JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 13, 111–117 (1991)

Jacobi-Type Algorithms for *LDU* and Cholesky Factorization

MARC MOONEN*,†

ESAT Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001 Heverlee, Belgium

PAUL VAN DOOREN

Philips Research Laboratory, Av. A. Einstein 4, 1348 Louvain-la-Neuve, Belgium

AND

JOOS VANDEWALLE*

ESAT Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001 Heverlee, Belgium

It is demonstrated how conventional algorithms for computing the *LDU* decomposition of a square matrix, or Cholesky factorization for symmetric positive definite matrices, can be reordered into Jacobi-type algorithms. For efficient parallel implementation on a systolic array, the resulting schemes compare favorably with earlier implementations.

INTRODUCTION

In this note it is demonstrated how conventional algorithms for *LU* and *LDU* factorization can be reorganized into Jacobi-type algorithms. First the Jacobi-type algorithm for *LU* decomposition, which largely corresponds to the *QR*D procedure of [4], is derived. This procedure is then transformed into an algorithm for computing the *LDU* decomposition which "preserves symmetry." The latter procedure is therefore suited for computing the Cholesky decomposition as well. The derivation is fairly straightforward, but it has not appeared in the literature before, as far as we know. Up until now, the *LU* and *LDU* factorizations were apparently the only matrix factorizations for which there was no Jacobi-type algorithm available. The advantage of having this type of algorithm available for any matrix decomposition is that then only one type of architecture is needed, which can be used for any operation, where each time only the cell functionality is slightly adapted. In this respect, our approach provides a useful alternative to earlier approaches for systolic *LU* (*LDU*) decomposition; see [2] and the references therein and [1, 3, 5].

* The work of these authors was sponsored in part by the BRA 3280 Project of the European Commission.

† Senior research assistant with the N.F.W.O. (Belgian National Fund for Scientific Research).

LU DECOMPOSITION

The *LU* decomposition is defined as

$$A = L \cdot U,$$

where an $n \times n$ matrix is decomposed as a product of a unit lower and an upper triangular matrix (L resp. U). We refer to [2] for standard algorithms to compute this decomposition. Our aim here is to derive a Jacobi-type algorithm for this decomposition. Such an algorithm is based on locally computed plane transformations and is particularly suited for parallel implementation, e.g., on a systolic array.

Our main source of inspiration is a similar algorithm for computing the *QR* decomposition [4]. This *QR*D procedure is straightforwardly turned into an algorithm for computing the *LU* decomposition, if we use only appropriate row transformations, i.e., embeddings of unit lower triangular 2×2 matrices instead of plane rotations. Briefly, the original matrix $A_{n\times n}$ is reduced to an upper triangular matrix U by applying a sequence of exactly $n(n-1)$ such transformations L_k^{-1} to the left, together with column permutations Π_k. Accumulating the left-hand transformations delivers the L matrix, while on the other hand, the accumulated permutations equal the identity. This can be cast as (see [4] for details)

$$A = I \cdot A \cdot I = \underbrace{I \cdot L_1 \cdots L_{n(n-1)}}_{L} \times \underbrace{L_{n(n-1)}^{-1} \cdots L_1^{-1} \cdot A \cdot \Pi_1 \cdots \Pi_{n(n-1)}}_{U} \times \underbrace{\Pi_{n(n-1)} \cdots \Pi_1 \cdot I}_{I}.$$

0743-7315/91 $3.00

Here L_k and Π_k differ from the identity only in

$$\{L_k\}_{i,i+1} = \begin{bmatrix} 1 & 0 \\ l_k & 1 \end{bmatrix}$$

$$\{\Pi_k\}_{i,i+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

where $\{M\}_{i,i+1}$ is a 2×2 submatrix of M on the intersection of rows i and $i+1$ and columns i and $i+1$, and l_k is chosen such that the $(i+1, i)$ entry in $L_{k-1}^{l} \cdots L_1^{l} \times A \cdot \Pi_1 \cdots \Pi_{k-1}\Pi_k$ is zeroed. The pivot i follows the *odd–even ordering* of [6], which means that, e.g., for odd values of n, i cycles through

$$i = \underbrace{1, 3, \ldots, n-2}_{\text{odd}}, \underbrace{2, 4, \ldots, n-1}_{\text{even}}$$

exactly n times. As both the odd-numbered transformations and the even-numbered transformations can be carried out in parallel, this algorithm requires roughly $2n$ time steps on a parallel architecture [4].

