The *CFS* and *ED* Data Distribution Schemes for Sparse Arrays on Distributed Memory Multicomputers[1]

Chun-Yuan Lin, Yeh-Ching Chung[2], and Jen-Shiuh Liu
Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, ROC
Tel: 886-4-24517250 x3765
Fax: 886-4-24516101
Email: {cylin, ychung, liuj}@iecs.fcu.edu.tw

Abstract

A data distribution scheme of sparse arrays on a distributed memory multicomputer, in general, is composed of three phases, data partition, data distribution, and data compression. In the data partition phase, a global sparse array is partitioned into some local sparse arrays. In the data distribution phase, these local sparse arrays are distributed to processors. In the data compression phase, a local sparse array is compressed by some data compression methods in order to obtain better performance for sparse array operations. To implement the data distribution scheme, methods proposed in the literature first perform the data partition phase, then the data distribution phase, followed by the data compression phase. We called this scheme as Send Followed Compress (SFC) scheme. In this paper, we propose two other data distribution schemes, Compress Followed Send (CFS) and Encoding-Decoding (ED), for sparse array distribution. In the CFS scheme, the data compression phase is performed before the data distribution phase. In the ED scheme, the data compression phase can be divided into two steps, encoding and decoding. The encoding step and the decoding step are performed before and after the data distribution phase, respectively. To evaluate the CFS and the ED schemes, we compare them with the SFC scheme. In theoretical analysis, we analyze the SFC, the CFS, and the ED schemes in terms of the data distribution time and the data compression time. In experimental test, for all test cases, we implemented these schemes on an IBM SP2 parallel machine. From the experimental results, the CFS and the ED schemes outperform the SFC scheme for all test cases. For the CFS and the ED schemes, the ED scheme outperforms the CFS scheme.

Index Terms — Data distribution schemes, Data compression methods, Partition methods, Sparse ratio, Distributed memory multicomputers

1. The work of this paper was partially supported by NSC under contract NSC90-2213-E-035-019
2. The corresponding author.

1. Introduction

Array operations are useful in a large number of important scientific codes, such as molecular dynamics [16], finite-element methods [23], climate modeling [40], etc. A data distribution scheme of sparse arrays on a distributed memory multicomputer, in general, is composed of three phases, data partition, data distribution, and data compression. In the data partition phase, a global sparse array is partitioned into some local sparse arrays. In the data distribution phase, these local sparse arrays are distributed to processors. In the data compression phase, a local sparse array is compressed by some data compression methods in order to obtain better performance for sparse array operations.

To implement the data distribution scheme, many methods have been proposed in the literature [2, 10-14, 37-42, 45]. Among them, the *Multiple Recursive Decomposition* (*MRD*) scheme [2, 42] have been popularly used to solve other important issues for sparse array problems [2-7, 10-14, 19-22, 37-42, 45]. In the data partition phase, the *MRD* scheme uses a 2D mesh partition with load-balancing method that is similar to (Block, Block) data distribution schemes used in Fortran 90 [1]. In the data distribution phase, both methods send local sparse arrays to processors. In the data compression phase, both methods use either the *Compressed Column Storage* (*CCS*) scheme [8] or the *Compressed Row Storage* (*CRS*) scheme [8] to compress the local sparse array in each processor.

For methods mentioned above, the three phases of the data distribution scheme are performed in the following order, the data partition phase, then the data distribution phase, followed by the data compression phase. A data distribution scheme with this order is called the *Send Followed Compress* (*SFC*) scheme. In this paper, we propose two data distribution schemes, *Compress Followed Send* (*CFS*) and *Encoding-Decoding* (*ED*), for sparse array distribution. In the *CFS* scheme, the data compression phase is performed before the data distribution phase. The three phases in the *CFS* scheme are performed in the following order, the data partition phase, then data compression phase, followed by the data distribution phase. The *ED* is a novel concept in which the data compression phase can be divided into two steps, *encoding* and *decoding*. The encoding step and the decoding step are performed before and after the data distribution phase, respectively. In encoding step, we encode information of nonzero array elements into a special buffer for each local sparse array. In decoding step, the special buffer is decoded into a compressed local sparse array. For the *ED* scheme, the data partition phase is performed first, then the encoding step, followed by the data distribution phase and the decoding step.

To evaluate the *CFS* and the *ED* schemes, we compare them with the *SFC* scheme. In the data partition phase, the 2D mesh partition with load-balancing method is used for these three schemes. In the data distribution phase, local sparse arrays, whether compressed or not, are sent to processors in sequence. In the compression phase, the *CRS/CCS* methods are used to compress sparse local arrays for the *SFC* and the *CFS* schemes while the encoding/decoding step is used for the *ED* scheme. Based on the methods used in the three phases, both theoretical analysis and experimental test were conducted. In theoretical analysis, we analyze the *SFC*, the *CFS*, and the *ED* schemes in terms of the data distribution time and the data compression time. Here, we do not consider the data partition time since the comparisons of the data distribution time and the data compression time of these three schemes are based on the same partition methods. In experimental test, we implemented the *SFC*, the *CFS*, and the *ED* schemes on an IBM SP2 parallel machine. From the experimental results, for all test cases, the *CFS* and the *ED* schemes outperform the *SFC* scheme. The reason is that we do not send entire local sparse arrays to processors in the *CFS* and the *ED* schemes. The data distribution time can be reduced. For the *CFS* and the *ED* schemes, the *ED* scheme outperforms the *CFS* scheme for all test cases. The reason is that, for the *ED* scheme, the data distribution time is less than that of the *CFS* scheme.

This paper is organized as follows. In Section 2, a brief survey of related work will be presented. Section 3 will describe the *SFC*, the *CFS*, and the *ED* schemes in detail. Section 4 will analyze the theoretical performance for the *SFC*, the *CFS*, and the *ED* schemes based on the 2D mesh partition with load-balancing method. The experimental results of these three schemes will be given in Section 5.

