# Data Distribution Schemes of Sparse Arrays on Distributed Memory Multicomputers

*Chun-Yuan Lin, Yeh-Ching Chung, and Jen-Shiuh Liu*
Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, ROC
TEL: 886-4-2451-7250x3765
FAX: 886-4-2451-6101
E-mail: {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. 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. 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. In experimental test, we implemented these schemes on an IBM SP2 parallel machine. From the experimental results, for most of test cases, the CFS and the ED schemes outperform the SFC scheme. For the CFS and the ED schemes, the ED scheme outperforms the CFS scheme for all test cases.*

Index Terms – Data distribution schemes, Data compression methods, Partition methods, Sparse ratio, distributed memory multicomputers.

## 1. Introduction

Array operations are useful in a large number of important scientific codes, such as molecular dynamics [7], finite-element methods [10], climate modeling [13], 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, 6, 13-14, 16]. Among them, the *Block Row Scatter* (*BRS*) scheme [2, 14] has been popularly used to solve other important issues for sparse array problems [2-3, 13-14]. In the data partition phase, the *BRS* scheme partition a global array into several blocks, all of the same spatial shape and size. In the data distribution phase, the *BRS* scheme send local sparse arrays to processors. In the data compression phase, the *BRS* scheme use either the *Compressed Column Storage* (*CCS*) scheme [4] or the *Compressed Row Storage* (*CRS*) scheme [4] to compress the local sparse array in each processor. For the *BRS* scheme, 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 *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.

To evaluate the *CFS* and the *ED* schemes, we compare them with the *SFC* scheme. In the data partition phase, many partition methods as block or cyclic partition methods can be used for these three schemes. For the page limitation, in this paper, the row partition, the column partition, and the 2D mesh partition methods that are similar to (Block, *), (*, Block), and (Block, Block) data distribution schemes used in Fortran 90 [1] are 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, many data compression methods in [4] can be used for these three schemes. In this paper, 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. 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. In experimental test, we implemented the *SFC*, the *CFS*, and the *ED* schemes on an IBM SP2 parallel machine. From the experimental results, for most of test cases, the *CFS* and the *ED* schemes outperform the *SFC* scheme. For the *CFS* and the *ED* schemes, the *ED* scheme outperforms the *CFS* scheme for all test cases.

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. 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 [2-3, 6, 13-14, 16]. Zapata *et al.* [2, 14] have proposed a data distribution scheme, *BRS*, for two-dimensional sparse arrays. Ziantz *et al.* [16] 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 that belongs to the *SFC* scheme. Lee *et al*. [6] presented an efficient library for parallel sparse computations with Fortran 90 array intrinsic operations. 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 data partition phase, a global sparse array is partitioned into local sparse arrays by some partition methods. In this paper, the row partition, the column partition, and the 2D mesh partition methods are used to partition a global sparse array. For simplicity, in the following, we use the row partition method as an example to describe the *SFC*, the *CFS*, and the *ED* data distribution schemes. The *SFC*, the *CFS*, and the *ED* data distribution schemes based on the column and 2D mesh partition methods are similar to those based on the row partition method.

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 row partition method is shown in Figure 2. 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. 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. The details for the *CRS* (*CCS*) method can be found in [4]. Figure 4 show 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, partition methods are 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*.

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

Sparse array $A$

**Figure 1: A 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 row partition method**.

$P_0$
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \end{pmatrix}$$

$P_2$
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \end{pmatrix}$$

$P_1$
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \end{pmatrix}$$

$P_3$
$$\begin{pmatrix} 0 & 11 & 12 & 0 & 13 & 0 & 0 & 0 \\ 14 & 0 & 0 & 15 & 0 & 0 & 16 & 0 \end{pmatrix}$$

