Data distributions for sparse matrix vector multiplication †

L.F. Romero, E.L. Zapata *

Dept. Arquitectura de Computadores, Universidad de Málaga, Plaza El Ejido s/n, 29013 Málaga, Spain

Received 15 September 1993; revised 21 December 1993

Abstract

Sparse matrix vector multiplication (SpMxV) is often one of the core components of many scientific applications. Many authors have proposed methods for its data distribution in distributed memory multiprocessors. We can classify these into four groups: Scatter, D-Way Strip, Recursive and Miscellaneous. In this work we propose a new method (Multiple Recursive Decomposition (MRD)), which 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 method. We will name Block Row Scatter (BRS) method 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.

Keywords: Sparse computation; Parallel algorithm; Mesh topology; Data distribution; Sparse matrix; Distributed memory multiprocessor; Performance evaluation

1. Introduction

One of the most challenging problems in distributed memory multiprocessors is to find good data distributions for irregular problems [22]. One solution to this problem is the one proposed by Saltz et al. [25] 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

† This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC92-0942.
* Corresponding author. Email: ezapata@ctima.uma.es

0167-8191/95/$09.50 © 1995 Elsevier Science B.V. All rights reserved
SSDI 0167-8191(94)00087-5
accessing a distributed data addressing table. In fact, the communications have a dominant impact on the performance of massively parallel processors [7]. 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 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 [27]. The idea is to define pseudoregular data distributions for efficient addressing of sparse matrices without expensive global tables.

Sparse matrix vector multiplication (SpMxV) constitutes one of the most important basic operations of numerical algebra and scientific computation: solution of equation systems by means of iterative methods; Sparse neural networks [17]; EM reconstruction on tomography [8], 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 [11 and VLSI mesh implementations [18]. 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 $A(p_{d-1}, p_{d-2}, \ldots, p_0)$, where $p_i$ is the size in dimension $i$. A PE located in $A(i_{d-1}, i_{d-2}, \ldots, i_0)$ is connected with the PEs located in $A(i_{d-1}, \ldots, i_j \pm 1, \ldots, i_0)$ 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.
We will also present a statistical analysis of the sparsity of a distribution and its
influence on the parameters that characterize it. 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 comparative analysis of the
methods described in the previous sections and other methods recently proposed.

2. Sparse matrix vector multiplication

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

$$c_i = \sum_{j=0}^{n-1} M_{ij} \cdot a_j$$

$$i = 0, 1, \ldots, m - 1$$

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 consider-
ning matrix $M$ as sparse, a significant savings both in computation time and local
storage requirements will be achieved. Vector $a$ and $c$, however, 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 [11]. Mathematically, matrix $A$ is obtained

$$\begin{pmatrix}
1 \\
1 \\
\vdots
\end{pmatrix} \cdot \begin{pmatrix}
a_1 & a_2 & \ldots \\
a_1 & a_2 & \ldots \\
\ldots & \ldots & \ldots
\end{pmatrix}$$

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 $R_{ij} = M_{ij} \cdot A_{ij}$

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

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.

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. In the next point we will see that statistics is a tool that can be of help, although with certain precautions.

### 2.1. Statistical analysis of the distribution

We will call sparsity rate $\beta$ the ratio between the number of non null elements (entries) of the sparse matrix $M$ (we will denote this quantity as $\alpha$) and the total number of elements ($m \cdot n$)

$$
\beta = \frac{\alpha}{m \cdot n} \quad (4)
$$

The value of $\beta$ required for considering a matrix sparse depends on the characteristics of the problem, on the distribution of the entries (pattern of the matrix) and even on the architecture. We will generally say that a matrix is sparse if it is advantageous to exploit its zeroes [10].

A sparse matrix of dimensions $m \times n$ with $\alpha$ entries can be thought of as a population in which a given event ('non null') occurs with a probability $\beta$, and the event 'null' has a probability of $1-\beta$. The distribution of the matrix among $p \cdot q$ PEs is equivalent to the extraction of $p \cdot q$ samples of the population, each of size $N = m \cdot n / p \cdot q$. For these samples to be representative of the population they must verify at least one of the following requirements: (a) The entries have to be randomly distributed in the matrix, (b) The elements of each sample must be randomly selected from the matrix. The first of these requirements is not always verified given the tendency of entries in the matrix to group. One way of alleviating
this problem consists in assigning to each PE elements that are distributed all over the matrix.