2. Related Work

Many methods have been proposed in the literature to implement the data distribution scheme [2, 10-14, 37-42, 45]. Zapata *et al.* [2, 42] have proposed a distribution scheme, *MRD*, for two-dimensional sparse arrays. Based on the *MRD* scheme, they solve other important problems based on sparse arrays [2-4, 6-7, 37-42, 44]. The *MRD* scheme can be considered as a generalization of the *Binary Recursive Decomposition* [9], a well-known data distribution scheme. For the *MRD* scheme, each processor has the same number of nonzero array elements, yet each processor has different size of local sparse array. The data compression time is determined by a processor, which has largest size of local sparse array. The data compression time for the *MRD* scheme will be large when nonzero array elements were in a part of a global sparse array. The reason is that, for the *MRD* scheme, there exists at least one processor whose local sparse array has the size similar to that of the global sparse array. Therefore, the data compression time for the *MRD* will be large in this situation.

Ziantz *et al.* [45] proposed a run-time optimization technique that was applied to sparse arrays compressed by the *CRS*/*CCS* methods for array distribution and off-processor data fetching to reduce both the communication and computation time. They used the block data distribution scheme with a bin-packing algorithm. The block data distribution scheme with a bin-packing algorithm belongs to the *SFC* scheme.

Lee *et al*. [10-14] presented an efficient library for parallel sparse computations with Fortran 90 array intrinsic operations. They provide a new data compression method, which is based on the *CRS*/*CCS* methods for two-dimensional sparse arrays, for multi-dimensional sparse arrays. Based on the *MRD* scheme, they also provide a new data distribution scheme for multi-dimensional sparse arrays. Their scheme is similar to (*, …, Block, Block) data distribution scheme used in Fortran 90. Their approach is promising in speeding up sparse array computations using array intrinsic functions on both sequential and distributed memory environments.

3. The *SFC*, *CFS* and *ED* Schemes

In the following, we describe the *SFC*, the *CFS*, and the *ED* schemes in detail. We assume that a two-dimensional global sparse array is given.

3.1 The *SFC* Scheme

The *SFC* is an intuitive data distribution scheme. In the *SFC* scheme, the data partition phase is performed first, then the data distribution phase, followed by the data compression phase.

In the data partition phase, a global sparse array is partitioned into local sparse arrays by some partition methods. In this paper, the 2D mesh partition with load-balancing method is used to partition a global sparse array. The 2D mesh partition with load-balancing method tries to partition a global sparse array into some local sparse arrays such that each local sparse array has the same number of nonzero array elements. The details can be found in [2, 42]. Assume that an 8×10 sparse array *A* with 16 nonzero array elements (Figure 1) and four processors are given. The partition result for the sparse array *A* by using the 2D mesh partition with load-balancing method is shown in Figure 2. For the 2D mesh partition with load-balancing method, the four processors are treated as a 2×2 processor array.

In the data distribution phase, local sparse arrays are packed and sent to processors in sequence. Figure 3 shows the corresponding local sparse arrays received by each processor for the partition result shown in Figure 2.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \\ 0 & 11 & 12 & 0 & 13 & 0 & 0 & 0 \\ 14 & 0 & 0 & 15 & 0 & 0 & 16 & 0 \end{pmatrix}$$

Sparse array *A*

Figure 1: An 8×10 sparse array *A* with 16 nonzero array elements.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & & 5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & 8 & 0 \\ & & & & & & & & \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 & \\ 0 & 11 & 12 & 0 & 13 & 0 & 0 & 0 & \\ 14 & 0 & 0 & 15 & 0 & 0 & 16 & 0 & \end{pmatrix}$$

Figure 2: The partition result for the sparse array *A* by using the 2D mesh partition with load-balancing method.

$$\mathbf{P_{0,0}} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{P_{0,1}} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \\ 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 8 & 0 \end{pmatrix}$$

$$\mathbf{P_{1,0}} \quad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 11 & 12 & 0 \\ 14 & 0 & 0 & 15 \end{pmatrix} \qquad \mathbf{P_{1,1}} \quad \begin{pmatrix} 9 & 0 & 0 & 10 \\ 13 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 \end{pmatrix}$$

Figure 3: The corresponding local sparse arrays received by each processor for the partition result shown in Figure 2.

In the data compression phase, a local sparse array in each processor is compressed by a data compression method. In this paper, the *CRS* and the *CCS* methods are used to compress sparse local arrays for the *SFC* and *CFS* schemes. The *CRS* (*CCS*) method uses two one-dimensional integer arrays, *RO* and *CO*, and one one-dimensional floating-point array, *VL*, to compress all of nonzero array elements along the rows (columns for *CCS*) of the sparse array. Array *RO* stores information of nonzero array elements of each row (column for *CCS*). The number of nonzero array elements in the *i*th row (*j*th column for *CCS*) can be obtained by subtracting the value of *RO*[*i*] from *RO*[*i*+1]. Array *CO* stores the column (row for *CCS*) indices of nonzero array elements of each row (column for *CCS*). Array *VL* stores the values of nonzero array elements of the sparse array. The base of these three arrays is 0. Figure 4 show the compressed results by using the *CRS* method for the received local sparse arrays shown in Figure 3.