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

$P_0$

| | | | | |
|---|---|---|---|---|
| $RO$ | 1 | 2 | 3 | 5 |
| $CO$ | 1 | 6 | 0 | 7 |
| $VL$ | 1 | 2 | 3 | 4 |

$P_1$

| | | | | |
|---|---|---|---|---|
| $RO$ | 1 | 2 | 3 | 4 |
| $CO$ | 5 | 3 | 4 | |
| $VL$ | 5 | 6 | 7 | |

$P_2$

| | | | |
|---|---|---|---|
| $RO$ | 1 | 2 | 4 |
| $CO$ | 6 | 4 | 7 |
| $VL$ | 8 | 9 | 10 |

$P_3$

| | | | | | | |
|---|---|---|---|---|---|---|
| $RO$ | 1 | 4 | 7 | | | |
| $CO$ | 1 | 2 | 4 | 0 | 3 | 6 |
| $VL$ | 11 | 12 | 13 | 14 | 15 | 16 |

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

Since values stored in *CO* are global array indices in the compression phase, when unpack the received buffer, values stored in *CO* may need to be converted to local array indices. We have the following cases.

Case 3.2.1: When the row (column) partition method and the *CRS* (*CCS* for column) method are used in the data partition phase and the data compression phase, respectively, the values stored in *CO* of the received buffer are desired local array indices. No conversion is needed.

Case 3.2.2: When the row (column) partition method and the *CCS* (*CRS* for column) method are used in the data partition phase and the data compression phase, respectively, each processor $P_i$ converts the values stored in *CO* of the received buffer to the corresponding local array indices by subtracting $N$ from each value stored in *CO* of the received buffer, where $N$ is the total number of columns (rows for column) in $P_0$, $P_1$, …, $P_{i-1}$.

Case 3.2.3: When the 2D mesh partition 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 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*).

An example of the *CFS* scheme is given in Figure 5 in which the row partition 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 row partition 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 $P_1$. In Figure 5(c), *RO*, *CO*, and *VL* for the first local sparse array are packed into a buffer and sent to $P_1$. After receiving the buffer, $P_1$ unpacks the received buffer to the corresponding *RO*, *CO*, and *VL*. According to Case 3.2.2 described above, $P_1$ converts the values stored in *CO* of the received buffer to the corresponding local array indices by subtracting 3 from each value stored in *CO* of the received buffer. For $P_0$, $P_2$, and $P_3$, the packing, send/receive, and unpacking procedures are similar to that of $P_1$.

## 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. In the data partition phase, the partition methods are 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. 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 $i = 0, 1, \ldots, n$ and $n$ is the number of rows in a local sparse array.

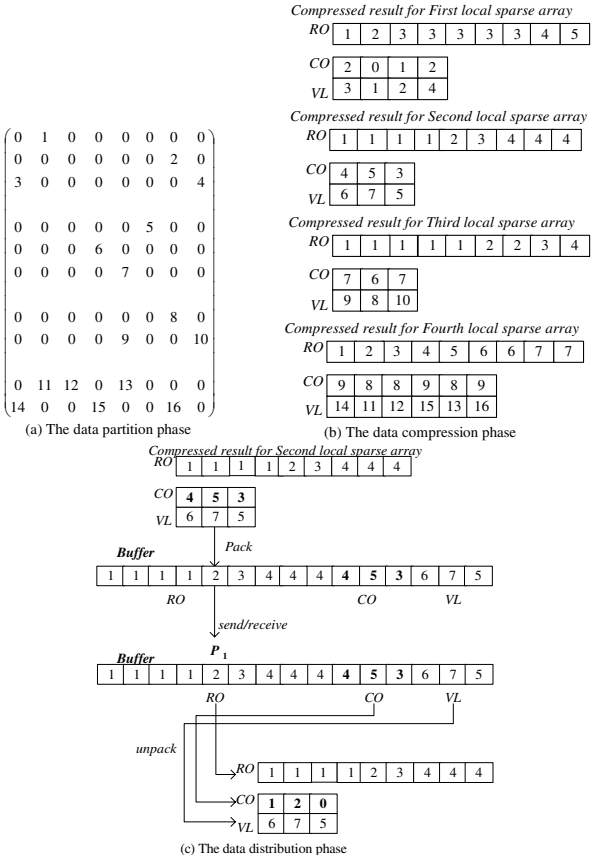
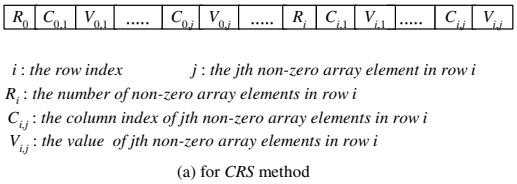
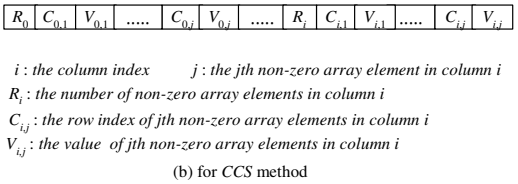



