J Supercomput (2007) 41: 63–87
DOI 10.1007/s11227-007-0104-x

Data distribution schemes of sparse arrays on distributed memory multicomputers

Chun-Yuan Lin · Yeh-Ching Chung

Published online: 10 March 2007

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, many methods proposed in the literature first perform the data partition phase, then the data distribution phase, followed by the data compression phase. We called a data distribution scheme with this order 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 the data partition phase, the row partition, the column partition, and the 2D mesh partition with/without load-balancing methods are 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. Both theoretical analysis and experimental tests were conducted. In the 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 tests, we implemented these three 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.

C.-Y. Lin (✉) · Y.-C. Chung
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, ROC
e-mail: cyulin@mx.nthu.edu.tw

Y.-C. Chung
e-mail: ychung@cs.nthu.edu.tw

Keywords 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 [10], finite-element methods [13], climate modeling [25], etc. To implement the data distribution scheme, many methods have been proposed in the literature [2, 7–9, 24–27, 30]. 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. For these methods in the literature, 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. 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 data compression methods in order to obtain better performance for sparse array operations [7, 15, 16, 18, 21, 23, 26, 30]. A data distribution scheme with this order is called the *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. 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 and the decoding steps are performed before and after the data distribution phase, respectively. In encoding step, we encode information of non-zero array elements into a special buffer for each local sparse array. In decoding step, a 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, many partition methods can be used for these three schemes. Different partition methods may lead to different performance of a data distribution scheme. In this paper, we use the row partition, the column partition, and the 2D mesh partition with/without load-balancing methods for these three schemes. The row partition, the column partition, and 2D mesh partition methods [20] whose are similar to (Block, *), (*, Block), and (Block, Block) data distribution schemes used in Fortran 90 [1, 14, 22]. The details of the load-balancing method for 2D mesh partition can be found in [2, 26]. For the row partition, the column partition, and the 2D mesh partition without load-balancing methods, each processor has the same size of local sparse array. However, each processor has different number of non-zero array elements. For the 2D mesh partition with load-balancing method, each processor has the same number of non-zero array elements. However, each processor has different size of local sparse array. In the data distribution phase, local sparse arrays, whether compressed or not, are sent to processors sequentially. In the compression

phase, many data compression methods, such as the *Compressed Row Storage* (*CRS*) [5, 28], *Compressed Column Storage* (*CCS*) [5, 28], *Jagged Diagonal format* (*JAD*) [5], etc., can be used for these three schemes. Different data compression methods are used in different sparse array applications. In this paper, the *CRS* and the *CCS* methods are used to compress sparse local arrays for the *SFC* and the *CFS* schemes while the encoding/decoding steps are used for the *ED* scheme.

Bases on the methods used in the three phases above, both theoretical analysis and experimental tests were conducted. In the 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 tests, 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. The reason is that we do not send entire local sparse arrays to processors in these two 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 Sect. 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 row partition, the column partition, and the 2D mesh partition with/without load-balancing methods. The experimental results of these three schemes will be given in Sect. 5.

2 Related work

Many methods have been proposed in the literature to implement the data distribution scheme [2, 7–9, 24–27, 30]. Zapata et al. [2, 26] have proposed two data distribution schemes, *Block Row Scatter* (*BRS*) and *Multiple Recursive Decomposition* (*MRD*), for sparse arrays. Based on the *BRS* and the *MRD* schemes, they solve other important problems based on sparse arrays [2–4, 24–26, 29]. The *BRS* scheme is based on the division of any computation domain into several blocks, all of the same spatial shape and size. The *MRD* scheme can be considered as a generalization of the *Binary Recursive Decomposition* [6], a well-known data distribution scheme. In the data partition phase, the *BRS* scheme uses block partition methods while the *MRD* scheme uses a 2D mesh partition with load-balancing method. In the data distribution phase, local sparse arrays are sent to processors. In the data compression phase, both schemes use the *CRS*/*CCS* methods to compress the local sparse array in each processor. For the *BRS* scheme, the data compression time is determined by a processor, which has largest number of non-zero array elements. For the *MRD* scheme, the data compression time is determined by a processor, which has largest size of local sparse array. The data compression time for the *BRS* and the *MRD* schemes will be large when non-zero array elements were concentrated in a portion of a global sparse array. The reason is that, for the *BRS* scheme, there exists at least one processor whose

local array is a dense array. 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.

Ziantz et al. [30] 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. Lee et al. [7–9] presented an efficient library for parallel sparse computations with Fortran 90 array intrinsic operations. Based on the *MRD* scheme, they provided a data distribution scheme for multi-dimensional sparse arrays [17]. Their scheme is similar to (*, ..., Block, Block) data distribution scheme used in Fortran 90.

3 The *SFC*, *CFS* and *ED* schemes

In the following, we describe the *SFC*, the *CFS*, and the *ED* schemes in detail. In the data partition phase, the row partition, the column partition, and the 2D mesh partition with/without load-balancing methods are used for these three schemes. In the data compression phase, the *CRS*/*CCS* methods are used to compress sparse local arrays for the *SFC* and the *CFS* schemes while the encoding/decoding steps are used for the *ED* scheme.

We assume that an 8×10 two-dimensional sparse array A with 16 non-zero array elements shown in Fig. 1 and four processors are given. For the 2D mesh partition with/without load-balancing methods, the four processors are treated as a 2×2 processor array.

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, the partition results of the row partition, the column partition, and the 2D mesh partition with/without load-balancing methods used on array A are shown in Fig. 2, respectively. In Fig. 2, we can see that array A is partitioned into four local sparse arrays by these four partition methods, respectively. In the data distribution phase, local sparse arrays are sent to processors sequentially. Figure 3 shows the local sparse arrays received by each processor. In the data compression phase, the local sparse array received by each processor

Fig. 1 An 8×10 sparse array A with 16 non-zero array elements

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

Sparse array A

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

(a) The row partition method

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

(b) The column partition method

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

(c) The 2D mesh partition without load-balancing method

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

(d) The 2D mesh partition with load-balancing method

Fig. 2 The partition results for array A by using these four partition methods

is compressed by the *CRS*/*CCS* methods. The *CRS* (*CCS*) method uses three one-dimensional arrays, *RO*, *CO*, and *VL*, to compress all of non-zero array elements along the rows (columns for *CCS*) of the sparse array.

