# Parallelization and Integration of the LU and ILU Algorithm in the LINSOL program package

Hartmut Häfner, Willi Schönauer, and Rüdiger Weiss

Numerical Research group for Supercomputers, Computing Center, Universität Karlsruhe, 76128 Karlsruhe, Germany,
haefner@rz.uni-karlsruhe.de, schoenauer@rz.uni-karlsruhe.de, weiss@rz.uni-karlsruhe.de,
WWW home page: http://www.rz.uni-karlsruhe.de/Uni/RZ/Forschung/Numerik/

**Abstract.** In order to provide generally applicable iterative linear solvers for the community of scientific computing the LINSOL program package has been designed. The focus of this package is on portability, robustness and on an efficient implementation on massively parallel systems. LINSOL uses iterative solvers as basic methods that are state of the art. Different normalization methods can be used to improve the convergence rates of the iterative solvers. Now preconditioners like the (in)complete Gaussian algorithm are being implemented. The paper is focussed on this type of algorithm. LINSOL is tuned to massively parallel systems with distributed memory. Therefore, the message passing programming style is used. LINSOL supports many matrix formats for the convenience of the users. Moreover, adaptive method selection schemes called polyalgorithms are implemented.

## 1 Background

LINSOL [1997] [1998] is a parallel iterative linear solver package solving

$$Ax = b.$$

The program package is adapted to the application of sparse matrices, but can also be efficiently applied to full matrices. It presently contains fourteen iterative methods and ten polyalgorithms of generalized Conjugate Gradient (CG) methods. The polyalgorithms use the fourteen iterative methods as basic algorithms and differ by switching automatically between different methods.

Ten different data structures are supported by LINSOL to ease the embedding of LINSOL into an application as well as the mapping of arbitrary sparse matrices to appropriate storage patterns.

The basic concepts of LINSOL are:

– **Flexibility or ease of use** - the design of LINSOL allows to use three types of interfaces and supports ten different data structures and storage patterns, respectively.

- **Portability** - by using the standardized Fortran77 and Fortran90 programming languages on single processor systems and additionally the *message passing* paradigm on parallel computers.
- **Robustness** - is obtained by fourteen different iterative solvers; they are suitable for different classes of matrices; additionally, adaptive polyalgorithms are provided that choose an appropriate solver from the fourteen methods. To enhance the robustness of LINSOL an (incomplete) LU algorithm embedded as a preconditioner into the iterative methods is being integrated into the program package.
- **Optimized code** for workstations (PCs), vectorcomputers and parallel computers - by reusing data in the highest memory hierarchy as far as possible and by using vector pipelining and parallel algorithms with a high *volume-to-surface* ratio. Two absolutely essential features for parallel software and thus for LINSOL are: scalability regarding the computation time and scalability regarding the memory. The second feature is achieved by the rowwise distribution of the matrix $A$ onto matrix blocks and by the distribution of all vectors of LINSOL in vector parts adapted to the distribution of the matrix.

There are three essential features of LINSOL. First LINSOL is designed as "Black Box" solver which means that as few parameters as possible should be user-defined. Second the convergence behaviour and the results of LINSOL do to not depend on the number of processors (at least if we neglect the roundoff errors). Third many storage patterns are supported by LINSOL. These basic data structures are: main diagonal (assumed full), full (non-main) diagonal, packed diagonal (elements and their indices, from the first to the last nonzero element), indexed column (one element per row and the corresponding column index, from the first to the last nonzero element), indexed row (similarly one element per column and its row index), starry sky (nonzero elements with row and column indices), full row, full column, packed row (nonzero elements in a row and their column indices from the first to the last nonzero element) and packed column (similarly nonzero elements in a column and their row indices). They can be assembled without cutback to the matrix $A$, if $A$ is nonsymmetric. If A is symmetric the only cutback is that the storage pattern "main diagonal" must exist and all main diagonal elements must be stored there.

