SOLVING SPARSE LINEAR LEAST SQUARES PROBLEMS ON SOME SUPERCOMPUTERS BY USING A SEQUENCE OF LARGE DENSE BLOCKS

P. C. Hansen * Tz. Ostromsky * A. Sameh † Z. Zlatev ‡

Abstract

It is relatively easy to obtain good computational speed on many high-speed computers when computations with dense matrix techniques are performed. This explains why very efficient subroutines have been developed for dense matrix computations. On some computers, as on the new CRAY models, the speed obtained in this situation is near the top-performance given by the manifactures. However, both the number of arithmetic operations and the number of locations needed in the computer memory grow very quickly when pure dense matrix technique is used and when the involved matrices become large. These two facts may cause problems; also when the new modern computers, which are very fast and very big, are used.

Assume that the matrices involved are sparse, and that the sparsity is exploited. Then both the number of arithmetic operations and the number of locations in the computer memory are reduced very considerably, because one operates only with the non-zero elements and stores only the non-zero elements. The reductions could be made more considerable by dropping "small" elements (some iterative procedure has to be used in the latter case in an attempt to regain the accuracy lost by dropping small elements). However, some price is to be paid for the reduction of the number of arithmetic operations and for the reduction of the storage needed. The price is a great degradation of the speed of computations when supercomputers are used (which is due to the use of indirect addresses, to the need to insert new non-zero elements in the sparse storage scheme, to the lack of data locality, etc.).

On many high-speed computers the dense matrix technique is more preferrable than the sparse matrix technique when the matrices are not very large, because the high computational speed compensates fully the disadvantages of using more arithmetic operations and more storage. If the sparse matrices are very large, one can still use dense matrix technique, but the computations must be organized as a sequence of tasks in each of which a dense block is treated. The blocks in this sequence must be large enough to achieve a high computational speed, but not too large, because too large blocks will lead to a very quick increase of both the computing time and the storage). A special algorithm, LORA, must be used to reorder the matrix (before the start of the computations) to a form that allows us to construct a sequence of dense blocks. Then the dense

*UNI•C, The Danish Computer Centre for Research and Education, The Technical University of Denmark, Bldg. 304, DK-2800 Lyngby, Denmark
e-mail: Per.Christian.Hansen@uni-c.dk
e-mail: Tzvetan.Ostromsky@uni-c.dk

†Computer Science Department, University of Minnesota
e-mail: sameh@cs.umn.edu

‡National Environmental Research Institute, Frederiksborgvej 399, DK-4000 Roskilde, Denmark
e-mail: luzz@sun2.dmu.dk

blocks can be treated with some standard software for dense matrices, say LAPACK or NAG.

The implementation of these ideas will be discussed in connection with linear least squares problems. It is assumed that the rectangular matrices (that appear in the linear least squares problems) are decomposed by some orthogonal method. Results obtained on a CRAY C90A computer demonstrate the efficiency of using a sequence of large dense blocks.

AMS Subject Classifications: 65F20, 65F25

Keywords: sparse matrix, general sparsity, orthogonal methods, drop-tolerance, reordering, preconditioned conjugate gradient methods, block algorithm, standard modules, speed-up.

1 STATEMENT OF THE PROBLEM

Linear least squares problems will be discussed in this paper. Such problems can be written in two equivalent forms:

(i) Solve

$$Ax = b - r \qquad \textit{with} \qquad A^T r = 0 \tag{1}$$

or

(ii) Minimize the two-norm of the residual vector

$$r = b - Ax. \tag{2}$$

It is clear that the problem is linear when the first representation is used (the unknown vectors x and r can be found by solving the well-known augmented system). The second representation is traditionally used and it states directly what is the objective of the computations. However, from this representation it is not immediately clear that the problem solved is linear (because the two-norm of a vector is a quadratic function of its coordinates).

The quantities involved in the linear least squares problem described either by (i) or by (ii) are:

$$A \in \mathbf{R}^{m\times n}, \quad x \in \mathbf{R}^{n\times 1}, \qquad b \in \mathbf{R}^{m\times 1}, \quad r \in \mathbf{R}^{m\times 1}. \tag{3}$$

In the whole paper it will be assumed that matrix A satisfies the following four conditions:

1. Matrix A is sparse (i.e. many of its elements are zeros).

2. Matrix A is large (for the computer on which the problem is solved).

3. $m \leq n$ (i.e. it is allowed to have $m = n$; in such a case the linear least squares problem is degenerated to a system of linear algebraic equations).

4. Matrix A has a full column rank.

Since dense matrix subroutines will be used in connection with a sequence of large dense blocks, the last assumption can probably be removed (in such a case rank revealing dense algorithms are to be used), however some technical difficulties may arise in the attempt to accomplish this. On the other hand, it must be emphasized that the fourth assumption is often imposed, but in fact a stronger assumption is needed when no rank revealing numerical procedure is used: one should assume that matrix A is not too close (in some sense) to a rank deficient matrix.

Important areas, in which linear least squares problems are to be handled, are:

1. tomography,

2. geodetic surveys,

3. structural analysis.

This list can be continued. It is important to mention that in the applications vector b contains normally measurements, while vector x contains some physical parameters that are to be determined experimentally.

High-speed computers have to be used when the linear least square problems are large. It is normally very difficult to use such computers in connection with general sparse matrices. Three major difficulties occur:

1. lack of data locality,

2. need of indirect addressing,

3. performance of dynamic transformation of the structure.

These difficulties disappear when dense matrices are to be handled on high-speed computers. The consequence is that the speed of the dense matrix computations is often close to the peak performance of the computer used. The following example can be given to confirm this statement. Consider the transposed matrix **gemat1** from the well-known Harwell-Boeing set of sparse matrices (see **Duff et al., [5]**). The non-zero elements of this matrix have been stored in a two-dimensional (10595 x 4929) array; the empty locations have been filled by zeros. The dense matrix so found has been treated by calling LAPACK (see **Anderson et al., [1]**), the subroutines SGEQRF, SORMQR and STRTRS, on a CRAY Y-MP C90A computer. The results are given in **Table 1**.

High efficiency can also be obtained on other fast computers when dense matrices are to be handled (and, of course, when high quality software, like LAPACK, is used). This is why dense matrix computations are rather popular also when sparse matrices are handled. The multifrontal method is often used. In this method one tries to carry out the computations by treatment a sequence of frontal matrices (which are either dense or are treated as dense); see, for example, **Matstoms, [10]**.

Computing time (in seconds)	511
Speed of computations in MFLOPS	878
Peak performance of the computer	902
Efficiency (in percent)	97%

Table 1
Results obtained when a matrix with 10595 **rows and** 4929
is run on a CRAY Y-MP C90A computer.

