# Chapter 1
# An Improved Algorithm for Parallel Sparse LU Decomposition on a Distributed-Memory Multiprocessor

Jacko Koster* Rob H. Bisseling†

**Abstract**

In this paper we present a new parallel algorithm for the LU decomposition of a general sparse matrix. Among its features are matrix redistribution at regular intervals and a dynamic pivot search strategy that adapts itself to the number of pivots produced. Experimental results obtained on a network of 400 transputers show that these features considerably improve the performance.

## 1 Introduction

This paper presents an improved version of the parallel algorithm for the LU decomposition of a general sparse matrix developed by van der Stappen, Bisseling, and van de Vorst [9]. The LU decomposition of a matrix $A = (A_{ij},\ 0 \leq i,j < n)$ produces a unit lower triangular matrix $L$, an upper triangular matrix $U$, a row permutation vector $\pi$ and a column permutation vector $\rho$, such that

$$A_{\pi_i,\rho_j} = (LU)_{ij}, \text{ for } 0 \leq i,j < n. \tag{1}$$

We assume that $A$ is sparse and nonsingular and that it has an arbitrary pattern of nonzeros, with all elements having the same (small) probability of being nonzero. A review of parallel algorithms for sparse LU decomposition can be found in [9].

We use the following notations. A *submatrix* of a matrix $A$ is the intersection of several rows and columns of $A$. The submatrix $A[I,J]$, $I,J \subseteq \{0,\ldots,n-1\}$, has domain $I \times J$. If $I = \{i\}$, we use $A[i,J]$ as shorthand for $A[\{i\},J]$. The concurrent assignment operator $c,d := a,b$ denotes the simultaneous assignment of $a$ to $c$ and $b$ to $d$. For any (sub)matrix $A$, $nz(A)$ denotes the number of nonzeros in $A$. For any set $I$, $|I|$ is the cardinality of $I$.

Our algorithm is aimed at a distributed-memory message-passing MIMD multiprocessor with an $M \times N$ mesh communication network. We identify each processor in the mesh with a pair $(s,t)$, $0 \leq s < M$, $0 \leq t < N$. A *Cartesian distribution* [1] of $A$ is a pair of mappings $(\phi,\psi)$ that assigns matrix element $A_{ij}$ to processor $(\phi_i,\psi_j)$, with $0 \leq \phi_i < M$ and $0 \leq \psi_j < N$. For processor $(s,t)$, the set $I(s)$ denotes the local set of row indices $I(s) = \{i : i \in I \wedge \phi_i = s\}$. Similarly, $J(t) = \{j : j \in J \wedge \psi_j = t\}$.

*CERFACS, 42 Ave G. Coriolis, 31057 Toulouse Cedex, France (Jacko.Koster@cerfacs.fr). Part of this work was done while this author was employed at Eindhoven University of Technology.

†Department of Mathematics, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, the Netherlands (bisseling@math.ruu.nl). Part of this work was done while this author was employed at Koninklijke/Shell-Laboratorium, Amsterdam.

## 2 The PARPACK-LU algorithm

In this section we briefly describe the previous algorithm [9], which we refer to as PARPACK-LU. This algorithm assigns the nonzero matrix elements to the processors according to the *grid distribution* defined by

$$\phi_i = i \bmod M \ \wedge \ \psi_i = i \bmod N, \text{ for } 0 \leq i < n. \tag{2}$$

Each step of the algorithm consists of a pivot search, row and column permutations, and a multiple-rank update of the *reduced matrix* $A[I, I]$. At the beginning of a step, $I = \{k, \ldots, n-1\}$, where $k$ is the number of pivots processed so far.

The pivot search determines a set $S$ of $m$ pivots from the reduced matrix with the following three properties. First, each element $(i, j)$ from $S$ satisfies the threshold criterion

$$|A_{ij}| \geq u \cdot \max_{l \in I} |A_{lj}|, \tag{3}$$

where $u$ is a user-defined parameter, $0 < u \leq 1$ [4, Ch. 9]. This ensures numerical stability. Second, the elements of $S$ have low Markowitz cost, to preserve sparsity. The *Markowitz cost* of a nonzero element $A_{ij}$ in a submatrix $A[I, J]$ equals $(nz(A[I, j]) - 1)(nz(A[i, J]) - 1)$. Third, the elements of $S$ are mutually *compatible* [2, 3], i.e.,

$$A_{i,j'} = 0 \ \wedge \ A_{i',j} = 0, \ \text{ for } \ (i, j), (i', j') \in S \ \wedge \ (i, j) \neq (i', j'). \tag{4}$$

The compatibility of the pivots enables the algorithm to process them in one step and to perform a single rank-$m$ update of the reduced matrix.