If all iterative methods fail or converge very slowly, the incomplete or full Gaussian algorithm is being provided as a preconditioner for the iterative methods. The full Gaussian algorithm can be used as "emergency exit", if all iterative solvers fail to converge. Incomplete stands for the dropping of small matrix elements before and during the elimination process. To enable the preconditioning of large matrices the algorithm from Gibbs, Poole, Stockmeyer is being inserted as a bandwidth optimizer and the elimination process is executed in a dynamically used buffer whose size is determined by the bandwidth of the matrix. The parallelization of the bandwidth optimizer will be presented in a separate subsequent paper from W. Schönauer, D. Zundel and H. Häfner. For the parallelization of the Gaussian elimination process the matrix and the dynamically used buffer are wrapped row by row onto the processors. The goal is to embed

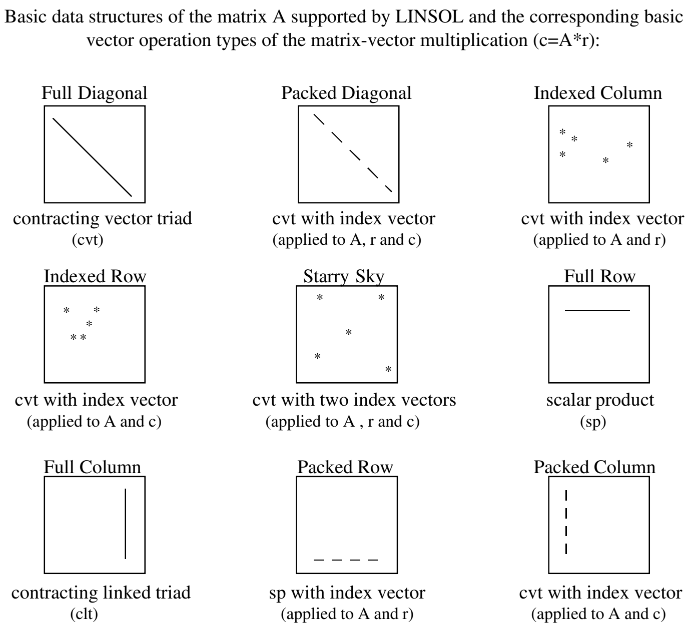


**Fig. 1.** Basic Data Structures of LINSOL

the LU/ILU algorithm into LINSOL so that the direct solver can be used as a preconditioner transparently to the user for all classes of matrices and for all processor numbers. C. Vuik, R.R.P. van Nooyen and P. Wesseling developed a parallel ILU-preconditioned GMRES method for matrices with nine non-zero diagonals [1998]; thus the algorithm only works for a special class of matrices. Within the PARASOL project (http://192.129.37.12/parasol) parallel sparse direct methods are being developed, too. But these methods can only be used as direct solvers; another difference in comparison to LINSOL is that the approach will be based on the multifrontal method as developed by Ian Duff and John Reid. The principal feature of the approach is that the matrix factorization can be represented by a tree where each edge represents the communication of data and each node some elimination operation on dense submatrices.

## 2 Mathematical outline

The embedding of the Gaussian algorithm into the iterative methods reads as follows. Instead of the linear system $Ax = b$ the preconditioned system $P_L A P P_R y = P_L b$ with $x = P_R y$ must be solved. $P_L$ $(P_R) \in R^{n \times n}$ is a non-singular left-hand (right-hand) preconditioning matrix acting on the equations (solutions); $P$ is a permutation matrix emerging from the partial pivoting in the Gaussian elimination. By the full Gaussian elimination we get the triangular matrices $U$ and $L$ with $AP = UL$. With $P_L = U^{-1}$ and $P_R = L^{-1}$ we get the optimal choice $P_R P_L = L^{-1}U^{-1} = (UL)^{-1} = (AP)^{-1} = PA^{-1}$. By the incomplete Gaussian elimination we get the triangular matrices $\tilde{U}$ and $\tilde{L}$ with $AP \approx \tilde{U}\tilde{L}$. With $P_L = \tilde{U}^{-1}$ and $P_R = \tilde{L}^{-1}$ we get the approach $P_R P_L = \tilde{L}^{-1}\tilde{U}^{-1} = (\tilde{U}\tilde{L})^{-1} \approx (AP)^{-1} = PA^{-1}$. Using $P_L$ in the iteration process, i.e. to get a preconditioned vector $r^P$ from the original vector $r$, means to perform a forward elimination step; using $P_R$ in the iteration process, i.e. to get the solution $x$ from the solution $y$ of the preconditioned system, means to perform a backward substitution step:

$$
\begin{aligned}
&Ur^P = r \Leftrightarrow r^P = U^{-1}r \qquad ,\\
&Lx = y \Leftrightarrow x = L^{-1}y \qquad .
\end{aligned}
$$