**Figure 5: An example of the *CFS* scheme.**




(a) for *CRS* method




(b) for *CCS* method

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

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 cases.

Case 3.3.1: When the row (column) partition method and the *CRS* (*CCS* for column) method are used in the data partition phase and the data compression phase, respectively, each $C_{i,j}$ of the received buffer is desired local array index. No conversion is needed.

Case 3.3.2: When the row (column) partition method and the *CCS* (*CRS* for column) method are used in the data partition phase and the data compression phase, respectively, each processor $P_i$ converts each $C_{i,j}$ of the received special buffer to the corresponding local array index by subtracting $N$ from each $C_{i,j}$ of the received special buffer, where $N$ is the total number of columns (rows for column) in $P_0$, $P_1$, …, $P_{i-1}$.

Case 3.3.3: When the 2D mesh partition 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 row partition 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 row partition 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 $P_1$. 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 $C_{3,0}$, $C_{4,0}$, and $C_{5,0}$ stored in the special buffer to *CO*. According to Case 3.3.2 described above, $P_1$ subtracts 3 from $C_{3,0}$, $C_{4,0}$, and $C_{5,0}$ of the received special buffer to convert them to the desired local array indices. To get *VL*, we move $V_{3,0}$, $V_{4,0}$, and $V_{5,0}$ stored in the special buffer to *VL*. For $P_0$, $P_2$, and $P_3$, the decoding step is similar to that of $P_1$.

## 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. For the page limitation, in this paper, we only list theoretical analysis results for these three schemes using the row partition method. However, we do give some experimental results for the cases where the column and the 2D mesh partition methods are used.

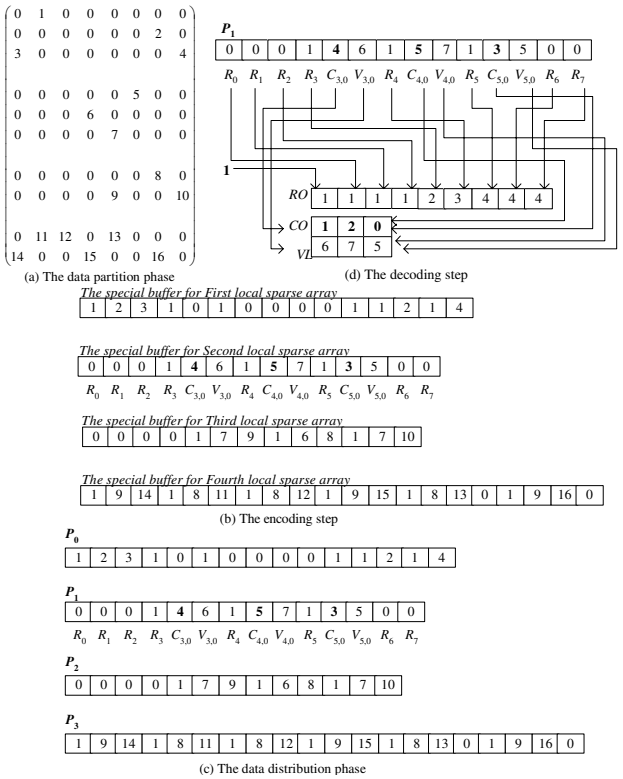



**Figure 7: An example of the *ED* scheme.**

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.
- $A$ is an $n \times n$ global sparse array.
- $p$ is the number of processors.
- $s$ is the sparse ratio of $A$.
- $S = \{s_i \mid i = 0,1,...p-1\}$ is the set of sparse ratios of local sparse arrays. The largest sparse ratio in $S$ is denoted as $s'$.