An alternative method, in which the same idea is applied (the sparse matrix computations are replaced by dense matrix computations), will be described in this paper. The method is based on an efficient preliminary reordering algorithm (**LORA**; see **van Duin et al., [7]**, **Gallivan et al., [8]**, **Ostromsky et al., [11]**), which allows us to get a sequence of large blocks that are considered as dense (by filling with zeros the empty locations in large two-dimensional arrays). This means that

- **the method is based on a trade off procedure: it is accepted to perform more computations, while the speed of computations is kept high.**

The fact that this strategy is very often successful will be demonstrated by many numerical experiments. It will be shown that a high speed of computations can be achieved when the new method is run on some fast computers. However, it is even more important to emphasize here the fact that the most time-consuming parts of the work are performed by calling subroutines for orthogonalization of dense matrices. This means that it should be expected to obtain good results on any computer for which high quality software for basic linear algebra operations is developed. As a matter of fact, the new method can be considered as templates for efficient treatment of orthogonal decomposition of general sparse matrices on a wide range of modern computers. Some templates for iterative methods for solving systems of linear algebra equations have been proposed in **Barrett et al., [2]**.

The presentation of the results in this paper is organized in the following way. A general description of the algorithm is given in **Section 2**. The major steps of the algorithm are discussed in **Section 3 - Section 7**. Numerical results are given in **Section 8**. Plans for future work on the algorithm are outlined in **Section 9**.

2 GENERAL DESCRIPTION OF THE ALGORITHM

The main objective is to perform the orthogonal decomposition of a large and sparse matrix A by handling a sequence of large dense blocks. The algorithm by which this can be done is sketched in **Table 2**.

Step	Task	Short description of the actions taken during the task
1	**LORA**	**Preliminary reordering of the non-zero elements of the matrix.**
2	**Scatter**	**Determine a large active block. Scatter the non-zeros of this block in a two-dimensional array (filling the appropriate locations with zeros).**
3	**Compute**	**Call some routines which perform dense orthogonal decomposition of the block determined in Step 2.**
4	**Gather**	**Gather the non-zeros of the decomposed block (without the non-zeros of the orthogonal matrix Q) in the original sparse arrays. Numerical dropping can optionally be used. If the remaining active sub-matrix is still sufficiently sparse go to Step 2.**
5	**Last block**	**If the active sub-matrix is relatively dense, then form the last block and switch to dense matrix technique in the classical way.**

Table 2
The five major step of the algorithm for performing orthogonal decomposition of a large sparse matrix by using a sequence of large dense blocks (note that Step 2 - Step 4 are carried out in a loop).

The call of **LORA**, which takes place in Step 1, is crucial. The preliminary reordering performed by this algorithm allows us to create successively large dense blocks and to replace all sparse computations by computations carried out by subroutines for dense orthogonal decomposition (see Section 4 - Section 7). After the call of **LORA** sparse matrix technique is used only to move data from the one-dimensional arrays where the structure of the matrix is kept in condensed form to the two-dimensional arrays which are actually used in the computations (Step 2 and the beginning of Step 5), and from the two-dimensional arrays back to the one-dimensional arrays (Step 4). The main part of the computational work is normally performed in Step 3 and Step 5. This will not be true only in two cases: either if the matrix is very sparse (and the sparsity is preserved very well during the computations) or if there are many processors (in the latter case Step 3 and Step 5 will efficiently be performed in parallel, while the other parts are essentially sequential; at present, at least).

The five major steps sketched in **Table 2** are discused in some more detail in the following five sections.

3 USING LORA FOR RECTANGULAR MATRICES

LORA stands for "locally optimized reordering algorithm". This algorithm has been proposed and studied by **Gallivan et al., [8]** in connection with coarse grain parallel computations during the LU factorization of square sparse matrices. Some improvements of the algorithm (again when it is applied to square matrices) have been proposed in **van Duin et al., [7]**. The same task (achieving coarse grain parallelism) can be solved when rectangular matrices are decomposed by the Givens plane rotations and by using sparse matrix technique only (see **Ostromsky et al., [11]**; the work in this direction is continuing).

LORA reorders the matrix in an upper block triangular form with rectangular diagonal blocks. In fact, there are three types of blocks in the reordered matrix (see also **Fig. 1**:

1. Dense blocks in the separator (the black boxes in **Fig. 1**).
2. Zero-blocks under the separator (the white boxes in **Fig. 1**).
3. Sparse blocks over the separator (the shaded boxes in **Fig. 1**). Some of these blocks may contain only zeros.

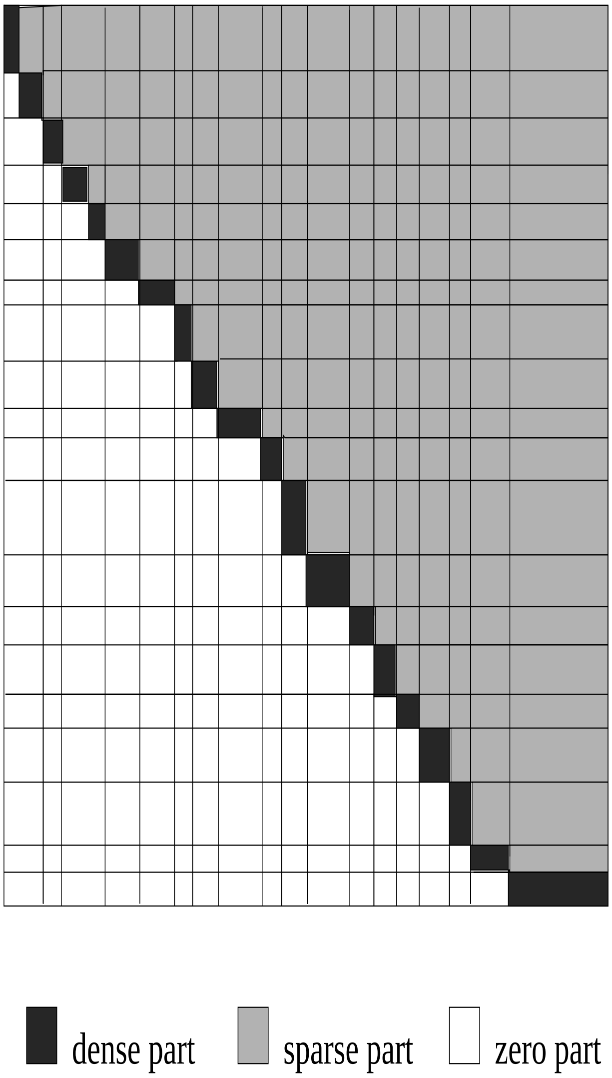

Figure 1
Sparsity pattern of a rectangular matrix obtained by LORA.

LORA attempts to put as many zeros as possible under the separator. Binary trees with leveled nodes are used to achieve this. Let NZ and n be the numbers of non-zeros and columns in matrix A. Then the following theorem holds.