$P_{0,0}$

RO	1	2	2	3	3	4	5	5
CO	1	0	3	4				
VL	1	3	6	7				

$P_{1,0}$

RO	1	1	3	5
CO	1	2	0	3
VL	11	12	14	15

$P_{0,1}$

RO	1	1	2	3	4	4	4	5
CO	1	2	0	1				
VL	2	4	5	8				

$P_{1,1}$

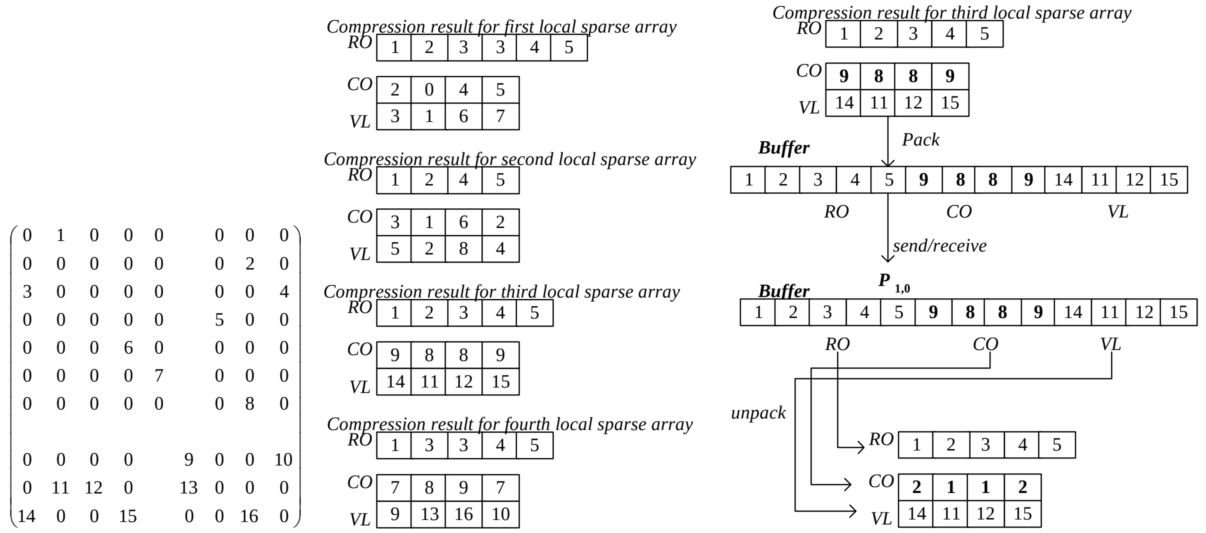
RO	1	3	4	5
CO	0	3	0	2
VL	9	10	13	16

Figure 4: The compressed results by using the *CRS* method for the received local sparse arrays shown in Figure 3.

3.2 The *CFS* Scheme

The *CFS* scheme is similar to the *SFC* scheme except that the data compression phase is performed before the data distribution phase. In the data partition phase, the 2D mesh partition with load-balancing method mentioned in the *SFC* scheme is used to partition a global sparse array. In the data compression phase, the *CRS*/*CCS* methods are used to compress local sparse arrays. In the compression, the values stored in *CO* are global array indices. In the data distribution phase, *RO*, *CO*, and *VL* for each local sparse array are packed and sent to its corresponding processor. After received the corresponding packed buffer, each processor unpacks the buffer to the corresponding *RO*, *CO*, and *VL*. Since the values stored in *CO* are global array indices in the compression phase, when unpack the received buffer, the values stored in *CO* may need to be converted to local array indices. We have the following case.

Case 3.2.1: When the 2D mesh partition with load-balancing method and the *CRS* (*CCS*) method is used in the data partition phase and the data compression phase, respectively, each processor $P_{i,j}$ converts the values stored in *CO* of the received buffer to the corresponding local array indices by subtracting *M* from each value stored in *CO* of the received buffer, where *M* is the total number of columns (rows for *CCS*) in $P_{i,0}$, $P_{i,1}$, …, $P_{i,j-1}$ ($P_{0,j}$, $P_{1,j}$, …, $P_{i-1,j}$ for *CCS*).



(a) The data partition phase (b) The data compression phase (c) The data distribution phase

Figure 5: An example of the *CFS* scheme.

An example of the *CFS* scheme is given in Figure 5 in which the 2D mesh partition with load-balancing method is used in the data partition phase and the *CCS* method is used in the data compression phase.

Figure 5(a) shows the partition result for the sparse array *A* (Figure 1) by using the 2D mesh partition with load-balancing method. Figure 5(b) shows the compressed results by using the *CCS* method for local sparse arrays shown in Figure 5(a). In Figure 5(b), the values stored in *CO* are global indices of global sparse array *A*, not local indices of a local sparse array. Figure 5(c) only shows the data distribution phase for $P_{1,0}$. In Figure 5(c), *RO*, *CO*, and *VL* for the first local sparse array are packed into a buffer and sent to $P_{1,0}$. After receiving the buffer, $P_{1,0}$ unpacks the received buffer to the corresponding *RO*, *CO*, and *VL*. According to Case 3.2.1 described above, $P_{1,0}$ converts the values stored in *CO* of the received buffer to the corresponding local array indices by subtracting 7 from each value stored in *CO* of the received buffer. For $P_{0,0}$, $P_{0,1}$, and $P_{1,1}$, the packing, send/receive, and unpacking procedures are similar to that of $P_{1,0}$.

3.3 The *ED* Scheme

The *ED* is a novel concept in which the data compression phase can be divided into two steps, encoding and decoding. The encoding step and the decoding step are performed before and after the data distribution phase, respectively. In the *ED* scheme, the data partition phase is performed first, then the encoding step, followed by the data distribution phase and the decoding step.

In the data partition phase, the 2D mesh partition with load-balancing method mentioned in the *SFC* scheme is used to partition a global sparse array. In the encoding step, each local sparse array is encoded into a special buffer *B*. Figure 6 shows the formats of the special buffer *B* for the *CRS*/*CCS* methods. In Figure 6, for the *CRS* (*CCS*) method, the R_i is used to store the number of nonzero array elements in a row (column for *CCS*) *i*. The $C_{i,j}$ and $V_{i,j}$ are used to store the column (row for *CCS*) index and the value of the *j*th nonzero array element in a row (column for *CCS*) *i*, respectively. The $C_{i,j}$ and $V_{i,j}$ are alternately stored in the buffer *B* and each $C_{i,j}$ is a global index of the global sparse array.

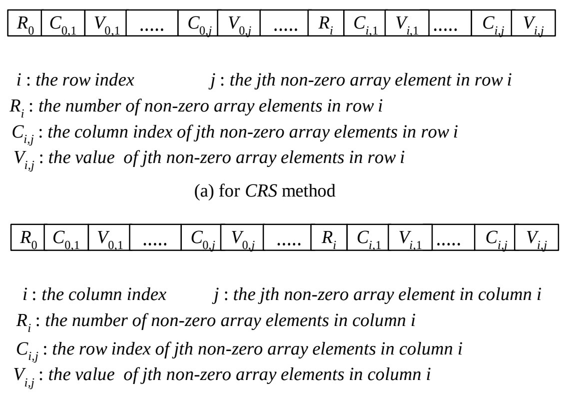


Figure 6: The formats of the special buffer *B*.