Figure 4 shows the compressed results by using the *CRS* method. In Fig. 4, array *RO* stores information of non-zero array elements of each row. The number of non-zero array elements in the ith row can be obtained by subtracting the value of $RO[i]$ from $RO[i+1]$. Array *CO* stores the column indices of non-zero array elements of each row. Array *VL* stores the values of non-zero array elements of the sparse array. The base of these three arrays is 0.

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 and the data compression phase, the process is the same as that of the *SFC* scheme, respectively. However, the values stored in array *CO* are global array indices since the partitioned local sparse arrays are compressed before sent to processors. In the data distribution phase, arrays *RO*, *CO*, and *VL* for each local sparse array are packed and then sent to its corresponding processor. After received the corresponding packed buffer, each processor unpacks the buffer to the corresponding arrays *RO*, *CO*, and *VL*. Since the values stored in array *CO* are global array indices in the compression

$$P_0 \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \end{pmatrix} \quad P_2 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 10 \end{pmatrix}$$

$$P_1 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \end{pmatrix} \quad P_3 \begin{pmatrix} 0 & 11 & 12 & 0 & 13 & 0 & 0 & 0 \\ 14 & 0 & 0 & 15 & 0 & 0 & 16 & 0 \end{pmatrix}$$

(a) The row partition method

$$P_0 \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 3 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 11 \\ 14 & 0 \end{pmatrix} \quad P_1 \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 6 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 12 & 15 \end{pmatrix} \quad P_2 \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 5 \\ 0 & 0 \\ 7 & 0 \\ 0 & 0 \\ 9 & 0 \\ 13 & 0 \\ 0 & 0 \end{pmatrix} \quad P_3 \begin{pmatrix} 0 & 0 \\ 2 & 0 \\ 0 & 4 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 8 & 0 \\ 0 & 10 \\ 0 & 0 \\ 16 & 0 \end{pmatrix}$$

(b) The column partition method

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

$$P_{1,0} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 11 & 12 & 0 \\ 14 & 0 & 0 & 15 \end{pmatrix} \quad P_{1,1} \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 9 & 0 & 0 & 10 \\ 13 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 \end{pmatrix}$$

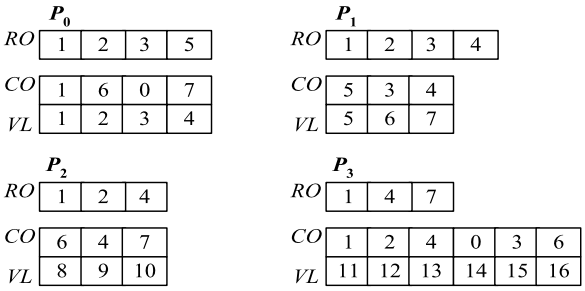
(c) The 2D mesh partition without load-balancing method

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

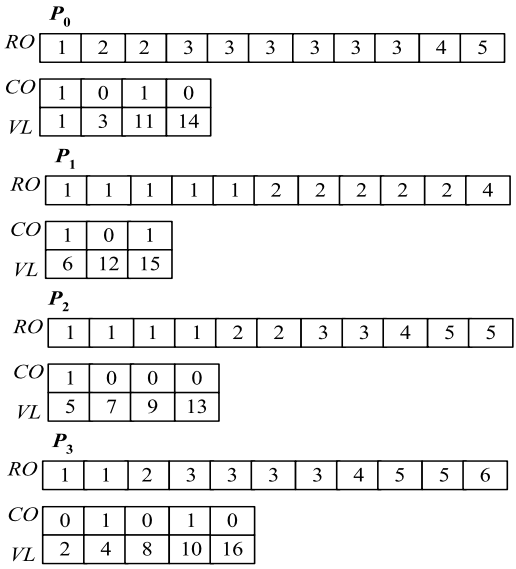
$$P_{1,0} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 11 & 12 & 0 \\ 14 & 0 & 0 & 15 \end{pmatrix} \quad P_{1,1} \begin{pmatrix} 9 & 0 & 0 & 10 \\ 13 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 \end{pmatrix}$$

(d) The 2D mesh partition with load-balancing method

Fig. 3 The corresponding local sparse arrays received by each processor



(a) The row partition method



(b) The column partition method

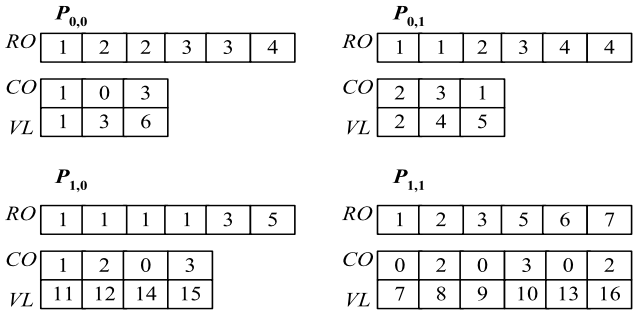
Fig. 4 The compressed results by using the *CRS* method for the received local sparse arrays

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

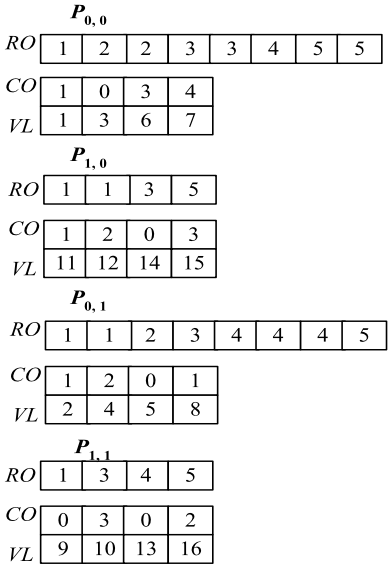
Case 3.2.1: When the row (column) partition and the *CRS* (*CCS* for column) methods are used in the data partition phase and the data compression phase, respectively, the values stored in array *CO* of the received buffer are desired local array indices. No conversion is needed.

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

Case 3.2.3: When the 2D mesh partition with/without load-balancing and the *CRS* (*CCS*) methods are used in the data partition phase and the data compression phase, respectively, each processor $P_{i,j}$ converts the values stored in array *CO* of the re-



(c) The 2D mesh partition without load-balancing method



(d) The 2D mesh partition with load-balancing method

Fig. 4 (Continued)

ceived buffer to the corresponding local array indices by subtracting M from each value stored in array CO of the received buffer, where M is the total number of columns (rows for CCS) in processors $P_{i,0}, P_{i,1}, \ldots, P_{i,j-1} (P_{0,j}, P_{1,j}, \ldots, P_{i-1,j}$ for CCS).