As examples preconditioning is described in connection with the *classical* CG algorithm for a symmetric positive definite matrix $A$ and in connection with the biconjugate gradients method (BCG) for a nonsymmetric matrix $A$. It must be mentioned that the preconditioning will work for all fourteen iterative methods and that the mathematical description of the preconditioning is varying for different methods. The fourteen iterative solvers implemented in LINSOL are described in detail in [1996]. In comparison to the *classical* CG algorithm the residual $r_k = Ax_k - b(k = 0, 1, 2, ...)$ has to be substituted by $r_k^P = P_L r_k$ and the matrix $A$ by $P_L A P P_R$. Thus the term $Ar_k$ must be substituted by $\hat{r}_k = P_L A P P_R r_k^P$.

For an arbitrary *initial guess* $x_0$ holds:

$$Ax = A(\tilde{x} + x_0) = b \Leftrightarrow A\tilde{x} = b - Ax_0 \Leftrightarrow A\tilde{x} = \tilde{b}.$$

Now we solve $P_L A P P_R \tilde{y} = P_L \tilde{b}$ with $\tilde{x} = P P_R \tilde{y}, \tilde{x}_0 = 0$ and calculate

$$
\begin{aligned}
r_0 &= -\tilde{b},\ r_0^P = P_L r_0,\ \hat{r}_0 = P_L A P P_R r_0^P \text{ and for } k \geq 0:\\
\alpha_{1,k}^P &= -\frac{\left(r_k^P\right)^T \hat{r}_k}{\left(r_k^P\right)^T r_k^P}\\
\alpha_{2,k}^P &= -\frac{\left(r_{k-1}^P\right)^T \hat{r}_k}{\left(r_{k-1}^P\right)^T r_{k-1}^P}\\
\phi_k^P &= \frac{1}{\alpha_{1,k}^P + \alpha_{2,k}^P}\\
r_{k+1}^P &= \phi_k^P (P_L A P P_R r_k^P + \alpha_{1,k}^P r_k^P + \alpha_{2,k}^P r_{k-1}^P)\\
&\quad \text{with } r_k^P = P_L r_k\ , \tilde{r}_k = P P_R r_k^P \text{ and } \hat{r}_k = P_L A P P_R r_k^P\\
\tilde{x}_{k+1} &= P P_R \tilde{y}_{k+1} = \phi_k^P (P P_R r_k^P + \alpha_{1,k}^P P P_R \tilde{y}_k + \alpha_{2,k}^P P P_R \tilde{y}_{k-1})\\
&= \phi_k^P (\tilde{r}_k + \alpha_{1,k}^P \tilde{x}_k + \alpha_{2,k}^P \tilde{x}_{k-1}).
\end{aligned}
$$

In BCG we solve $P_L A P P_R \tilde{y} = P_L \tilde{b}$ with $\tilde{x} = P P_R \tilde{y}, \tilde{x}_0 = 0$ and calculate