An algorithmic description is thus as follows. The matrix L initially equals the identity and remains unit lower triangular throughout (the product of lower triangular matrices is lower triangular). The U matrix initially equals A and is gradually reduced to upper triangular form. The zeros in the lower triangular part are introduced in the same fashion as in the R matrix in the QRD algorithm; see Fig. 2 in [4].

Let us now slightly modify this algorithm, such that it can be extended for the LDU case. For a 6×6 matrix, for instance, the pivot index in the above algorithm takes up the following values, where transformations on the same line can be performed in parallel:

$$\begin{array}{l} i = \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{ccccc} 1 & & 3 & & 5 \\ & 2 & & 4 & \\ 1 & & 3 & & 5 \\ & 2 & & 4 & \\ 1 & & 3 & & 5 \\ & 2 & & 4 & \\ 1 & & 3 & & 5 \\ & 2 & & 4 & \\ 1 & & 3 & & 5 \\ & 2 & & 4 & \end{array}$$

Some of these transformations introduce zeros, which are filled in in subsequent steps. Alternatively, one could also perform only those transformations which create zeros that are not filled in afterward. One can check that the algorithm is then reduced to the following:

$$\begin{array}{l} i = \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{ccccc} \cdot & & \cdot & & 5 \\ & \cdot & & 4 & \\ \cdot & & 3 & & 5 \\ & 2 & & 4 & \\ 1 & & 3 & & 5 \\ & 2 & & 4 & \\ \cdot & & 3 & & 5 \\ & \cdot & & 4 & \\ \cdot & & \cdot & & 5 \\ & \cdot & & \cdot & \end{array}$$

This is called a *backward sweep*. Figure 1 illustrates this procedure. The odd–evens are indicated (frames), but only the transformations of the backward sweep (double frames) are actually carried out. Note however that, as half of the column permutations are left out as well, the resulting product $L \cdot U$ equals A up to a column permutation.

$$A = I \cdot A \cdot I = \underbrace{I \cdot L_1 \cdots L_{(n/2)(n-1)}}_{L} \times \underbrace{L_{(n/2)(n-1)}^{-1} \cdots L_1^{-1} \cdot A \cdot \Pi_1 \cdots \Pi_{(n/2)(n-1)}}_{U} \cdot \underbrace{\Pi_{(n/2)(n-1)}^{t} \cdots \Pi_1^{t} \cdot I}_{\Pi},$$

where

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

In order to compute an LU decomposition for A, one should therefore start from $A \cdot \Pi$, to end up with