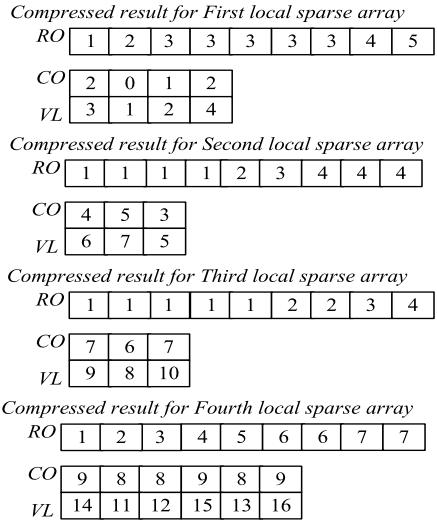
An example of the CFS scheme by applying the row partition method and the CCS method to array A is given in Fig. 5.

Figure 5a shows the partition result of the row partition method used on array A. Figure 5b shows the compressed results of the CCS method. Figure 5c only shows the data distribution phase for processor P_1. In Fig. 5c, arrays RO, CO, and VL for the first local sparse array are packed into a buffer and then sent to processor P_1. After receiving the buffer, processor P_1 unpacks the received buffer to the corresponding arrays RO, CO, and VL. According to Case 3.2.2 described above, processor P_1 converts

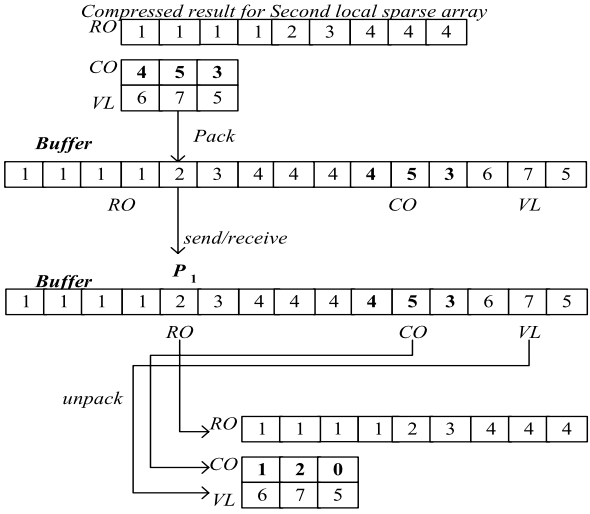
Fig. 5 An example of the *CFS* scheme

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

(a) The data partition phase



(b) The data compression phase



(c) The data distribution phase

the values stored in array *CO* of the received buffer to the corresponding local array indices by subtracting 3 from each value stored in array *CO* of the received buffer. For processors P_0, P_2, and P_3, the packing, send/receive, and unpacking procedures are similar to that of processor 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 *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 process is the same as that of the *SFC* scheme. 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 in the *CRS*/*CCS* formats. In Fig. 6, for the *CRS* (*CCS*) format, the R_i is used to store the number of non-zero 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 jth non-zero 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 sequentially.

In the decoding step, the special buffer B is decoded to get arrays *RO*, *CO*, and *VL* in each processor. To get array *RO*, in each processor, *RO*[0] is first initialized to 1. Then other values of array *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. To get array *CO*, in each processor, we move $C_{0,0}, C_{0,1}, \ldots, C_{0,j}, C_{1,0}, C_{1,1}, \ldots, C_{1,j}, \ldots, C_{i,0}, C_{i,1}, \ldots, C_{i,j}$ stored in the special buffer to array *CO*, where $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, m$, n is the number of rows of the local sparse array of a processor, and m is the number of non-zero array elements in row i. To get array *VL*, we move all $V_{i,j}$ to array *VL* in a similar manner as that of getting array *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.

R_0	$C_{0,0}$	$V_{0,0}$	…	$C_{0,j}$	$V_{0,j}$	…	R_i	$C_{i,0}$	$V_{i,0}$	…	$C_{i,j}$	$V_{i,j}$

i: *the row index* j: *the jth non-zero array element in row i*

R_i: *the number of non-zero array elements in row i*

$C_{i,j}$: *the column index of jth non-zero array elements in row i*

$V_{i,j}$: *the value of jth non-zero array elements in row i*

(a) for *CRS* format

R_0	$C_{0,0}$	$V_{0,0}$	…	$C_{0,j}$	$V_{0,j}$	…	R_i	$C_{i,0}$	$V_{i,0}$	…	$C_{i,j}$	$V_{i,j}$

i: *the column index* j: *the jth non-zero array element in column i*

R_i: *the number of non-zero array elements in column i*

$C_{i,j}$: *the row index of jth non-zero array elements in column i*

$V_{i,j}$: *the value of jth non-zero array elements in column i*

(b) for *CCS* format

Fig. 6 The formats of the special buffer B

Case 3.3.1: When the row (column) partition method and the *CRS* (*CCS* for column) format are used in the data partition phase and the encoding step, 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) format are used in the data partition phase and the encoding step, 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 rows (columns for column) in processors $P_0, P_1, \ldots, P_{i-1}$.

Case 3.3.3: When the 2D mesh partition with/without load-balancing methods and the *CRS* (*CCS*) format are used in the data partition phase and the encoding step, 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 processors $P_{i,0}, P_{i,1}, \ldots, P_{i,j-1}$($P_{0,j}, P_{1,j}, \ldots, P_{i-1,j}$ for *CCS*).

An example of the *ED* scheme by applying the row partition method and the *CCS* format to array A is given in Fig. 7. Figure 7a shows the partition result of the row partition method used on array A. Figure 7b shows the special buffers for local sparse arrays. Figure 7c shows the special buffers received by each processor.

Figure 7d only shows the decoding step for processor P_1. After receiving the special buffer, to get arrays *RO*, *RO*[0] is first set to 1. Then other values of array *RO* are computed according to the formula $RO[i+1] = RO[i] + R_i$, where $i = 0, 1, 2, 3, 4, 5, 6$, and 7. To get array *CO*, we move $C_{3,0}$, $C_{4,0}$, and $C_{5,0}$ stored in the special buffer to array *CO*. According to Case 3.3.2 described above, processor 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 array *VL*, we move $V_{3,0}$, $V_{4,0}$, and $V_{5,0}$ stored in the special buffer to array *VL*. For processors P_0, P_2, and P_3, the decoding step is similar to that of processor 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. 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.
- $T_{\text{Operation}}$ is the average operation time for an array element. In order to simplify the analysis, we use $T_{\text{Operation}}$ to present average operation cost for an array element, such as memory access, addition or subtraction operations, etc.
- $T_{\text{Distribution}}$ is the data distribution time for the data distribution phase. The data distribution time include the packing/unpacking time and send/receive time.
- $T_{\text{Compression}}$ 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.

