Sparse Block and Cyclic Data Distributions for Matrix Computations

R. Asenjo
L.F. Romero
M. Ujaldon
E.L. Zapata

June 1994
Technical Report No: UMA-DAC-94/06

Published in:

Adv. Workshop in High Performance Computing: Technology, Methods and Applications
Cetraro, Italy, June 1994, pp. 359-377
(Elsevier Science, Amsterdam, 1995)

University of Malaga
Department of Computer Architecture
C. Tecnologico • PO Box 4114 • E-29080 Malaga • Spain

Sparse block and cyclic data distributions for matrix computations*

R. Asenjo, L.F. Romero, M. Ujaldón and E.L. Zapata

Dept. Arquitectura de Computadores, Universidad de Málaga, Spain
Plaza El Ejido S/N, 29013 Málaga. Spain - (ezapata.atc.ctima.uma.es)

A significant part of scientific codes consist of sparse matrix computations. In this work we propose two new pseudoregular data distributions for sparse matrices. The Multiple Recursive Decomposition (MRD) partitions the data using the prime factors of the dimensions of a multiprocessor network with mesh topology. Furthermore, we introduce a new storage scheme, storage-by-row-of-blocks, that significantly increases the efficiency of the Scatter distribution. We will name Block Row Scatter (BRS) distribution this new variant. The MRD and BRS methods achieve results that improve those obtained by other analyzed methods, being their implementation easier. In fact, the data distributions resulting from the MRD and BRS methods are a generalization of the Block and Cyclic distributions used in dense matrices.

1. INTRODUCTION

One of the most challenging problems in distributed memory multiprocessors is to find good data distributions for irregular problems [19]. The main characteristics of the irregular problems are: Low spatial locality (irregular data access and multiple levels of indirection); low temporal locality (limited data reusability) and the sparse matrices need to be represented in a compact way so that the storage requirements and computational time are kept to reasonable levels.

One solution to this problem is the one proposed by Saltz et al [21] that consists in endowing the compiler with a run-time library (PARTI) that facilitates the search and capture of data located in the distributed memory. The most important drawback of this approach is the large number of messages that are generated as a consequence of accessing a distributed data addressing table, and its associated overhead of memory (value based distributions [13]). In fact, the communications have a dominant impact on the performance of massively parallel processors [6]. Besides, this table occupies a relevant amount of memory. In order to enable the compiler to apply more optimizations and simplify the task of the programmer, Bick and Wijshoff [4] have implemented a restructuring compiler which automatically converts programs operating on dense matrices into sparse code. This method postpones the selection

* This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC92-0942-C03 and by the Human Capital and Mobility programme of the European Union (ERB4050P1921660)

of a data structure until the compile phase.

Another alternative, the one we will follow in this work, consists in defining heuristics that perform an efficient mapping of the data and can be incorporated to the data parallel languages in the same way as the popular block and cyclic distributions [23]. The idea is to define pseudoregular data distributions for efficient addressing of sparse matrices without expensive global tables. A pseudoregular data distribution must preserve a compact representation of matrices and vectors, exploit the locality of data and computations, obtain an efficient load balance, and minimize the communications.

Sparse matrix vector multiplication (SpMxV) constitutes one of the most important basic operations of numerical algebra and scientific computation [9], [18]: solution of equation systems by means of iterative methods; Sparse neural networks [14]; EM reconstruction on tomography [7]; etc. The computation of the SpMxV product is completely different from the general case (dense matrices). Sparse methods are mainly based on compacting the data in order to reduce the memory needs and optimize the arithmetic operations. As the SpMxV algorithm is highly computation intensive, there has been great interest in developing parallel formulations for it and test its performance in different parallel architectures [1] and VLSI mesh implementations [15]. In this paper we will concentrate on distributed memory mesh multiprocessors.

Multiprocessor systems with mesh topology present a simple interconnection network that makes them attractive for massively parallel computation. An important number of real machines based on this architecture are currently available. The multiprocessors with mesh topology are made up by a network of processing elements (PEs) arranged as a d-dimensional matrix $A(p_{d-1}, p_{d-2}, ..., p_0)$, where p_i is the size in dimension i. A PE located in $A(i_{d-1}, i_{d-2}, ..., i_0)$ is connected with the PEs located in $A(i_{d-1}, ..., i_j \pm 1, ..., i_0)$ with $0 \leq j < d$ (if they exist). In this work we will consider two dimensional meshes of size $p \times q$. In what follows, we will call the processor located in row r and column s of the mesh PE[r,s]. Using simple indexing, PE[t] represents the t-th PE, $0 \leq t < p \times q$.

In designing parallel sparse algorithms, a key issue is the distribution of the workload among the PEs. Usually we have to solve the tradeoff between a balanced distribution of workload and a minimal communication and synchronization overhead. The complexity of the parallel algorithm for the SpMxV product is strongly conditioned by the distribution of the data. Choosing of a good partitioning for the sparse matrix is crucial in order to balance the load and minimize communications. In this work we present a new distribution method we call Multiple Recursive Decomposition (MRD). The MRD method performs the data partitioning using the prime factor decomposition of the dimensions of the multiprocessor. Furthermore, we introduce a new variant of the Scatter distribution (we will name Block Row Scatter (BRS)), which organizes the storage of data using a storage-by-row-of-blocks. We will analyze and compare the performance of the MRD and BRS methods with several alternative methods for obtaining the SpMxV product in 2D meshes.

The organization of this work is as follows. In section 2 we define the basic steps of an iterative algorithm that includes the SpMxV product as a basic core. The MRD method is described in section 3. A new storage-by-row-of-blocks for the Scatter method it is presented in section 4. Finally, in section 5 we will carry out a performance analysis of the two new data distributions and their specification on HPF (High Performance Fortran).

2. SPARSE MATRIX VECTOR MULTIPLICATION

Given a matrix *M* of dimensions m×n, and a vector *a*, the Sparse Matrix-Vector product (SpMxV) $c=M \cdot a$ is mathematically characterized by expression

$$c_i = \sum_{j=0}^{n-1} M_{ij} \cdot a_j \qquad i = 0, 1, \ldots, m-1 \qquad \textbf{(1)}$$

In most cases, as in the iterative methods for solving equation systems, the resulting vector *c* of a SpMxV product is converted into an operand vector *a*, either directly or modified, of a new product (this requires that m=n). When considering matrix *M* as sparse, a significant savings both in computation time and in local storage requirements will be achieved. We will call *Sparsity rate ß* the ratio between the non null elements (*entries*) of *M* (we will denote this quantity as α) end the total number of elements ($m \cdot n$). Vectors *a* and *c*, will be taken from now on as dense (there is no special treatment of the null elements). The complete algorithm for the SpMxV product is

Preprocessing (M, a^1)
for v=1 *to* number of iterations *do*
    Product (M, a^v)
    Collection (c^v)
    Redistribution (c^v, a^{v+1})
end for

Once matrix *M* has been distributed (in the *Preprocessing* stage), each elements of the operand vector *a* is distributed to each column of matrix *M* (specifically, to the PEs where its elements are stored). The distribution of vector *a* is equivalent to the creation of a matrix *A* consisting in m copies of vector a^T (T means transpose), which contains the same number of rows and columns as *M*. This process is known as array expansion [10]. Mathematically, matrix *A* is obtained

$$\begin{pmatrix} 1 \\ 1 \\ .. \end{pmatrix} \cdot \begin{pmatrix} a_1 & a_2 & \ldots \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \ldots \\ a_1 & a_2 & \ldots \\ \ldots & \ldots & \ldots \end{pmatrix} \qquad \textbf{(2)}$$

Each element A_{ij} will be stored in the PE that contains elements M_{ij}.