Theorem 1. **LORA** *can be implemented in* $O(NZ\ logn)$ *operations when binary trees with leveled nodes are used.*

Proof. The theorem is proved for square matrices (i.e. when $m = n$) in **Gallivan et al., [8]**; see Theorem 3 in [**8**]. Checking carefully the proof of Theorem 3 as well as the proofs of some previous theorems in [**8**], one can establish that all these proofs hold also when the matrix is rectangular. □

The complexity can be reduced to $O(NZ)$ by using some other techniques (as, for example, some devices used in the codes **MA28** and **Y12M** in connection with some algorithms for dynamical updating the order of the rows in the active sub-matrix during the LU-factorization of a square matrix; see **Duff et al., [4]**, **Duff and Reid, [6]** and **Østerby and Zlatev, [14]**). However, the use of binary trees with leveled nodes allows us to introduce some additional criteria, which can be used in an attempt to improve the quality of the reordered structure. In the present study a secondary criterion, which attempts to put many of the non-zeros in the sparse blocks close to the separator, is applied. This secondary criterion, and several others, will be described elsewhere.

The non-zero elements of the matrix are stored in the first NZ locations of a one-dimensional real array AQR. If a non-zero element a_{ij} is stored in position J, $J = 1, 2, \ldots, NZ$, then its column number j is stored in the same position, position J, of a one-dimensional integer array $CNQR$. There are two integer arrays (of length m) which contain pointers for the row starts and row ends in array AQR. This structure is the same as the structure used in the codes discussed in **Zlatev, [12]** and very similar to that used in **MA28**; see **Duff et al., [4]** and **Duff and Reid, [6]**.

Some additional ordering is performed in order to facilitate some modifications. Consider any two non-zero elements a_{ij} and a_{ik} in row i, $i = 1, 2, \ldots, m$. If $j < k$, then a_{ij} is located, in array AQR, before a_{ik} . This means that not only are the non-zero elements of matrix A ordered by rows in array AQR, but also their column indices are in an increasing order within each row.

The information about the storage of the non-zero elements, which has been sketched above, is quite sufficient for the purposes in this paper; more details can be found in **Duff et al., [4]**, **Duff and Reid, [6]** and **Zlatev, [12]**.

4 SCATTERING THE NON-ZERO ELEMENTS IN A TWO DIMENSIONAL ARRAY

The first task that is to be solved after the call of **LORA** is to determine a large block (that will be treated by using dense matrix technique). An obvious choice is to take the block-row containing the first dense block of the separator (see **Fig. 1**). Assume that p_1 and q_1 are the numbers of rows and columns in the first dense block of the separator. Assume that c_1 is the length of the common sparsity pattern of the rows contained in the block-row corresponding to the first block of the separator. Then the non-zero elements in the first block-row are stored in a two-dimensional array with p_1 rows and c_1 columns. The number of orthogonal stages that are to be performed, by a dense matrix technique in Step 3, is q_1 (it is easy to see that if matrix A has a full column rank, then both $q_1 \leq p_1$ and $q_1 \leq c_1$ are satisfied).

Assume now that the second dense block of the separator contains p_2 rows and q_2 columns. The second block-row can be formed by taking this dense block and adding the relevant "unfinished rows" (after the treatment of the first block-row in Step 3 and Step 4). The total number of unfinished rows is $p_1 - q_1$ but some of them will in general have no non-zero element in the first q_2 columns Therefore the number of relevant unfinished rows is $q_1^* \leq p_1 - q_1$. This means that the second block-row will contain $p_2 + q_1^*$ rows. Assume that c_2 is the length of the common sparsity pattern of the rows contained in the second block-row. Then the non-zero elements of this block-row are stored in a two-dimensional array with $p_2 + q_1^*$ rows and c_2 columns. The number of orthogonal stages that are to be performed, by a dense matrix technique in Step 3, is q_2 (again, it is easy to see that if matrix A has a full column rank, then both $q_2 \leq p_2 + q_1^*$ and $q_2 \leq c_2$ are satisfied).

The process can be continued until all relevant dense blocks of the separator have been treated. However, this is, normally, not a good choice, because the block-rows obtained in this way are rather small. Therefore one has to take several consecutive dense blocks from the separator at a time. It is clear that the technique described above can be used also when several dense blocks from the separator are taken at a time. In both cases (when only one dense block from the separator is taken or when several consecutive dense blocks are united) one has to solve, many times, three major tasks:

1. to determine the the number of rows in the current block-row,
2. to find the common sparsity pattern of the rows in the current block-row,
3. to initialize the two-dimensional array with r_i rows and c_i columns that will be used in Step 3 (where r_i is the number of rows in the i'th block-row of matrix A, while c_i is the common sparsity pattern of the i'th block-row).

The first task is an easy one. One can prescribe some minimal number of rows in a block-rows (say, 100) and take as many dense blocks from the separator as needed to reach this number. There is a danger that the block-rows determined in this way may require only a few stages. Therefore it is worthwhile to introduce another parameter which imposes a requirement for a minimal number of stages (say, 50). By using these two parameters, the minimal number of rows $LROWS$ and the minimal number of stages $LSTAGE$, one can ensure that the work that is to be performed in the next step, Step 3, is not too small. The complexity of this operation is negligible; $O(k_i)$, where k_i is the number of dense blocks from the separator incorporated in the i'th block-row ($i = 1, 2, \ldots, NBLOCK$; $NBLOCK$ being the number of block-rows used in the orthogonal decomposition of matrix A).

The second task is more complicated. However, this is a well-known operation which is often needed in sparse matrix computations and there are fast algorithms for performing this task. The complexity of this operation is $O(NZ_i)$, where NZ_i is the number of non-zeros in the i'th block-row, $i = 1, 2, \ldots, NBLOCK$.

The third task can be performed in an obvious way. Its complexity is $O(r_i c_i)$. This task can be carried out in parallel.

Two remarks are needed here. The first remark is related to the size of the blocks. It is increased when the number of blocks is increased, because the number of unfinished rows is normally gradually increased. Therefore it is necessary to reserve a sufficiently large working space for the two-dimensional arrays.

The second remark concerns the numerical stability of the algorithm. It is well-known that partitioning algorithms often lead to poor stability properties. This is not the case here. The numerical stability of the algorithm (when no dropping is used) is not affected by the creation of large dense blocks.

5 CALLING SUBROUTINES FOR DENSE MATRICES

At the end of Step 2 everything is prepared to call some subroutines for dense orthogonal decomposition. One can use different subroutines (in fact, as mentioned before, the global algorithm can be considered as a preparation of templates for handling large sparse linear least squares problems by using kernels based on dense matrix computations. LAPACK subroutines are currently used (but other subroutines can also be called).

Assume that the i'th block with r_i rows and c_i columns is to be handled. Assume that q_i stages are to be performed. Then the LAPACK subroutine SGEQRF is called to produce zeros under the diagonal elements of first q_i columns of the $(r_i \times c_i)$ dense matrix. After that the LAPACK subroutine SORMQR is called to modify the last $c_i - q_i$ columns of the i'th dense block-row by using the orthogonal matrix Q_i obtained during the call of SGEQRF.