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

(a) The data partition phase

The special buffer for First local sparse array

1	2	3	1	0	1	0	0	0	0	1	1	2	1	2	4

The special buffer for Second local sparse array

0	0	0	1	**4**	6	1	**5**	7	1	**3**	5	0	0
R_0	R_1	R_2	R_3	$C_{3,0}$	$V_{3,0}$	R_4	$C_{4,0}$	$V_{4,0}$	R_5	$C_{5,0}$	$V_{5,0}$	R_6	R_7		

The special buffer for Third local sparse array

0	0	0	0	1	7	9	0	1	6	8	1	7	10

The special buffer for Fourth local sparse array

1	9	14	1	8	11	1	8	12	1	9	15	1	8	13	0	1	9	16	0

(b) The encoding step

P_0

1	2	3	1	0	1	0	0	0	0	1	1	2	1	2	4

P_1

0	0	0	1	**4**	6	1	**5**	7	1	**3**	5	0	0
R_0	R_1	R_2	R_3	$C_{3,0}$	$V_{3,0}$	R_4	$C_{4,0}$	$V_{4,0}$	R_5	$C_{5,0}$	$V_{5,0}$	R_6	R_7		

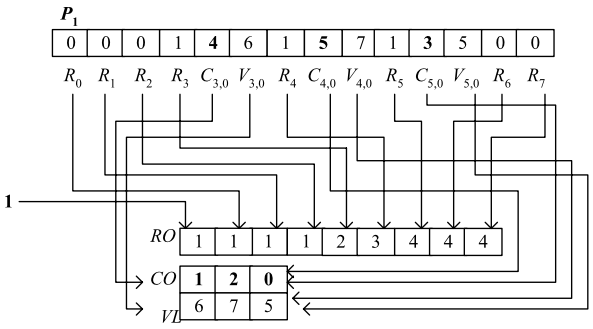
P_2

0	0	0	0	1	7	9	0	1	6	8	1	7	10

P_3

1	9	14	1	8	11	1	8	12	1	9	15	1	8	13	0	1	9	16	0

(c) The data distribution phase



(d) The decoding step

Fig. 7 An example of the *ED* scheme

- A is an $n \times n$ global sparse array.
- p is the number of processors. For the 2D mesh partition with/without load-balancing methods, p processors are treated as an $r \times q$ processor array.
- s is the sparse ratio of array A.
- $S = \{s_i | i = 0, 1, \ldots, p-1\}$ is the set of sparse ratios of local sparse arrays. The largest sparse ratio in S is denoted as s'.
- $\alpha = \{\alpha_i | i = 0, 1, \ldots, p-1\}$ is set of space ratios of local sparse arrays. 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'.

4.1 The row partition method

4.1.1 The CRS method

A The SFC scheme Assume that array A and p processors are given. The number of non-zero array elements in array A is sn^2. For the *SFC* scheme, the row partition method partitions array 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 non-zero array elements among these local sparse arrays is $\lceil n/p \rceil \times n \times s'$. In the data distribution phase, local sparse arrays are sent to processors sequentially. 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. $T_{\text{Distribution}}$ is $(p \times T_{\text{Startup}} + n^2 \times T_{\text{Data}})$ that is determined by the size of array A. In the data compression phase, local sparse arrays are compressed by the *CRS* method simultaneously. Therefore, $T_{\text{Compression}}$ is $(\lceil n/p \rceil \times n \times (1 + 3s')) \times T_{\text{Operation}}$ that is determined by a processor with the largest number of non-zero array elements. When non-zero array elements are concentrated in a portion of array A, the data compression time is large since s' is close to 1.

B The CFS scheme For the *CFS* scheme, the row partition method partitions array A into p local sparse arrays. In the data compression phase, local sparse arrays are compressed by the *CRS* method sequentially. This phase is similar to compress a global sparse array by the *CRS* method. Therefore, $T_{\text{Compression}}$ is $(n^2 \times (1 + 3s)) \times T_{\text{Operation}}$ that is determined by the size of array A. In the data distribution phase, the compressed results are first packed into buffers and then sent to the corresponding processors. After receiving the corresponding buffer, each processor unpacks the buffer to get the desired arrays *RO*, *CO*, and *VL*. The values stored in array *CO* do not need to be converted to local sparse indices in each processor according to Case 3.2.1. The packing time is $(2n^2 s + n + p) \times T_{\text{Operation}}$, the send/receive time is $p \times T_{\text{Startup}} + (2n^2 s + n + p) \times T_{\text{Data}}$, and the unpacking time is $(\lceil n/p \rceil \times n \times (2s' + (1/n)) + 1) \times T_{\text{Operation}}$. Therefore, $T_{\text{Distribution}}$ is $p \times T_{\text{Startup}} + (2n^2 s + n + p) \times T_{\text{Data}} + (2n^2 s + (\lceil n/p \rceil \times n \times (2s' + (1/n))) + n + p + 1) \times T_{\text{Operation}}$. The number of non-zero array elements in array A determines the packing time and the send/receive time. The unpacking time is determined by a processor that has the largest number of non-zero array elements. When non-zero array elements are concentrated in a portion of array A, the unpacking time is large. However, for the data distribution time, the above effect is slight since the sum of the packing time and the send/receive time is much larger than the unpacking time.

Table 1 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the row partition and the *CRS* methods

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

C The ED scheme For the *ED* scheme, the row partition method partitions array A into p local sparse arrays. In the encoding step, local sparse arrays are encoded into special buffers sequentially. The encoding time is $(n^2 \times (1 + 3s)) \times T_{\text{Operation}}$ that is determined by the size of array A. In the data distribution phase, these special buffers are sent to processors sequentially. $T_{\text{Distribution}}$ is $(p \times T_{\text{Startup}} + (2n^2 s + n) \times T_{\text{Data}})$ that is determined by the number of non-zero array elements of array A. In the decoding step, the special buffer B in each processor is decoded simultaneously. 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 $(\lceil n/p \rceil \times n \times (2s' + (1/n)) + 1) \times T_{\text{Operation}}$ that is determined by a processor with the largest number of non-zero array elements. The data compression time $T_{\text{Compression}}$ is $(n^2 \times (1 + 3s) + \lceil n/p \rceil \times n \times (2s' + (1/n)) + 1) \times T_{\text{Operation}}$. When non-zero array elements are concentrated in a portion of array A, the decoding time is large. However, for the data compression time, the above effect is slight since the encoding time is much larger than the decoding time.