During the second phase, the *Product* of each element of matrix *M* with the corresponding element of vector *a* is obtained. Using the previous analogy, it consists in the product of

components $R=M\otimes a$, where $R_{ij} = M_{ij} \cdot A_{ij}$

$$\begin{pmatrix} M_{11} & M_{12} & \dots \\ M_{21} & M_{22} & \dots \\ \dots & \dots & \dots \end{pmatrix} \otimes \begin{pmatrix} a_1 & a_2 & \dots \\ a_1 & a_2 & \dots \\ \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} M_{11} \cdot a_1 & M_{12} \cdot a_2 & \dots \\ M_{21} \cdot a_1 & M_{22} \cdot a_2 & \dots \\ \dots & \dots & \dots \end{pmatrix} \quad \textbf{(3)}$$

The computation of the product is carried out within each PE without communications.

In the *Collection* stage, the elements of vector *c* are obtained by collecting and adding the products of the second phases. Each element c_i is obtained as the sum of the i-th row of matrix $R=M\otimes a$. Finally, in the *Redistribution* phase, the elements c^v (result of the v-th iteration of the product) are transferred from the place where they are stored to the places where the a^{v+1} are required for the *Product* stage of the v+1 iteration. Note that in most cases, the *Redistribution* could partially overlap the *Collection* phase. Figure 1.a shows an example of sparse matrix (m=n=8, α=13, ß=20.3); figure 1.b represents the sequential SpMxV product for a storage by row of matrix *M* (see figure 1.c, where vectors Data, Column and Row store entries, columns and number of entries per row, respectively).

The distribution of matrix *M* is the operation with the greatest impact on the complexity and efficiency of the SpMxV product, being the one that requires a more precise analysis. However, sparse matrices appear with very different patterns, and so it is very difficult to establish comparative parameters a priori. Nevertheless, as we will see now, statistics is a tool that can be of help, although with certains precautions.

When a population, as the elements of a sparse matrix *M*, with two kinds of individuals ("entries" and "zeroes"), is randomly distributed in any number of equally sized groups (p.e., $p \cdot q$ groups), it can be easily shown that, with a probability of being right of $\prod$, the maximum number of individuals of the kind "entries" in a group is

$$w = \frac{\alpha}{p \cdot q} + z_\pi \left(\frac{\alpha}{p \cdot q} \right)^{1/2} = \left\langle \frac{\alpha}{p \cdot q} \right\rangle^{z_\pi}, \qquad (\pi = \sqrt[p \cdot q]{\Pi}) \quad \textbf{(4)}$$

being α the number of entries, and z_π the variable corresponding to the area π under the Gaussian curve. From now on, we will follow the notation $\langle V \rangle^z = V + z \cdot V^{1/2}$, where z is a parameter that modifies the mean value in all PEs of any expression (the expected for a perfect balance) to obtain the maximum expected value between all PEs.

3. MULTIPLE RECURSIVE DECOMPOSITION

Recently, Berger and Bokhari [3] have proposed the *Binary Recursive Decomposition (BRD)*, a well-known distribution algorithm where the matrix *M* is recursively bisected, alternating vertical and horizontal partitions until we have as many submatrices as PEs. Other possibilities for performing these divisions consist in altering the order of the partitions so that horizontal and vertical partitions are not alternated, introducing other arrangements [22]. This distribution method, apart from achieving a good load balance, permits a simple assignment

of the submatrices to a PE network with hypercube or binary tree topology. However, there are serious problems in communications, as adjacent elements in the matrix may be projected onto PEs that are not directly communicated. Besides, the BRD method is only applicable to PE networks with a number of PEs that is a power of two.

In this section we present a new method for the distribution of matrix *M*. We call it *Multiple Recursive Decomposition (MRD)*. It can be considered a generalization of the BRD method for an arbitrary number of PEs.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \\ 0 & 11 & 0 & 0 & 0 & 12 & 13 & 0 \end{pmatrix}$$

1.a. $\alpha = 13$, ß= 20.3%

```
DO I=1, N
  DO J=Row(I), Row(I+1)-1
    Y(I)= Y(I)+Data(J)* X(Column(J))
  ENDDO
ENDDO
```

1.b. Sequential SpMxV Code

Data	*Column*
1	*1*
2	*7*
3	*3*
4	*7*
5	*8*
6	*5*
7	*4*
8	*2*
9	*5*
10	*8*
11	*2*
12	*6*
13	*7*

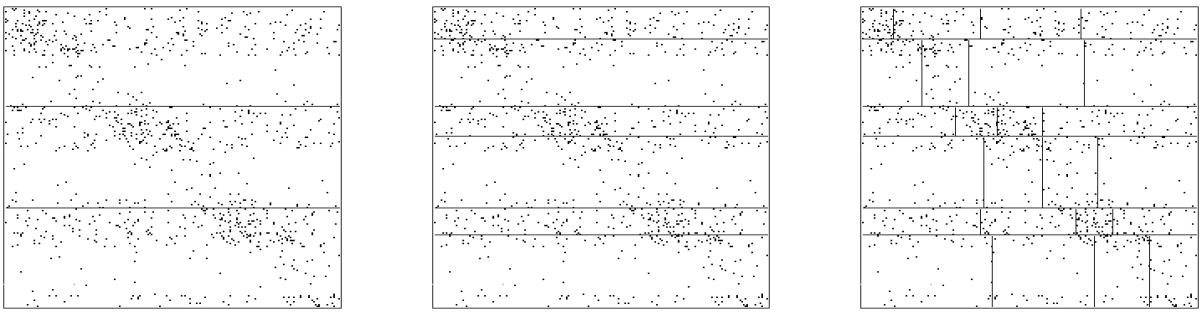
Row
1
3
5
6
7
8
9
11
14

1.c. Data Storage

1. Example of a Sparse Matrix

Let us assume a p processor network and let $P_1 \cdot P_2 \cdot ... P_k$ be the prime factor decomposition of p. The MRD distribution method performs p partitions of matrix *M* in k levels, operating in a very similar way to the BRD method. During the level 1 partition, matrix *M* will be divided into P_1 submatrices with the same load (with the same number of entries) by means of divisions in the horizontal (or vertical) direction. Each submatrix is divided (in a perpendicular direction to the one used in the previous level) into P_2 submatrices during the level 2 partition. This process continues until the level k partition is reached, alternating horizontal and vertical divisions.