$$
\begin{aligned}
r_0 &= -\tilde{b},\ r_0^P = p_0^P = P_L r_0,\ (r_0^P)^* = (p_0^P)^* = P_R^T r_0^* \text{ and for } k \geq 0: \\
\delta_k^P &= -\frac{\left(r_k^P\right)^T \left(r_k^P\right)^*}{\left(p_k^P\right)^T P_R^T P A^T P_L^T \left(p_k^P\right)^*} = -\frac{\left(r_k^P\right)^T \left(r_k^P\right)^*}{\left(p_k^P\right)^T \left(\hat{p}_k^P\right)^*} \\
r_{k+1}^P &= r_k^P + \delta_k^P P_L A P P_R p_k^P = r_k^P + \delta_k^P P_L A \tilde{p}_k^P \\
\left(r_{k+1}^P\right)^* &= \left(r_k^P\right)^* + \delta_k^P \left(\hat{p}_k^P\right)^* \\
\beta_k^P &= \frac{\left(r_{k+1}^P\right)^T \left(r_{k+1}^P\right)^*}{\left(r_k^P\right)^T \left(r_k^P\right)^*} \\
p_{k+1}^P &= r_{k+1}^P + \beta_k^P p_k^P \\
\left(p_{k+1}^P\right)^* &= \left(r_{k+1}^P\right)^* + \beta_k^P \left(p_k^P\right)^* \\
\tilde{x}_{k+1} &= P P_R \tilde{y}_{k+1} = P P_R (\tilde{y}_k + \delta_k^P p_k^P) \\
&= \tilde{x}_k + \delta_k^P \tilde{p}_k^P .
\end{aligned}
$$

Thus the CG algorithm needs one forward elimination step and one backward substitution step per iteration and the BCG algorithm two forward elimination steps and two backward substitution steps per iteration. To apply a vector to the permutation matrix $P$ means to perform a permutation of the vector elements by an index list.

If the matrix $A$ is symmetric we can use a Cholesky decomposition and get the triangular matrices $L^T$ and $L$ with $AP = L^T L$. Thus the elimination can be performed completely on $L$ reducing the storage requirements to the halfth. It holds $P_L = P_R$ and the forward elimination takes place on $P_L$ and the backward substitution on $P_L^T$.

To minimize the storage requirements of the preconditioning matrices $P_L$ and $P_R$ for sparse matrices they are stored in a packed storage pattern. During the elimination process the rows of the matrix to be eliminated are wrapped around onto the processors on parallel computers to obtain scalability and a good load balancing.

## 3 Brief description of the LU/ILU-preconditioner

The preconditioning process within the program package LINSOL consists of four parts:

- preparation of the LINSOL-matrix,
- application of the bandwidth optimizer,
- the elimination process and
- the repeated application of the forward elimination and of the backward substitution within the iterative solution process.

Between these four steps several permutation algorithms resorting the matrix and the right hand side respectively must be applied.

### 3.1 Preparation of the LINSOL-matrix

The LINSOL-matrix ($A$) can be made up of the ten different data structures supported by LINSOL. However the preconditioning matrix ($A^P$) only works on the storage pattern "packed row". Thus the matrix $A$ is completely reassembled "in place" in the storage pattern "packed row", and zero entries are eliminated to reduce the storage requirements. On parallel computers this algorithm can be performed locally on all processors. In a second preparation step rows with only one non-zero entry ("explicit" variables) are used to eliminate entries of all other rows in the column, corresponding to the single non-zero entry, by shifting the entries multiplied by the solution component of the row with the single non-zero entry to the right hand side. Thus new rows with only one non-zero entry can arise and again entries in the matrix $A$ can be eliminated. If the matrix would be a permuted triangular matrix, this algorithm is equivalent to a backward substitution step on the original triangular matrix. In the next step a graph of the matrix $A$ is created. As the LINSOL-matrix $A$ is distributed in blocks of rows onto the processors and the graph has to be "symmetrized" for the application of the bandwidth optimizer, the generation process of the graph requires communication over all processors.

### 3.2 Application of the bandwidth optimizer

In the graph the row numbers of the matrix $A$ are interpreted as nodes and the row indices (not equal to their row number) are interpreted as undirected edges. Thus we get a finite undirected graph without loops or multiple edges, on which the bandwidth optimization process is performed. In paper [1976] the used algorithm of Gibbs, Poole, Stockmeyer (GPS) is described in detail. The GPS-algorithm returns a permutation vector that is used to resort both rows and columns of the matrix $A$ and the right hand side. Running LINSOL on more than one processor resorting of the rows of matrix $A$ requires communication over all processors (see also Fig. 3).