$$L \cdot U = (A \cdot \Pi) \cdot \Pi = A$$

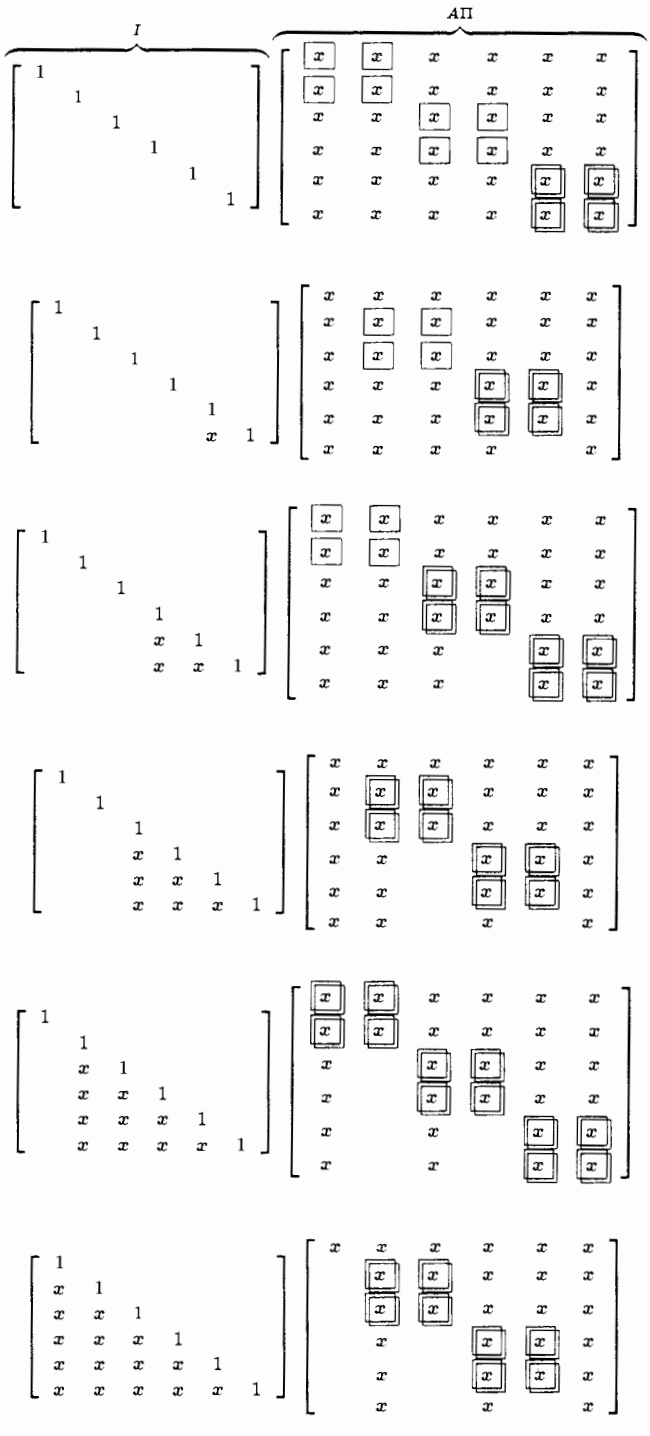

FIG. 1. *LU* decomposition.

As for the *LU* algorithm, we are now clearly complicating things. However, in the *LDU* algorithm of the next section, the redundant transformations *need* be left out, as otherwise there will be fill-ins in the upper (lower) triangular part of L (U).

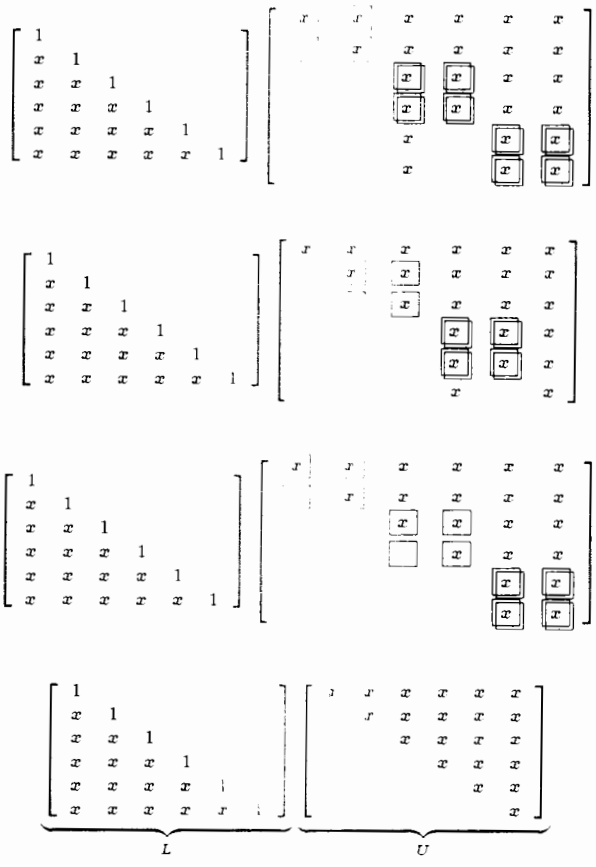

FIG. 1—*Continued*

LDU AND CHOLESKY DECOMPOSITION

The *LDU* decomposition is defined as

$$A = L \cdot D \cdot U,$$

where D is a diagonal matrix and L and U are unit upper and unit lower triangular matrices. For symmetric matrices $A = A^t$, it follows that $U = L^t$ in the *LDU* decomposition. Furthermore, for symmetric positive definite matrices, the matrix D has positive diagonal entries and $L \cdot D^{1/2}$ is referred to as the *Cholesky factor*.