The complexity of the numerical algorithm used during the call of SGEQRF to decompose the i'th block is $O(r_i q_i^2 - q_i^3/3)$. The complexity of the numerical algorithm used in SORMQR is $O(r_i q_i (c_i - q_i) - q_i^2 (c_i - q_i)/2)$.

The comparison of the complexity of the algorithms at Step 3 with the complexity of the algorithms used in the previous steps shows the computational cost of Step 3 is considerably higher than computational cost of the previous steps.

6 GATHERING THE NON-ZERO ELEMENTS IN ONE-DIMENSIONAL ARRAYS

Assume that the third step has successfully been completed for the i'th block. When the non-zero elements (without the non-zero elements of matrix Q_i have to be gathered into the one dimensional sparse array AQR. This operation can cause difficulties. The major problem is that the number of non-zero elements per row is in general changed during the third step. Therefore some operations that are traditionally used in sparse techniques for general matrices (performing copies of rows at the end of array AQR or even garbage collections) must be used during Step 4; the performance of such operations is discussed, for example, in **Duff and Reid, [6]** and **Zlatev, [12]**.

One can try to reduce the number of non-zero elements by dropping non-zero elements which are in some sense small. If dropping is used, then one should try to regain the accuracy lost (because some non-zero elements have been removed) by using the upper triangular matrix R as a preconditioner in a conjugate gradients iterative method. More precisely, the following notation can be introduced:

$$C = (R^T)^{-1} A^T A R^{-1}, \tag{4}$$

$$z = Rx, \tag{5}$$

$$d = (R^T)^{-1} A^T b. \tag{6}$$

It is easily seen that matrix C is symmetric and positive definite. Therefore the conjugate gradient method can be applied to the preconditioned system

$$Cz = d. \tag{7}$$

It must be added here that matrix C is never formed explicitly; one works the whole time with the matrices A and R. It must also be added that the orthogonal matrix Q which is normally rather dense is neither stored nor used in the computations during the iterative process.

This approach is very successful for some matrices, but it should not be used if the matrix is ill-conditioned. Direct methods may work better in the latter case. It should be noted that the storage of orthogonal matrix Q can be avoided also in this situation. One can calculate the vector

$$c = Q^T b \tag{8}$$

during the orthogonal decomposition and then the unknown vector x can be found by solving the system

$$Rx = c. \tag{9}$$

The storage of matrix Q can be avoided even in the case where vector c has not been calculated. In this situation the so-called "semi-normal" system

$$R^T R x = A^T b \tag{10}$$

has to be solved. This may lead to calculating more inaccurate results (comparing with the case where $Rx = Q^T b$ is solved).

7 FINAL SWITCH TO DENSE MATRIX TECHNIQUE

Assume that the first s rows, where $s < min(m, n)$, have been "finished" after the last performance of Step 2 - Step 4. Consider the remaining $m - s \times n - s$ sub-matrix. If the density of the non-zero elements in this matrix (in percent) is greater than some prescribed parameter (say, 30 %), then Step 5 has to be performed (instead of the sequence Step 2 - Step 3 - Step 4) in order to complete the decomposition. In principle, the same work as in Step 2 - Step 3 - Step 4 has to be carried out. However, two simplifications are used to reduce the amount of computational work:

1. There is no need to determine the common sparsity pattern of the rows involved in this step.
2. No gather step (no Step 4) is needed after the calculation of the dense orthogonal decomposition of the last block.

It is important to find out when to switch to Step 5. Experimentally it has been established that the switch when the density of the active matrix becomes greater than 30%

is a good choice; some numerical results, which confirm this statement, will be presented in the next section.

8 SOME NUMERICAL RESULTS

Numerical results will be presented in this section. Most of the results have been obtained on a **CRAY Y-MP C90A** computer. The code is now run on other computers. Some preliminary results indicate that the same (or, at least similar) conclusions can be drawn when other computers on which LAPACK is efficiently implemented are in use. However, much more experiments are needed in order to give a precise answer to the question: "When is the new algorithm most efficient". Such systematic investigations will be carried out in the future. In this section, it will be shown that the new algorithm is rather efficient when it is run on **CRAY Y-MP C90A**.

8.1 Matrices used in the runs

Rectangular matrices from the well-known Harwell-Boeing set of sparse matrices (see **Duff et al.,** [**5**]) have been used in the runs. Some characteristics of these matrices are listed in **Table 3**. It should be mentioned that matrix **gemat1** is in fact the transposed matrix of the matrix with the same name in the Harwell-Boeing set. Furthermore, the two matrices **amoco1** and **bellmedt** have been proposed by **M. Berry,** [**3**].

Matrix	Rows	Columns	Non-zero
abb313	313	176	1557
ash219	219	85	438
ash331	331	104	662
ash608	608	188	1216
ash958	958	292	1916
illc1033	1033	320	4732
well1033	1033	320	4732
illc1850	1850	712	8758
well1850	1850	712	8758
amoco1	1436	330	35210
bellmedt	5831	1033	52012
gemat1	10595	4929	47369

Table 3
The rectangular Harwell-Boeing matrices used in the runs.

Only the sparsity pattern of the non-zero elements is given for the first five matrices in **Table 3**. Non-zero elements have been produced for these matrices by using a random number generator.

The code has also been run with square matrices from the Harwell-Boeing set; 27 such matrices have been selected (the same matrices have been extensively used in **Zlatev,** [**12**]).

The Harwell-Boeing sparse matrices are rather small (both the rectangular matrices and the square matrices) for the modern high-speed computers. Therefore some large sparse matrices created by one of the sparse matrix generators proposed in **Zlatev et al., [13]**, see also **Zlatev, [12]**. have also been used in the runs.

It is desirable to be able to check the accuracy of the computed solution. Therefore the right-hand-side vector b has been created (for all matrices used) with a vector x all components of which are equal to one (i.e. $x_i = 1$ for $i = 1, 2, \ldots, n$).

8.2 Experiments with rectangular Harwell-Boeing matrices

Only the last two matrices in **Table 3** are sufficiently large for **CRAY Y-MP C90A**. Some results obtained when these matrices has been run will be discussed in this sub-section.

The new algorithm (based on the use of large dense blocks) has been compared with three other algorithms:

1. **Totally dense**. In this algorithm the matrix is stored in a two-dimensional array (the empty locations being filled with zeros), and the subroutines of package LAPACK are directly used to solve the problem.
2. **Partially sparse**. This algorithm is similar to the algorithm based on the use of large dense blocks. The large dense blocks are created in the same way. When these large blocks are scattered in two-dimensional arrays, the empty locations are filled with zeros. Some of these zeros may be located in positions where one must create zeros. Therefore instead of calling LAPACK routines (where this fact is totally ignored), some routines based on the use of plane rotations for dense rows are used. A plane rotation is performed only if both leading elements in the two involved rows are non-zeros. Some computations may be saved in this way (it is not necessary to perform a plane rotation if at least one of the leading elements is equal to zero), but the speed of computations is not so high as in the case where LAPACK routines are called.
3. **Pure sparse**. No attempt to create dense blocks is made. Subroutines of the package described in **Zlatev, [12]** are called when this option is specified.