After the pivot search, the rows and columns of the matrix are permuted such that the $m \times m$ submatrix $A[I_S, I_S]$, $I_S = \{k, \ldots, k+m-1\}$, turns into a diagonal submatrix with the $m$ pivots positioned on the diagonal. This is followed by the rank-$m$ update, which contains the bulk of the floating point operations. In this part, the set $I_S$ is subtracted from $I$, each *multiplier column* $j$ in $A[I, I_S]$ is divided by the corresponding pivot value $A_{jj}$ and the matrix product $A[I, I_S] \cdot A[I_S, I]$ is subtracted from $A[I, I]$.

## 3 The new algorithm

An outline of the new algorithm is given below; a detailed description and a program text can be found in [7].

ALGORITHM 1 (Parallel LU decomposition).
$I(s), J(t) := \{i : 0 \leq i < n \wedge \phi_i = s\}, \{j : 0 \leq j < n \wedge \psi_j = t\}$;
$L, U, k := 0, 0, 0$;
**while** $k < n$ **do begin**
  find pivot set $S = \{(i_r, j_r) : k \leq r < k + m\}$;
  $I_S, J_S := \{i_r : k \leq r < k + m\}, \{j_r : k \leq r < k + m\}$;
  $I(s), J(t) := I(s) \setminus I_S, J(t) \setminus J_S$;
  register pivots in $\pi$ and $\rho$;
  perform multiple-rank update;
  store multipliers in $L$ and pivots and update rows in $U$;
  $k := k + m$;
  row-redistribute$(A, L)$;
  col-redistribute$(A, U)$
**end**;
row-permute$(L, \pi)$;
col-permute$(U, \rho)$.

In the search for pivot candidates, we vary the number *ncol* of matrix columns searched per processor column, on the basis of the estimated density of the reduced matrix. In the first steps of the algorithm, the reduced matrix is still sparse and the candidate pivots will most likely be compatible and hence most of them will become pivots. As the computation proceeds, fill-in causes the reduced matrix to become gradually denser. Consequently, the probability of nonzero elements being compatible decreases and expensive searches for large sets of candidate pivots will yield only relatively few compatible pivots. Therefore, for higher densities of the reduced matrix, we decrease *ncol*. When the reduced matrix becomes so dense that the pivot search repeatedly produces only one pivot, the algorithm switches to a simple search for only one pivot.

The PARPACK-LU algorithm distributes the work load by maintaining the grid distribution and performing *explicit* row and column permutations [1, 6] at every step. Therefore, it often requires the exchange of rows between processor rows, and similarly for columns, because many of the explicit permutations cannot be performed locally. This induces a certain amount of added communication time. When *implicit* permutations [8] are used, the rows and columns of the matrix remain in place. The matrix elements are addressed indirectly and no communication for row or column movements is required. This, however, may lead to a poor load balance, depending on the sequence of pivot choices.

We distribute the load by redistributing the reduced matrix at regular intervals such that $|J(t)| \leq \lceil (n-k)/N \rceil$ for all $t$ and $|I(s)| \leq \lceil (n-k)/M \rceil$, for all $s$. It suffices to move columns from processor columns $(*,t)$ for which $|J(t)| > \lceil (n-k)/N \rceil$ to processor columns for which $|J(t)| < \lceil (n-k)/N \rceil$, and similarly for the rows of the matrix. The number of columns to be sent left and right is determined using the criterion: a column to be disposed of is sent in the direction that has the least average surplus per processor column. The first processor column that has space available accepts a passing column. This heuristic reduces the distance over which the rows and columns are communicated. The column redistribution by processor column $(*,t)$ is done as follows:

ALGORITHM 2 (Column redistribution).
**if** redistribution needed **then begin**
*ceiling* := $\lceil (n-k)/N \rceil$;
*surplus* := $|J(t)|$ − *ceiling*;
**if** *surplus* > 0 **then** determine $n_l, n_r \geq 0$ such that $n_l + n_r =$ *surplus*
**else** $n_l, n_r := 0, 0$;
redistribute ($n_r$, right);
redistribute ($n_l$, left)
**end**;

The theoretical analysis of [9] shows that the unordered two-dimensional doubly linked list is the best local data structure for parallel sparse LU decomposition, among several plausible candidates. We use this data structure for the implementation of the new algorithm; the previous algorithm was implemented using the ordered two-dimensional singly linked list. The present data structure facilitates insertion and deletion of nonzeros.