### 3.3 The elimination process

Before the elimination process can be started, the matrix $A$ must be copied into the preconditioning matrix $A^P$. To determine the size of the buffer in which the Gaussian elimination is processed on unpacked rows we need two definitions. The bandwidths $\beta_U$ and $\beta_L$ of the matrix $A^P$ are defined by

$$\beta_U = \max_{a_{ij}\neq 0, j>i} (j-i) \text{ and } \beta_L = \max_{a_{ij}\neq 0, i>j} (i-j).$$

$\beta_U$ is the maximal distance of elements above the main diagonal to the main diagonal; $\beta_L$ is the maximal distance of elements below the main diagonal to the main diagonal. In the elimination process pivoting by columns is performed. Then the size of the buffer must be set to $(\beta_L + 2\beta_U)(\beta_L + \beta_U)$. $(\beta_L + 2\beta_U)$ is the buffer size of each row and the maximal number of rows in the buffer is $(\beta_L + \beta_U)$. The buffer consists of the pivot row and all further rows that have non-zero entries in the pivot column (in Fig. 2 the elimination process for the pivot rows 1 to 4 is depicted). The buffer is a one-dimensional array that is managed by an information array containing a pointer to the row elements, the global row number and the length of the row for each row stored in the buffer. In Fig. 2 the global row number can be seen below the entry "Row" for the elimination process of the pivot rows 1 to 4. Row by row is stored contiguously in the buffer till the end of the buffer is reached. Then a "garbage collection" is performed; this means that "active" rows (rows with row indices greater than the pivot row index) are shifted to a contiguous array at the beginning of the buffer.

The pivot column is the column with the maximal element in the pivot row. The permutation of column indices - caused by column pivoting - is stored in two permutation vectors assigned to the global matrix $A^P$ (in Fig. 2 $P_{global}$) and the local buffer (in Fig. 2 $P_{local}$). The global permutation vector is used in the iteration process instead of the matrix $P$. The local permutation vector is used to assign matrix elements to the buffer.

For each pivot row the preconditioning matrix $A^P$ must be searched for rows with non-zero entries in the pivot column. In parallel mode the processor with the pivot row has to broadcast the index of the pivot column to all processors and then each processor searches for rows in the locally stored matrix $A^P$. These rows must be unpacked and put into the buffer. E.g. in Fig. 2 the rows with the global row number 1,2 and 4 must be put into the buffer for pivot row 1 and pivot column 4. The buffering is done by the permutation vector $P_{local}$ assigned to the local buffer. Thus the buffered part of the matrix $A^P$ corresponds to the permuted matrix $(A^P P)$. The elimination process then runs on the unpacked rows in the buffer with the vectorlength of the pivot row reduced by 1. In Fig. 2 the computation area is surrounded by bold lines in the tables. After one computation step in the buffer the pivot row and the actual leftmost column of the buffer are restored to the preconditioning matrix $A^P$ as packed row and packed column respectively. Restored values are depicted by a grey background in Fig. 2.

The parallelization of the elimination process is done by wrapping around the rows of the preconditioning matrix onto the processors. Unpacked rows in the buffer are stored on the same processor as the corresponding packed rows of the preconditioning matrix $A^P$. This kind of parallelization leads to a simple communication pattern. Only two "broadcasts" of the pivot indices and of the elimination coefficients, that are processed by the pivot processor, are necessary per one elimination step in the buffer for the update of the unpacked matrix rows in the buffers of all processors.

Actually the elimination is processed on unpacked rows in the buffer. If the matrix is very sparse within the bandwidth of the matrix $A^P$, the buffer contains

Elimination process: A = U * L * P

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | | 4 | | | | | |
| 1 | 2 | 3 | | | | | | |
| | 20 | 3 | 4 | | 6 | | | |
| | 2 | | 4 | 5 | 6 | | | |
| | 2 | | | 5 | 6 | | | |
| | | | | | | 7 | 8 | |
| | | | | | 6 | 7 | 8 | 9 |
| | | | | 5 | 6 | 7 | 80 | |
| | | | | | | 7 | | 90 |