## 4.1 The Row Partition Method

Assume that $A$ and $p$ 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 row partition method partition $A$ into $p$ local sparse arrays and the size of each local sparse array is $\lceil n/p \rceil \times n$. The largest number of nonzero array elements among local sparse arrays is $\lceil n/p \rceil \times n \times s'$. For a two-dimensional spare array in the row partition method, array elements in a local sparse array are continuous. Therefore, local sparse arrays are sent to processors without packing into buffers. The data distribution time $T_{Distribution}$ is $(p \times T_{Startup} + n^2 \times T_{Data})$. In the data compression phase, local sparse arrays are compressed by the *CRS* method. Therefore, the data compression time $T_{Compression}$ is $(\lceil n/p \rceil \times n \times (1 + 3s'))$ $\times T_{Operation}$.

### B. The *CFS* Scheme

For the *CFS* scheme, the row partition method partition $A$ into $p$ local sparse arrays and the size of each local sparse array is $\lceil n/p \rceil \times n$. The largest number of nonzero array elements among local sparse arrays is $\lceil n/p \rceil \times n \times s'$. 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 $(n^2 \times (1 + 3s)) \times T_{Operation}$. 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* do not need to be converted to local sparse indices in each processor according to Case 3.2.1. The packing time is $(2n^2 s + n + p) \times T_{Operation}$, the send/receive time is $p \times T_{Startup} + (2n^2 s + n + p) \times T_{Data}$, and the unpacking time is $(\lceil n/p \rceil \times n \times (2s' + (1/n))) + 1) \times T_{Operation}$. Therefore, the data distribution time $T_{Distribution}$ is $p \times T_{Startup} + (2n^2 s + n + p) \times T_{Data} + (2n^2 s + (\lceil n/p \rceil \times n \times (2s' + (1/n))) + n + p + 1) \times T_{Operation}$.

### C. The *ED* Scheme

For the *ED* scheme, the row partition method partition $A$ into $p$ local sparse arrays and the size of each local sparse array is $\lceil n/p \rceil \times n$. The largest number of nonzero array elements among local sparse arrays is $\lceil n/p \rceil \times n \times s'$. In the encoding step, the encoding time is $(n^2 \times (1 + 3s)) \times T_{Operation}$. In the data distribution phase, the data distribution time $T_{Distribution}$ is $(p \times T_{Startup} +$

$(2n^2s+n)\times T_{Data}$). In the decoding step, the special buffer $B$ in each processor is decoded. The $C_{i,j}$ stored in the special buffer do not need to be converted to local sparse indices in each processor according Case 3.3.1. The decoding time is $(\lceil n/p \rceil \times n \times (2s' + (1/n))) + 1) \times T_{Operation}$. The data compression time $T_{Compression}$ is $(n^2 \times (1 + 3s) + \lceil n/p \rceil \times n \times (2s' + (1/n))) + 1) \times T_{Operation}$. Table 1 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the row partition method and the *CRS* method.