The probability $\beta_t$ of the 'non null' event in $\text{PE}[t]$ presents a binomial sampling probability for a finite population and sampling without replacing, and has a mean value of $\beta$, and a variance:

$$\sigma^2 = \frac{\beta \cdot (1 - \beta)}{N} \cdot \frac{N_p - N}{N_p - 1}$$

being $N_p$, the size of the population ($N_p = m \cdot n$). When $N$ is a large number (in general $N > 10$ is enough), the binomial distribution can be approximated by a normal or Gaussian distribution.

The probability of $\beta_t$ being lower than $\beta_{\text{max}}$ for all $t$ is given by the area under the Gaussian curve between $-\infty$ and $z_{\text{max}}$, where $z_{\text{max}} = (\beta_{\text{max}} - \beta)/\sigma$ is the normalized variable $\beta_{\text{max}}$. If this area is $\pi$, the possibility of this event not happening in any of the $p \cdot q$ PEs (assuming they are independent events) is

$$\Pi = \pi^{p \cdot q}$$

Consequently, we can say, with a probability of being right of $\Pi$, that the proportion of entries to be stored in a PE is always lower than:

$$\beta_{\text{max}} = \beta + z_\pi \cdot \sigma \quad (\pi = \frac{p \cdot q}{\sqrt{\Pi}})$$

being $z_\pi$ the variable corresponding to the area $\pi$ under the Gaussian curve.

In sparse matrices, $\beta$ is close to 0 and $N_p/N = p \cdot q$ is generally a large number, so, $\sigma$ can be approximated to its upper bound

$$\sigma = \left(\frac{\beta}{N}\right)^{1/2}$$

Substituting in (7) it can be stated, within the confidence limits and number of PEs established by $z_\pi$, that the number of entries in all of the PEs is less than

$$w = \frac{\alpha}{p \cdot q} + z_\pi \cdot \left(\frac{\alpha}{p \cdot q}\right)^{1/2}$$

<table>
<thead>
<tr>
<th>$\Pi$ values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10$</td>
</tr>
<tr>
<td>$90%$</td>
</tr>
<tr>
<td>2.31</td>
</tr>
<tr>
<td>3.08</td>
</tr>
<tr>
<td>3.72</td>
</tr>
</tbody>
</table>
In Table I we present some of the values for $z$. Analyzing the values obtained for the Scatter distribution, which performs a periodic assignment of data distributed throughout matrix $M$ [1], we note that in those cases in which certain randomness conditions are verified the results are very similar to those obtained by means of Eq. (9) for $z = 4.27$ (1024 PEs, 99% probability).

Although Eq. (9) has been calculated in order to estimate the number of entries stored in each PE it is possible to obtain similar estimations for other parameters (sparsity rate, number of rows, etc...). Many of them follow the general expression

$$\langle V \rangle^z = V + z \cdot (V)^{1/2}$$

(10)

These expressions, as has already been pointed out, are only valid under certain conditions. When these are not met, any expression of the form $\langle V \rangle^z$ must be considered as the maximum expected distance for $V$, endowing $z$ with a new meaning. This parameter becomes a 'shape factor' that indicates the extent to which entries are grouped in regions.

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 [26]. This distribution method, apart from achieving a good load balance, permits a simple assignment of the submatrix.
Fig. 2. Multiple recursive decomposition. (a) Level 1 horizontal partition, (b) level 2 horizontal partition, (c) vertical partitions.
ces to a PE network with hypercube or binary tree topology. However, as can be noted in Fig. 1, 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 Distribution (MRD). It can be considered a generalization of the BRD method for an arbitrary number of PEs.

Let us assume a $p$ processor network and let $P_1 \cdot P_2 \cdots P_k$ 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 $p \times q$, where the decomposition in prime factors of $p$ and $q$ is $p = P_1 \cdot P_2 \cdots P_a$ and $q = Q_1 \cdot Q_2 \cdots Q_b$, 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. Fig. 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 Fig. 2(c), where $q = Q_1 \cdot Q_2 = 4$).