Matrix A with an upper bandwidth of 3 and a lower bandwidth of 3 stored in the storage scheme of „packed rows“

For row 1

| $P_{global}$ | 4 | 2 | 3 | 1 | 5 | 6 |
|---|---|---|---|---|---|---|
| Row/$P_{local}$ | 4 | 2 | 3 | 1 | 5 | 6 |
| 1 | **4** | **2** | **0** | **1** | | |
| 3 | **4** | **20** | **3** | **0** | 0 | 6 |
| 4 | **4** | **2** | **0** | **0** | 5 | 6 |

| $P_{global}$ | 4 | 2 | 3 | 1 | 5 | 6 |
|---|---|---|---|---|---|---|
| Row/$P_{local}$ | 4 | 2 | 3 | 1 | 5 | 6 |
| 1 | **4** | **0.5** | **0** | **0.25** | | |
| 3 | **4** | 18 | 3 | -1 | 0 | 6 |
| 4 | **4** | 0 | 0 | -1 | 5 | 6 |

3 elements stored as 1th column of U in storage scheme „packed columns“
2 elements stored as 1th row of L in storage scheme „packed rows“

For row 2

| $P_{global}$ | 3 | 2 | 1 | 5 | 6 |
|---|---|---|---|---|---|
| Row/$P_{local}$ | 3 | 2 | 1 | 4 | 5 |
| 2 | **3** | **2** | **1** | | |
| 3 | **3** | **18** | **-1** | 0 | 6 |
| 4 | 0 | 0 | -1 | 5 | 6 |

| $P_{global}$ | 3 | 2 | 1 | 5 | 6 |
|---|---|---|---|---|---|
| Row/$P_{local}$ | 3 | 2 | 1 | 4 | 5 |
| 2 | **3** | **0.66** | **0.33** | | |
| 3 | **3** | 16 | -2 | 0 | 6 |
| 4 | 0 | 0 | -1 | 5 | 6 |

2 elements stored as 2th column of U
2 elements stored as 2th row of L

For row 3

| $P_{global}$ | 2 | 1 | 5 | 6 |
|---|---|---|---|---|
| Row/$P_{local}$ | 2 | 1 | 3 | 4 |
| 3 | **16** | **-2** | **0** | **6** |
| 4 | 0 | -1 | 5 | 6 |
| 5 | **2** | **0** | **5** | **60** |

| $P_{global}$ | 2 | 1 | 5 | 6 |
|---|---|---|---|---|
| Row/$P_{local}$ | 2 | 1 | 3 | 4 |
| 3 | **16** | **-0.125** | **0** | **0.375** |
| 4 | 0 | -1 | 5 | 6 |
| 5 | **2** | 0.25 | 5 | 59.25 |

2 elements stored as 3th column of U
2 elements stored as 3th row of L

For row 4

| $P_{global}$ | 6 | 5 | 1 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|
| Row/$P_{local}$ | 3 | 2 | 1 | 4 | 5 | 6 |
| 4 | **6** | **5** | **-1** | | | |
| 5 | **59.25** | **5** | **0.25** | | | |
| 7 | **6** | **0** | **0** | 7 | 8 | 9 |
| 8 | **6** | **5** | **0** | 7 | 80 | |

| $P_{global}$ | 6 | 5 | 1 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|
| Row/$P_{local}$ | 3 | 2 | 1 | 4 | 5 | 6 |
| 4 | **6** | **0.83** | **-0.16** | | | |
| 5 | 59.25 | -44.37 | 10.125 | | | |
| 7 | **6** | -5 | 1 | 7 | 8 | 9 |
| 8 | **6** | 0 | 1 | 7 | 80 | |