The *LDU* decomposition of A—when it exists—can be computed from the *LU* decomposition, simply by scaling the U to a unit upper triangular matrix. However, for the special case where A is symmetric the *LU* algorithm destroys symmetry after one step. Let us therefore try to derive a true *LDU* algorithm, which furthermore preserves symmetry throughout. Initially, D could be set equal to A, while L and U equal the identity. The aim is then to reduce D to diagonal form by applying *row and column transformations*. Accumulating these transformations should then deliver L and U.

As for the reduction of D to diagonal form, the row and column transformations L_k and U_k to be applied should at

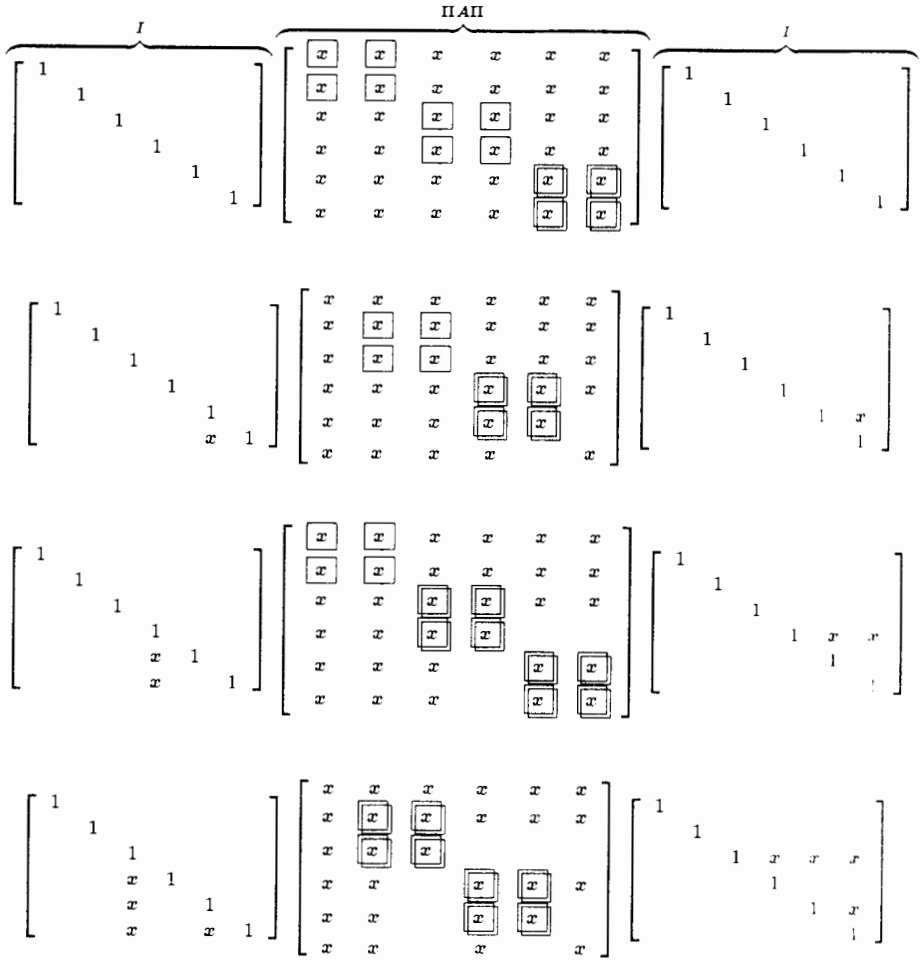

FIG. 2. *LDU* decomposition.

first sight correspond to 2×2 *LDU*s on the main diagonal as

$$\begin{bmatrix} D_{i,i} & 0 \\ 0 & D_{i+1,i+1} \end{bmatrix} \leftarrow \underbrace{\begin{bmatrix} 1 & 0 \\ \dfrac{D_{i+1,i}}{D_{i,i}} & 1 \end{bmatrix}^{-1}}_{\{L_k\}_{i,i+1}^{-1}} \cdot \begin{bmatrix} D_{i,i} & D_{i,i+1} \\ D_{i+1,i} & D_{i+1,i+1} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & \dfrac{D_{i,i+1}}{D_{i,i}} \\ 0 & 1 \end{bmatrix}^{-1}}_{\{U_k\}_{i,i+1}^{-1}}.$$