Table 1 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the row partition and the *CRS* methods.

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. It is very important that, in [18], we have shown that the sparse ratio s of a k-dimensional sparse array must be less than $1/k$ if we want to use *CRS* and *CCS* to compress it. According to the Harewell-Boeing Sparse Matrix Collection [11, 12], it shows that over 80% sparse array applications in which the sparse ratio of a sparse array is less than 0.1. Therefore, the data distribution time of the *ED* scheme is less than that of the *SFC* scheme. We have the following remark.

Remark 1 $T_{\text{Distribution}}(ED) < T_{\text{Distribution}}(SFC)$ and $T_{\text{Distribution}}(CFS)$.

For the data distribution time of the *CFS* scheme, it is less than that of the *SFC* scheme if the condition $T_{\text{Data}} > (2s/1 - 2s)T_{\text{Operation}}$ is satisfied. In general, T_{Data} is less than or equal to $T_{\text{Operation}}$ in a distributed memory multicomputer. If we assume that T_{Data} is equal to $T_{\text{Operation}}$, $T_{\text{Data}} > (2s/1 - 2s)T_{\text{Operation}}$ when s is less than 0.25. We have the following remark.

Remark 2 $T_{\text{Distribution}}(CFS) < T_{\text{Distribution}}(SFC)$ for most of sparse array applications.

For the data compression time of these three schemes, we have the following remark.

Remark 3 $T_{\text{Compression}}(SFC) < T_{\text{Compression}}(CFS) < T_{\text{Compression}}(ED)$.

From Table 1, for the overall performance of these three 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 $T_{\text{Data}} > (1 + 3s/1 - 2s)T_{\text{Operation}}$ and $T_{\text{Data}} > (1 + 5s/1 - 2s)T_{\text{Operation}}$ 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 by using the row partition and the *CCS* methods. The main difference between Table 1 and Table 2 is that, for the *CFS* and the *ED* schemes, the values stored in array *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, we have similar observations as those of Remarks 1, 2, 3, 4, and 5.

4.2 The column partition method

4.2.1 The CRS method

Table 3 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition and the *CRS* methods. There

Table 2 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the row partition and the *CCS* methods

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

Table 3 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition and the *CRS* methods

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

Table 4 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition and the *CCS* methods

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

are two differences between the Table 1 and the Table 3. First, for two-dimensional sparse arrays by using the column partition method, array elements in a local sparse array are not continuous. For the *SFC* scheme, these local sparse arrays are sent to processors after packing into buffers. Second, for the *CFS* and the *ED* schemes, the values stored in array *CO* and each $C_{i,j}$ stored in the special buffer need to be converted to local sparse indices in each processor according Case 3.2.2 and Case 3.3.2, respectively.

From Table 3, for the data distribution time and the data compression time of these three schemes, we have similar observations as those of Remarks 1, 2, and 3. For the overall performance of the *CFS* and the *ED* schemes, we have similar observation as that of Remark 4. For the overall performance of these three schemes, we have following remark.

Remark 6 The *ED* and the *CFS* schemes outperform the *SFC* scheme if the conditions $T_{\text{Data}} > (3s/1 - 2s)T_{\text{Operation}}$ and $T_{\text{Data}} > (5s/1 - 2s)T_{\text{Operation}}$ are satisfied, respectively.

Table 5 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without load-balancing and the *CRS* methods

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

Table 6 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without load-balancing and the *CCS* methods

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

4.2.2 The CCS method

Table 4 lists the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition and the *CCS* methods. The main difference between Tables 3 and 4 is that, for the *CFS* and the *ED* schemes, the values stored in array *CO* and each $C_{i,j}$ stored in the special buffer do not need to be converted to local sparse indices in each processor according Case 3.2.1 and Case 3.3.1, respectively. From Table 4, for the data distribution time, the data compression time, and the overall performance, we have similar observations as those of Remarks 1, 2, 3, 4, and 6.

4.3 The 2D mesh partition without load-balancing method

Tables 5 and 6 list the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without load-balancing and the *CRS*/*CCS* methods, respectively. For the *CFS* and the *ED* schemes, the values stored in array *CO* and each $C_{i,j}$ stored in the special buffer need to be converted to

local sparse indices in each processor according Case 3.2.3 and Case 3.3.3, respectively. From Tables 5 and 6, for the data distribution time, the data compression time, and the overall performance, we have similar observations as those of Remarks 1, 2, 3, 4, and 6.

4.4 The 2D mesh partition with load-balancing method

Assume that the dimension of the largest local sparse array is $r' \times q' = \alpha' n^2$. Tables 7 and 8 list the data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition with load-balancing and the *CRS*/*CCS* methods, respectively. From Tables 7 and 8, we can see that, for the *SFC* scheme, the data distribution time is determined by the size of a global sparse array. The data compression time is determined by the processor that has the largest size of local sparse array. For the *CFS* scheme, the number of non-zero array elements of a global sparse array determines the data distribution time. The data compression time is determined by the size of a global sparse array. For the *ED* scheme, the number of non-zero array elements of a global sparse array determines the data distribution time. The data compression time in encoding step is determined by the size of a global sparse array. The number of non-zero array element of a local spare array determines the data compression time in decoding step. When non-zero array elements are concentrated in a portion of a global sparse array, the data compression

Table 7 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition with load-balancing and the *CRS* methods

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

Table 8 The data distribution time and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition with load-balancing and the *CCS* methods

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

time for the *SFC* scheme is large. From Tables 7 and 8, for the data distribution time, the data compression time, and the overall performance, we have similar observations as those of Remarks 1, 2, 3, 4, and 6.

5 Experimental results

In experimental tests, 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 with/without load-balancing methods all are implemented in these three schemes. In the data compression phase, the *CRS* method is implemented in these three schemes. All methods are written in C + MPI (*Message Passing Interface*) codes. In order to observe the general cases for the *SFC*, the *CFS*, and the *ED* schemes, in this paper, we perform all the experimental results using the random data for two-dimensional sparse arrays. 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 9 shows the data distribution and the data compression time for the *SFC*, the *CFS*, and the *ED* schemes by using the row partition method. From Table 9, for the data distribution time, we have the following observation.