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 ($i \in \{1, \ldots, a\}$), each PE $[r, s]$ divides the matrix it owns after the level $i - 1$ partition into $P_i$ equal submatrices, keeping submatrix $[r/(P_{i+1} \cdot P_{i+2} \cdots P_a)]$, and rejecting the rest. Next, (2) in the $i$th vertical partitions ($i \in \{1, \ldots, b\}$), each PE $[r, s]$ divides the matrix it owns after the previous partition into $Q_i$ equal submatrices, keeping submatrix $[s/(Q_{i+1} \cdot Q_{i+2} \cdots Q_b)]$, 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 $m \cdot n$.

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, according to Section 2.1 is $(m/p)^z_r$ (where $z_r$, as 'shape factor' depends on the concentration of entries in certain rows). After
the vertical division, this unbalance is partially corrected, resulting the maximum number of entries in a PE:

\[ w = \frac{\alpha}{p \cdot q} + \frac{m}{2 \cdot q} + \frac{1}{2} \cdot \left( \frac{m}{p} \right)^{r/2} \]  

(11)

In the Preprocessing stage each PE receives as many elements of vector \( a \) as columns it has in its local submatrix \( M_{rs} \). Thus, the MRD method induces \( q \) partitions of vector \( a \) for each PE row, assigning \( \langle n/q \rangle \) elements to each PE (where \( z_c \), as 'second level shape factor' depends on the concentration of entries in certain columns inside of certain rows). The Product stage is carried out in each PE, that will contain a subvector \( c' \) of length given by the \( p \) partitions of the rows of matrix \( M \). In this case, the MRD method induces \( p \) partitions of vector \( c \), assigning \( \langle m/p \rangle^{r/2} \) elements to each PE. Fig. 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 \( \langle m/p \rangle^{r/2} \) are required. After this stage, all of the PEs in a row of the mesh have a copy of the corresponding subvector. This information redundancy is useful in iterative applications that require the Redistribution stage.

The Redistribution of vector \( c^r \) 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 \( c^r \), in order to transform them into elements \( a^{r+1} \). \( q \) messages of size \( \langle n/q \rangle^{r/2} \) 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 \( p \times p \) and \( m \times m \), respectively. Eq. (11) 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 \( w + \langle m/p \rangle^{r/2} \cdot \log(p/2) \) 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 \( c \): \( w + \langle m/p \rangle^{r/2} + \langle m/p \rangle^{r/2} \) floating point data and \( w + \langle m/p \rangle^{r/2} \) 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 \( \langle m/p \rangle^{r/2} \) and \( \langle m/p \rangle^{r/2} \).

It can be easily seen that the MRD distribution scheme encompasses, as particular cases, the BRD [3,26] and One Way Strip Partitioning (OSP) [2,6,9,24] 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.
Fig. 3. Scatter decomposition. (a) matrix partitioning, (b) matrix distribution, (c) local memories.

(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.

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 $p \times p$ PE network, so that each PE contains a fraction of each block.

Let us consider a sparse matrix M of size $m \times m$ partitioned into a set of submatrices $B(k, l)$ of size $p \times p$, so that $M_{ij} = B_{rs}$ where $i = p \cdot k + r$, $j = p \cdot l + s$ ($0 \leq i, j < m$); $k = \lfloor i/p \rfloor$, $l = \lfloor j/p \rfloor$ ($0 \leq k, l \leq m/p$); $r = i - k \cdot p$, $s = j - l \cdot p$ ($0 \leq r, s < p$). 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 $p \times p$ onto the PE mesh. The scatter method admits larger block sizes [14], but this increase in the grain causes a bigger load unbalance. Fig. 3 shows the distribution of the blocks of M onto a mesh.

