Using dense matrix computations in the solution of sparse problems

Tz. Ostromsky[1] and Z. Zlatev[2]

[1] Central Laboratory for Parallel Information Processing,
Bulgarian Academy of Sciences,
Acad. G.Bonchev str., bl. 25-A, 1113 Sofia, Bulgaria;
e-mail: ceco@iscbg.acad.bg
[2] National Environmental Research Institute,
Frederiksborgvej 399, P. O. Box 358, DK-4000 Roskilde, Denmark;
e-mail: luzz@sun2.dmu.dk

Abstract. On many high-speed computers the dense matrix technique is preferable to 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. Dense matrix techniques can still be used if the computations are successively carried out in a sequence of large dense blocks. A method based on this idea will be discussed.

1 Statement of the problem

Consider: $Ax = b - r$ with $A^T r = 0$, where $A \in \mathbf{R}^{m\times n}$, $m \geq n$, $rank(A) = n$, $b \in \mathbf{R}^{m\times 1}$, $r \in \mathbf{R}^{m\times 1}$ and $x \in \mathbf{R}^{n\times 1}$. It is difficult to use efficiently high-speed computers for solving the above problem if matrix A is general sparse. For dense matrices all difficulties disappear and the speed of the dense matrix computations is normally close to the peak performance of the computer used; see Table 1.

matrix gemat1 (10595×4929)	
Computing time (in seconds)	511
Speed of computations in MFLOPS	878
Peak performance of the computer	902
Efficiency (in percent)	97%

Table 1. Results on CRAY C92A by using LAPACK dense subroutines.

A new method will be described. The method is based on a reordering algorithm LORA, [3], [4], which allows us to form easily a sequence of relatively large blocks. The blocks are handled as dense matrices. This is **a trade off procedure: it is accepted to perform more computations, but with higher**

speed. All computations are performed by dense kernels (LAPACK subroutines, [1] are used at present, but other dense subroutines may also be applied). The dense kernels perform the most time-consuming part of the work; Table 3. Therefore, it should be expected to obtain good results on any computer for which high quality software for dense matrices is available.

The new method is described in Section 2. Numerical results, obtained on CRAY C92A and POWER CHALLENGE (Silicon Graphics) are given in Section 3. Plans for future work on the algorithm are outlined in Section 4.

2 Description of the algorithm

The main objective is to compute the QR-factorization of A by applying dense orthogonal transformations to a sequence of large blocks. The algorithm consists of five steps: (i) reordering by LORA, (ii) scatter, (iii) compute, (iv) gather and (v) deal with the last block. The actions performed during the five steps are discussed in §2.1 - §2.5)

2.1 Using LORA for rectangular matrices

LORA ("locally optimized reordering algorithm"; [3, 4]) reorders the matrix to a block upper triangular form (Fig. 1) with an important additional requirement to put as many zeros as possible under the diagonal blocks ("the separator").

The complexity of LORA is $O(NZ \ \log \ n)$ assuming that A has NZ non-zeros and n columns. The complexity can be reduced to $O(NZ)$. The more expensive version allows us to introduce additional criteria, [3], by which the quality of the ordering is improved. A criterion that puts more non-zeros close to the diagonal blocks is applied here.

2.2 Scattering the non-zeros in a two dimensional array

The next task is to form large dense block-rows. It is easier to explain the main ideas by taking the simplest (but not the best) case: each block-row is formed by taking a dense block of the separator (Fig. 1). If the first dense block of the separator contains r_1 rows and q_1 columns, then the non-zeros in the first block-row are stored in a two-dimensional array with r_1 rows and c_1 columns, where c_1 is the union of the sparsity patterns of its r_1 rows. The number of orthogonal stages is q_1 ($rank(A) = n$ implies $q_1 \leq r_1$).

Assume that the second dense block of the separator contains r_2 rows and q_2 columns. The second block-row can be formed by taking this dense block and adding the relevant "unfinished rows" from the first block-row. The total number of unfinished rows is $r_1 - q_1$ but there may be some with only zeros in the first q_2 columns. Therefore the number of relevant (when the second block is treated) unfinished rows is $r_1^* \leq r_1 - q_1$. The remaining $r_1 - q_1 - r_1^*$ unfinished rows are moved to the end of the second block-row. They will be treated in some of the following block-rows. The non-zeros of the relevant rows of the second

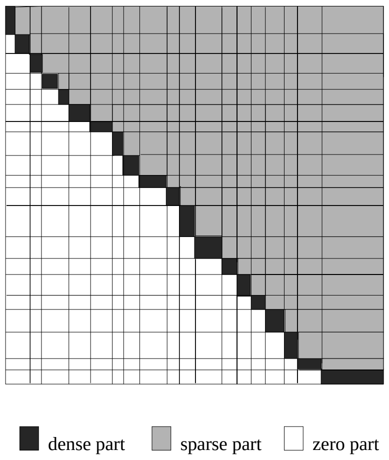

Fig. 1. Sparsity pattern of a rectangular matrix, reordered by LORA.

block-row (the first $r_2 + r_1^*$ rows) are stored in a two-dimensional array with $r_2 + r_1^*$ rows and c_2 columns, where c_2 is the union of the sparsity patterns of its $r_2 + r_1^*$ relevant rows. The number of orthogonal stages is q_2 (again, $rank(A) = n$ implies $q_2 \leq min(r_2 + r_1^*, c_2)$).

