Local Pivoting for Sparse Factorizations on Multiprocessors *

Juan Touriño and Ramón Doallo
{juan,doallo}@udc.es

Dep. Electrónica y Sistemas. University of La Coruña
Campus de Elviña, s/n. 15071 La Coruña, Spain

Abstract

Many linear algebra algorithms require explicit row/column swapping mainly when pivoting operations are involved. Assuming an MIMD distributed-memory environment, these pivoting operations imply a communication overhead between the processors which store the corresponding rows/columns to be swapped. These communications suppose an appreciable bottleneck in the algorithms and reduce parallelism.

In order to eliminate this drawback, we propose a local explicit pivoting strategy based on constraining the candidate pivot row/column set to those rows/columns located in the the same processors that contain the row/column to be swapped. We show experimentally on the Fujitsu AP1000 multiprocessor the reduction of the execution times, the increase of the efficiencies and the numerical stability of the results using this approach.

1 Introduction

Exploitation of locality, which implies the reduction of communications in message passing environments, is essential to obtain efficient parallel algorithms. This is the aim of parallel linear algebra libraries, such as *ScaLAPACK* [1], and run-time libraries such as *DDLY* (Data Distribution Layer) [7], which provide a simple interface to manage and distribute dense and sparse data structures.

The explicit pivoting procedure is very expensive (in terms of communications) both for dense and sparse factorizations. As a contribution to solve this problem, we present a local pivoting criterion in the sparse Modified Gram-Schmidt (MGS) QR factorization. A similar approach can be also applied to other problems, such as LU and Cholesky factorizations, as well as Householder and Givens QR decompositions.

This work is organized as follows: in Section 2 we focus, as a specific example of a sparse factorization, on the MGS QR factorization; the local pivoting strategy applied to this algorithm is described in Section 3; finally, we present in Section 4 experimental results on the Fujitsu AP1000 about this strategy.

2 Modified Gram-Schmidt Factorization

The MGS QR factorization [4, Chapter 5] involves the decomposition $A = Q \times R$, $A \in \Re^{m\times n}$, $Q \in \Re^{m\times n}$ (orthogonal matrix) and $R \in \Re^{n\times n}$ (upper triangular matrix). Figure 1 presents the pseudo-code of a MGS QR factorization with column pivoting, in which matrix Q is obtained in-place on matrix A. In (1), the squares of the euclidean norms of the columns of matrix A are calculated; the column swapping stage (explicit pivoting) corresponds with (2)-(4); and (5)-(9) is the MGS procedure in the strict sense.

In our version for sparse matrices we use the pivoting criterion shown in expression (2) (widely explained in [6]), in order to preserve the sparsity of the coefficient matrix A during the factorization. Basically, this criterion selects as pivot column (p) the one with the least num-

*This work was supported by the Galician Government (Xunta de Galicia) under project XUGA20605B96

ber of nonzero elements, setting the parameter $\epsilon = 1$ ($zero_j$ is the number of zero entries in column j of the matrix).

This column swapping is also used to determine the rank of the matrix to be factorized (rank-revealing factorization), as done in expression (3). Numerical stability is also preserved (avoiding divisions by numbers close to zero) by discarding as pivot column that one with euclidean norm less than a certain threshold α, as reflected in (2).

Regarding the parallel MGS algorithm, it is executed on a mesh topology of $npey \times npex$ processors. The entries of matrix A are distributed on the mesh in a cyclic way (per rows and per columns), as if the matrix were dense, but only the nonzero elements are stored in one-dimensional column-oriented doubly linked lists. That is, each column of the sparse matrix A is represented by one linked list, as shown in Figure 2 for a 5×5 sparse matrix. This parallelization of the sparse MGS procedure, which exploits data locality, is described in detail in [2].