Even though the MRD method still presents the communications problems pointed out for the BRD, it is possible to adapt it to the requirements of the SpMxV product in a PE mesh. Let us assume a mesh of size p×q, where the decomposition in prime factors of p and q is $p=P_1 \cdot P_2 \cdot .. \cdot P_a$ and $q=Q_1 \cdot Q_2 \cdot .. \cdot Q_b$, respectively. The MRD decomposition of matrix *M* into p submatrices will be executed as indicated for the BRD, but without alternating the direction of the partitions, always taking the horizontal direction. Figure 2.b shows the resulting p submatrices for the p=6 (P_1=3, P_2=2) case. In a second phase, we will perform a partition into q submatrices of each of the p submatrices generated in the horizontal division using the same technique with the prime factors of q but in the vertical direction (see figure 2.c, where $q=Q_1 \cdot Q_2$=4).



2.a. (P_1=3) **2.b.** (P_2=2) **2.c.** ($Q_1 \cdot Q_2 = 2 \cdot 2$)

2. Multiple Recursive Decomposition (p×q= 6×4 PEs)

As it will now be shown, the MRD distribution method is enough and efficient for the requirements of the SpMxV product. During the *Preprocessing* stage, matrix *M* is divided and distributed as we have indicated. Also, this phase can be carried out in parallel as follows. Initially, every PE is owner of the matrix *M*. Then, 1) in the i-th horizontal partitions (i∈{1,..,a}), each PE[r,s] divides the matrix it owns after the level i-1 partition into P_i equal submatrices, keeping submatrix $\lfloor r/(P_{i+1} \cdot P_{i+2} \cdot .. \cdot P_a) \rfloor$,and rejecting the rest. Next, 2) in the i-th vertical partitions (i∈{1,..,b}), each PE[r,s] divides the matrix it owns after the previous partition into Q_i equal submatrices, keeping submatrix $\lfloor s/(Q_{i+1} \cdot Q_{i+2} \cdot .. \cdot Q_b) \rfloor$, and rejecting the rest. Even though this procedure seems to require large amounts of local memory for storing *M* in each PE, in practice this decomposition can be carried out over the symbolic matrix. The algorithmic complexity is m·n.

During the horizontal division, each one of the p submatrices generated contains, at most, $\alpha/p+m/2$ entries of the matrix (m/2 is the maximum possible unbalance, as the division does not affect the contents of a row). The maximum number of rows in the submatrices, $<m/p>^{z_r}$

(where z_r, is a "shape factor" depends on the concentration of entries in certains rows). After the vertical division, this unbalance is partially corrected, resulting the maximum number of entries in a PE

$$w=\frac{\alpha}{p\cdot q}+\frac{m}{2\cdot q}+\frac{1}{2}\cdot\left\langle\frac{m}{p}\right\rangle^{z_r} \quad \textbf{(5)}$$

In the *Preprocessing* stage each PE receives as many elements of vector *a* as columns it has in its local submatrix $M_{r,s}$. Thus, the MRD method induces q partitions of vector *a* for each PE row, assigning $<n/q>^{z_c}$ elements to each PE (where z_c, as "second level shape factor" depends on the concentration of entries in certains columns inside of certains rows). Figure 3.a shows the MRD partitioning of the matrix example of figure 1.a, considering a mesh of 2×2 PEs. Note that the MRD distribution splits *M* into four sparse rectangular blocks. Each rectangular block is stored in its corresponding PE as shows figure 3.b. The local storage by row of figure 3.b preserve the structure of the sequential algorithm (see figure 1.c). The only difference between figures 1.c and 3.b is which this last stores in vector *Column* the local column of each rectangular block.

The *Product* stage is carried out in each PE, that will contain a subvector c^r of length given by the p partitions of the rows of *M*. In this case, the MRD method induces p partitions of vector *c*, assigning $<m/p>^{z_r}$ elements to each PE. Figure 2.c shows the result of applying the MRD method to matrix *M* and vectors *a* and *c*, considering a mesh with 24 PEs (p=6,q=4). Note that the horizontal partition only affects vector *c*, whereas the vertical partition decomposes vector *a*.

In the *Collection* stage, all of the elements R_{ij} of the same row are added in each PE. These partial results are collected and added (using the cascade addition algorithm) in each of the rows of the mesh without any need for moving data between PE rows. q message exchanges of size $<m/p>^{z_r}$ are required. After this stage, all of the PEs in a row of the mesh have a copy of the corresponding *c* subvector.This data redundance is useful in iterative applications requiring the *Redistribution* stage.

The *Redistribution* of vector c^v can be carried out by means of an exchange of data between rows so that each PE obtains this way all the elements it requires from c^v, in order to transform them into elements of a^{v+1}. q messages of size $<n/q>^{z_c}$ are needed.

We now summarize some of the most significant parameters of the iterative SpMxV multiplication based on the MRD distribution method. From now on, and without any loss of generality we will consider that the mesh and matrix *M* are square and of dimensions p×p and m×m, respectively. Equation (5) specifies the number of entries assigned to each PE. This expression allows us to state that the MRD method generates a good computational load balance, performing w multiplications and $w+<m/p>^{z_r}\cdot\log(p/2)$ additions. The size of the local memory for each PE is given by the number of entries of local submatrix *M* (w), its addressing, and local subvectors *a* and *c*: $w+<m/p>^{z_r}+<m/p>^{z_c}$ floating point data and $w+<m/p>^{z_r}$ integers, necessary for addressing the entries using a storage-by-row of submatrix *M*. The communications are concentrated in the *Collection* and *Redistribution* stages, generating a total of $2\cdot p$ messages of size $<m/p>^{z_r}$ and $<m/p>^{z_c}$.

It can be easily seen that the MRD distribution scheme encompasses, as particular cases, the BRD [3], [22] and One Way Strip Partitioning (OSP) [2], [5], [8],[20] methods. When the

number of PEs of the mesh is a power of two, the MRD method coincides with the BRD method. The OSP method coincides with the MRD method when it is just applied to the rows (Row OSP method) or columns (Column OSP method) of matrix *M*. In section 5 we will compare the MRD method to other alternative methods for the SpMxV product in multiprocessors with mesh topology.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \\ 0 & 11 & 0 & 0 & 0 & 12 & 13 & 0 \end{pmatrix}$$

3.a. Matrix Decomposition

#0

Data	*Column*
1	*1*
3	*3*
6	*5*
-	-

Row
1
2
3
3
4

#1

Data	*Column*
2	*2*
4	*2*
5	*3*
-	-

Row
1
2
3
4
4

#2

Data	*Column*
7	*4*
8	*2*
11	*2*
-	-

Row
1
2
3
3
4

#3

Data	*Column*
9	*1*
10	*4*
12	*2*
13	*3*

Row
1
1
1
3
5

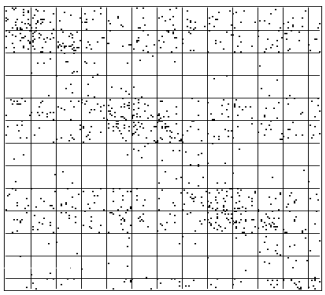
3.b. Local Data Storage_by_Row

3. Multiple Recursive Decomposition (p×q=2×2, N×N=8×8)

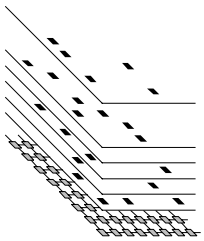
4. BLOCK ROW SCATTER METHOD

The Scatter distribution techniques are based on the division of any computation domain (as the case of a sparse matrix) into several blocks, all of the same spatial shape and size. Each of these blocks is uniformly distributed over the p×p PE network, so that each PE contains a fraction of each block.