Numerical results obtained by the new algorithm and by the three additional algorithms sketched above are given in **Table 4** for the matrix **gemat1** and in **Table 5** for the matrix **bellmedt**.

Characteristics measured	Totally dense	Dense blocks	Partially sparse	Pure sparse
Computing time	511	28	10	34
MFLOPS	878	662	196	4
Peak performance	902	902	902	902
Efficiency	97%	73%	22%	0.4%
Number of blocks	1	47	49	-

Table 4
Results obtained when matrix gemat1 has been run by four algorithms: "Totally dense" refers to the direct use of LAPACK when matrix gemat1 is stored as dense. "Dense blocks" refers to the algorithm studied in this paper. Dense matrix computations are carried out by plane rotations when the "Partially sparse" method is used, but a plane rotation is performed only when both leading elements are non-zeros. No dense matrix computations are used in the "Pure sparse" method.

Characteristics measured	Totally dense	Dense blocks	Partially sparse	Pure sparse
Computing time	14	8	16	1490
MFLOPS	870	678	458	6
Peak performance	902	902	902	902
Efficiency	96%	75%	51%	0.7%
Number of blocks	1	8	8	-

Table 5
Results obtained when matrix bellmedt has been run by four algorithms: "Totally dense" refers to the direct use of LAPACK when matrix gemat1 is stored as dense. "Dense blocks" refers to the algorithm studied in this paper. Dense matrix computations are carried out by plane rotations when the "Partially sparse" method is used, but a plane rotation is performed only when both leading elements are non-zeros. No dense matrix computations are used in the "Pure sparse" method.

Characteristics measured	gemat1	bellmedt
Non-zeros in A	47369	52012
Max. non-zeros kept	295526	1612665
Non-zeros in R	266824	519521

Table 6
The number of non-zero elements in the original matrix A, the maximal number of non-zero elements kept in the one-dimensional array AQR (found at any stage of the orthogonal decomposition) and the number of non-zero elements in the upper triangular matrix R.

The main conclusions which can be drawn from the experiments (not only the two experiments with the matrices **gemat1** and **bellmedt**) can be summarized as folloes:

1. If the matrix is relatively small (several thousand rows and no more than about thousand columns when the **CRAY Y-MP C90A** computer is used), then the direct use of LAPACK, ignoring the sparsity (the "Totally dense" method) is quite competitive with the methods based on the use of a sequence of large dense blocks (the "Dense blocks" method); see **Table 5**. The situation changes when the matrix become large; this is demonstrated in **Table 4**.

2. The speed of computations and the efficiency of the "Dense blocks" method are quite good (see the results given in **Table 4** and **Table 5**).

3. The method based on "Partial sparsity" performs better than the method based on "Dense blocks" for **gemat1**. The opposite is true for **bellmedt**. The results displayed in **Table 6** explain why this is so. Matrix **bellmedt** is denser than matrix **gemat1**: it contains more non-zero elements in the beginning although it is smaller. Moreover, the number of intermediate fill-ins in **bellmedt** is much greater than that in **gemat1**. A very crude estimation of the number of intermediate fill-ins can be obtained by taking the difference between the corresponding numbers in the second and third rows in **Table 6**. It is seen that this estimation gives a number less than 30000 for matrix **gemat1**, while the corresponding number is greater than one million for matrix **bellmedt**.

4. The "Pure sparse" method is quite competitive with the "Dense blocks" method for **gemat1**, while the use of this method is very bad for **bellmedt**. The large number of intermediate fill-ins produced when **bellmedt** is decomposed (about 30 times greater than that produced when **gemat1** is treated) explains the reason for this effect; see also **Table 6** and the explanations given in the previous paragraph.

The general conclusion, drawn by using results from many other runs, is that the application of the algorithm based on the use of a sequence of large dense blocks is often the best one. If this is not true, then the difference is not very big and, what is even more important, a big amount of work is normally to be used to optimize the "Partially sparse" code in the transition from one high-speed computer to another. The use of a sequence of large dense blocks is in fact equivalent to using templates. Therefore one should expect to achieve good results when the computer is changed if the dense subroutines are optimized (and this is certainly true for the LAPACK modules). The conclusion is that the use of a sequence of large dense blocks should be preferred when a standard package, which gives good results on many different high-speed computers, is needed. This is why only the algorithm based on the use of a sequence of large dense blocks will be discussed in the following part of this paper.

8.3 When to switch to Step 5

Step 5 in the algorithm presented in Section 2 is cheaper than one sweep in the loop containing the sequence Step 2 - Step 3 -Step 4 (because some simplifications are made during the treatment of this step; see Section 7). Therefore it is worthwhile to investigate when to switch. Many runs have been performed in an attempt to find a good criterion for switching to Step 5. Some results obtained with the two largest Harwell-Boeing matrices used in the previous sub-section are given in **Table 7**.

Percent	gemat1	bellmedt
5%	90	13
10%	51	11
20%	37	11
30%	28	10
40%	30	9
50%	30	9
60%	30	8
70%	30	8

Table 7
Computing times achieved for different switches to Step 5.

The major conclusions from the runs with these two matrices (and from some other runs) are:

1. The switch to Step 5 is as a rule not very critical, but the density of the non-zero elements in the active sub-matrix, which is used in the switching criterion, should not be too small; the computing times obtained if the switch to Step 5 is performed when the density of the non-zero elements in the active sub-matrix is only 5% are normally rather large (see **Table 7**).
2. The results are rather good also when the switch to Step 5 is delayed (even until the density of the non-zero elements in the active sub-matrix becomes as high as 70%).
3. The results on other computers may be different.

8.4 Determination of the sizes of the blocks

Two parameters, $LROWS$ and $LSTAGE$, are currently used in the determination of the sizes of the blocks. A lower limit of the rows in any block is introduced by $LROWS$. A lower limit of the number of stages that is to be performed in any block is imposed by $LSTAGE$. These two parameter must be selected carefully. Some larger matrices are needed in the experiments concerning the choice of the parameters $LROWS$ and $LSTAGE$. The Harwell-Boeing matrices are too small (the only exception being **gemat1**) for this purpose. Therefore matrices of class F2 (see **Zlatev et al., [13]**, and **Zlatev, [12]**) have been used. Five parameters can be selected when the matrix generator for matrices of class F2 is used:

1. the number of rows (parameter M),
2. the number of columns (parameter N),
3. the sparsity pattern (parameter C),

4. the density of the non-zero elements (parameter R; R being the average number of non-zero elements per row),

5. the condition number of the matrix (parameter $ALPHA$; the condition number is normally increased when $ALPHA$ is increased).