Continuing in this way, one can factorize the whole matrix. This may be inefficient, because the blocks of the separator are in general too small. One can combine several blocks of the separator into a larger diagonal block and to apply the same technique, as above. It is appropriate to use two parameters: *LROWS* and *LSTAGE* as lower limits respectively for the number of rows and the number of stages in a composite block.

2.3 Calling subroutines for dense matrices

The i'th dense block contains r_i rows and c_i columns, while q_i stages are to be performed. 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 by Householder reflections. 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 SGEQRF is $O(r_i q_i^2 - q_i^3/3)$. The complexity of SORMQR is $O(r_i q_i (c_i - q_i) - q_i^2 (c_i - q_i)/2)$. The cost of Step 3 is the dominant part of the work; Table 3.

2.4 Gathering the non-zeros in one-dimensional arrays

After Step 3 for block-row i, the non-zeros must be gathered in the sparse arrays. This can cause difficulties, because the number of non-zeros per row is in general changed in the Step 3. Therefore some operations (moving rows to the end of the sparse arrays and even performing occasionally garbage collections; [5]) that are traditionally used in sparse techniques for general matrices must be carried in Step 4. This extra work can be reduced by (i) dropping small elements and (ii) avoiding the storage of Q_i .

Dropping has two effects: (i) the numbers of copies and/or garbage collections is often reduced and (ii) the sizes of some of the next blocks are sometimes reduced (when c_i is reduced for some values of i). The second effect is more important than the first one. If dropping is used, then one should try to regain the accuracy lost by using R in a preconditioned conjugate gradients (PCG) method. The preconditioned system is $Cz = d$, where $C = (R^T)^{-1}A^TAR^{-1}$, $z = Rx$ and $d = (R^T)^{-1}A^Tb$. The PCG method is applicable, because C is symmetric and positive definite. C is never formed explicitly; one works the whole time with A and R. Q, which is normally rather dense, is neither stored nor used in the iterative process (see [5]).

Dropping is very successful for some matrices (see §3.2), but it should not be used if the matrix is very ill-conditioned. Direct methods may work better in the latter case. The storage of Q can be avoided by calculating $c = Q^Tb$ during the decomposition and then x can be found from $Rx = c$.

2.5 Final switch to dense matrix technique

If the density of the non-zeros in the active submatrix (in percent) becomes greater than some parameter (30% is normally a good choice), then the algorithm switches to Step 5. The same work as in Step 2 - Step 4 has to be done, with two simplifications: (i) no need to determine the union of the sparsity patterns of the rows in the last block; (ii) no need of gather step after the dense orthogonal decomposition (the part of R calculated during Step 5 can be used in the solution process directly from the dense array).

3 Numerical results

Most of the results have been obtained on CRAY C92A. A few results on POWER CHALLENGE from Silicon Graphics will also be presented.

3.1 Experiments with rectangular Harwell-Boeing matrices

Results obtained when the two largest rectangular matrices from [2] have been run, are given in Table 2. The new algorithm has been compared with three other algorithms: (i) **Totally dense**. Matrix A is stored in a full-size two-dimensional array (the empty locations are filled with zeros). LAPACK is directly used to solve the problem. (ii) **Partially sparse**. The large block-rows are created in the same way (as explained in §2.2). Givens rotations are used to produce zero elements. A Givens rotation is performed only if both leading elements of the two rows involved are non-zeros. (iii) **Pure sparse**. Without using dense matrix technique (the sparse algorithm used is discussed in [5]).

Matrix	**Characteristics measured**	Totally dense	New algorithm	Partially sparse	Pure sparse
gemat1	Computing time	511	28	10	34
	MFLOPS	878	662	196	4
	Efficiency	97%	73%	22%	0.4%
	Number of blocks	1	47	49	-
bellmedt	Computing time	14	10	16	1490
	MFLOPS	870	685	458	6
	Efficiency	96%	76%	51%	0.7%
	Number of blocks	1	8	8	-

Table 2. Results of comparing the four algorithms on CRAY C92A (one processor, peak performance 902 MFLOPS).

If the matrix is relatively small, then the direct use of LAPACK ("Totally dense") is quite competitive with the new algorithm. The situation changes when the matrix becomes larger. The general conclusion is that the use of a sequence of large dense blocks together with dense kernels should be preferred when a standard package is needed.

In Table 3 the computing times spent in the different steps of the new algorithm are given. Most of the time is spent in the LAPACK subroutines.

Step	**Action**	gemat1	bellmedt
Step 1	LORA	0.67	0.25
Step2 and the beginning of Step 5	Scatter	1.30	0.07
Step 3 and the second part of Step 5	Compute	22.55	9.38
Step 4	Gather	3.22	0.14
	Solve	0.09	0.03
	Total time	**27.79**	**9.91**

Table 3. Computing times (in seconds) spent in the different steps. "Solve" refers to the solution part carried out after the factorization (the direct solver is used in this example).