## 4 Experimental results

The parallel computer used for our experiments (and those of [9]) is a Parsytec SuperCluster FT-400 consisting of a square mesh of 400 INMOS T800-20 transputers. The new algorithm is implemented in ANSI C with extensions for communication and parallelism. This

TABLE 1
*Time (in s) of LU decomposition on p processors using a static pivot search strategy.*

| Matrix | $p = 1$ | $p = 4$ | $p = 9$ | $p = 16$ | $p = 25$ | $p = 49$ | $p = 100$ | $p = 400$ |
|---|---|---|---|---|---|---|---|---|
| IMPCOL B | 0.25 | 0.19 | 0.16 | 0.15 | 0.13 | 0.13 | 0.12 | 0.13 |
| WEST0067 | 0.32 | 0.25 | 0.24 | 0.19 | 0.19 | 0.18 | 0.16 | 0.18 |
| FS 541 1 | 8.85 | 3.97 | 2.65 | 2.24 | 1.89 | 1.53 | 1.28 | 1.14 |
| STEAM2 | 48.9 | 16.1 | 9.75 | 6.22 | 5.27 | 3.83 | 3.07 | 2.51 |
| SHL 400 | 2.51 | 1.68 | 1.28 | 1.09 | 0.94 | 0.79 | 0.71 | 0.61 |
| BP 1600 | 4.78 | 3.38 | 2.69 | 2.45 | 2.20 | 1.96 | 1.79 | 1.63 |
| JPWH 991 | 125. | 43.6 | 20.6 | 12.6 | 11.2 | 7.06 | 5.64 | 4.28 |
| SHERMAN1 | 17.1 | 11.5 | 7.18 | 5.87 | 4.35 | 2.97 | 2.79 | 2.42 |
| SHERMAN2 | 1294. | | | 108. | 64.7 | 46.1 | 30.1 | 13.6 |
| LNS 3937 | 1430. | | | 128. | 82.9 | 55.7 | 34.6 | 20.5 |
| GEMAT11 | 41.9 | 21.8 | 15.6 | 12.6 | 10.9 | 8.78 | 7.40 | 5.71 |

implementation allows us to use the parallel program also for experiments on an ordinary sequential computer. We did not optimise the parallel program for these sequential runs. The sequential experiments were performed on a SUN SPARC10 model 30 workstation. The test set of sparse matrices that we use for our experiments is taken from [9]. It consists of eleven unsymmetric matrices from the Harwell-Boeing sparse matrix collection [5].

The user of the program has to specify six input parameters. The parameter $ncol_0$ is the initial number of matrix columns that is searched for pivot candidates by each processor column. The parameter $u$ is used for threshold criterion (3). Pivot candidates with a Markowitz cost higher than $\alpha M_{\min} + \beta$ are discarded. Here $\alpha$ and $\beta$ are input parameters, and $M_{\min}$ is the lowest Markowitz cost of a pivot candidate. After $nlast$ successive pivot searches produce only one pivot, a switch is made to a single-pivot strategy. Each time a multiple of $f$ pivots has been processed, the matrix is redistributed.

First, we repeated the experiments from [9, Table 4] to compare the performance of the new algorithm to that of the PARPACK-LU algorithm. The standard static pivot search procedure in [9] is most closely approximated by fixing $ncol = 1$, and using $u = 0.1$, $\alpha = 4$, $\beta = 0$, and $nlast = \infty$. The matrix is redistributed at every step, i.e., $f = 1$, to mimic the explicit row and column permutations of the PARPACK-LU algorithm. This experiment allows us to examine the effect of choosing a different data structure, without regard to other improvements. Table 1 shows the time needed for parallel LU decomposition on a square mesh of $p$ transputers. A comparison with the results for the PARPACK-LU algorithm [9] shows that the new algorithm is superior. It outperforms the previous one for small $p$, with a gain of up to a factor of 2.4 (for SHERMAN1 and $p = 1$). For large $p$, the two algorithms perform equally well, because communication time, which dominates for such $p$, is similar.