However, additional permutations should be included, so that all off-diagonal elements can be annihilated. Each iteration can then be cast as

$$D \leftarrow L_k^{-1} \cdot \Pi_k \cdot D \cdot \Pi_k \cdot U_k^{-1}$$

with

$$\{L_k\}_{i,i+1} \leftarrow \begin{bmatrix} 1 & 0 \\ D_{\cdot,\cdot} & 1 \\ D_{\cdot,\cdot} & \end{bmatrix}$$

$$\{U_k\}_{i,i+1} \leftarrow \begin{bmatrix} 1 & D_{\cdot,\cdot} \\ & D_{\cdot,\cdot} \\ 0 & 1 \end{bmatrix}.$$

In view of efficient implementation, one should introduce additional permutations for the accumulation of row and column transformations as well, i.e.,

$$L \leftarrow \Pi_k \cdot L \cdot \Pi_k \cdot L_k$$

$$U \leftarrow U_k \cdot \Pi_k \cdot U \cdot \Pi_k.$$

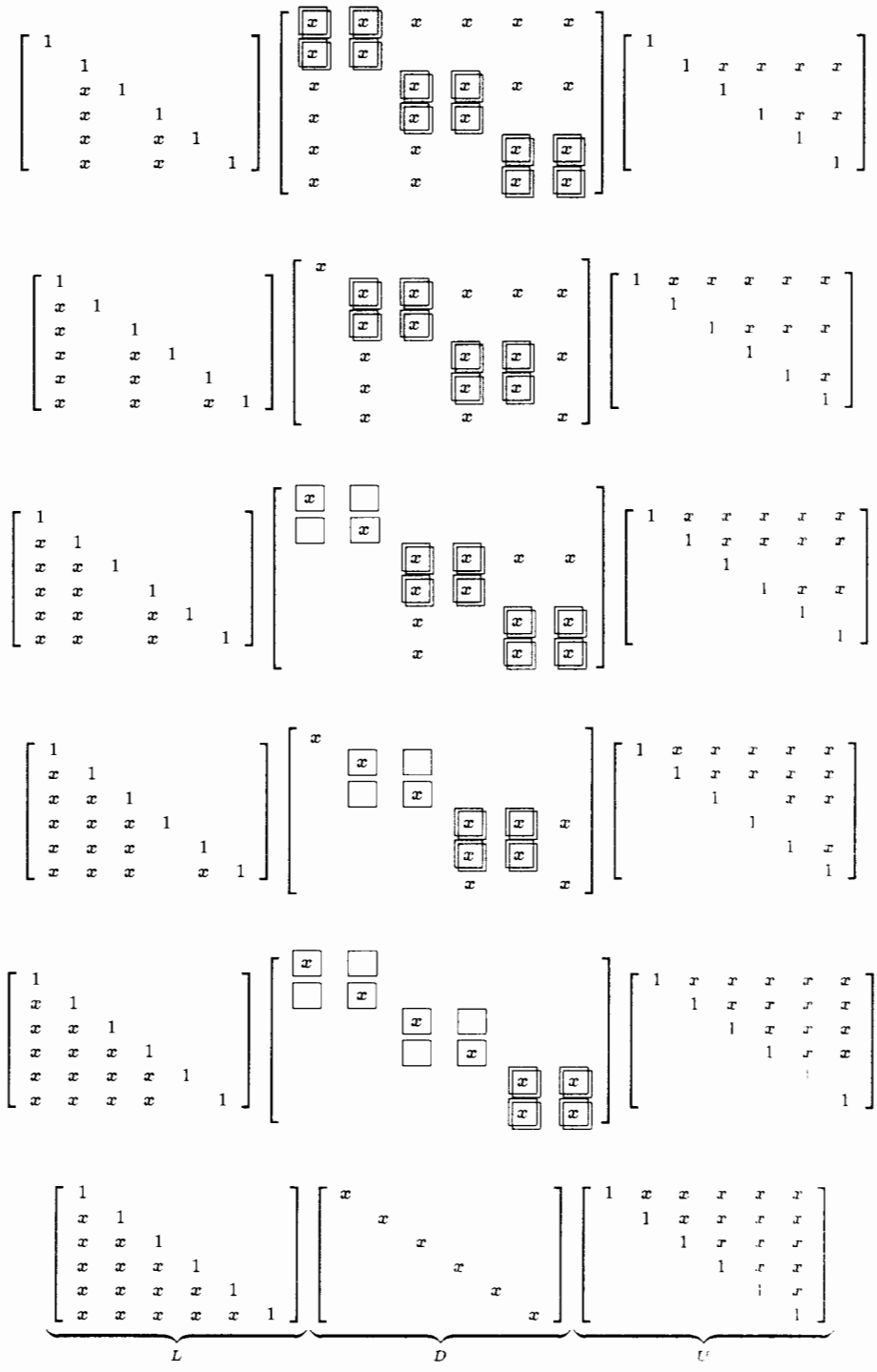

FIG. 2—*Continued*

Figure 2 shows that one backward sweep reduces the initial A matrix to diagonal form, while L and U remain lower resp. upper triangular. Clearly, after the backward sweep one has

$$L \cdot D \cdot U = \Pi \cdot A \cdot \Pi.$$

In order to compute an LDU decomposition for A, one should therefore start from $\Pi \cdot A \cdot \Pi$, to end up with

$$L \cdot D \cdot U = \Pi \cdot (\Pi \cdot A \cdot \Pi) \cdot \Pi = A.$$

REMARKS

1. The algorithm *preserves symmetry*, in a sense that if the A is symmetric, D is symmetric at all stages and likewise $L = U^T$ throughout. For symmetric positive definite matrices, the Cholesky factor is obtained in factorized form, $L \cdot D^{1/2}$.

2. At any stage, the matrices L, D, and U have nontrivial entries ($\neq 0, 1$) in complementary patterns. Hence the memory requirement is only n^2 for nonsymmetric and $\frac{1}{2}n^2$ for symmetric matrices.