**Table 1: The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes.**

| *Method* | *Complexity* | *Cost* |
|---|---|---|
| *SFC* | $T_{Distribution}$ | $p \times T_{Startup} + n^2 \times T_{Data}$ |
| | $T_{Compression}$ | $(\lceil n/p \rceil \times n \times (1 + 3s')) \times T_{Operation}$ |
| *CFS* | $T_{Distribution}$ | $p \times T_{Startup} + (2n^2s+n+p) \times T_{Data} +$ $(2n^2 s + (\lceil n/p \rceil \times n \times (2s' + (1/n))) + n + p + 1) \times T_{Operation}$ |
| | $T_{Compression}$ | $(n^2 \times (1 + 3s)) \times T_{Operation}$ |
| *ED* | $T_{Distribution}$ | $p \times T_{Startup} + (2n^2s+n) \times T_{Data}$ |
| | $T_{Compression}$ | $(n^2 \times (1 + 3s) + \lceil n/p \rceil \times n \times (2s' + (1/n)) + 1) \times T_{Operation}$ |

### 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 condition $T_{Data} > (2s/1-2s)T_{Operation}$ is satisfied. In general, $T_{Data}$ is less than or equal to $T_{Operation}$ in a distributed memory multicomputer. If we assume that $T_{Data}$ is equal to $T_{Operation}$, $T_{Data} > (2s/1-2s)T_{Operation}$ when $s$ is less than 0.25. According to the Harewell-Boeing Sparse Matrix Collection [8, 9], it shows that over 80% sparse array applications in which the sparse ratio of a sparse array is less than 0.1. We have the following remark.

**Remark 2.** The data distribution time of the *CFS* scheme is less than that of the *SFC* scheme for most of sparse array applications.

For the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the row partition method and the *CRS* method, 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 using the row partition method and the *CRS* method, 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 $T_{Data} > (1+3s/1-2s)T_{Operation}$ and $T_{Data} > (1+5s/1-2s)T_{Operation}$ are satisfied, respectively. ( $T_{Data} > (3s/1-2s)T_{Operation}$ and $T_{Data} > (5s/1-2s)T_{Operation}$ in the column and the 2D mesh partition methods)

### 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 row partition method and the *CCS* method.

**Table 2: The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes.**

| *Method* | *Complexity* | *Cost* |
|---|---|---|
| *SFC* | $T_{Distribution}$ | $p \times T_{Startup} + n^2 \times T_{Data}$ |
| | $T_{Compression}$ | $(\lceil n/p \rceil \times n \times (1+3s')) \times T_{Operation}$ |
| *CFS* | $T_{Distribution}$ | $p \times T_{Startup} + (2n^2s+n+p) \times T_{Data} +$ $(2n^2 s + (\lceil n/p \rceil \times n \times (3s') + pn + p + n + 1) \times T_{Operation}$ |
| | $T_{Compression}$ | $(n^2 \times (1 + 3s)) \times T_{Operation}$ |
| *ED* | $T_{Distribution}$ | $p \times T_{Startup} + (2n^2s+pn) \times T_{Data}$ |
| | $T_{Compression}$ | $(n^2 \times (1 + 3s) + \lceil n/p \rceil \times n \times (3s') + n + 1) \times T_{Operation}$ |

The main difference between Table 1 and Table2 is that, for the *CFS* and the *ED* schemes, the values stored in *CO* and each $C_{i,j}$ stored in the special buffer need to be converted to local array indices in each processor according to Case 3.2.2 and Case 3.3.2, respectively. 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 row partition, the column partition, and the 2D mesh partition methods are 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 Row Partition Method

Table 3 shows the data distribution and the data compression time for the *SFC*, the *CFS*, and the *ED* schemes using the row partition method.