Let us focus on the column pivoting stage of the parallel algorithm. We find three possible situations in each iteration k of the algorithm:

- If $k = p$ (that is, the current column k being processed is the pivot column), no action is carried out.
- If $k \neq p$ and these columns are located in the same column of processors (that is, k *mod* $npex = p$ *mod* $npex$), then the column swap in matrices Q and R (see expression (4)) is performed by means of a simple exchange of local pointers (represented by the array of pointers *ptr* of Figure 2), taking advantage of the storage scheme we have chosen based on linked lists.
- If $k \neq p$ and these columns are located in different processors (k *mod* $npex \neq p$ *mod* $npex$), then message passing is required to perform the column pivoting. Obviously, this last situation is not desired because it involves a communication overhead for the parallel algorithm, which reduces efficiencies.

$rank = n;$

for (j=0; j<n; j++)

$$norm_j = \sum_{i=0}^{m-1} a_{ij}^2; \qquad (1)$$

for (k=0; k<n; k++) {

Find p, $k \le p < n$, such that

$$\left\{ \epsilon \left(\frac{zero_p}{\max\limits_{k \le j < n} zero_j} \right) + (1-\epsilon) \left(\frac{norm_p}{\max\limits_{k \le j < n} norm_j} \right) \right\}$$

is maximum and $norm_p \ge \alpha$ (2)

if $(\forall p,\ k \le p < n,\ norm_p < \alpha)$ { (3)

$rank = k;$ break;

}

else swap $(norm_k,\ a_{0:m-1,k},\ r_{0:n-1,k})$

and $(norm_p,\ a_{0:m-1,p},\ r_{0:n-1,p})$; (4)

$$r_{k\,k} = \sqrt{norm_k}; \qquad (5)$$

for (i=0; i<m; i++)

$$a_{i\,k} = a_{i\,k} / r_{k\,k}; \qquad (6)$$

for (j=k+1; j<n; j++) {

$$r_{k\,j} = \sum_{i=0}^{m-1} a_{i\,k} \cdot a_{ij}; \qquad (7)$$

$$norm_j = norm_j - r_{k\,j}^2; \qquad (8)$$

for (i=0; i<m; i++)

$$a_{ij} = a_{ij} - a_{i\,k} \cdot r_{k\,j}; \qquad (9)$$

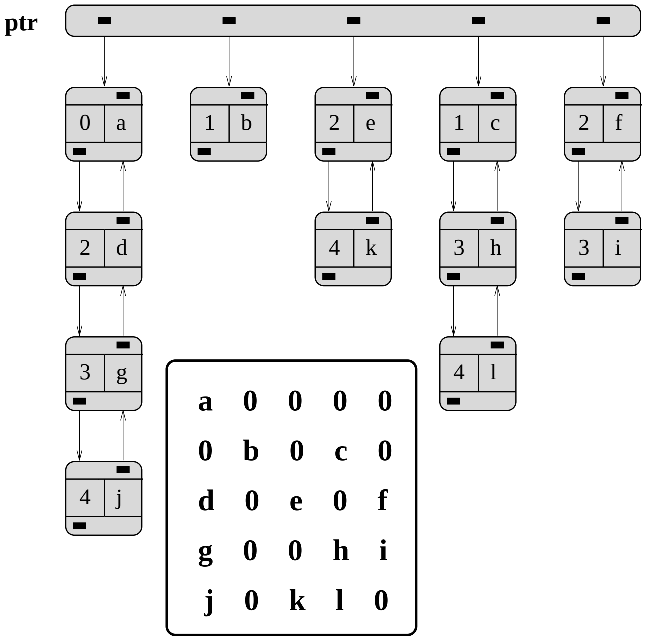
}

}

Figure 1: *Sequential MGS algorithm*

3 A Local Pivoting Approach

The pivoting criterion explained in the previous section is a global pivoting strategy, to the effect that the candidate pivot columns may be located in any column of processors. Therefore, if the pivot column p and the current column k are located in different processors, a data exchange is necessary. Assuming the equi-probability of each column of processors to contain the se-




```
struct matrix
  {
  struct entry *ptr[n];
  };

struct entry
  {
   int i;
   double a;
   struct entry *previ, *nexti;
  };
```