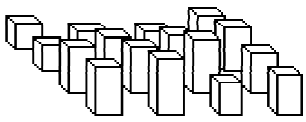
Let us consider a sparse matrix *M* of size m×m partitioned into a set of submatrices *B(k,l)* of size p×p, so that $M_{ij}={}^{kl}B_{rs}$ where $i=p\cdot k+r$, $j=p\cdot l+s$ $(0\le i,j<m)$; $k=\lfloor i/p \rfloor$, $l=\lfloor j/p \rfloor$ $(0\le k,l\le m/p)$; $r=i-k\cdot p$, $s=j-l\cdot p$ $(0\le r,s<p)$. The pairs (i,j), (r,s) and (k,l) are the global indices, local indices and block indices, respectively. For matrix *M* to be divisible into submatrices *B(k,l)*, it may be necessary to add rows or columns with null elements to it. The distribution of the elements of matrix *M* among the PEs is by projecting each one of the blocks of size p×p onto the PE mesh. The scatter method admits larger block sizes [12], but this increase in the grain causes a bigger load unbalance. Figure 4 shows the distribution of the blocks of *M* onto a mesh.



4.a. Matrix Partitioning



4.b. Matrix Distribution



4.c. Local Memories

4. Scatter Decomposition (16×16 PEs)

Once the partition of matrix *M* among the PEs has been carried out as indicated, the entries distributed to each of the PEs can be stored in different ways in order to optimize the number of calculations and the memory requirements. In [1] we find a review of the storage technique proposed by Morjaria and Makinson [16] (MM method) and two new schemes are proposed by Andersen et al, the Extended Stacking Scheme (ESS) and the Block Banded Scheme (BBS). The BBS scheme contains the other two as particular cases. The more significant differences between this three schemes appears in the collection stage of a SpMxV product. In the MM scheme all the elements of the k-th block row are added in each PE and the corresponding element of *c* is obtained in each row before increasing the k index. So, the number of additions is considerably increased. To reduce it, the ESS scheme [1] does not compact in the local memory the elements coming from all the blocks, only those belonging to the same block row (k index). In this case, the local memories are organized into $\lfloor m/p \rfloor$ different local bands. However, the ESS scheme can produce an excessive increase in the requirements of memory when the matrices are very sparse.

The BBS scheme permits grouping the elements of different consecutive row blocks, which, according to ESS, are in different local bands of memory, into a single band (*block band*). A

set of local bands are united into a block band only if the number of entries of that group of row blocks is smaller than a parameter Φ that was predetermined. The local memory required by a BBS scheme is a function of parameter Φ. Given the broad variability interval of Φ we must perform a statistical estimation of it. In general we will consider equation

$$w(\Phi)=\frac{\alpha}{p^2}+z(\Phi)\cdot\left(\frac{\alpha}{p^2}\right)^{1/2} \quad \textbf{(6)}$$

where $z(\Phi)$ varies between z_π (when $\Phi=\alpha$) and $z_\pi \cdot \lfloor m/p \rfloor^{1/2}$ (if $\Phi=0$). It can be easily seen that equation (6) is a very approximated estimation of the memory requirements experimentally reported in [1]. This way, if $\Phi=0$, there is no grouping of local bands (the method coincides with ESS); if $\Phi=\alpha$, the grouping is complete, that is, all of the elements are in a single memory area, as suggested by the MM method. An additional problem of the BBS scheme is that in [1] no technique is described for determining the most adequate Φ parameter, although it is known that it depends, to a great extent, on the sparsity rate of the matrices, being small in matrices that are not very sparse and large in very sparse matrices.

We introduce in this section a new scheme, we will name Block Row Scatter Method (BRS), that significantly increases the efficiency of the Scatter distribution of sparse matrices. We call u plane the set of elements located in the u-th position of the local memory of each PE. During the *Preprocessing* stage of the BRS method, the entries of M will be stored in row major order in consecutives locations of the corresponding PE memory. It is easy to see that in each plane we may find entries coming from different blocks (k,l). The maximum number of entries in a PE is

$$w=\frac{\alpha}{p^2}+z_\pi\cdot\left(\frac{\alpha}{p^2}\right)^{1/2} = \left\langle\frac{\alpha}{p^2}\right\rangle^{z_\pi} \quad \textbf{(7)}$$

which coincides with the number of planes.

The distribution of vector *a* is carried out in block row major order. That is, vector *a* is divided into $\lfloor m/p \rfloor$ blocks of size p, where each block a^l multiplies the l-th block column of matrix *M*. Each element of block a^l is only stored in those planes that reference elements of the l-th block column of matrix *M*. Consequently, in order to carry out the distribution we have to store with each element the index l corresponding to the block column to which it belongs (w integers). It is also necessary to have w floating point memory positions in each PE initially for storing the elements of *a* and afterwards for storing R_{ij}.

In the *Product* stage of the BRS method, each PE performs as many multiplications as memory planes there are in its local memory. It is possible to define a Scatter method which drastically reduces the number of additions in the *Collection* stage. A first approach consists in storing with each entry of matrix *M* the indices (k,l) of the block to which it belong. This makes the organization of data as block bands unnecessary, although we are introducing redundance in the storage of data. A more efficient alternative is to organize the storage of the data using a storage-by-row-of-blocks. The Block Row Scatter method allows us to consider matrix *M* as a matrix of blocks of size p×p. Besides, each PE can only have a single element of each block of matrix *M*, defined by the position of the PE in the mesh, assigned

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \\ 0 & 11 & 0 & 0 & 0 & 12 & 13 & 0 \end{pmatrix}$$

5.a. Matrix Decomposition

#0

Data	*Column*
1	*1*
2	*4*
9	*3*
-	-

Row
1
3
3
3
4

#1

Data	*Column*
5	*4*
7	*2*
10	*4*
-	-

Row
1
1
2
3
4

#2

Data	*Column*
3	*2*
4	*4*
6	*3*
13	*4*

Row
1
3
4
4
5

#3

Data	*Column*
8	*1*
11	*1*
12	*3*
-	-

Row
1
1
1
2
4

5.b. Local Data Storage_by_Row_of_Blocks

5. Block Row Scatter (p×q=2×2, N×N=8×8)

to it. Consequently, the entries of *M* assigned to a PE are identified if we store them in a vector and if in two auxiliary vectors we store the number of entries in each block row and the index of the block column. The first auxiliary vector will have m/p elements and the second's length is equal to the number of entries assigned to the PE (see equation (7)).