**Table 3: The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes.**

| *No. of Processors* | *Array Sizes* / *Methods-Costs* | | 200×200 | 400×400 | 800×800 | 1000×1000 | 2000×2000 |
|---|---|---|---|---|---|---|---|
| 4 | *SFC* | $T_{Distribution}$ | 5.648 | 19.009 | 68.798 | 94.542 | 383.718 |
| | | $T_{Compression}$ | 2.527 | 7.604 | 26.959 | 38.778 | 160.579 |
| | *CFS* | $T_{Distribution}$ | 4.119 | 10.591 | 31.377 | 39.265 | 134.291 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 1.716 | 6.132 | 18.781 | 27.618 | 103.443 |
| | | $T_{Compression}$ | 6.878 | 21.001 | 83.453 | 127.398 | 520.574 |
| 16 | *SFC* | $T_{Distribution}$ | 7.234 | 22.154 | 71.642 | 97.234 | 388.184 |
| | | $T_{Compression}$ | 0.887 | 2.380 | 8.406 | 12.647 | 40.814 |
| | *CFS* | $T_{Distribution}$ | 4.120 | 14.204 | 48.825 | 61.640 | 187.761 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 3.302 | 8.343 | 21.625 | 30.309 | 106.922 |
| | | $T_{Compression}$ | 4.886 | 19.575 | 92.187 | 146.024 | 530.092 |
| 32 | *SFC* | $T_{Distribution}$ | 8.676 | 25.083 | 74.066 | 100.102 | 392.763 |
| | | $T_{Compression}$ | 0.689 | 2.069 | 4.882 | 8.179 | 31.427 |
| | *CFS* | $T_{Distribution}$ | 6.542 | 14.908 | 54.463 | 71.368 | 197.496 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 4.704 | 11.272 | 24.049 | 33.177 | 111.235 |
| | | $T_{Compression}$ | 4.832 | 17.964 | 95.188 | 147.834 | 530.887 |

*Time*: *ms*

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.

From experimental tests, we can estimate that $T_{Data} \approx 1.2 \times T_{Operation}$. Therefore, for the *CFS* scheme, the condition $T_{Data} > (1/4)T_{Operation}$ shown in Table 1 is satisfied. 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.

For the overall performance, from Table 3, we have the following observations.

1. The *ED* scheme outperforms the *CFS* scheme.
2. The *SFC* outperforms the *CFS* and the *ED* schemes since the conditions $T_{Data} > (15/8)T_{Operation}$ and $T_{Data} > (13/8)T_{Operation}$ shown in Table 1 are not 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.

## 5.2 The Column Partition Method

Table 4 shows the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the column partition method. From Table 4, for the data distribution time and the data compression time, the experimental results match Remarks 1, 2, 3, and 4. For the overall performance of these schemes, we have the following observations.

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 $T_{Data} > (3/8)T_{Operation}$ are satisfied, respectively.

These results match Remarks 4 and 5.

**Table 4: The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes.**

| *No. of Processors* | *Array Sizes* / *Methods-Costs* | | 200×200 | 400×400 | 800×800 | 1000×1000 | 2000×2000 |
|---|---|---|---|---|---|---|---|
| 4 | *SFC* | $T_{Distribution}$ | 12.208 | 45.155 | 179.714 | 292.231 | 909.207 |
| | | $T_{Compression}$ | 1.914 | 6.536 | 24.003 | 38.606 | 147.746 |
| | *CFS* | $T_{Distribution}$ | 4.734 | 14.787 | 61.085 | 84.134 | 289.102 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 1.741 | 6.182 | 18.880 | 27.742 | 103.691 |
| | | $T_{Compression}$ | 6.763 | 24.848 | 97.887 | 152.643 | 597.112 |
| 16 | *SFC* | $T_{Distribution}$ | 14.727 | 47.457 | 188.987 | 301.999 | 925.376 |
| | | $T_{Compression}$ | 0.704 | 1.76 | 7.260 | 9.691 | 38.179 |
| | *CFS* | $T_{Distribution}$ | 6.983 | 17.173 | 77.401 | 109.220 | 334.324 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 3.427 | 8.593 | 22.724 | 32.433 | 110.170 |
| | | $T_{Compression}$ | 7.711 | 26.319 | 108.886 | 166.119 | 630.521 |
| 32 | *SFC* | $T_{Distribution}$ | 16.057 | 48.399 | 196.915 | 310.999 | 935.492 |
| | | $T_{Compression}$ | 0.561 | 1.305 | 5.188 | 6.212 | 22.273 |
| | *CFS* | $T_{Distribution}$ | 8.373 | 18.970 | 83.835 | 126.788 | 346.495 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 4.729 | 10.022 | 25.148 | 35.301 | 116.483 |
| | | $T_{Compression}$ | 8.099 | 27.005 | 115.503 | 176.134 | 644.641 |

*Time*: *ms*

## 5.3 The 2D Mesh Partition Method

Table 5 shows the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes using the 2D mesh partition method. For the data distribution time and the data compression time, the experimental results match Remarks 1, 2, and 3. For the overall performance, the *ED* scheme outperforms the *CFS* scheme that outperforms the *SFC* scheme. These results match Remarks 4 and 5.