3.2 Running larger problems with matrices of class F2

LAPACK can be used directly for solving problems with Harwell-Boeing matrices (only for gemat1 the results are inefficient; Table 2). Therefore some very large matrices must also be tested. Matrices of class F2 ([5]) are used. Five parameters determine a matrix of that class: (i) *the number of rows M*, (ii) *the number of columns N*, (iii) *the sparsity pattern C*, (iv) *the average number of non-zeros per row NR*, (v) *the condition parameter $ALPHA$*. The results in Table 4 show that there is no difficulty with the computing times (they are small even when $M = 500000$).

Matrix identifiers			**Computing times**		
M	N	C	NR=5	NR=10	NR=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 4. Computing times (on CRAY C92A) spent for the factorization of 30 matrices of class F2 ($ALPHA$=1 , NZ=$NR$$*$$M + 110$) with the new algorithm.

3.3 Dropping small elements and using PCG

Dropping small elements and computing an approximate QR-factorization is discussed in [5]. It can be used (often with a great positive effect) in our new algorithm. The approximate R (obtained by dropping small non-zeros in Step 4) is used as a preconditioner; §2.4 . Sometimes this leads to considerable reductions of the computing time and/or the storage; see Tables 5, 6. Large drop tolerance may be inefficient for very ill-conditioned matrices, For such matrices it is better to apply the direct method (this is true for gemat1).

3.4 Preliminary results obtained on POWER CHALLENGE

Table 7 contain some results for the performance of our algorithm on a POWER CHALLENGE computer from Silicon Graphics. The results are obtained by just taking the CRAY version of the sparse code and calling the LAPACK routines

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

Table 5. Results of both the direct and the iterative solution (with drop-tolerance TOL= 0.0625) of 10 matrices from class F2 on CRAY C92A. The total time for iterative solution of the problem with using drop-tolerance TOL=0.0625 is given in the third column (togetger with the time spent in the iterative procedure and the number of iterations, given in brackets). The two-norms of the evaluated by the code error and the exact error of the iterative solution are given in the last two columns. (the required accuracy is 10^{-7}).

TOL	Time		Itera-	Evaluated	Exact
	Fact.	Sol.	tions	error	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^{-4}	7.9	0.26	18	5.9E-08	1.6E-08
2^{-8}	10.1	0.10	5	1.8E-08	4.9E-10
2^{-12}	10.1	0.06	3	6.0E-09	3.2E-12
2^{-16}	10.1	0.04	2	7.3E-10	2.4E-13

Table 6. Results of running bellmedt on CRAY C92A with different values of the drop-tolerance TOL and required accuracy 10^{-7}. The two-norms of the evaluated by the code error and the exact error are given in the last two columns.

available on POWER CHALLENGE. Therefore, the results are quite satisfactory: the main idea (to obtain standard modules that perform reasonably well on different high-speed computers on which the LAPACK modules perform well) seems to work (we get reasonably good time, speed-up, MFLOPS and efficiency). The results could be improved.

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 7. Results on a POWER CHALLENGE computer from Silicon Graphics for a problem with coefficient matrix `gemat1`.

4 Conclusions and plans for the future work

Only the computations in Step 3 and Step 5 have been optimized. These two steps are the most time-consuming parts of the computational work (which has been illustrated by two examples in Table 3), but it is nevertheless necessary to optimize also the other steps of the computational work.

The main conclusion is that although the new method will not always give best results it is **a standard tool for solving efficiently large sparse linear least squares problems**. It will produce good results on any high-speed computer on which the LAPACK modules perform well.

Acknowledgements

This research was partially supported by the BRA III Esprit project APPARC (# 6634) and by Grant I-505/95 from the Bulgarian Academy of Sciences. Dr. Guodong Zhang from the Application Group at Silicon Graphics sent us the last updated version of the LAPACK routines tuned on POWER CHALLENGE.

References

1. Anderson, E., Bai, Z., Bischof C., Demmel J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D, "LAPACK: Users' guide", SIAM, Philadelphia, 1992.
2. Duff, I. S., Grimes, R. G. and Lewis, J. G., "Sparse matrix test problems", ACM Trans. Math. Software, 15 (1989), 1-14.
3. Duin, A. C. N. van, Hansen, P. C., Ostromsky, Tz., Wijsoff, H. and Zlatev, Z., "Improving the numerical stability and the performance of a parallel sparse solver", Comput. Math. Applics., Vol. 30, No. 12 (1995), 81-96.
4. Gallivan, K., Hansen, P. C., Ostromsky, Tz. and Zlatev, Z., "A locally optimized reordering algorithm and its application to a parallel sparse linear system solver", Computing, 54 (1995), 39-67.
5. Zlatev, Z., "Computational methods for general sparse matrices", Kluwer Academic Publishers, Dordrecht-Toronto-London, 1991.

This article was processed using the LaTeX macro package with LLNCS style