3. It is seen that once an off-diagonal element is generated in L or U, it is not changed by subsequent transformations. Instead of, e.g., right multiplying L with L_k, it therefore suffices to copy the new entry. As a result, essentially only permutations are applied to L and U. From this it can also be seen that the Jacobi-type algorithm is only a reorganized version of the conventional algorithm for LDU [2]. Hence flop counts and stability result straightforwardly carry over to our case.

4. Instead of starting from $\Pi \cdot A \cdot \Pi$ and applying a backward sweep, one could alternatively start from A and apply a *forward sweep*. For a 6×6 matrix, a forward sweep consists of the transformations

$$
\begin{array}{cccccc}
i = 1 & & \cdot & & \cdot & \\
& 2 & & \cdot & & \\
1 & & 3 & & \cdot & \\
& 2 & & 4 & & \\
1 & & 3 & & 5 & \\
& 2 & & 4 & & \\
1 & & 3 & & \cdot & \\
& 2 & & \cdot & & \\
1 & & \cdot & & \cdot &
\end{array}
$$

Note that in order to avoid fill-ins, tranformations and permutations should be interchanged. The iteration formulas then read

$$
\begin{aligned}
\tilde{L} &\leftarrow \Pi_k \cdot \tilde{L} \cdot \tilde{L}_k \cdot \Pi_k \\
\tilde{D} &\leftarrow \Pi_k \cdot \tilde{L}_k^{-1} \cdot \tilde{D} \cdot \tilde{U}_k^{-1} \cdot \Pi_k \\
\tilde{U} &\leftarrow \Pi_k \cdot \tilde{U}_k \cdot \tilde{U} \cdot \Pi_k .
\end{aligned}
$$

As a result of this, $\tilde{L}$ and $\tilde{U}$ are upper resp. lower triangular matrices. The LDU decomposition of A is then determined as

$$
\tilde{L} \cdot \tilde{D} \cdot \tilde{U} = \Pi \cdot A \cdot \Pi
$$

$$
\underbrace{\Pi \tilde{L} \Pi}_{L} \cdot \underbrace{\Pi \tilde{D} \Pi}_{D} \cdot \underbrace{\Pi \tilde{U} \Pi}_{U} = A .
$$

One can verify that with both algorithms, exactly the same operations are performed. Only the presentation is different.