Figure 5.a shows the BRS partitioning of the matrix example of figure 1.a, considering a mesh of 2×2 PEs. Note that the BRS distribution organizes *M* into a sparse block matrix (16 blocks). Each entry of a block is automatically assigned to a specific PE as shows figure 5.b. The local storage by row of blocks of figure 5.b preserve the structure of the sequential algorithm (see figure 1.c). The only difference between figures 1.c and 5.b is which this last stores in vector *Column* the local block column of each entry.

Using the storage-by-row-of-blocks the PEs know the value of block row index (k) associated with each entry. This allows us to compute the *Collection* stage in two phases. In the first phase each PE calculates all $\lfloor m/p \rfloor$ partial additions without communications. Next, using the cascade addition algorithm all the PEs of the r-th row obtain the $\lfloor m/p \rfloor$ elements of vector *c*. This phase requires p messages of size $\lfloor m/p \rfloor$. Summarizing, in the *Collection* stage of the BRS method $w+\lfloor m/p \rfloor \cdot \log_2(p/2)$ additions are carried out.

In the *Redistribution* stage ($a^{v+1}=f(c^v)$) the $\lfloor m/p \rfloor$ values of c^v stored in the r-th row of the PE mesh have to be distributed to the PEs in the r-th column. This distribution can be efficiently carried out if the PE of indices (r,r) is the one providing the data to each column. Therefore, only p messages of size $\lfloor m/p \rfloor$ are required for this operation.

5. EVALUATION

In this section, we will carry out a comparative analysis of both methods proposed in the two previous sections and other methods recently appeared in the literature. We also include a High Performance Fortran specification of the iterative SpMxV multiplication and some experimental results. In order to facilitate the comparison of these, we will substitute the variables until they are reduced to three: α, p, and m (the mesh and matrix *M* are square). A stricter measure for the calculation of the comparison parameters consists in the determination of the worst possible case. However, we have discarded this option as it does not reflect the characteristics of most of the methods. On the other hand, an analysis of the pattern of the sparse matrix can be carried out using only m^2 boolean steps in order to precisely determine the exact value of each one of the parameters. This makes sense when we are interested in a specific type of sparsity. Finally, we consider that a simultaneous exchange of messages between two adjacent PEs is possible. In this case, the additions performed in the *Collection* stage can be simultaneously executed in all the PEs that supply the terms to be added.

The result of the analysis performed is specified in table I. In the first column of the table we have expressed the difficulty of implementation for each distribu-

tion method. The next three columns of the table show the complexities of the *Preprocessing* and *Product* stages and the number of additions performed in the *Collection* stage. The values found in the table for each parameter are a estimation to their real values. In those cases in which it has been impossible to establish an estimation of the parameter a priori we have indicated the worst cases. This is what happens, for instance, with the number of additions calculated for the BBS Scatter method when $\Phi=\alpha$, whose value is obtained from the number of planes that reference the k-th block row. The optimum value for $\Phi=\alpha$ is the expected value for $\Phi=0$. The real number of additions for $\Phi=\alpha$ (and, in general, for any value of Φ) is a quantity between the two extreme cases presented in the table. We have also made use of the worst possible case in the OSP and MRD methods, although with less influence on the approximations. Due to the "border effect" we have added m/2 entries in the OSP method and <m/p>/2 entries in the MRD method.

The Snake method [14], [15] is the one that presents a smaller number of products and sums due to the optimal load balance it obtains, but it requires a disproportionate number of messages (see fifth and sixth columns of table I) and more local memory in each PE (see seventh column). The MRD method exhibits the smallest arithmetic complexity of the three. The difference between the MRD and Row OSP is due to the border effect we have mentioned before, which is more pronounced in partitions with elongated shapes and which can be critical in very sparse matrices with dense rows.

The BBS Scatter(α) method presents an optimal behavior in the number of products but generates a disproportionate number of additions. This problem was already pointed out in [1]. When Φ decreases, the number of additions is significantly reduced at the price of an increase in the number of products. The BRS method drastically reduces the number of additions and yields a smaller arithmetic complexity than the MRD method, although it slightly increases the size of the local memory (see seventh column). It also simplifies programming of the Scatter distributions.

The number of messages, together with their sizes, of the *Collection* (r) and *Redistribution* (d) stages are reflected in the fifth and sixth columns of table I. The Snake method is the one presenting a larger number of messages as a consequence of the change of storage from Snake-like Column Major Order to Snake-like Row Major Order. The rest of the methods require a much smaller number of messages ($2 \cdot p$), and the Row OSP method is the one that exhibits a smaller message size in inter-row communications. We must also point out the regularity in the size of the messages of the Scatter methods.

The seventh column of table I shows the local memory requirements. The memory needed for the MRD and Row OSP methods is much smaller than for the rest as a consequence of the simplicity of the local storage scheme (storage-by-row). Again, the MRD method obtains smaller values as a consequence of the minimization of the lateral effects. Note that the BRS method needs an intermediate amount of memory between the two extremes of the BBS Scatter method.

Figure 6 shows the normalized number of arithmetic computations in a logarithmic scale (fig. 6.a) and the normalized memory size (fig. 6.b) for a PE as a function of the number of PEs in the mesh. We have considered a sparsity rate

of ß=0.1 and m=10000. The Snake method exhibits a constant behavior that is independent from p^2. The Scatter and MRD methods present a smooth increase in the computations and memory requirements, being the Row OSP method the most sensitive to the mesh size. The bigger the mesh, the more difficult it will be to obtain a good load balance with it. Besides, as the number of PEs is increased, the impact of vector *a* in the size of the local memory grows. If we decreases the sparsity rate, the slope of all the plots in figure 6 is accentuated.

The last column of table I represents the number of evaluations of f (function that transforms the elements of vector c^v into a^{v+1}). The Row (Column) OSP method is the one that performs the smallest number of operations of this type. Nonetheless, it is possible to modify the other methods so that they perform this operation in approximately the same number of cycles ($<m/p^2>$). For example, in the MRD method it is necessary for each region to know the dimensions of the other regions, in which case the size of the messages can also be reduced.