To investigate the influence of the pivot strategy, we replaced the standard pivot strategy of [9] by a dynamic one: $ncol_0 = 10$ and at the end of each step $ncol$ is adjusted according to

$$\textbf{if } m > \frac{ncol \cdot N}{2} \textbf{ then } ncol := ncol + 1 \textbf{ else } ncol := ncol - 1$$

This strategy strives for generating twice as many pivot candidates as pivots. This way, a relatively large set of pivot candidates, each with a reasonable probability of becoming a pivot, is used in constructing the pivot set, while the time required for the compatibility checks is kept within bounds. (The other parameters in this experiment are $u = 0.1$, $\alpha = 4$,

TABLE 2

*Time (in s) of LU decomposition on a sequential computer and on p processors of a parallel computer, using a dynamic pivot search strategy.*

| Matrix | SPARC10 | $p = 1$ | $p = 16$ | $p = 100$ | $p = 400$ |
|---|---|---|---|---|---|
| IMPCOL B | | 0.19 | 0.11 | 0.10 | 0.11 |
| WEST0067 | | 0.27 | 0.18 | 0.15 | 0.14 |
| FS 541 1 | 1.25 | 7.99 | 1.86 | 1.08 | 0.96 |
| STEAM2 | 7.05 | 45.6 | 6.88 | 2.55 | 1.95 |
| SHL 400 | 0.12 | 0.77 | 0.46 | 0.39 | 0.36 |
| BP 1600 | 0.48 | 3.01 | 1.27 | 1.06 | 0.96 |
| JPWH 991 | 13.9 | 82.2 | 13.0 | 4.34 | 2.57 |
| SHERMAN1 | 2.98 | 18.8 | 4.53 | 2.00 | 1.54 |
| SHERMAN2 | 206. | 1145. | 127. | 25.9 | 10.6 |
| LNS 3937 | 277. | 1405. | 120. | 33.2 | 16.4 |
| GEMAT11 | 4.13 | 26.6 | 6.68 | 4.54 | 3.83 |

$\beta = 10$, $nlast = 25$, and $f = 100$.) Table 2 shows that the dynamic pivot search strategy is superior. We attribute this mainly to the higher rank $m$ of the matrix updates, which leads to less frequent synchronisation of the processors. Even for $p = 1$ there is a gain: for example, the computing time is reduced by a factor of three for the matrix SHL 400. This confirms the theoretical analysis from [9] that multiple-rank updates are beneficial also in sequential sparse LU decomposition algorithms.

## References


[1] R. H. Bisseling and J. G. G. van de Vorst, *Parallel* LU *decomposition on a transputer network*, Lecture Notes in Computer Science 384, Springer-Verlag, New York, 1989, pp. 61–77.

[2] D. A. Calahan, *Parallel solution of sparse simultaneous linear equations*, Proc. 11th Annual Allerton Conf. on Circuits and System Theory, 1973, pp. 729–735.

[3] T. A. Davis and P.-C. Yew, *A nondeterministic parallel algorithm for general unsymmetric sparse* LU *factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 383–402.

[4] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, UK, 1986.

[5] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[6] G. A. Geist and C. H. Romine, LU *factorization algorithms on distributed-memory multiprocessor architectures*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 639–649.

[7] J. Koster, *Parallel solution of sparse systems of linear equations on a mesh network of transputers*, Final Report, Institute for Continuing Education, Eindhoven University of Technology, Eindhoven, The Netherlands, July 1993.

[8] A. Skjellum, *Concurrent dynamic simulation: multicomputer algorithms research applied to ordinary differential-algebraic process systems in chemical engineering*, Ph. D. Thesis, California Institute of Technology, Pasadena, CA, May 1990.

[9] A. F. van der Stappen, R. H. Bisseling, and J. G. G. van de Vorst, *Parallel sparse* LU *decomposition on a mesh network of transputers*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 853–879.