In the data distribution phase, these special buffers are sent to processors in sequence. In the decoding step, the special buffer *B* is decoded to get *RO*, *CO*, and *VL* in each processor. To get *RO*, in each processor, *RO*[0] is first initialized to 1. Then other values of *RO* are computed according to the formula $RO[i+1] = RO[i] + R_i$, where i = 0, 1, …, n and n is the number of rows in a local sparse array. To get *CO*, in each processor, we move $C_{0,0}$, $C_{0,1}$, …, $C_{0,j}$, $C_{1,0}$, $C_{1,1}$, …, $C_{1,j}$, …, $C_{i,0}$, $C_{i,1}$, …, $C_{i,j}$ stored in the special buffer to *CO*, where i = 0, 1, …, n, j = 0, 1, …,m, n is the number of rows of the local sparse array of a processor, and m is the number of nonzero array elements in row i. To get *VL*, we move all $V_{i,j}$

to *VL* in a similar manner as that of getting *CO*. Since each $C_{i,j}$ is a global array index in the encoding step, to decode the received special buffer in the decoding step, each $C_{i,j}$ may need to be converted to a local array index. We have the following case.

Case 3.3.1: When the 2D mesh partition with load-balancing method and the *CRS* (*CCS*) method are used in the data partition phase and the data compression phase, respectively, each processor $P_{i,j}$ converts each $C_{i,j}$ of the received special buffer to the corresponding local array index by subtracting M from each $C_{i,j}$ of the received special buffer, where M is the total number of columns (rows for *CCS*) in $P_{i,0}$, $P_{i,1}$, …, $P_{i,j-1}$ ($P_{0,j}$, $P_{1,j}$, …, $P_{i-1,j}$ for *CCS*).

An example of the *ED* scheme is given in Figure 7 in which the 2D mesh partition with load-balancing method is used in the data partition phase and the local sparse arrays are in *CCS* format. Figure 7(a) shows the partition result for the sparse array A (Figure 1) by using the 2D mesh partition with load-balancing method. Figure 7(b) shows the special buffers for local sparse arrays shown in Figure 7(a). In Figure 7(b), each $C_{i,j}$ is a global index of global sparse array A. Figure 7(c) shows the special buffers received by each processor. Figure 7(d) only shows the decoding step for $P_{1,0}$. After receiving the special buffer, to get *RO*, *RO*[0] is first set to 1. Then other values of *RO* are computed according to the formula $RO[i+1] = RO[i] + R_i$, where i = 0, 1, and 2. To get *CO*, we move $C_{0,0}$, $C_{1,0}$, $C_{2,0}$, and $C_{3,0}$ stored in the special buffer to *CO*. According to Case 3.3.1 described above, $P_{1,0}$ subtracts 7 from $C_{0,0}$, $C_{1,0}$, $C_{2,0}$,and $C_{3,0}$ of the received special buffer to convert them to the desired local array indices. To get *VL*, we move $V_{0,0}$, $V_{1,0}$, $V_{2,0}$, and $V_{3,0}$ stored in the special buffer to *VL*. For $P_{0,0}$, $P_{0,1}$, and $P_{1,1}$, the decoding step is similar to that of $P_{1,0}$.

4. Theoretical Analysis

In this section, we analyze the *SFC*, the *CFS*, and the *ED* schemes for two-dimensional sparse arrays in terms of the data distribution time and the data compression time. Here, we do not consider the data partition time since the comparisons of the data distribution time and the data compression time of these three schemes are based on the same partition methods. In the data partition phase, the 2D mesh partition with load-balancing method is used for these three schemes. In the compression phase, the *CRS/CCS* methods are used to compress sparse local arrays for the *SFC* and the *CFS* schemes while the encoding/decoding step is used for the *ED* scheme. In the following, we list the notations used in the theoretical analysis.

- $T_{Startup}$ is the startup time for a communication channel.
- T_{Data} is the transmission time for sending an array element through a communication channel.
- $T_{Operation}$ is the operation time for an array element. In order to simplify the analysis, we use $T_{Operation}$ to present any operation cost for an array element, such as memory access, addition or subtraction operations, etc.
- $T_{Distribution}$ is the data distribution time for the data distribution phase. The data distribution time includes the packing/unpacking time and send/receive time.
- $T_{Compression}$ is the data compression time for the data compression phase. For the *ED* scheme, the data compression time is the sum of the encoding time and the decoding time in the encoding and the decoding steps, respectively.
- A is an $n\times n$ global sparse array.
- p is the number of processors.
- s is the sparse ratio of A.

$\alpha = \{\alpha_i \mid i = 0,1,...p-1\}$ is the set of sparse ratios for each local sparse array. The space ratio for a local sparse array is the size of a local sparse array divided by the size of the global sparse array A. The largest space ratio in α is denoted as α'. The dimension of the largest local sparse array is $r'\times q'$ = $\alpha' n^2$.

4.1 The 2D Mesh Partition with Load-Balancing Method

Assume that A and $p = r\times q$ processors are given. The number of nonzero array elements in A is sn^2.

4.1.1 The *CRS* method

A. The *SFC* scheme

For the *SFC* scheme, the 2D mesh partition with load-balancing method partition A into $r\times q$ local sparse arrays and the number of nonzero array elements for each local sparse array is $sn^2/r\times q$. In the data distribution phase, local sparse arrays are sent to processors. For a two-dimensional spare array in the 2D mesh partition with load-balancing method, array elements in a local sparse array are not continuous. Therefore, local sparse arrays are sent to processors after packing into buffers. The data distribution time $T_{Distribution}$ is ($r\times q\times T_{Startup}$ + $n^2\times T_{Data}$ + $n^2\times T_{Operation}$) that is determined by the size of a global sparse array. In the data compression phase, local sparse arrays are compressed by the *CRS* method. Therefore, the data compression time $T_{Compression}$ is $((n^2\times(\alpha' + (3/_{r\times q})\, s))\times T_{Operation})$ that is determined by the processor that has the largest size of local sparse array.