1. $T_{\text{Distribution}}(ED) < T_{\text{Distribution}}(CFS) < T_{\text{Distribution}}(SFC)$.

From experimental tests, we can estimate that $T_{\text{Data}} \approx 1.2 \times T_{\text{Operation}}$. Therefore, for the *CFS* scheme, the condition $T_{\text{Data}} > (1/4)T_{\text{Operation}}$ shown in Table 1 is satisfied. These results match Remarks 1 and 2. For the data compression time, from Table 9, we have the following observation.

1. $T_{\text{Compression}}(SFC) < T_{\text{Compression}}(CFS) < T_{\text{Compression}}(ED)$.

This result matches Remark 3. For the overall performance, from Table 9, 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 $T_{\text{Data}} > (15/8)T_{\text{Operation}}$ and $T_{\text{Data}} > (13/8)T_{\text{Operation}}$ shown in Table 1 are not satisfied, respectively.

These results match Remarks 4 and 5. From Table 9, we can see that the experimental results match the theoretical analysis in Table 1.

5.2 The column partition method

Table 10 shows the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition method. From Table 10, 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.

Table 9 The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the row partition method on an IBM SP2 parallel machine

No. of processors	Methods-costs		Array sizes				
			200 × 200	400 × 400	800 × 800	1000 × 1000	2000 × 2000
2	*SFC*	$T_{\text{Distribution}}$	5.341	18.078	66.600	93.371	380.030
		$T_{\text{Compression}}$	**2.262**	**8.223**	**33.521**	**59.370**	**232.054**
	CFS	$T_{\text{Distribution}}$	3.173	9.666	34.107	40.532	131.744
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.609**	**4.601**	**16.520**	**25.447**	**99.755**
		$T_{\text{Compression}}$	6.051	21.830	84.572	129.711	520.936
4	*SFC*	$T_{\text{Distribution}}$	5.648	19.009	68.798	94.542	383.718
		$T_{\text{Compression}}$	**2.527**	**7.604**	**26.959**	**38.778**	**160.579**
	CFS	$T_{\text{Distribution}}$	4.119	10.591	31.377	39.265	134.291
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.716**	**6.132**	**18.781**	**27.618**	**103.443**
		$T_{\text{Compression}}$	6.878	21.001	83.453	127.398	520.574
8	*SFC*	$T_{\text{Distribution}}$	6.353	20.551	69.502	95.583	385.987
		$T_{\text{Compression}}$	**1.316**	**4.036**	**14.524**	**20.342**	**91.654**
	CFS	$T_{\text{Distribution}}$	3.654	12.313	35.742	41.197	149.302
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**2.421**	**6.774**	**19.485**	**28.659**	**104.725**
		$T_{\text{Compression}}$	5.285	20.353	81.886	123.018	513.534
16	*SFC*	$T_{\text{Distribution}}$	7.234	22.154	71.642	97.234	388.184
		$T_{\text{Compression}}$	**0.887**	**2.380**	**8.406**	**12.647**	**40.814**
	CFS	$T_{\text{Distribution}}$	4.120	14.204	48.825	61.640	187.761
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.302**	**8.343**	**21.625**	**30.309**	**106.922**
		$T_{\text{Compression}}$	4.886	19.575	92.187	146.024	530.092
32	*SFC*	$T_{\text{Distribution}}$	8.676	25.083	74.066	100.102	392.763
		$T_{\text{Compression}}$	**0.689**	**2.069**	**4.882**	**8.179**	**31.427**
	CFS	$T_{\text{Distribution}}$	6.542	14.908	54.463	71.368	197.496
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**4.704**	**11.272**	**24.049**	**33.177**	**111.235**
		$T_{\text{Compression}}$	4.832	17.964	95.188	147.834	530.887

Time: ms.

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

These results match Remarks 4 and 6. From Table 10, we can see that the experimental results match the theoretical analysis in Table 3.

Table 10 The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the column partition method

No. of processors	Methods-costs		Array sizes				
			200 × 200	400 × 400	800 × 800	1000 × 1000	2000 × 2000
2	*SFC*	$T_{\text{Distribution}}$	11.824	44.206	176.368	289.902	905.458
		$T_{\text{Compression}}$	**3.029**	**12.647**	**46.951**	**79.926**	**294.525**
	CFS	$T_{\text{Distribution}}$	4.525	13.911	53.488	76.046	267.728
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.633**	**4.650**	**16.614**	**25.571**	**100.002**
		$T_{\text{Compression}}$	6.683	23.69	93.745	146.884	579.769
4	*SFC*	$T_{\text{Distribution}}$	12.208	45.155	179.714	292.231	909.207
		$T_{\text{Compression}}$	**1.914**	**6.536**	**24.003**	**38.606**	**147.746**
	CFS	$T_{\text{Distribution}}$	4.734	14.787	61.085	84.134	289.102
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.741**	**6.182**	**18.880**	**27.742**	**103.691**
		$T_{\text{Compression}}$	6.763	24.848	97.887	152.643	597.112
8	*SFC*	$T_{\text{Distribution}}$	13.265	46.216	183.403	296.529	917.343
		$T_{\text{Compression}}$	**1.032**	**3.164**	**12.928**	**19.496**	**75.094**
	CFS	$T_{\text{Distribution}}$	5.937	15.512	72.387	97.710	311.543
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**2.546**	**6.924**	**19.884**	**29.783**	**106.973**
		$T_{\text{Compression}}$	7.603	25.502	101.076	161.341	619.507
16	*SFC*	$T_{\text{Distribution}}$	14.727	47.457	188.987	301.999	925.376
		$T_{\text{Compression}}$	**0.704**	**1.76**	**7.260**	**9.691**	**38.179**
	CFS	$T_{\text{Distribution}}$	6.983	17.173	77.401	109.220	334.324
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.427**	**8.593**	**22.724**	**32.433**	**110.170**
		$T_{\text{Compression}}$	7.711	26.319	108.886	166.119	630.521
32	*SFC*	$T_{\text{Distribution}}$	16.057	48.399	196.915	310.999	935.492
		$T_{\text{Compression}}$	**0.561**	**1.305**	**5.188**	**6.212**	**22.273**
	CFS	$T_{\text{Distribution}}$	8.373	18.970	83.835	126.788	346.495
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**4.729**	**10.022**	**25.148**	**35.301**	**116.483**
		$T_{\text{Compression}}$	8.099	27.005	115.503	176.134	644.641

Time: ms.