In Figure 7 we include the HPF code [23] for the parallel SpMxV multiplication based on the BRS data distribution. We have used a new pair of directives (highlighted with !!!!!) in order to indicate to the compiler that this data distribution must be generated. The HPF code for the parallel SpMxV multiplication based on the MRD data distribution is exactly the same of figure 7, but changing these two directives by the following ones:

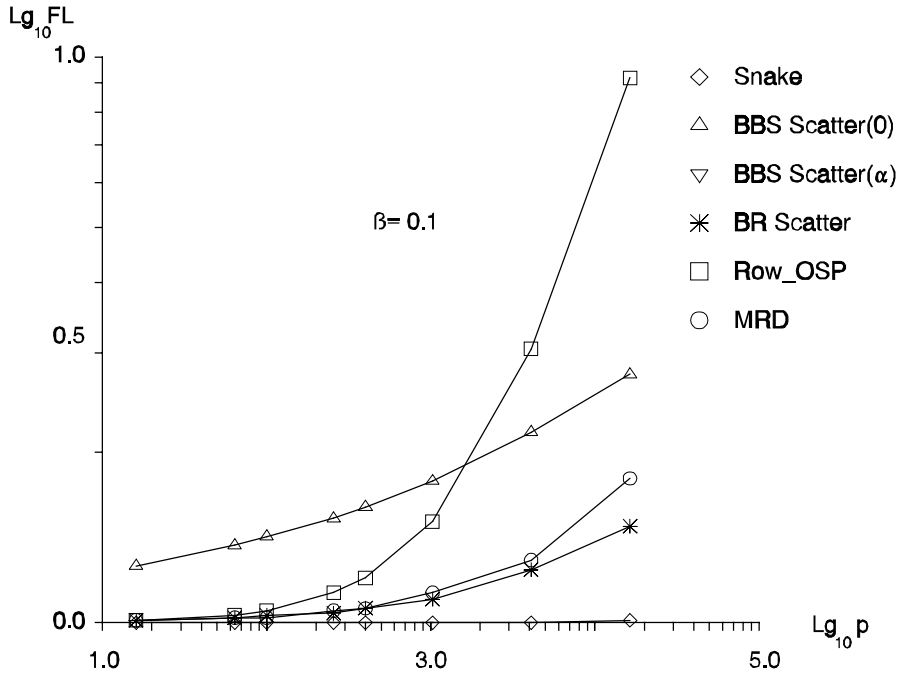
```
!!!!! DISTRIBUTE A(SPARSE BLOCK, SPARSE BLOCK) ONTO MESH
!!!!! DISTRIBUTE Data,Column,Row SPARSE A
```

Looking at the code, we can separate it into two parts: First, we find the declaration part in which we capture the hardware topology we are using, and declare, align and distribute the variables we need to compute the SpMxV; then we place a interface block where we declare the extrinsic subroutine we are going to use. Second, in the execution part, we begin calling the code that performs the local product using SPMD model with local index onto each distributed memory. Then, a global loop will be computed to collect all the partial sums we have performed locally before. This last stage needs communications to obtain the partial sum from processors in the same mesh row. Finally, we realign the solution vector in order to use it as input in the following iteration.

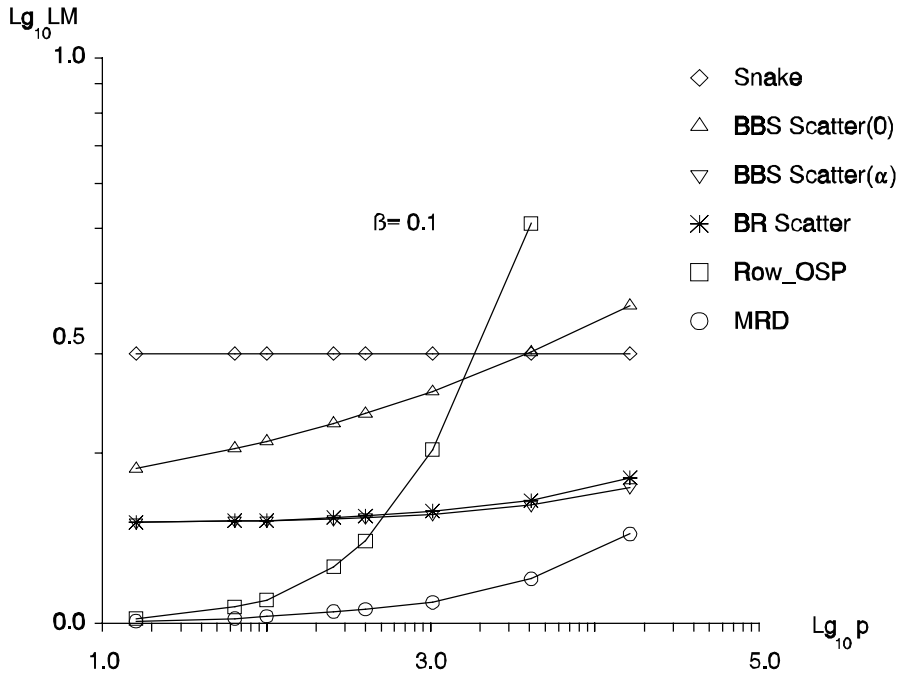
In figure 8 we present the isoefficiency [11] for differents squared meshes performed on a PARAMID 16/i860SYS machine with 16 i860 nodes. It can be observed that the good scalability which presents the parallel SpMxV multiplication is a direct consequence of exploit the spatial locality exhibed by MRD and BRS sparse data distributions.