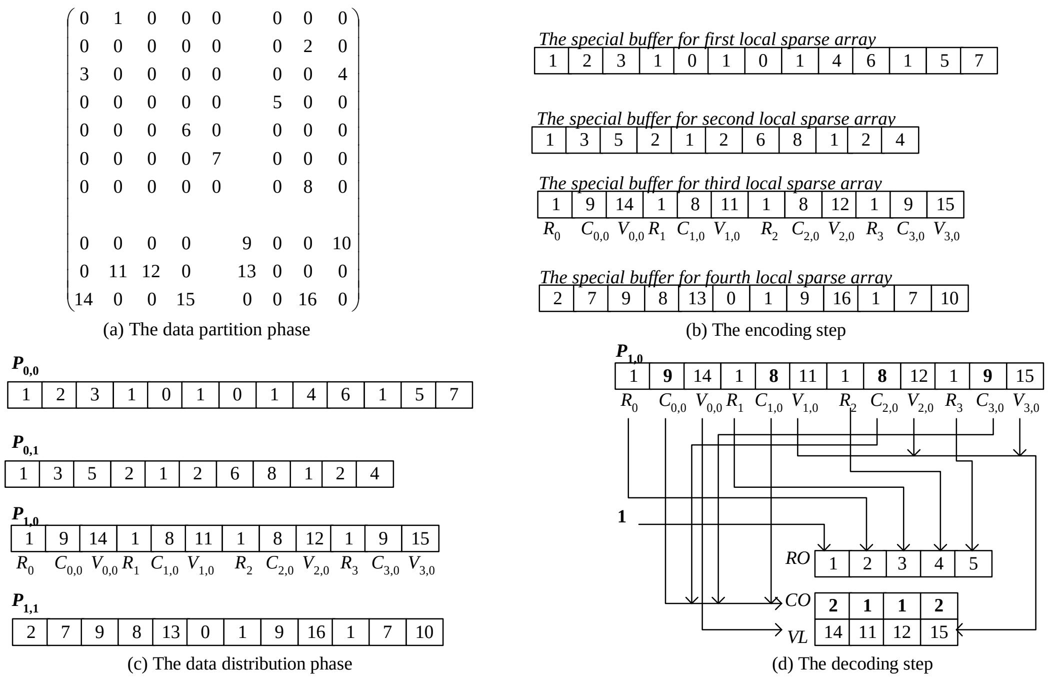


Figure 7: An example of the *ED* scheme.

B. The *CFS* scheme

In the data compression phase, local sparse arrays are compressed by the *CRS* method. This phase is similar to compress a global sparse array by the *CRS* method. Therefore, the data compression time $T_{Compression}$ is $(n^2\times(1+3s))\times T_{Operation}$ that is determined by the size of a global sparse array. In the data distribution phase, the compressed results are first packed into buffers. These buffers are then sent to the corresponding processors. After receiving the corresponding buffer, each processor unpacks the buffer to get the desired *RO*, *CO*, and *VL*. The values stored in *CO* need to be converted to local sparse indices in each processor according to Case 3.2.1. The packing time is $(n^2\times(3/(r\times q))s+r'+1)\times T_{Operation}$, the send/receive time is $r\times q\times T_{Startup}+(2n^2s+qn+rq)\times T_{Data}$, and the unpacking time is $(2n^2s+qn+rq)\times T_{Operation}$. Therefore, the data distribution time $T_{Distribution}$ is $r\times q\times T_{Startup}+(2n^2s+qn+rq)\times T_{Data}+(2n^2s+n^2\times(3/(r\times q))s+r'+qn+rq+1)\times T_{Operation}$. The number of nonzero array elements of a global sparse array determines the data distribution time.

C. The *ED* scheme

In the encoding step, local sparse arrays are encoded into special buffers. The encoding time is $(n^2\times(1+3s))\times T_{Operation}$ that is determined by the size of a global sparse array. In the data distribution phase, these special buffers are sent to processors. The data distribution time $T_{Distribution}$ is $(r\times q\times T_{Startup}+(2n^2s+qn)\times T_{Data})$ that is determined by the number of nonzero array elements of a global sparse array. In the decoding step, the special buffer *B* in each processor is decoded. The $C_{i,j}$ stored in the special buffer need to be converted to local sparse indices in each processor according Case 3.3.1. The decoding time is $(n^2\times(3/(r\times q))s+r'+1)\times T_{Operation}$ that is determined by the number of nonzero array element of a local spare array. The data compression time $T_{Compression}$ is $(n^2\times(1+3s)+n^2\times(3/(r\times q))s+r'+1)\times T_{Operation}$.

Table 1 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method and the *CRS* method.

D. Discussions

From Table 1, we can see that the data distribution time of the *ED* scheme is less than that of the *CFS* scheme. The data distribution time of the *ED* scheme is less than that of the *SFC* scheme if the sparse ratio of a global sparse array is less than 0.5. Since the sparse ratio of a global sparse array is less than 0.5, the data distribution time of the *ED* scheme is less than that of the *SFC* scheme. We have the following remark.

Remark 1. The data distribution time of the *ED* scheme is less than that of the *SFC* and the *CFS* schemes.

For the data distribution time of the *CFS* scheme, it is less than that of the *SFC* scheme if the sparse ratio of a global sparse array is less than 0.5. We have the following remark.

Remark 2. The data distribution time of the *CFS* scheme is less than that of the *SFC* scheme.

Table 1: The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method and the *CRS* method.

Method	Complexity	Cost
SFC	$T_{Distribution}$	$r \times q \times T_{Startup} + n^2 \times T_{Data} + n^2 \times T_{Operation}$
	$T_{Compression}$	$(n^2 \times (\alpha' + (3/r \times q)\ s)) \times T_{Operation}$
CFS	$T_{Distribution}$	$r \times q \times T_{Startup} + (2n^2 s + qn + rq) \times T_{Data} + (2n^2 s + n^2 \times (3/r \times q) s + r' + qn + rq + 1) \times T_{Operation}$
	$T_{Compression}$	$(n^2 \times (1 + 3s)) \times T_{Operation}$
ED	$T_{Distribution}$	$r \times q \times T_{Startup} + (2n^2 s + qn) \times T_{Data}$
	$T_{Compression}$	$(n^2 \times (1 + 3s) + n^2 \times (3/r \times q) s + r' + 1) \times T_{Operation}$

Table 2: The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method and the *CCS* method.