Figure 2: *Doubly linked list data structure for a sparse matrix*

lected pivot column p (this assumption can be made due to the utilization of a cyclic distribution), and assuming, to simplify, that the coefficient matrix to be factorized is maximum-rank, the mean number of communications (data exchanges) required to perform the pivoting is $\left(1 - \frac{1}{npex}\right) \times n$. Obviously, as the X dimension of the mesh ($npex$) increases, a greater number of communications will be necessary because the probability of columns k and p to be located in the same column of processors decreases.

Our proposal to reduce the pivoting communication overhead lies in applying a local pivoting, so that the candidate pivot columns are restricted to those ones located in the column of processors which contains column k. Therefore, we should append the condition:

'*and (k mod npex) = (p mod npex)*'

to the criterion (2) of Figure 1. Using this strategy, pivoting is simply reduced to a pointer exchange, and communications are eliminated; it is, therefore, a local operation. Besides, as linked lists are used as storage scheme, we also save the execution times employed by the pivoting operation to dump/reconstruct the lists in/from a buffer, when they are located in different processors (as the data to be sent must be stored in adjacent memory positions).

Nevertheless, in the last iterations of the QR algorithms, there are few candidate pivot columns in the corresponding column of processors and, in this situation, it would be advisable to apply the global pivoting to assure numerical stability and sparsity preservation. In order to include this feature, we define a parameter called γ as the number of iterations of the algorithm in which local pivoting is applied divided by the total number of iterations (n if the matrix is maximum-rank). Taking into account the previous assumptions, the mean number of communications required by the pivoting strategy becomes $\left(1 - \frac{1}{npex}\right) \times (1-\gamma) \times n$. In practice, as we work with large sparse matrices (n high), we use γ values greater than 0.9. Therefore, in the code of Figure 1 we should add the following condition at the beginning of loop k:

'*if* $(k < \gamma\ n)$ *apply local pivoting*'.

We have also considered those cases in which, for a certain iteration, none of the candidate pivot columns in the corresponding column of processors satisfy criterion (2) (for instance, for all the candidate pivot columns, their norm is less than the threshold α). In this situation, which may happen in any iteration, global pivoting is applied in that iteration, taking advantage of the calculations previously performed in

Matrix	Origin	m×n	#A	% #A
SHL400	Linear programming problems	663×663	1712	0.39%
BP1600	Linear programming problems	822×822	4841	0.72%
JPWH991	Circuit physics modelling	991×991	6027	0.61%
SHERMAN1	Oil reservoir modelling	1000×1000	3750	0.37%
MAHINDAS	Economic modelling	1258×1258	7682	0.49%

Table 1: *Harwell-Boeing sparse matrices*

Matrix	*Global pivoting*			*Local pivoting*			%Red.Ex.
	Ex.time	**Error**	**Effic.**	**Ex.time**	**Error**	**Effic.**	
SHL400	3.05s	0	0.06	1.94s	0	0.10	36%
BP1600	7.65s	1.4E-7	0.27	6.17s	3.2E-7	0.34	19%
JPWH991	54.02s	6.8E-15	0.70	47.68s	8.2E-15	0.79	12%
SHERMAN1	15.91s	6.4E-13	0.50	12.73s	5.2E-13	0.63	20%
MAHINDAS	14.56s	6.7E-7	0.27	11.14s	2.9E-7	0.35	23%

Table 2: *Global pivoting vs local pivoting*

the local pivoting. That is, actually, all the columns of processors, not only the one which owns column k, carry out the calculations of local pivoting in parallel, just in case global pivoting is necessary. This prospective calculation does not increase the computation times. It is also important to note that the fact of using local pivoting does not eliminate the rank-revealing condition of the factorization.