First the matrices of class F2 will be used to test the choice of $LSTAGE$. The parameters chosen in this experiment are: $M = 500000$, $N = 30000$, $C = 15000$ (this means that the diagonals, whose first elements are $a_{1,15000}$, $a_{1,15001}$ and $a_{1,15002}$ contain non-zero elements), $R = 10(10)100$ and $ALPHA = 1$ (this means that the matrices created are not very ill-conditioned). The parameter $LROWS$ is kept fixed; $LROWS = min(500, M/5)$. The parameter $LSTAGE$ has been varied; $LSTAGE = min(k, N/10)$, where $k = 25(25)200$. It is clear that $LROWS = 500$ and $LSTAGE = k$ when M and N are sufficiently large. It should also be stressed that the limit $LSTAGE$ has been dynamically changed in this experiments: if the current dense block contains more than $3 * LROWS$ rows, then $LSTAGE$ is reduced by five until $LSTAGE$ becomes equal to 25 (i.e. it is not allowed to decrease further $LSTAGE$ when it becomes 25). Some of the results are presented in **Table 8**.

Density	$k = 25$	$k = 50$	$k = 75$	$k = 100$	$k = 150$	$k = 200$
$R = 10$	**9**	**9**	**10**	**12**	**10**	**17**
$R = 20$	**14**	**15**	**15**	**16**	**20**	**23**
$R = 30$	**21**	**21**	**20**	**22**	**26**	**34**
$R = 40$	**33**	**31**	**30**	**33**	**43**	**46**
$R = 50$	**44**	**46**	**44**	**45**	**51**	**59**
$R = 60$	**60**	**61**	**64**	**59**	**65**	**70**
$R = 70$	**71**	**77**	**74**	**78**	**81**	**86**
$R = 80$	**90**	**93**	**88**	**95**	**97**	**97**
$R = 90$	**108**	**108**	**105**	**107**	**115**	**118**
$R = 100$	**125**	**129**	**124**	**130**	**127**	**134**

Table 8
Computing times achieved in the decomposition of ten matrices ($M = 50000$, $N = 30000$ and $R = 10(10)100$) with several minimal values of $LSTAGE$; $LSTAGE = min(k, N/10)$.

The major conclusions from the runs with these ten matrices is that the choice of a lower limit of the number of stages that are to be performed is not very critical. There seems to be a slight trend of increasing the computing time when $LSTAGE$ is increased. For large values of R, the relative differences are negligible.

The matrices of class F2 are also used to test the choice of $LROWS$. The parameters used in this experiment are: $M = 100000$, $N = 40000(10000)100000$, $C = 20000$, $R = 15$, $ALPHA = 1$, $LROWS = min(k, M/5)$ with $k = 300(100)700$ and $LSTAGE = min(100, N/10)$. Thus, seven matrices are used (the number of columns is gradually increased, while the number of rows is kept fixed; the last matrix is square). The number of non-zero elements in any of the seven matrices is also fixed: $NZ = R * M + 110 = 1500110$. Since both the rows and the non-zero elements are kept constant, the computational work depends mainly on the preservation of sparsity (and, more precisely, on the success of **LORA** to find a good preliminary ordering). Some results are given in **Table 9**.

Columns	$k = 300$	$k = 400$	$k = 500$	$k = 600$	$k = 700$
$N = 40000$	**23**	**23**	**23**	**23**	**23**
$N = 50000$	**52**	**61**	**61**	**60**	**61**
$N = 60000$	**32**	**32**	**31**	**32**	**32**
$N = 70000$	**57**	**54**	**53**	**53**	**54**
$N = 80000$	**29**	**30**	**30**	**30**	**30**
$N = 90000$	**41**	**42**	**42**	**43**	**44**
$N = 100000$	**22**	**25**	**34**	**24**	**30**

Table 9
Computing times achieved in the decomposition of seven matrices of order $M = 100000$ **with several minimal values of** $LROWS$; $LROWS = min(k, M/5)$.

It is seen that also the choice of $LROWS$ is not very critical. In many experiments the computing times are the same for all five values used. In all other experiments $LROWS = min(500, M/5)$ has been used.

8.5 Running some very large problems

Subroutines from package LAPACK can be used directly (i.e. the sparsity can be neglected) when the Harwell-Boeing rectangular sparse matrices (see **Table 3**) are to be run (only for the largest of these matrices, **gemat1**, the results will not be very efficient; see **Table 4**). Therefore it is interesting to try the new method on some very large matrices, for which the subroutine from LAPACK cannot be directly applied. Results from such runs are shown in **Table 10**.

Matrix identifiers			Computing times		
M	**N**	**C**	$R = 5$	$R = 10$	$R = 15$
50000	**25000**	**15000**	**10**	**18**	**35**
100000	**50000**	**30000**	**18**	**34**	**60**
150000	**75000**	**45000**	**27**	**40**	**76**
200000	**100000**	**60000**	**36**	**54**	**74**
250000	**125000**	**75000**	**46**	**73**	**109**
300000	**150000**	**90000**	**59**	**81**	**107**
350000	**175000**	**105000**	**70**	**98**	**141**
400000	**200000**	**120000**	**82**	**117**	**167**
450000	**225000**	**135000**	**88**	**125**	**162**
500000	**250000**	**150000**	**100**	**142**	**179**

Table 10

Computing times spent for the decomposition of 30 matrices ($ALPHA = 1$). The number of non-zero elements is $NZ = R * M + 110$.

It is easily seen that there is no difficulties with the computing times needed on **CRAY Y-MP C90A** (the computing times are small even when linear least square problems with half million equations are solved). However, if the problems become very big (bigger than half million equations) there arise difficulties with the storage that is needed (difficulties arise also when the equations are less than half a million, but the matrices are not very sparse; for the matrices of class F2, when $R > 20$). This observation indicates that some efforts are needed in order to reduce the storage requirements.

8.6 Comparing the new method with a multifrontal technique

Multifrontal techniques are rather popular for general sparse matrices. While the main idea used in the two approaches, the method based on the use of large dense blocks and the multifrontal techniques, is the same (an extensive use of dense matrix technique computations), the methods, by which this is achieved, are different. In the method discussed in this paper, the construction of **large** dense blocks is the main purpose (the main ordering procedure, **LORA**, is called in the very beginning; before the start of the actual computations). In the multifrontal technique for rectangular matrices, one uses the non-zero structure of $A^T A$ to "compute an elimination tree with nodes associated to small and and dense matrices" (**Matstoms, [10], p. 20**). In other words, in the former method one works the whole time with the sparsity pattern of matrix A, while the sparsity pattern of $A^T A$ is used in the multifrontal methods for rectangular matrices. Moreover, in the former method the emphasis is on forming a sequence of large dense blocks, while the classical use of the multifrontal methods leads to small dense matrices.