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 [19] (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 \([m/p]\) 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 \((\text{block band})\). 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 \( \phi \) that was predetermined. The local memory required by a BBS scheme is a function of parameter \( \phi \). Given the broad variability interval of \( \phi \) we must perform a statistical estimation of it. In general we will consider equation

\[
 w(\phi) = \frac{\alpha}{p^2} + z(\phi) \cdot \left( \frac{\alpha}{p^2} \right)^{1/2}
\]  

(12)

where \( z(\phi) \) varies between \( z_\pi \) (when \( \phi = \alpha \)) and \( z_\pi \cdot [m/p]^{1/2} \) (if \( \phi = 0 \)), and \( z_\pi \) is obtained as shown in Section 2.1. It can be easily seen that Eq. (12) 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 \( \phi \) 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 \textbf{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

\[
 w = \frac{\alpha}{p^2} + z_\pi \cdot \left( \frac{\alpha}{p^2} \right)^{1/2} = \left( \frac{\alpha}{p^2} \right) z_\pi
\]  

(13)

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 \( [m/p] \) blocks of size \( p \), where each block \( a' \) multiplies the \( l \)th block column of matrix \( M \). Each element of block \( a' \) 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 redundancy 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 \( p \times p \). 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 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 Eq. (13)).

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 \([m/p]\) partial additions without communications. Next, using the cascade addition algorithm all the PEs of the \( r \)th row obtain the \([m/p]\) elements of vector \( c \). This phase requires \( p \) messages of size \([m/p]\). Summarizing, in the Collection stage of the BRS method \( w + [m/p] \cdot \log_2(p/2) \) additions are carried out.

In the Redistribution stage \((a^{r+1} = f(c^r))\) the \([m/p]\) values of \( c^r \) 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 \([m/p]\) 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. In order to facilitate the comparison of these, we will substitute the variables until they are reduced to three: \( a \), \( p \), and \( m \) (the mesh and matrix \( M \) are square). To these, we will add the constant \( z \). This constant, the meaning and values of which are described in previous sections, provides a modification of the
sizes of the local submatrices. Any expression of the form $\langle \text{mean} \rangle^2$ must be taken as the maximum expected distance from the mean value. 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.

In the next paragraphs we briefly describe two more methods, which we will compare with the MRD and BRS proposed in Sections 3 and 4, respectively: D-way Strip Partitioning and Snake. Other methods not included in this analysis we name Miscellaneous. As miscellaneous methods we include regular and irregular distribution methods. The objective of a regular method is just adapts to the topology of the network (as the Boxwise Decomposition [3,15]); however it does not attempt to balance the computational workload. In practice, a large number of regular distributions for meshes can be considered as particular instances of the Scatter distribution following the general method proposed in [16]. Among the methods with irregular distribution we must point out Hinz's N4 partitioning [16], that uses the Boxwise Decomposition as input. The vertices of the subdomains are moved in the two dimensions preserving the topology of the mesh but balancing the load in each subdomain. However, the irregularity of these squares is not very compatible with moving data by rows or columns as required for the SpMxV product. We can also include in the group of irregular methods those proposed by Fox [13]: Simulated Annealing (SA), Neural Nets (NN) and Orthogonal Recursive Bisection (ORB). The decompositions based on SA or NN obtain very good load balances but are not very practical when the size of the problem and/or the PE network is significant, the ORB method is the extension of the BRD method [3] to irregular partitions. Finally, there are other techniques that, by means of permutations of the rows and columns of the matrix, rearrange the matrix with the object of adapting it to a specific algorithm [5,21].

Under the D-way Strip Partitioning heading we find a broad spectrum of methods [2,6,9,24], based on the division of a computation domain into $N$ subdomains ('stripes') with similar computational loads and where each subdomain $k$ is connected to regions $k - 1$ and $k + 1$. This type of partitions (1-Way or 1-D Strip Partitioning) adapt perfectly to one dimensional PE networks. When these techniques are applied to sparse matrices, the two most evident options are row by row partition or column partition (see Fig. 4 considering a 12 PE network).

When the dimension of the PE network (D) is higher than one, the proposed techniques combine D one dimensional partitions, carried out in orthogonal (or almost orthogonal) directions, generating a partition that is perfectly adapted to the communications model of the network, but exhibits poor computational load
distribution. Consequently, the solution of the SpMxV product in a two dimensional mesh is more efficient if we divide matrix $M$ by means of partitions of a single dimension (one dimensional mesh) and distribute each submatrix to one PE. Out of the two options (by rows or columns) we will only analyze the first one, as it is the most adequate for the calculations required.