Method	Complexity	Cost
SFC	$T_{Distribution}$	$r \times q \times T_{Startup} + n^2 \times T_{Data} + n^2 \times T_{Operation}$
	$T_{Compression}$	$(n^2 \times (\alpha' + (3/r \times q)\ s)) \times T_{Operation}$
CFS	$T_{Distribution}$	$r \times q \times T_{Startup} + (2n^2 s + rn + rq) \times T_{Data} + (2n^2 s + n^2 \times (3/r \times q) s + q' + rn + rq + 1) \times T_{Operation}$
	$T_{Compression}$	$(n^2 \times (1 + 3s)) \times T_{Operation}$
ED	$T_{Distribution}$	$r \times q \times T_{Startup} + (2n^2 s + rn) \times T_{Data}$
	$T_{Compression}$	$(n^2 \times (1 + 3s) + n^2 \times (3/r \times q) s + q' + 1) \times T_{Operation}$

For the data compression time of the *SFC*, the *CFS*, and the *ED* schemes, we have the following remark.

Remark 3. The data compression time of the *SFC* scheme is less than that of the *CFS* scheme that is less than that of the *ED* scheme.

From Table 1, for the overall performance of the *SFC*, the *CFS*, and the *ED* schemes, we have two remarks.

Remark 4. The *ED* scheme outperforms the *CFS* scheme.

Remark 5. The *ED* and the *CFS* schemes outperform the *SFC* scheme if the conditions $T_{Data} > (3s - \alpha' / 1 - 2s) T_{Operation}$ and $T_{Data} > (5s - \alpha' / 1 - 2s) T_{Operation}$ are satisfied, respectively.

4.1.2 The *CCS* method

Table 2 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method and the *CCS* method. From Table 2, for the data distribution time, the data compression time, and the overall performance of these three schemes, we have similar observations as those of Remarks 1, 2, 3, 4, and 5.

5. Experimental Results

In the experimental test, we implement the *SFC*, the *CFS*, and the *ED* schemes on an IBM SP2 parallel machine. In the partition phase, the 2D mesh partition with load-balancing method is implemented. In the compression phase, the *CRS* method is implemented. All methods are written in *C* + MPI (*Message Passing Interface*) codes. The sparse ratio is set to 0.1 for all two-dimensional sparse arrays used as test samples.

5.1 The 2D Mesh Partition with Load-Balancing Method

Table 3 shows the data distribution and the data compression time for the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method. From Table 3, for the data distribution time, we have the following observations.

1. The data distribution time of the *ED* scheme is less than that of the *SFC* and the *CFS* schemes.
2. The data distribution time of the *CFS* scheme is less than that of the *SFC* scheme.

These results match Remarks 1 and 2. For the data compression time, from Table 3, we have the following observation.

1. The data compression of the *SFC* scheme is less than that of *CFS* scheme is less than that of the *ED* scheme.

This result matches Remark 3. From experimental tests, we can estimate that $T_{Data} \approx 1.2 \times T_{Operation}$. For the overall performance, from Table 3, we have the following observations.

Table 3: The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition with load-balancing method.

No. of Processors	*Array Sizes* / *Methods-Costs*		120×120	240×240	480×480	960×960	1920×1920
2×2	*SFC*	$T_{Distribution}$	11.905	48.543	167.326	259.691	905.85
		$T_{Compression}$	0.665	2.565	9.515	34.905	136.747
	CFS	$T_{Distribution}$	3.538	9.82	35.644	55.252	204.104
		$T_{Compression}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{Distribution}$	1.659	4.701	16.718	25.695	100.251
		$T_{Compression}$	4.893	19.967	75.023	124.171	515.103
3×3	*SFC*	$T_{Distribution}$	13.219	50.372	169.201	263.424	913.466
		$T_{Compression}$	0.421	1.492	4.224	18.452	71.559
	CFS	$T_{Distribution}$	4.195	10.245	37.422	58.724	210.189
		$T_{Compression}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{Distribution}$	2.621	7.174	20.277	29.854	106.109
		$T_{Compression}$	5.245	20.542	76.542	127.254	521.524
4×4	*SFC*	$T_{Distribution}$	14.522	52.696	173.702	266.785	918.182
		$T_{Compression}$	0.339	0.998	3.355	10.742	38.227
	CFS	$T_{Distribution}$	5.803	12.298	42.391	63.154	220.962
		$T_{Compression}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{Distribution}$	3.702	9.143	23.209	32.293	110.895
		$T_{Compression}$	6.296	21.367	78.619	131.496	528.426
5×5	*SFC*	$T_{Distribution}$	16.656	55.321	177.226	273.247	925.524
		$T_{Compression}$	0.267	0.737	1.938	6.724	24.254
	CFS	$T_{Distribution}$	6.279	14.098	47.088	70.722	229.254
		$T_{Compression}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{Distribution}$	3.927	9.593	24.100	33.733	113.724
		$T_{Compression}$	6.667	22.459	80.426	140.741	540.141
6×6	*SFC*	$T_{Distribution}$	18.285	60.028	183.293	285.124	938.247
		$T_{Compression}$	0.184	0.588	1.228	4.425	19.827
	CFS	$T_{Distribution}$	7.155	15.895	52.006	79.752	240.841
		$T_{Compression}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{Distribution}$	4.177	10.093	25.090	34.649	115.602
		$T_{Compression}$	7.425	23.852	85.722	148.424	551.541

Time: *ms*

1. The *ED* scheme outperforms the *CFS* scheme.
2. The *CFS* and the *ED* schemes outperform the *SFC* scheme since the conditions $T_{Data} > (5/8)T_{Operation}$ and $T_{Data} > (3/8)T_{Operation}$ shown in Table 1 are satisfied, respectively.