5.3 The 2D mesh partition without/with load-balancing method

Tables 11 and 12 show the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without/with load-balancing method. For the data distribution time, the data compression time, and the overall performance, the experimental results match Remarks 1, 2, 3, 4, and 6, respectively. From Tables 11 and 12, we can see that the experimental results match the theoretical analysis in Tables 5 and 6, respectively.

Table 11 The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without load-balancing method

No. of processors	Methods-costs		Array sizes				
			120×120	240×240	480×480	960×960	1920×1920
2×2	*SFC*	$T_{\text{Distribution}}$	11.191	46.565	162.632	250.151	902.477
		$T_{\text{Compression}}$	**0.633**	**2.789**	**8.898**	**32.556**	**136.174**
	CFS	$T_{\text{Distribution}}$	3.498	8.192	32.737	54.128	200.717
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.659**	**4.701**	**16.718**	**25.695**	**100.251**
		$T_{\text{Compression}}$	4.926	19.861	75.475	123.114	517.207
3×3	*SFC*	$T_{\text{Distribution}}$	12.219	48.372	165.604	253.992	907.066
		$T_{\text{Compression}}$	**0.430**	**1.392**	**4.208**	**15.480**	**61.559**
	CFS	$T_{\text{Distribution}}$	3.695	9.776	37.823	58.692	210.389
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**2.621**	**7.174**	**20.277**	**29.651**	**106.709**
		$T_{\text{Compression}}$	5.157	20.102	76.651	129.383	527.959
4×4	*SFC*	$T_{\text{Distribution}}$	14.522	50.696	170.702	265.641	914.282
		$T_{\text{Compression}}$	**0.339**	**0.998**	**2.750**	**9.792**	**36.127**
	CFS	$T_{\text{Distribution}}$	4.303	12.298	44.391	67.015	220.96
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.702**	**9.143**	**23.209**	**32.293**	**110.89**
		$T_{\text{Compression}}$	5.096	20.367	74.619	133.49	532.396
5×5	*SFC*	$T_{\text{Distribution}}$	16.656	55.321	175.266	279.886	925.445
		$T_{\text{Compression}}$	**0.267**	**0.737**	**1.738**	**7.577**	**24.984**
	CFS	$T_{\text{Distribution}}$	5.279	13.298	49.088	76.932	231.227
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.927**	**9.593**	**24.121**	**33.409**	**113.122**
		$T_{\text{Compression}}$	5.667	23.859	80.426	141.447	561.653
6×6	*SFC*	$T_{\text{Distribution}}$	17.785	60.028	183.293	285.791	938.527
		$T_{\text{Compression}}$	**0.184**	**0.588**	**1.228**	**5.376**	**18.973**
	CFS	$T_{\text{Distribution}}$	6.155	15.295	53.006	86.23	245.821
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**4.177**	**10.093**	**25.09**	**34.649**	**115.602**
		$T_{\text{Compression}}$	6.249	25.414	82.027	150.997	570.591

Time: ms.

6 Conclusions and future work

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 sparse array applications, the data distribution time of the *CFS* scheme is less than that of the *SFC* scheme.

Table 12 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	Methods-costs		Array sizes				
			120 × 120	240 × 240	480 × 480	960 × 960	1920 × 1920
2 × 2	*SFC*	$T_{\text{Distribution}}$	11.905	48.543	167.326	259.691	905.85
		$T_{\text{Compression}}$	**0.665**	**2.565**	**9.515**	**34.905**	**136.747**
	CFS	$T_{\text{Distribution}}$	3.538	9.82	35.644	55.252	204.104
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**1.659**	**4.701**	**16.718**	**25.695**	**100.251**
		$T_{\text{Compression}}$	4.893	19.967	75.023	124.171	515.103
3 × 3	*SFC*	$T_{\text{Distribution}}$	13.219	50.372	169.201	263.424	913.466
		$T_{\text{Compression}}$	**0.421**	**1.492**	**4.224**	**18.452**	**71.559**
	CFS	$T_{\text{Distribution}}$	4.195	10.245	37.422	58.724	210.189
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**2.621**	**7.174**	**20.277**	**29.854**	**106.109**
		$T_{\text{Compression}}$	5.245	20.542	76.542	127.254	521.524
4 × 4	*SFC*	$T_{\text{Distribution}}$	14.522	52.696	173.702	266.785	918.182
		$T_{\text{Compression}}$	**0.339**	**0.998**	**3.355**	**10.742**	**38.227**
	CFS	$T_{\text{Distribution}}$	5.803	12.298	42.391	63.154	220.962
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.702**	**9.143**	**23.209**	**32.293**	**110.895**
		$T_{\text{Compression}}$	6.296	21.367	78.619	131.496	528.426
5 × 5	*SFC*	$T_{\text{Distribution}}$	16.656	55.321	177.226	273.247	925.524
		$T_{\text{Compression}}$	**0.267**	**0.737**	**1.938**	**6.724**	**24.254**
	CFS	$T_{\text{Distribution}}$	6.279	14.098	47.088	70.722	229.254
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**3.927**	**9.593**	**24.100**	**33.733**	**113.724**
		$T_{\text{Compression}}$	6.667	22.459	80.426	140.741	540.141
6 × 6	*SFC*	$T_{\text{Distribution}}$	18.285	60.028	183.293	285.124	938.247
		$T_{\text{Compression}}$	**0.184**	**0.588**	**1.228**	**4.425**	**19.827**
	CFS	$T_{\text{Distribution}}$	7.155	15.895	52.006	79.752	240.841
		$T_{\text{Compression}}$	4.573	18.295	73.183	119.348	507.399
	ED	$T_{\text{Distribution}}$	**4.177**	**10.093**	**25.090**	**34.649**	**115.602**
		$T_{\text{Compression}}$	7.425	23.852	85.722	148.424	551.541

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.

Acknowledgements The work in this paper was partially supported by National Science Council of the Republic of China under contract NSC91-2213-E-007-104. Parts of this work have previously appeared in the Proceedings of International Conference on Parallel Processing Workshops on Compile/Runtime Techniques for Parallel Computing [19].