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. For most of cases, the data distribution time of the *CFS* scheme is less than that of the *SFC* scheme.

**Table 5: The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes.**

| *No. of Processors* | *Methods-Costs* | *Array Sizes* | 120×120 | 240×240 | 480×480 | 960×960 | 1920×1920 |
|---|---|---|---|---|---|---|---|
| 2×2 | *SFC* | $T_{Distribution}$ | 11.191 | 46.565 | 162.632 | 250.151 | 902.477 |
| | | $T_{Compression}$ | 0.633 | 2.789 | 8.898 | 32.556 | 136.174 |
| | *CFS* | $T_{Distribution}$ | 3.498 | 8.192 | 32.737 | 54.128 | 200.717 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 1.659 | 4.701 | 16.718 | 25.695 | 100.251 |
| | | $T_{Compression}$ | 4.926 | 19.861 | 75.475 | 123.114 | 517.207 |
| 4×4 | *SFC* | $T_{Distribution}$ | 14.522 | 50.696 | 170.702 | 265.641 | 914.282 |
| | | $T_{Compression}$ | 0.339 | 0.998 | 2.750 | 9.792 | 36.127 |
| | *CFS* | $T_{Distribution}$ | 4.303 | 12.298 | 44.391 | 67.015 | 220.96 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 3.702 | 9.143 | 23.209 | 32.293 | 110.89 |
| | | $T_{Compression}$ | 5.096 | 20.367 | 74.619 | 133.49 | 532.396 |
| 6×6 | *SFC* | $T_{Distribution}$ | 17.785 | 60.028 | 183.293 | 285.791 | 938.527 |
| | | $T_{Compression}$ | 0.184 | 0.588 | 1.228 | 5.376 | 18.973 |
| | *CFS* | $T_{Distribution}$ | 6.155 | 15.295 | 53.006 | 86.23 | 245.821 |
| | | $T_{Compression}$ | 4.573 | 18.295 | 73.183 | 119.348 | 507.399 |
| | *ED* | $T_{Distribution}$ | 4.177 | 10.093 | 25.09 | 34.649 | 115.602 |
| | | $T_{Compression}$ | 6.249 | 25.414 | 82.027 | 150.997 | 570.591 |

*Time: ms*

**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

In this paper, we have proposed two data distribution schemes, *CFS* and *ED*, for the distribution of sparse arrays on distributed memory multicomputers. 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 the experimental tests, for most 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 to work on the following directions. (1) Analyze the performance of the *SFC*, the *CFS*, and the *ED* schemes for other partition and data compression methods. (2) Developing efficient data distribution schemes for multi-dimensional sparse arrays based on the *extended Karnaugh map representation* (*EKMR*) scheme [11-12]. We believe that these directions are of importance in parallel sparse array operations.

## Acknowledgments

The work in this paper was partially supported by National Science Council of the Republic of China under contract NSC90-2213-E-035-019.

## References

[1] Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, Brain T. Smith, and Jerrold L. Wagener. *Fortran* 90 *Handbook*. 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] Gerardo Bandera and Emilio L. Zapata, "Sparse Matrix Block-Cyclic Redistribution," *In Proceedings of IEEE International Parallel Processing Symposium*, 1999.

[4] 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.

[5] 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.

[6] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Parallel Sparse Supports for Array Intrinsic Functions of Fortran 90," *The Journal of Supercomputing*, 18(3):305-339, March 2001.

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.