The One Way Row Strip Partitioning (Row OSP) method decomposes matrix $M$ of dimensions $m \times m$ into $p^2$ submatrices of dimensions $m_r \times m$, where $m = \Sigma m_r$. A way of carrying out the partition consists in applying the MRD method performing the successive partitions in the same direction. Using the statistical approximation of Section 2.1, the maximum number of rows in a submatrix is

$$m_r = \left( \frac{m}{p^2} \right)^{1/2} \cdot \frac{m}{2} + z_r \cdot \left( \frac{m}{p^2} \right)^{1/2}$$  \hspace{1cm} (14)$$

The number of entries in each PE obtained is always lower than $\alpha/p^2 + m/2$. The quantity $m/2$ appears as a consequence of the fact that the division will not affect elements belonging to the same row of the matrix. To the memory needed for storing this floating point data we must add the quantities $m$ and $m_r$ for storing the elements of vectors $a$ and $c$ in each PE. Using storage-by-row $\alpha_r + m_r$ integers are needed for data addressing.

No communications are necessary during the Product and Collection phases, as they are completed inside each PE. The Redistribution of vector $c^r$, using the Row OSP method requires 'all to all' communications, producing $2 \cdot p$ messages in the two dimensional mesh. The size of all the messages for horizontal and vertical communications is $\langle m/p^2 \rangle z_r$ and $\langle m/p \rangle z_r$, respectively. The computation of $a^{r+1} = f(c^r)$ is carried out before the Redistribution stage (approximately $\langle m/p^2 \rangle z_r$ times). Finally, it can be noted that the Row OSP partition is specially effective
during the Collection stage, but forces the inclusion of a copy of vector a in each PE, reducing the scalability of the method.

The last method we will consider is the one proposed by Misra and Kumar [17,18], we name Snake Method, for the iterative solution of sparse linear systems [18] and the implementation of a sparse artificial neural network [17]. In this method the Matrix M is partitioned and distributed in a Column OSP, but the partition will also affect entries belonging to a single column. All the entries in a column are mapped in a column connected set of PEs following a snaked assignment (Snake-Like Column Major Order). 3 \cdot p messages are the maximum required messages for the Distribution. Exactly \( \alpha/p^2 \) products are carried out in each PE during the Product stage, without any communications.

For the Collection stage, the matrix R will be partitioned and distributed in a Row OSP, but the partition will also affect entries belonging to a single row. All the entries in a row are mapped in a row connected set of PEs (Snake-like row major order). However, to move the \( R_{ij} \) from the initial Snake-like Column Major Order to the Snake-like Row Major Order can be critical if it is not done in an effective manner. Misra and Kumar proposed a technique that perform this move in three stages, which require a total of \( 3 \cdot \alpha/p \) messages between PEs and an additional calculation in the Preprocessing stage executed in \( O(\alpha \cdot \log_2 \alpha) \) in one PE. Once the values of \( R_{ij} \) are in row connected regions, the collection is carried out in \( 3 \cdot p \) steps, performing the additions in parallel using the cascade addition algorithm in the new regions so that all of the PEs of each region \( j \) finally contains the results of the product \( (c_i = \Sigma R_{ij}) \). In the Redistribution stage the results stored in the row connected regions (elements of \( c^r \)) have to be sent to the PEs of the column connected regions. As the regions in both distributions are randomly distributed among all of the PEs and all of the planes, the complexity of the redistribution is very high. The technique proposed by the authors is the following: Establish beforehand which PEs are the 'owners' of elements \( a_i \). Each element \( c^r_j \) will be sent to this PE and from it, once \( a_i^{r+1} \) has been computed, it will be distributed to any PE of the column connected region. As each PE is the owner of, at most, \( m/p^2 \) elements of \( a \), the number of computations of function \( f \) is \( m/p^2 \). The number of messages in each iteration is \( 6 \cdot \alpha/p + 3 \cdot p \) of size 1 (or \( 5 \cdot p \) messages of size \( \alpha/p^2 \)).