References

1. Adams JC, Brainerd WS, Martin JT, Smith BT, Wagener JL (1992) FORTRAN 90 handbooks. Intertext Publications/McGraw-Hill Inc
2. Asenjo R, Romero LF, Ujaldon M, Zapata EL (1994) Sparse block and cyclic data distributions for matrix computations. In: Proc. High Performance Comput. Technology, Methods and Applications, 1994, pp 6–8
3. Asenjo R, Plata O, Tourino J, Doallo R, Zapata EL (1998) HPF-2 support for dynamic sparse computations. In: Proc. Int. Workshop Languages and Compilers for Parallel Comput., 1998, pp 230–246
4. Bandera G, Zapata EL (1996) Extending CRAFT data-distributions for sparse matrices. In: Proc. European Cray MPP Workshop, 1996
5. Barrett R, Berry M, Chan TF, Demmel J, Dongarra J, Eijkhout V, Pozo R, Romine C, Van der Vorst H (1994) Templates for the solution of linear systems: building blocks for the iterative methods, 2nd edn. SIAM
6. Berger MJ, Bokhari SH (1987) A partitioning strategy for nonuniform problems on multiprocessors. IEEE Trans Comput 36:570–580
7. Chang R-G, Chung T-R, Lee JK (2001) Parallel sparse supports for array intrinsic functions of Fortran 90. J Supercomput 18:305–339
8. Chang R-G, Chung T-R, Lee JK (2004) Support and optimization for parallel sparse programs with array intrinsics of Fortran 90. Parallel Comput 30:527–550
9. Chang R-G, Chung T-R, Lee JK (1997) Towards automatic support of parallel sparse computation in java with continuous compilation. Concurrency: practice and experiences. Parallel Comput 9:1101–1111
10. Cullum JK, Willoughby RA (1985) Lanczos algorithms for large symmetric eignenvalue computations. Birkhauser, Boston
11. Duff I, Grimes R, Lewis J (1989) Sparse matrix test problems. ACM Trans Math Soft 15:1–14
12. Duff I, Grimes R, Lewis J (1992) User's giude for the harwell-boeing sparse matrix collection (Release I), Technical Report RAL 92-086, Rutherford Appleton Laboratory
13. Golub GH, Van Loan CF (1989) Matrix computations, 2nd edn. The John Hopkins University Press, Baltimore
14. High Performance Fortran Forum (1997) High performance fortran language specification, 2nd edn. Rice University
15. Kebler CW, Smith CH (1999) The SPARAMAT approach to automatic comprehension of sparse matrix computations. In: Proc. International Workshop Program Comprehension, 1999, pp 200–207
16. Kotlyar V, Pingali K, Stodghill P (1997) Compiling parallel code for sparse matrix applications. In: Proc. Supercomputing Conference, 1997, pp 20–38
17. Lin C-Y, Liu J-S, Chung Y-C (2002) Efficient representation scheme for multi-dimensional array operations. IEEE Trans Comput 51:327–345
18. Lin C-Y, Chung Y-C, Liu J-S (2003) Efficient data compression methods for multi-dimensional sparse array operations based on the EKMR scheme. IEEE Trans Comput 52:1640–1646
19. Lin C-Y, Chung Y-C, Liu J-S (2002) Data distribution schemes of sparse arrays on distributed memory multicomputers. In: Proc. ICPP Workshops Compile/Runtime Techniques Parallel Comput., 2002, pp 551–558
20. Lin C-Y, Chung Y-C, Liu J-S (2003) Efficient data parallel algorithms for multi-dimensional array operations based on the EKMR scheme for distributed memory multicomputers. IEEE Trans Parallel Distrib Syst 14:625–639
21. Mateev N, Pingali K, Stodghill P, Kotlyar V (2000) Next-generation generic programming and its application to sparse matrix computations. In: Proc. International Conference on Supercomput, 2000, pp 88–99
22. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1996) Numerical recipes in Fortran 90: the art of parallel scientific computing. Cambridge University Press
23. Sulatycke PD, Ghose K (1998) Caching efficient multithreaded fast multiplication of sparse matrices. In: Proc. Merged Int. Parallel Process. Symposium and Symposium Parallel Distributed Process., 1998, pp 117-124
24. Ujaldon M, Zapata EL, Chapman BM, Zima HP (1995) New data-parallel language features for sparse matrix computations. In: Proc. IEEE Int. Parallel Process. Sympos., 1995, pp 742–749
25. Ujaldon M, Zapata EL, Sharma SD, Saltz J (1996) Parallelization techniques for sparse matrix applications. J Parallel Distrib Comput 38:256–266

26. Ujaldon M, Zapata EL, Chapman BM, Zima HP (1997) Vienna-fortran/hpf extensions for sparse and irregular problems and their compilation. IEEE Trans Parallel Distrib Syst 8:1068–1083
27. Vastenhouw B, Bisseling RH (2005) A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Review 47:67–95
28. White JB, Sadayappan P (1997) On improving the performance of sparse matrix-vector multiplication. In: Proc. Int. Confer. High-Performance Comput., 1997, pp 711–725
29. Zapata EL, Plata O, Asenjo R, Trabado GP (1999) Data-parallel support for numerical irregular problems. J Parallel Comput 25:1911–1944
30. Ziantz LH, Ozturan CC, Szymanski BK (1994) Run-time optimization of sparse matrix-vector multiplication on SIMD machines. In: Proc. Int. Conference Parallel Architectures and Languages, 1994, pp 313–322

Chun-Yuan Lin was born in 1977. He received a B.S. degree in Information Engineering and Computer Science from Feng Chia University in 1999, and the M.S. and Ph.D. degrees in Department of Information Engineering and Computer Science from Feng Chia University in 2000 and 2003, respectively. He joined the Institute of Molecular and Cellular Biology and the Department of Computer Science at National Tsing Hua University as a post-doctoral fellow in 2003 and 2006, respectively. His research interests are in the areas of parallel and distributed computing, parallel algorithms, algorithm analysis, information retrieve, proteomics, and bioinformatics. He is a member of the IEEE computer society and ACM.

Yeh-Ching Chung received a B.S. degree in Information Engineering from Chung Yuan Christian University in 1983, and the M.S. and Ph.D. degrees in Computer and Information Science from Syracuse University in 1988 and 1992, respectively. He joined the Department of Information Engineering at Feng Chia University as an associate professor in 1992 and became a full professor in 1999. From 1998 to 2001, he was the chairman of the department. In 2002, he joined the Department of Computer Science at National Tsing Hua University as a full professor. His research interests include parallel and distributed processing, pervasive computing, cluster computing, grid computing, embedded software, and system software for SOC design. He is a member of the IEEE computer society and ACM.