	PROGRAMMING	PREPROCESS	PRODUCTS	ADDITIONS	N^{er} MESSAGES*	SIZE OF MESSAGES*	LOCAL MEMORY: INTEGERS FLOATS	$a^{v+1}=f(c^v)$ EVALUATIONS
Snake	Complex	$m^2+\alpha\cdot\log\ \alpha$	$\frac{\alpha}{p^2}$	$\frac{\alpha}{p^2}+\log(3\cdot p)$	$(r)\ \ \frac{3\cdot\alpha}{p}+3\cdot p$ $(d)\ \ 2\cdot p$	$(r)\ \ 1$ $(d)\ \ \frac{\alpha}{p^2}$	$\frac{4\cdot\alpha}{p^2}$ $\frac{2\cdot\alpha}{p^2}$	$\frac{\alpha}{p^2}$
Scatter (BBS,Φ=0)	Half	$m^2\cdot\left\langle\frac{\alpha}{m\cdot p}\right\rangle^{Z_\alpha}$	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha\cdot(m/p)^{1/2}}$	$\frac{m}{p}\cdot\log\frac{p}{2}+\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha\cdot(m/p)^{1/2}}$	$(r_r)\ \ p$ $(d_c)\ \ p$	$(r_r)\ \ \frac{m}{p}$ $(d_c)\ \ \frac{m}{p}$	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha\cdot(m/p)^{1/2}}$ $2\cdot\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha\cdot(m/p)^{1/2}}$	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha\cdot(m/p)^{1/2}}$
Scatter (BBS,Φ=α)	Half	$m^2\cdot\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$\frac{m}{p}\cdot\left(\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}+\log\frac{p}{2}\right)$	$(r_r)\ \ p$ $(d_c)\ \ p$	$(r_r)\ \ \frac{m}{p}$ $(d_c)\ \ \frac{m}{p}$	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$ $2\cdot\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$\left\langle\frac{m}{p}\right\rangle^{Z_\alpha}$
Block Row Scatter	Easy	m^2	$\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$\frac{m}{p}\cdot\log\frac{p}{2}+\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$(r_r)\ \ p$ $(d_c)\ \ p$	$(r_r)\ \ \frac{m}{p}$ $(d_c)\ \ \frac{m}{p}$	$2\cdot\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$ $2\cdot\left\langle\frac{\alpha}{p^2}\right\rangle^{Z_\alpha}$	$\left\langle\frac{m}{p}\right\rangle^{Z_\alpha}$
1_Way Str. Row	Easy	m^2	$\frac{\alpha}{p^2}+\frac{m}{2}$	$\frac{\alpha}{p^2}+\frac{m}{2}-\left\langle\frac{m}{p^2}\right\rangle^{Z_r}$	$(d_r)\ \ p$ $(d_c)\ \ p$	$(d_r)\ \ \left\langle\frac{m}{p^2}\right\rangle^{Z_r}$ $(d_c)\ \ \left\langle\frac{m}{p}\right\rangle^{Z_r}$	$\frac{\alpha}{p^2}+\frac{m}{2}+\left\langle\frac{m}{p^2}\right\rangle^{Z_r}$ $\frac{\alpha}{p^2}+\frac{3\cdot m}{2}+\left\langle\frac{m}{p^2}\right\rangle^{Z_r}$	$\left\langle\frac{m}{p^2}\right\rangle^{Z_r}$
Multiple Recurs. Decomp.	Easy	m^2	$\frac{\alpha}{p^2}+\frac{m}{2\cdot p}+\frac{1}{2}\cdot\left\langle\frac{m}{p}\right\rangle^{Z_r}$	$\frac{\alpha}{p^2}+\left\langle\frac{m}{p}\right\rangle^{Z_r}\cdot(\log\ p-\frac{1}{2})+\frac{m}{2\cdot p}$	$(r_r)\ \ p$ $(d_c)\ \ p$	$(r_r)\ \left\langle\frac{m}{p}\right\rangle^{Z_r}$ $(d_c)\ \left\langle\frac{m}{p}\right\rangle^{Z_{c'}}$	$\frac{\alpha}{p^2}+\frac{m}{2\cdot p}+\frac{3}{2}\cdot\left\langle\frac{m}{p}\right\rangle^{Z_r}$ $\frac{\alpha}{p^2}+\frac{m}{2\cdot p}+\frac{3}{2}\cdot\left\langle\frac{m}{p}\right\rangle^{Z_r}+\left\langle\frac{m}{p}\right\rangle^{Z_{c'}}$	$\left\langle\frac{m}{p}\right\rangle^{Z_r}$

(*) (r) recollection of c^v. (d) redistribution from c^v to a^{v+1}. (the r and c subindices represents row and column data movements)



6.a. Floating Points Operations (FL= $f(p^2)$)



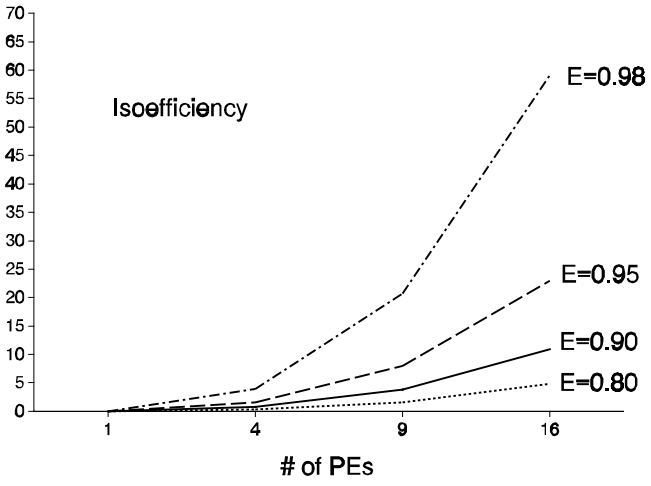
6.b. Local Memories (LM=$f(p^2)$)

6. Scalability (m= 10^4)

```
      PARAMETER(P=NUMBER_OF_PROCESSORS(DIM=1))
      PARAMETER(Q=NUMBER_OF_PROCESSORS(DIM=2))
      PARAMETER(ALPHA=8000,N=1000,K=10)
      REAL X(N), Y(N), P_Y(N,Q), Data(ALPHA)
      INTEGER Column(ALPHA), Row(N+1), V, I, IP
      CHAR DIST_P_Y(9)

      USE HPF_LIBRARY


!HPF$     PROCESSORS MESH(P,Q)
!HPF$     TEMPLATE A(N,N)
!HPF$     ALIGN WITH A(*,:) :: X(:)
!HPF$     DYNAMIC, ALIGN WITH A(:,*) :: Y(:)

!!!!! DISTRIBUTE A(SPARSE CYCLIC,SPARSE CYCLIC) ONTO MESH
!!!!! DISTRIBUTE Data,Column,Row SPARSE A

      CALL HPF_DISTRIBUTION(A,AXIS_TYPE(1)=DIST_P_Y)

!HPF$     DISTRIBUTE P_Y(DIST_P_Y,I) ONTO MESH(I)

      INTERFACE
        EXTRINSIC(HPF_LOCAL) SUBROUTINE SPMXV(X,Y,D,C,R)
              REAL, DIMENSION(:,:), INTENT(OUT) :: Y
              REAL, DIMENSION(:), INTENT(IN) :: X, D
              INTEGER, DIMENSION(:), INTENT(IN) :: C, R
        END SUBROUTINE SPMXV
      END INTERFACE
```

```
          DO V=1, K
C --- Perform product and partial sum locally
            CALL SPMXV(X,P_Y,Data,Column,Row)
C --- Collection into each row of PEs
            DO IP = 1, Q
              DO I = LBOUND(Row,1), UBOUND(Row,1)-1
                Y(I) = Y(I) + P_Y(I,IP)
              END DO
            END DO
C --- Redistribution
!HPF$           REALIGN Y(:) WITH X(:)
            X = Y
          END DO
        END


      --- EXTRINSIC SUBROUTINE USING SPMD MODEL ---
      EXTRINSIC(HPF_LOCAL) SUBROUTINE SPMXV(X,Y,D,C,R)
            REAL, DIMENSION(:,:), INTENT(OUT) :: Y
            REAL, DIMENSION(:), INTENT(IN) :: X, D
            INTEGER, DIMENSION(:), INTENT(IN) :: C,R
            INTEGER I,J
C --- Local Product
            DO I = LBOUND(R,1), UBOUND(R,1)-1
              DO J = R(I), R(I+1)-1
                Y(I) = Y(I) + D(J)*X(C(J))
              END DO
            END DO
        END
```