4 elements stored as 4th column of U
2 elements stored as 4th row of L

And so on for rows 5 to 9

**Fig. 2.** The elimination process for a nonsymmetric sparse 9x9-matrix

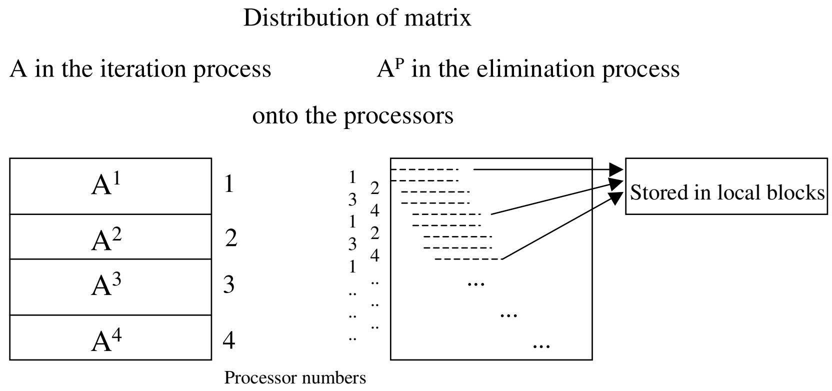


**Fig. 3.** Distribution of matrices $A$ and $A^P$ onto the processors

many zero elements and many unnecessary computations on zero elements are performed. Therefore we are just now implementing the elimination process in a second version so that it can take place on packed rows in the buffer. Nevertheless we will not use index lists for the access to the buffered elements, but will resort the rows stored in the buffer according to the entries of the index list of the pivot row (e.g. in Fig. 2 for pivot row 1 we will drop the zero entry with column index 3 in row 1; this means that the entries of row 3 with the column indices 3 and 1 must be exchanged and the column width of the computation window is reduced by 1). By this strategy the buffer size can be reduced and unnecessary computations on zero elements can be avoided.

### 3.4 Forward elimination and backward substitution

After the elimination process there are the two eliminated matrices $U$ and $L$ ($A^P = ULP$) for nonsymmetric matrices and one eliminated matrix $L$ ($A^P = L^T LP$) for symmetric matrices. The matrix $U$ is a lower triangular matrix and the matrix $L$ is a columnwise permuted upper triangular matrix.

The matrix $U$ is stored in packed columns. As the matrix $L$ is stored in packed rows it must be converted into the storage pattern "packed column". The benefit of the columnwise storing of $U$ and $L$ is that the update of the right hand sides can be done by vector operations. The forward elimination process on $U$ is a "usual" forward elimination and the backward substitution process differs from the "usual" backward substitution by the access to the columns of $L$ via the permutation vector corresponding to $P$. After calling the backward substitution the result vector (e.g. $\tilde{r}$) must be permuted by the above mentioned permutation vector (e.g. $L\tilde{r} = b, r = P\tilde{r}$) (see also Fig. 4).

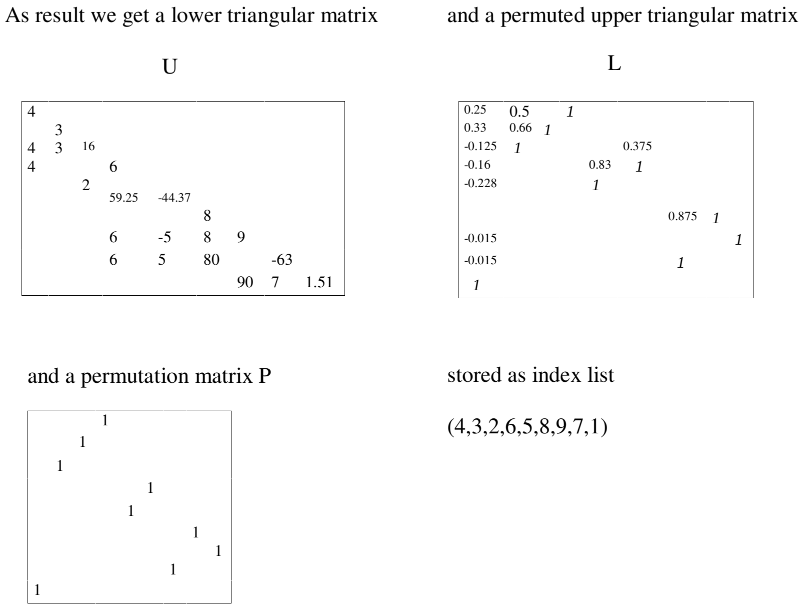


**Fig. 4.** The resulting matrices $U$, $L$ and $P$

The parallelization of these two algorithms is similar to the parallelization of the elimination process. The computed solution component is broadcasted to all other processors and each processor updates the components of the right hand side according to the list of the row indices stored in the matrix for the column corresponding to the index of the solution component.