It is necessary to try to compare the results obtained by using large dense blocks with some multifrontal method. The multifrontal code **qr27**, which is described in **Matstoms, [10]**, has been chosen as a typical representative of the multifrontal methods for rectangular matrices. Results from some comparisons are given in **Table 11**.

Density	$k = 75$	**QR27**
$R = 10$	**10**	**15 (1.5)**
$R = 20$	**15**	**34 (2.3)**
$R = 30$	**20**	**64 (3.2)**
$R = 40$	**30**	**104 (3.5)**
$R = 50$	**44**	**153 (3.5)**
$R = 60$	**64**	**211 (3.3)**
$R = 70$	**74**	**280 (3.8)**
$R = 80$	**95**	**356 (3.7)**
$R = 90$	**105**	**444 (4.2)**
$R = 100$	**124**	**526 (4.2)**

Table 11
Computing times achieved in the decomposition of ten matrices ($M = 50000$, $N = 30000$ and $R = 10(10)100$) by the method discussed in this paper with

$LSTAGE = min(k, N/10)$ and a multifrontal method. Note that while the dimension of the matrices remains the same, the number of non-zeros, $NZ = R * M + 110$, is increased, when R becomes larger.

The results indicate that the method based on the use of large dense blocks is at least competitive with the multifrontal methods. Much more experiments (also experiments on other high-speed computers are needed in order to find classes of matrices for which one of the techniques will consistently give better results. It is intuitively clear that if the matrix is very sparse and remains very sparse during the computations, then the multifrontal techniques will perform better (the other method will carry out many unnecessary operations in this case, because there are many zeros in the sequence of large dense blocks), while if this is not the case, then the method proposed in this paper will give better results. This trend (the method based on the use of a sequence of large dense blocks becomes better when the matrix becomes denser; for very dense matrices, obtained by setting R to two or three, the multifronral technique becomes the better choice) is seen in **Table 11**, but it must be reiterated here that much more experiments are needed in order to be able to draw some more definite conclusions.

8.7 Using preconditioned conjugate gradients

An approximate matrix R (obtained by dropping some small non-zero elements when the non-zero elements are gathered in the sparse arrays; Step 4) can be used as a preconditioner in a preconditioned conjugate gradients method; see Section 6. Sometimes this will lead to considerable reductions of the computing time and/or the storage needed. Some results obtained when this approach is applied are given in **Table 12** (for the matrices used in **Table 8** and **Table 11**) and in **Table 13** (for the Harwell-Boeing matrix **bellmedt**).

Density	Direct	Iterative	Eval.error	Exact error
$R = 10$	**10**	**6.9 (0.9, 5)**	**1.1E-08**	**4.5E-10**
$R = 20$	**15**	**8.4 (1.2, 7)**	**4.9E-08**	**1.2E-09**
$R = 30$	**20**	**10.1 (1.3, 7)**	**1.3E-08**	**2.4E-09**
$R = 40$	**30**	**11.1 (1.3, 7)**	**7.9E-08**	**1.5E-10**
$R = 50$	**44**	**12.6 (1.4, 7)**	**2.9E-08**	**6.1E-10**
$R = 60$	**64**	**14.4 (2.1,10)**	**2.1E-08**	**8.6E-10**
$R = 70$	**74**	**16.0 (2.9,13)**	**2.4E-08**	**1.3E-09**
$R = 80$	**95**	**18.3 (3.0,13)**	**2.3E-08**	**1.4E-09**
$R = 90$	**105**	**20.1 (3.1,13)**	**2.4E-08**	**1.4E-09**
$R = 100$	**124**	**24.2 (5.5,23)**	**1.1E-08**	**4.0E-11**

Table 12

Results obtained in running the ten matrices from Table 11. The computing times obtained by the direct method with $LSTAGE = min(75, N/10)$ are given in the column under "Direct". The total computing times obtained by the iterative method are given in the column under "Iterative" (the computing times spent during the iterative procedure and the numbers of iterations being given in brackets). The two-norms of the evaluated by the code eror

vector and the exact error vector are given in the last two columns. The accuracy required was $ACCUR = 10^{-7}$. The drop-tolerance used to calculate the preconditioner R was $TOL = 0.0625$.

It is seen that the reductions of the computing time achieved when the preconditioned conjugate gradient method is used are considerable (up to a factor of five). Also the storage needed can be reduced considerably (and some very large problems, which cannot be solved by the direct method, can be solved when the preconditioned conjugate gradients are used). However, if the matrices are very ill-conditioned then the preconditioned iterative method begins to converge when the drop tolerance becomes very small, and it is better to apply the direct method (this happens for the Harwell-Boeing matrix **gemat1**).

The choice of an optimal value of the drop-tolerance may be difficult. The next experiment shows, however, that in some cases it is not difficult to choose a good value of the drop-tolerance (such a value by which the preconditioned conjugate gradients will give better results than the simple use of the direct method).

TOL	Fact.	Sol.	Iters	Eval.error	Exact error
2^{-1}	1.6	1.39	113	9.5E-08	3.7E-09
2^{-2}	5.0	0.49	36	8.4E-08	9.0E-09
2^{-3}	6.8	0.33	25	6.8E-08	1.9E-08
2^{-4}	7.9	0.26	18	5.9E-08	1.6E-08
2^{-5}	9.4	0.22	14	3.6E-08	4.5E-08
2^{-6}	8.3	0.16	10	9.8E-08	3.6E-08
2^{-7}	9.1	0.13	8	1.0E-08	2.2E-10
2^{-8}	10.1	0.10	5	1.8E-08	4.9E-10
2^{-9}	10.1	0.09	4	8.4E-09	2.3E-11
2^{-10}	10.1	0.07	3	2.8E-08	6.7E-11
2^{-11}	10.1	0.06	3	3.8E-08	6.3E-11
2^{-12}	10.1	0.06	3	6.0E-09	3.2E-12
2^{-13}	10.1	0.06	2	3.3E-11	2.6E-13
2^{-14}	10.1	0.04	2	1.7E-08	2.7E-11
2^{-15}	10.1	0.06	2	2.3E-09	2.8E-13
2^{-16}	10.1	0.04	2	7.3E-10	2.4E-13

Table 13

Results obtained in running the Harwell-Boeing matrix bellmedt with 16 values of the drop-tolerance TOL. The factorization times, the solution times and the numbers of iterations are given in the columns under "Fact.", "Sol." and "Iters" respectively. The two-norms of the evaluated by the code eror vector and the exact error vector are given in the last two columns. The accuracy required was $ACCUR = 10^{-7}$.