7. HPF Code for SpMxV Multiplication based on BRS Distribution



8. Isoefficiency for squared meshes (BRS and MRD data distributions)

6. CONCLUSIONS

From the analysis of the different techniques for executing the SpMxV product in multiprocessors with mesh topologies we deduce that it is difficult to obtain a perfect equilibrium of the number of entries stored in each PE with an optimum communications cost (as happens in the Snake method). However, values that are very close to this equilibrium can be very satisfactory. Also, the simplicity in the storage of the elements positively influences both the size of the local memory and the number of computations. This causes potentially good distribution methods to loose their efficiency because of their storage complexity (as is the case of MM, ESS and BBS Scatter methods). There is a close relationship between the geometry of the computations and the geometry of the distributions. The computation of the SpMxV product requires a larger number of calculations by rows than by columns and, therefore, it is a good solution to keep the rows together in a single PE, or, at least, to find regularity in communications by rows. For this reason the results obtained with the Row OSP method were better than those of the Column OSP. This idea has contributed to making us opt in the MRD method for performing the first subdivision by rows and then by columns, facilitating this way horizontal communications and reducing the number of additions. Other applications of this method may use variations in the order of the partitions, even using new dimensions.

The Scatter methods are specially effectives when the elements of the computation domain that require a larger number of calculations are grouped in certain zones of the domain. However, this can produce strong load imbalance when matrix *M* contains structures that are periodically repeated with a period that is an integer multiple of the dimensions of the network [17]. We can prevent the coupling between the matrix and the network by means of the reduction of the size of the latter. On the other hand, a dense row or column can also lead to an inefficient storage in a PE row or column.

The MRD method we propose in this work is presented as a general method in which the OSP and BRD methods are some of its particular cases. It has a good load balance (similar to the BRD method) and a good disposition of message exchange (characteristic of the OSP methods). Consequently, this method achieves results that improve those obtained by the rest of the methods analyzed.

Finally, we want to point out the analogy of the MRD and BRS distributions to the Block and Cyclic distributions used in parallel algorithms with dense matrices. In fact, these last can be considered particular cases that do not require additional storage for addressing the data. The MRD and BRS distributions can also be used to determine good data distributions for irregular problems avoiding expensive phases of communication for addressing of nonlocal data because the PEs have sufficient local information to know where are allocated the data. Moreover, their incorporation to a data parallel language is immediate.

REFERENCES

1. J. Andersen, G. Mitra, D. Parkinson, "The Scheduling of Sparse Matrix-Vector Multiplication on a Massively Parallel DAP Computer", Par. Computing, vol.18, pp.675-697, 1992.
2. C.Aykanat, F.Ozgüner, F.Ercal, P.Sadayappan, "Iterative Algorithms for Solution of Large Sparse Systems of Linear Equations on Hypercubes", IEEE Trans. Comput., vol.37, no.12,

pp.1554-1568, 1988.
3. M.J. Berger and S.H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors", IEEE Trans. Comput., vol. 36, no. 5, pp. 570-580, 1987.
4. A.J.C. Bik and H.A.G. Wijshoff, "Compilation Techniques for Sparse Matrix Computations", ACM Int'l Conf. on Supercomputing (Tokyo), pp. 416-424, july 1993.
5. J.E. Boillat, "Load Balancing and Poisson Equation in a Graph", Concurrency, Practice and Experience, vol. 2, no. 4, pp. 289-313, 1990.
6. E.L. Boyd, J.D. Wellman, S.G. Abraham and E.S. Davidson, "Evaluating the Communication Performance of MPPs Using Synthetic Sparse Matrix Multiplication Workloads", ACM Int'l Conf. on Supercomputing, (Tokyo), July 1993.
7. C.M. Chen, S.Y. Lee and Z.H. Cho, "Parallelization of the EM Algorithm for 3-D PET Image Reconstruction", IEEE Trans. on Medical Imaging, vol.10, no.4, pp.513-522, 1991.
8. Y.C. Chung and S. Ranka, "Mapping Finite Element Graphs on Hypercubes", J. of Supercomputing, vol. 6, no. 3/4, pp. 257-282, 1992.
9. I.S. Duff, A.M.Erisman, J.K.Reid, "Direct Methods for Sparse Matrices", Clarendon Press, Oxford, 1986.
10. P. Feautrier, "Array expansion". ACM Int'l Conf. on Supercomputing, July, 1988.
11. A. Gupta, V. Kumar, "The Scalability of FFT on Parallel Computers", IEEE Trans. on Parallel and Distr. Systems, vol. 4, no. 8, pp. 922-932, August 1993.
12. M. Gupta, P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers", IEEE Trans. on Parallel and Distr. Systems, vol. 3, no. 2, pp. 179-193, 1992.
13. R.V. Hanxleden, K. Kennedy, J. Saltz, "Value-Based Distributions in Fortran D - A Preliminary Report", Tech. Rep. TR93365-S, CRPC, December 1993.
14. M. Misra, P. Kumar, "Implementation of Sparse Neural Networks on Fixed Size Arrays", in Parallel Algorithms and Architectures for DSP Applications, M.A. Bayoumi (Ed.), pp. 255-279, 1991.
15. M. Misra, D. Nassimi, V.K. Prasanna, "Efficient VLSI implementation of iterative solutions to sparse linear systems", Parallel Computing, vol. 19, no. 5, pp. 525-544, 1993.
16. M. Morjaria, G.J. Makinson, "Unstructured Sparse Matrix Vector Multiplication on the DAP". Super Computers and Parallel Computation. pp.157-166, Clarendon Press, Oxford 1984.
17. D.M. Nicol and J.H. Saltz "An Analysis of Scatter Decomposition", IEEE Trans Comput., vol. 39, no. 11, pp. 1337-1345, November 1990.
18. A.T. Ogielski, W. Aiello, "Sparse Matrix Computations on Parallel Processor Array", SIAM J. Sci. Comput., vol. 14, no. 3, pp. 519-530, 1993
19. R. Ponnusamy, J. Saltz, R. Das, C. Koelbel, A. Choudhary. Embedding Data Mappers with Distributed Memory Machine Compilers. ACM SIGPLAN Notices, vol.28, no.1, pp.52-55, 1993.
20. P. Sadayappan and F. Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes", IEEE Trans Comput., vol. 36, no. 12, pp. 1408-1424, 1987.
21. J. Saltz, R. Das, R. Ponnusamy, D. Mavriplis, H. Berryman and J. Wu. PARTI Procedures for Realistic Loops. 6th Int'l Conf. on Distributed Memory Computing, 1991.
22. D.W. Walker, "Characterizing the Parallel Performance of a Large-Scale, Particle-in-Cell Plasma Simulation Code", Concurrency, Pract. and Exper., vol.2, no.4, pp.257-288, 1990.
23. "High Performance Fortran Language Specification". Version 1.0, Technical Report TR92-225, CRPS, Rice University, May 1993.