These results match Remarks 4 and 5. From Table 3, we can see that the experimental results match the theoretical analysis in Table 1.

From the theoretical analysis and experimental results, for the *SFC*, the *CFS*, and the *ED* schemes, we have the following conclusions.

Conclusion 1: For the data distribution phase, the data distribution time of the *ED* scheme is less than that of the *SFC* and the *CFS* schemes. The data distribution time of the *CFS* scheme is less than that of the *SFC* scheme.

Conclusion 2: For the data compression phase, the data compression time of the *SFC* is less than that of the *CFS* scheme that is less than that of the *ED* scheme.

Conclusion 3: For the overall performance, the *ED* scheme outperforms the *CFS* scheme. For most of cases, the *CFS* and the *ED* schemes outperform the *SFC* scheme.

6. Conclusions and Future Work

In this paper, we have proposed two data distribution schemes, *CFS* and *ED*, for the distribution of sparse arrays on distributed memory multicomputers. To evaluate the *CFS* and the *ED* schemes, we compare them with the *SFC* scheme. In the data partition phase, the 2D mesh partition with load-balancing method is used for these three schemes. In the data distribution phase, local sparse arrays, whether compressed or not, are sent to processors in sequence. In the compression phase, the *CRS/CCS* methods are used to compress sparse local arrays for the *SFC* and the *CFS* schemes while the decoding/encoding step is used for the *ED* scheme. Based on the methods used in the three phases, both theoretical analysis and experimental test were conducted. In theoretical analysis, we analyze the *SFC*, the *CFS*, and the *ED* schemes in term of the data distribution time and the data compression time. In experimental test, we implemented the *SFC*, the *CFS*, and the *ED* schemes on an IBM SP2 parallel machine. From the experimental results, for all of test cases, the *CFS* and the *ED* schemes outperform the *SFC* scheme. The reason is that we do not send entire local sparse arrays to processors in the *CFS* and the *ED* schemes. The data distribution time can be reduced. For the *CFS* and the *ED* schemes, the *ED* scheme outperforms the *CFS* scheme for all test cases. The reason is that, for the *ED* scheme, the data distribution time is less than that for the *CFS* scheme.

In the future, we plan to work on the following directions. (1) Analyze the performance of the *SFC*, the *CFS*, and the *ED* schemes for multi-dimensional sparse arrays. (2) Develop efficient data distribution schemes for multi-dimensional sparse arrays based on the *extended Karnaugh map representation* (*EKMR*) scheme [30-33]. (3) Develop efficient data partition methods for multi-dimensional sparse arrays. We believe that these directions are of importance in parallel sparse array operations.

References

[1] Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, Brain T. Smith, and Jerrold L. Wagener. *Fortran* 90 *Handbook*. Intertext Publications/ McGraw-Hill Inc., 1992.

[2] R. Asenjo, L.F. Romero, Manuel Ujaldon and Emilio L. Zapata, "Sparse Block and Cyclic Data Distributions for Matrix Computations," *In Proceedings of High Performance Computing*: *Technology, Methods and Applications Advanced Workshop*, June 1994.

[3] R. Asenjo, O. Plata, J. Tourino, R. Doallo, and Emilio L. Zapata, "HPF-2 Support for Dynamic Sparse Computations," *Languages and Compilers for Parallel Computing*, 1998.

[4] Gerardo Bandera and Emilio L. Zapata, "Extending CRAFT Data-Distributions for Sparse Matrices," *2nd. European Cray MPP Workshop*, July 1996.

[5] Gerardo Bandera, Manuel Ujaldon, M.A. Trenas and Emilio L. Zapata, "The Sparse Cyclic Distribution against its Dense Counterparts," *11th International Parallel Processing Symposium* (*IPPS'97*), April 1997.

[6] Gerardo Bandera, Pablo P. Trabado and Emilio L. Zapata, "Local Enumeration Techniques for Sparse Algorithms," *In Proceedings of IEEE International Parallel Processing Symposium* (*IPPS'98*), Apr. 1998.

[7] Gerardo Bandera and Emilio L. Zapata, "Sparse Matrix Block-Cyclic Redistribution," *In Proceedings of IEEE International Parallel Processing Symposium* (*IPPS'99*), April 1999.

[8] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for the Iterative Methods*, 2nd Edition, SIAM, 1994.

[9] M.J. Berger and S.H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 570-580, 1987.

[10] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Compiler Optimization for Parallel Sparse Programs with Array Intrinsics of Fortran 90," *In the International Conference on Parallel Processing*, September, 1999.

[11] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Parallel Sparse Supports for Array Intrinsic Functions of Fortran 90," *accepted by Journal of Supercomputing*.

[12] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Towards Automatic Support of Parallel Sparse Computation in Java with Continuous Compilation," to appear in Concurrency: Practice and Experiences.

[13] Tyng-Ruey Chung, Rong-Guey Chang, and Jenq Kuen Lee, "Sampling and Analytical Techniques for Data Distribution of Parallel Sparse Computation," *In SIAM Conference on Parallel Processing for Scientific Computing*, March, 1997.

[14] Tyng-Ruey Chung, Rong-Guey Chang, and Jenq Kuen Lee, "Efficient Support of Parallel Sparse Computation for Array Intrinsic Functions of Fortran 90," *In the* 12*th* *ACM International Conference on Supercomputing*, July, 1998.

[15] M. Cierniak and W. Li, "Unifying Data and Control Transformations for Distributed Shared Memory Machines," *Technical Report*, November 1994.

[16] J.K. Cullum and R.A. Willoughby, "Algorithms for Large Symmetric Eignenvalue Computations," vol. 1 (birkhauser, Boston 1985).

[17] I. Duff, R.Grimes, and J. Lewis: Sparse matrix test problems. *ACM Transaction on Mathematical Software*, 15, 1-14, 1989.

[18] I. Duff, R.Grimes, and J. Lewis: *User's Giude for the Harwell-Boeing Sparse Matrix Collection*. CERFACS, Toulouse, France: Cedex 1992.