Local pivoting presents also some drawbacks. These drawbacks lie in, as the candidate pivot column set is restricted, fill-in may increase in some occasions. This is because the criterion to reduce fill-in has less candidate pivot columns to select, among them, the one which causes the maximum preservation of sparsity. This fact could imply higher execution times to treat this new nonzero entries, and this overhead in the execution time can counteract the profits in the running times obtained by reducing the communications using local pivoting. Nevertheless, good results were obtained, as we will show in the next section.

4 Experimental Results and Conclusions

Experimentally, we have proved that the local pivoting strategy provides satisfactory results for high values of *npex* and, in general, for large matrices (n high), in which the effect of the communication reduction is appreciable within the whole factorization execution time. The test matrices (see Table 1) were chosen from the Harwell-Boeing collection [3], where $\#A$ is the number of nonzero entries of the matrices and $\%\#A$ is the percentage of these entries with respect to the total number of matrix elements.

Table 2 shows a comparison between global pivoting and local pivoting, for the parallel sparse MGS algorithm, on an 8×8 mesh (*npex*=8) of the Fujitsu AP1000, setting parameters γ=0.99 and ϵ=1. Execution times, numerical errors (obtained by solving least squares problems through the MGS factorization) and efficiencies are the measures taken into consideration.

As we can see in Table 2, execution times are significantly reduced by 22% on average using local pivoting, for this set of sparse matrices. Consequently, the efficiencies of the parallel algorithm increase. Moreover, the numerical errors are practically the same for global and local pivoting, as desired. Additional experiments about local pivoting on the Cray T3D supercomputer are presented in [5].

As a last remark, better results can be obtained in multiprocessors in which the ratio *Processor peak performance/Peak interprocessor communication rate* is high (that is, multiprocessors in which the interprocessor network is very slow in relation to the processors). Therefore, multiprocessor architectures such as clusters of workstations, for instance, would be very appropriate for applying local pivoting.

Acknowledgments

We gratefully thank the Imperial College/Fujitsu Parallel Computing Research Centre (IFPC) in London and the Fujitsu Parallel Computing Research Facilities (FPCRF) in Kawasaki for giving us access to the AP1000 machine.

References

[1] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker and R.C. Whaley. *ScaLAPACK: a Portable Linear Algebra Library for Distributed Memory Computers -Design Issues and Performance.* Technical Report CS-95-283, Dep. Computer Science, University of Tennessee, Knoxville, 1995.

[2] R. Doallo, B.B. Fraguela, J. Touriño and E.L. Zapata. Parallel Sparse Modified Gram-Schmidt QR Decomposition. In *Int'l Conference on High-Performance Computing and Networking, HPCN'96*, Springer-Verlag LNCS no. 1067, pages 646-653. Brussels, Belgium, Apr 1996.

[3] I.S. Duff, R.G. Grimes and J.G. Lewis. *User's Guide for the Harwell-Boeing Sparse Matrix Collection.* Technical Report TR-PA-92-96. CERFACS, Oct 1992.

[4] G.H. Golub and C.F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, second edition, 1989.

[5] J. Touriño. *Parallelization and Compilation Issues for Sparse QR Algorithms.* Technical Report UDC-DES-97/02, Dept. Electrónica y Sistemas, University of La Coruña, Spain, 1997.

[6] J. Touriño and R. Doallo. An Approach to Decrease Fill-in in Sparse Orthogonalizations on a MIMD Computer. In *Sixth Parallel Computing Workshop, PCW'96*, pages S-B-1 to S-B-5. Kawasaki, Japan, Nov 1996.

[7] G.P. Trabado and E.L. Zapata. Exploiting Locality on Parallel Sparse Matrix Computations. In *Third Euromicro Workshop on Parallel and Distributed Processing, PDP'95*, IEEE Computer Society Press, pages 2-9. San Remo, Italy, 1995.