The result of the analysis performed is specified in Table 2. In the first column of the table we have expressed the difficulty of implementation for each distribution 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 \( \phi \)) is a quantity between the two extreme cases presented in the table. We
| **Table 2**  
Comparative parameters |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM/MDNG</td>
</tr>
<tr>
<td>Snake</td>
</tr>
<tr>
<td>Scatter (BSS, ( \Psi = 0 ))</td>
</tr>
<tr>
<td>Scatter (BSS, ( \Psi = 0 ))</td>
</tr>
<tr>
<td>Block Row Scatter</td>
</tr>
<tr>
<td>1_Way Str. Row</td>
</tr>
<tr>
<td>Multiple Recurs. Decomp.</td>
</tr>
</tbody>
</table>
Fig. 5. Floating points operations. (a) Sparsity rate $\beta = 0.1$, (b) sparsity rate $\beta = 0.01$. 
Fig. 6. Local memories. (a) Sparsity rate $\beta = 0.1$, (b) sparsity rate $\beta = 0.01$. 
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.

Fig. 5 shows the normalized total number of floating point operations generated by each distribution method in a square mesh with 256 PEs \((p = 16)\) and different problem sizes \((m)\). Both coordinates are in a logarithmic scale \((\log_{10})\) and the normalization is obtained by dividing the estimated number of computations by \(2^{a/p^2}\). We have considered two sparsity rates, \(\beta = 0.1\) (Fig. 5(a)) and \(\beta = 0.01\) (Fig. 5(b)). The Snake method 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 II) and more local memory in each PE (see Fig. 6). 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 \((\alpha)\) method presents an optimal behavior in the number of products but generates a disproportionate number of additions, out of the range of the variables for Fig. 5. This problem was already pointed out in [1]. When \(\phi\) decreases, the number of additions is significantly reduced at the price of an increase in the number of products. In Fig. 5 we have included the plot associated with the new Block Row Scatter method we propose to distribute sparse matrices. 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 Fig. 6). It also simplifies programming of the Scatter distributions.

The number of messages, together with their size, of the Collection \((r)\) and Redistribution \((d)\) stages are reflected in the fifth and sixth columns of Table 2. 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. In order to alleviate its excessive number of messages, we have modified the Redistribution stage proposed by Misra and Kumar [17]. Modification that consists in the propagation of all the elements \(a_i^{\nu+1}\), computed in the regions connected by rows (owners of \(c_i^{\nu}\)), to all the PEs. Each PE only picks up those it needs, making the three step distribution used in [17] unnecessary. In each iteration, before carrying out the Product stage, \(\alpha/p^2\) computations of function \(f\) and \(2 \cdot p\) messages of size \(\alpha/p^2\) are required. 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.

Fig. 6 shows the normalized size of the local memory of a PE as a function of the size of the problem (logarithmic scale) considering a square mesh with 256 PEs and the two sparsity rates of Fig. 5. The memory required for the MRD and Row
Fig. 7. Scalability. (a) Floating points operations versus size of the mesh ($p^2$), (b) local memories versus size of the mesh ($p^2$).
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 Block Row Scatter method needs an intermediate amount of memory between the two extremes of the BBS Scatter method. Finally, we must point out that the normalized memory size associated with the Snake method is not dependent on the size of the problem \(m\) or the sparsity rate \(\beta\).

Fig. 7 shows the normalized number of arithmetic computations in a logarithmic scale (Fig. 7(a)) and the normalized memory size (Fig. 7(b)) for a PE as a function of the number of PEs in the mesh. We have considered a sparsity rate of \(\beta = 0.1\) 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 Fig. 7 is accentuated.

The last column of Table 2 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 \(\langle m / p^2 \rangle\). 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.

Finally, we want to point out that the results shown in Figs. 5, 6 and 7 have been obtained assuming that matrix \(M\) exhibits random sparsity. Nevertheless, the values reported here are also, in general, valid for matrices whose data are grouped (included in the banded matrices). This is so because the values of the shape parameters of the matrix \(z_z, z_r,\) and \(z_c\) scarcely influence the determination of the number of products and additions or the sizes of the local memories. They influence, however, the size of the messages and the number of calculations of \(f\), allowing this way the Snake and Scatter techniques to be less sensitive to the possible groupings of entries in the matrix. For reasons of space, we will not continue with the analysis of other sparse matrices. More information on this topic can be obtained elsewhere [23].

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 lose 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 [20]. 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