## 4 Results

This is a report on work in progress. We have designed and implemented all algorithms. Now we are stabilizing the system and doing tests for the complete and incomplete Gaussian elimination. Thus up to the conference we will have many test examples for quite different matrices. We want to mention that the parallel iterative solver package is running stable and that measurements have been published in the papers [1997] and [1998].

At least we want to show that it can be reasonable to use the complete Gaussian algorithm for a bandmatrix even if the implemented iterative solvers could solve the problem. To simplify the model we assume that $\beta_U = \beta_L$ holds ($\beta = \beta_U = \beta_L$); $l$ be the dimension of the matrix. Then one matrix-vector multiplication needs $ops_{iter}(l, \beta) = 2(l(1+\beta) - \beta(\beta+1))$ operations; the Gaussian elimination needs in the best case, if the pivot elements are always the main diagonal elements, $ops_{Gauss}(l, \beta) = (l-\beta)(2\beta^2 - 3\beta + 1) + ((\beta-1)(\beta-2)/2)(1 + (\beta-2)(\beta-1))$ operations. In the worst case the buffer is six times as large as the buffer in the best case; thus the Gaussian elimination needs six times more opertations than in the best case. Now we want to compute the number of matrix-vector multiplications ($mvm$) with $mvm * ops_{iter}(l, \beta) > ops_{Gauss}(l, \beta)$. E.g. for $l = 10000$ and $\beta = l/10 = 1000$ we get $mvm \approx 13000$ and for $l = 10000$ and $\beta = l/100 = 100$ we get $mvm \approx 3$, if we assume the best case for the Gaussian elimination. Thus one can say that for a bandmatrix with a broad band the complete Gaussian elimination never pays. For bandmatrices with smaller bands one has to consider that usually bandmatrices within the band are also sparse; this means that the number of operations for a matrix-vector multiplication must be reduced by the sparsity factor within the band of the bandmatrix and thus $mvm$ must be divided by the sparsity factor. Nevertheless the complete Gaussian algorithm can pay; if we set for example the sparsity factor to 0.1 and take the worst case for the Gaussian algorithm, $mvm$ will be approximately $(3 * 6)/0.1 = 180$; this number says that the complete Gaussian algorithm pays, if the iterative solver needs more than 180 matrix-vector multiplications. Note that the number of matrix-vector multiplications required to achieve a certain accuracy depends on inner properties of the matrix.

## References


[1998] C. Vuik, R.R.P. van Nooyen, P. Wesseling: Parallelism in ILU- preconditioned GMRES, Parallel Computing 24, 1998, pp. 1927-1946, Elsevier Science B.V.

[1997] W. Schönauer, H. Häfner, R. Weiss: LINSOL, a parallel iterative linear solver package of generalized CG-type for sparse matrices, in M. Heath et al. (Eds.), Proceedings of the Eigth SIAM Conf. on Paralel Processing for Scientific Computing, SIAM, Philadelphia 1997, CD-ROM (ISBN 0-89871-395-1), 8 pages.

[1998] H. Häfner, W. Schönauer, R. Weiss: The parallel and portable linear solver package LINSOL, in H. Lederer, F. Hertweck (Eds.), Proceedings of the Fourth European SGI/Cray MPP Workshop, IPP R/46, October 1998, Max-Planck-Institut für Plasmaphysik, Garching bei München, Germany, pp. 242-251.

[1996] R. Weiss: Parameter-Free Iterative Linear Solvers, Akademie Verlag, Berlin 1996, Book (ISBN 3-05-501763-3).

[1976] N.E. Gibbs, W.G. Poole, P.K. Stockmeyer: An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. of Numerical Analysis, Vol. 13, No. 2, 1976, pp. 236-250.