The set of stopping criteria as well as **the use** of the stopping criteria are two critical issues when preconditioned conjugate gradient methods are used. Their importance (especially the importance of the way in which the stopping criteria are used) is often underestimated when different preconditioned algorithms are defined. The most important stopping criterion chosen is

(11) $$\frac{|\alpha_i| \|p_i\|}{1 - RATE_i} \le ACCUR,$$

where $ACCUR$ is some accuracy prescribed the user, $RATE_i$ is the current evaluation of the convergence rate and $\alpha_i p_i$ is coming from the formula by which the solution vector is updated in the conjugate gradient method (and in many conjugate gradient type methods):

(12) $$x_{i+1} = x_i + \alpha_i p_i.$$

While it is well known that the error evaluation at the i'th iteration should be performed only when $0 < RATE_i < 1$, it is not well known that one should also study very carefully the behaviour of parameter α_i during the iterative process. This parameter:

1. should not increase too quickly,
2. should not decrease too quickly,
3. should not oscillate too much.

The behaviour of α_i is checked by comparing the last four values of this parameter. The criterion defined by (12) is actually used to evaluate the current error in the solution vector **only if** the comparison indicates that the three conditions stated above are satisfied. No attempt to evaluate the error at the current step is made when the comparison indicates that at least one of the above three conditions is not satisfied. This is a rather conservative stopping criterion, but the experiments show that it is rather reliable (see more details in **Zlatev, [12]**). It should be added that there are some additional stopping criteria and that the stopping criterion (12) is slightly modified in order to obtain an estimation of the relative error.

8.8 Preliminary results obtained on another high-speed computer

All results presented in the previous sub-sections have been obtained on a **CRAY Y-MP C90A** computer. Several runs on a **POWER CHALLENGE** computer from Silicon Graphics have been performed. Some results are given in **Table 14**.

Processors	Comp. time	Speed-up	MFLOPS	Efficiency
1	416	-	167	64%
2	266	1.7	262	50%
4	157	3.3	443	43%
8	94	4.4	741	36%

Table 14
Some results obtained on a POWER CHALLENGE computer from Silicon Graphics when a problem whose coefficient matrix is gemat1 is run.

It should be stressed that the results have been obtained by just taking the **CRAY** version of the sparse code ("the sparse templates") and calling the LAPACK routines available at the **POWER CHALLENGE** computer. From this point of view, the results are quite satisfactory: it is seen that the main idea (to obtain standard modules that perform reasonably well on many high speed computers on which the LAPACK modules perform well) seems to work (we get reasonably good times, speed-ups, MFLOPS and efficiency). On the other hand, it should also be emphasized that the results could be improved by tunning the sparse code according to the requirements on this computer.

9 CONCLUSIONS AND PLANS FOR THE FUTURE WORK

The most important task which has to be done in the future is to check systematically the efficiency of the code on different high-speed computers; especially computers with shared memory and computers with distributed memory (on many computersi with distributed memory **PVM**, see **Geist et al., [9]**, could be applied). The work on this task has been started.

The work has been optimized at Step 3 and at Step 5. This is natural, because these steps are the most time-consuming parts of the computational work. The other steps (Step 1, Step 2 and Step 3) are at present carried out mainly in a sequential mode. Some attempts to improve the efficiency of these parts (and, thus, the efficiency of the whole code) will also be carried out in the future. This will be an important issue for computers with many processors (and for matrices that are very sparse and remain very sparse during the orthogonal decomposition).

Some attempts to improve the performance of the preconditioned conjugate gradients are necessary. Until now most of the work in this part is carried out sequentially. Even this version is in some cases very efficient; see the results given in Sub-section 8.7. The performance can be improved (and the class of matrices for which the conjugate gradients perform better than the direct method can be increased) if parallel and vector computations are achieved in this part of the solution process.

It seems to be necessary to make some efforts to reduce the storage requirements. This is especially needed when very large problems are to be treated.

The main conclusion is that although the methods will not always give best results it is **a standard tool for solving efficiently large sparse linear least squares problems** and will produce good results on all high-speed computers on which the LAPACK modules gives good results. It should also be emphasized that other modules for handling dense matrices can also be used. Thus, the method can be considered also as templates for efficient treatment of general sparse matrices on different types of high-speed computers.

ACKNOWLEDGEMENTS

This research was partially supported by the BRA III Esprit project **APPARC** (# 6634) and by NATO (ENVIR.CGR 930449).

Dr. Guodong Zhang from the Application Group at Silicon Graphics sent us the last updated version of the LAPACK routines tunned on POWER CHALLENGE. We should

like to thank him very much.

References

[1] **E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen:** *"LAPACK: Users' guide"*. SIAM (Society for Industrial and Applied Mathematics, Philadelphia, **1992**

[2] **R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst:** *"Templates for the solution of linear systems: Building blocks for iterative methods"*. SIAM, Philadelphia, **1994**.

[3] **M. Berry:** *Private communication.* Department of Computer Science, University of Tennessee, Knoxville, Tennessee, TN 37996-1301, **1994**.

[4] **I. S. Duff, A. M. Erisman, and J. K. Reid:** *"Direct methods for sparse matrices"*. Monographs on Numerical Analysis, Oxford University Press, London-Oxford, **1986**.

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

[6] **I. S. Duff and J. K. Reid:** *"Some design features of a sparse matrix code"*. ACM Trans. Math. Software, **5(1979)**, 18-35.

[7] **A. C. N. van Duin, P. C. Hansen, Tz. Ostromsky, H. Wijsoff and Z. Zlatev:** *"Improving the numerical stability and the performance of a parallel sparse solver"*. Computers and Mathematics with Applications; to appear.

[8] **K. Gallivan, P. C. Hansen, Tz. Ostromsky and Z. Zlatev:** *"A locally optimized reordering algorithm and its application to a parallel sparse linear system solver"*. Computing, **54(1995)**, 39-67.

[9] **A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam:** *"PVM 3 user's guide and reference manual"*. Report No. ORNL/TM-12187. Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, USA, **1994**.

[10] **P. Matstoms:** *"The multifrontal solution of sparse linear least squares problems"*. Thesis. Department of Mathematics, Linköping University, Linköping, Sweden, **1991**.

[11] **Tz. Ostromsky, P. C. Hansen and Z. Zlatev:** *"A parallel sparse QR-factorization algorithm"*. In: **"Applied Parallel Computing in Physics, Chemistry and Engineering Science"** (J. Dongarra, K. Madsen and J. Waśniewski, eds). Springer-Verlag, Berlin; to appear.

[12] **Z. Zlatev:** *"Computational methods for general sparse matrices"*. Kluwer Academic Publishers, Dordrecht-Toronto-London, **1991**.

[13] **Z. Zlatev, J. Wasniewski and K. Schaumburg:** *"A testing scheme for subroutines solving large linear systems"*. Comput. Chem., **5(1981)**, 91-100.

[14] **O. Østerby and Z. Zlatev:** *"Direct methods for sparse matrices"*. Springer, Berlin, **1983**.