to the size of systolic p and length or order $3p$. Thus, we can proceed exactly in the same way as shown in section 3.3 case (ii). Observe that after the first block column

has been annihilated, the transformed matrix A has the following form

$$\bar{A} = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} & \bar{A}_{13} & & \\ & \bar{A}_{22} & \bar{A}_{23} & & \\ & A_{32} & A_{33} & A_{34} & \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

This form indicates how to handle the right hand side vector b. Let us partition the vector b consistent with the decomposition of the matrix A, so $b^T = [b_1^T, \ldots, b^T_{\lceil n/p \rceil}]$. One can consider the vector $[b_1^T, b_2^T]^T$ as the last column of the $2p \times (3p+1)$ matrix A_1 where

$$A_1 = \begin{array}{|c|c|c|c|} \hline A_{11} & A_{12} & & b_1 \\ \hline A_{21} & A_{22} & A_{23} & b_2 \\ \hline \end{array}$$

We apply the triangularization scheme to the matrix A_1. After the first cycle of the triangularization is finished, the vector $\bar{b}_1$ will not be modified and can be discarded from our consideration. The second cycle is applied to the $2p \times (3p+1)$ matrix A_2 where

$$A_2 = \begin{array}{|c|c|c|c|} \hline \bar{A}_{22} & \bar{A}_{23} & & \bar{b}_2 \\ \hline A_{32} & A_{33} & A_{34} & b_3 \\ \hline \end{array}$$

and the scheme is repeated until the whole matrix $\hat{A} = [A,b]$ is triangularized.

It is easy to see that the triangularization of the matrix $\hat{A} = [A, b_1, \ldots, b_k]$ can be accomplished in

$$t = (10p - 6 + 2k)\lceil n/p \rceil$$

units of time, where p is the bandwidth of the matrix A.

REFERENCES

[1] Bartels, R.H. and Golub, G.H., "The Simplex method for linear programming using LU decomposition", *Communications of ACM* 12 (1969) 266-268.

[2] Bojanczyk, A., Brent, R.P. and Kung, H.T., "Numerically stable solution of dense system of linear equations using mesh-connected processors", *SIAM Journal on Scientific and Statistical Computing* 5 (1984) 95-104.

[3] Gentelman, W.M. and Kung, H.T., "Matrix triangularization by systolic arrays", *Proceedings of SPIE Symposium*, vol. 298, *Real-Time Signal Processing* IV, the Society of Photo-optical Instrumentation Engineers, August 1981.

[4] Kung, H.T., "Let's design algorithms for VLSI systems", Tech. Report, Carnegie-Mellon University, Computer Science Department, January 1979.

[5] Kung, H.T. and Leiserson, C.E., "Systolic arrays for VLSI", *Sparse Matrix Proceedings* 1978, Duff, I.S. and Stewart, G.W., eds., Society for Industrial and Applied Mathematics, 1979, pp. 256-282.

[6] Wilkinson, J.H., *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.