5. The above algorithms require roughly $2n$ steps, similar to the QRD algorithm. When several LDU decompositions have to be computed, one can alternately use forward and backward sweeps. As these can be pipelined, see, e.g., [6], the marginal cost for each additional LDU is then only n steps. For comparison, the algorithms in [3] and [1] require respectively $4n$ and $3n$ time steps for one single LU decomposition. Similarly the algorithm in [1] requires n time steps per decomposition when different decompositions can be pipelined. In conclusion, the Jacobi-type algorithm thus compares favorably with existing methods.

REFERENCES

1. Gentleman, W. M., and Kung, H. T. Matrix triangularization by systolic arrays. *Real-Time Signal Processing IV. Proc. SPIE* **298** (1981), 19–26.
2. Golub, G. H., and Van Loan, C. F. *Matrix computations*. North Oxford Academic Publishing, Johns Hopkins Univ. Press, 1989.
3. Kung, H. T., and Leiserson, C. E. Algorithms for VLSI processor arrays. In Mead, C., and Conway, L. (Eds.) *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA, 1980.
4. Luk, F. T. A rotation method for computing the QR-decomposition. *SIAM J. Sci. Statist. Comput.* **7**, 2 (1986), 452–459.
5. Schreiber, R. Cholesky factorization by systolic array. *RPI Computer Science Tech. Rep.* **87-14.**
6. Stewart, G. W. A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. *SIAM J. Sci. Statist. Comput.* **6**, 4 (1985), 853–863.

MARC MOONEN received his electromechanical engineering degree in electronics from the Katholieke Universiteit Leuven, Belgium in 1986 and his Doctoral degree in applied sciences from the Katholieke Universiteit Leuven in 1990. He is currently employed as a senior research assistant with the N.F.W.O. (Belgian National Fund for Scientific Research) and works in the Department of Electrical Engineering, Katholieke Universiteit Leuven. His research interests are in mathematical system theory and signal processing.

PAUL VAN DOOREN received his engineering degree in computer science from the Katholieke Universiteit Leuven, Belgium in 1974 and his Doctoral degree in applied sciences from the Katholieke Universiteit Leuven in 1979. From September 1974 to February 1978 he was a teaching and research assistant in the Department Computer Science, Katholieke Universiteit Leuven, Belgium. From April 1978 to March 1979 he was a research associate in the Department Electrical Engineering–Systems, University of

Southern California. From April 1979 to July 1979 Dr. Van Dooren was a teaching and research assistant in the Department Computer Science, Katholieke Universiteit Leuven, Belgium and he was a postdoctoral fellow in the Computer Science Department and Information Systems Laboratory, Stanford University, August 1979 to July 1980. For the period January 1985–May 1985 Dr. Van Dooren was a visiting fellow at the Centre of Mathematical Analysis and Department of Systems Engineering, Australian National University, and as of August 1980 he has been a research associate at Philips Research Laboratory, Brussels, Belgium. His main field of research is linear algebra. Other areas of interest include linear system theory and digital signal processing and parallel algorithms.

JOOS VANDEWALLE received his electromechanical engineering degree in electronics from the Katholieke Universiteit Leuven, Belgium in 1971. He earned his doctorate in applied sciences from Katholieke Universiteit Leuven in 1976 and became a special doctor in applied sciences (Katholieke Universiteit Leuven) in 1984. From 1976 to 1978 Dr. Vandewalle was a research associate and from 1978 to 1979 he was a visiting assistant professor at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. For the years 1980–1986, he was an assistant professor and since 1986 has been a full professor at the Department of Electrical Engineering, Katholieke Universiteit Leuven, Belgium. His research interests include system theory, digital signal processing, and cryptography.

Received August 27, 1990; revised February 6, 1991; accepted March 14, 1991