[19] B.B. Fraguela, R. Doallo, and Emilio L. Zapata, "Cache Misses Prediction for High Performance Sparse Algorithms, " 4*th* *International Euro-Par Conference* (*Euro-Par'*98), September 1998.

[20] B.B. Fraguela, R. Doallo, and Emilio L. Zapata, "Cache Probabilistic Schemeling for Basic Sparse Algebra Kernels Involving Matrices with a Non-Uniform Distribution, " 24*th* *IEEE Euromicro Conference*, pp.345-348, August 1998.

[21] B.B. Fraguela, R. Doallo, and Emilio L. Zapata, "Schemeling Set Associative Caches Behaviors for Irregular Computations, " *ACM International Conference on Measurement and Schemeling of Computer Systems* (*SIGMETRICS'*98), pp.192-201, June 1998.

[22] B.B. Fraguela, R. Doallo, and Emilio L. Zapata, "Automatic Analytical Schemeling for the Estimation of Cache Misses, " *International Conference on Parallel Architectures and Compilation Techniques* (*PACT'*99), October, 1999.

[23] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd ed. (Johns Hopkins Univ.Press, Baltimore, 1989)

[24] High Performance Fortran Forum, "High Performance Fortran Language Specification (version 2.0)," Rice Univ., Jan. 1997.

[25] Christoph W. Kebler, "Applicability of Automatic Program Comprehension to Sparse Matrix Computations," *In Proceedings of 3rd International Euro-Par Conference*, Aug 1997.

[26] Christoph W. Kebler and Craig H. Smith, "The SPARAMAT Approach to Automatic Comprehension of Sparse Matrix Computations," *In Proceedings of the Seventh International Workshop on Program Comprehension*, pp. 200-207, IEEE Computer Society Press, 1999.

[27] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill,

"Compiling Parallel Sparse Code for User-Defined Data Structures," *In Proceedings of* 8*th* *SIAM Conference on Parallel Processing for Scientific Computing*, March, 1997.

[28] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill, "A Relation Approach to the Compilation of Sparse Matrix Programs," *In Euro Par*, August, 1997.

[29] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill, "Compiling Parallel Code for Sparse Matrix Applications," *In Proceedings of the Supercomputing Conference*, August, 1997.

[30] Chun-Yuan Lin, Jen-Shiuh Liu, and Yeh-Ching Chung, "Efficient Representation Scheme for Multi-Dimensional Array Operations, " *IEEE Transactions on Computers,* Vol. 51, No. 3, pp. 327-345, March 2002.

[31] Chun-Yuan Lin, Jen-Shiuh Liu, and Yeh-Ching Chung, "Efficient Data Compression Methods for Multi-Dimensional Sparse Array Operations Based on the *EKMR* Scheme," *Accepted by The Eighth Workshop on Compiler Techniques for High Performance Computing* (*CTHPC*), March 2002.

[32] Jen-Shiuh Liu, Jiun-Yuan Lin, and Yeh-Ching Chung, "Efficient Representation for Multi-Dimensional Matrix Operations," *Proceedings of Workshop on Compiler Techniques for High Performance Computing* (*CTHPC*), pp. 133-142, March 2000.

[33] Jen-Shiuh Liu, Jiun-Yuan Lin, and Yeh-Ching Chung, "Efficient Parallel Algorithms for Multi-Dimensional Matrix Operations," *Proceedings of IEEE International Symposium on Parallel Architectures, Algorithms and Networks* (*I-SPAN*), Dec. 2000.

[34] Nikolay Mateev, Keshav Pingali, Paul Stodghill, and Vladimir Kotlyar, "Next-generation Generic Programming and its Application to Sparse Matrix Computations," *In Proceedings of International Conference on Supercomputing*, 2000.

[35] W. H. Press, S. A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in Fortran* 90*:* The Art of Parallel Scientific Computing. (Cambridge University Press, 1996)

[36] Peter D. Sulatycke and Kanad Ghose, "Caching Efficient Multithreaded Fast Multiplication of Sparse Matrices," *In Proceedings of the* 1*st* *Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, 1998.

[37] Manuel Ujaldon, Emilio L. Zapata, Barbara M. Chapman and Hans P. Zima, "New Data-Parallel Language Features for Sparse Matrix Computations," *9th IEEE International Parallel Processing Symposium* (*IPPS'95*), April 1995.

[38] Manuel Ujaldon, Shamik D. Sharma, Joel Saltz, and Emilio L. Zapata, "Run-time techniques for parallelizing sparse matrix problems," *Workshop on Parallel Algorithms for Irregularly Structured Problems*, 1995

[39] Manuel Ujaldon and Emilio L. Zapata, "Efficient Resolution of Sparse Indirections in Data-Parallel Compiler," *9th ACM International Conference on Supercomputing*, July 1995.

[40] M. Ujaldon, Emilio L. Zapata, Shamik D. Sharma, and Joel Saltz, "Parallelization Techniques for Sparse Matrix Applications," *Journal of parallel and distribution computing*, 1996.

[41] Manuel Ujaldon, Shamik D. Sharma, Emilio L. Zapata, and Joel Saltz, "Experimental Evaluation of Efficient Sparse Matrix Distributions," *In Proceedings of International Conference on Supercomputing*, 1996.

[42] Manuel Ujaldon, Emilio L. Zapata, Barbara M. Chapman, and Hans P. Zima, "Vienna-Fortran/HPF Extensions for Sparse and Irregular Problems and Their Compilation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no 10, October 1997.

[43] James B. White, and P. Sadayappan, "On Improving the Performance of Sparse Matrix-Vector Multiplication," *International Conference on High-Performance Computing*, 1997.

[44] Emilio L. Zapata, O. Plata, R. Asenjo and G.P. Trabado, "Data-Parallel Support for Numerical Irregular Problems," *Journal of Supercomputing*, 1999.

[45] Louis H. Ziantz, Can C. Ozturan, and Boleslaw K. Szymanski, "Run-Time Optimization of Sparse Matrix-Vector Multiplication on SIMD Machines," *In Proceedings International Conference of Parallel Architectures and Languages*, Athens, July, 1994.