The CFS and ED Data Distribution Schemes for 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: 886-4-24517250 x3765
Fax: 886-4-24516101
Email: {cylin, ychung, liuj}@iecs.fcu.edu.tw

Abstract

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

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

1. Introduction

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

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

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

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

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

2. Related Work

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

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

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

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

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

3.1 The *SFC* Scheme

The *SFC* is an intuitive data distribution scheme. In the *SFC* scheme, the data partition phase is performed first, then the data distribution phase, followed by the data compression phase. In the data partition phase, a global sparse array is partitioned into local sparse arrays by some partition methods. In this paper, the 2D mesh partition with load-balancing method is used to partition a global sparse array. The 2D mesh partition with load-balancing method tries to partition a global sparse array into some local sparse arrays such that each local sparse array has the same number of nonzero array elements. The details can be found
in [2, 42]. Assume that an 8x10 sparse array A with 16 nonzero array elements (Figure 1) and four processors are given. The partition result for the sparse array A by using the 2D mesh partition with load-balancing method is shown in Figure 2. For the 2D mesh partition with load-balancing method, the four processors are treated as a 2x2 processor array.

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

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

Figure 1: An 8x10 sparse array A with 16 nonzero array elements.

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

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

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

$$\begin{bmatrix}
P_{i,0} & P_{i,1} \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 2 & 3 & 4 & 5 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
1 & 0 & 1 & 2 & 0 & 1 & 0 & 1 & 2 & 0 & 1 & 1 & 2 & 0 \\
1 & 1 & 2 & 3 & 4 & 5 & 1 & 1 & 2 & 3 & 4 & 5 & 1 & 1 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
\end{bmatrix}$$

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

$$\begin{bmatrix}
P_{i,0} & P_{i,1} \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 2 & 3 & 4 & 5 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\
1 & 0 & 1 & 2 & 0 & 1 & 0 & 1 & 2 & 0 & 1 & 1 & 2 & 0 \\
1 & 1 & 2 & 3 & 4 & 5 & 1 & 1 & 2 & 3 & 4 & 5 & 1 & 1 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
\end{bmatrix}$$

3.2 The CFS Scheme

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

Case 3.2.1: When the 2D mesh partition with load-balancing method and the CRS (CCS) method is used in the data partition phase and the data compression phase, respectively, each processor P_{ij} 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}, \ldots, P_{i,j-1}, (P_{i,j}, P_{i,j+1}, \ldots, P_{i,4} for CCS).
An example of the CFS scheme is given in Figure 5 in which the 2D mesh partition with load-balancing method is used in the data partition phase and the CCS method is used in the data compression phase.

Figure 5(a) shows the partition result for the sparse array A (Figure 1) by using the 2D mesh partition with load-balancing method. Figure 5(b) shows the compressed results by using the CCS method for local sparse arrays shown in Figure 5(a). In Figure 5(b), the values stored in CO are global indices of global sparse array A, not local indices of a local sparse array. Figure 5(c) only shows the data distribution phase for P1,0. In Figure 5(c), RO, CO, and VL for the first local sparse array are packed into a buffer and sent to P1,0. After receiving the buffer, P1,0 unpacks the received buffer to the corresponding RO, CO, and VL. According to Case 3.2.1 described above, P1,0 converts the values stored in CO of the received buffer to the corresponding local array indices by subtracting 7 from each value stored in CO of the received buffer. For P0,0, P0,1, and P1,1, the packing, send/receive, and unpacking procedures are similar to that of P1,0.

3.3 The ED Scheme

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

In the data partition phase, the 2D mesh partition with load-balancing method mentioned in the SFC scheme is used to partition a global sparse array. In the encoding step, each local sparse array is encoded into a special buffer B. Figure 6 shows the formats of the special buffer B for the CRS/CCS methods. In Figure 6, for the CRS (CCS) method, the R(i) is used to store the number of nonzero array elements in a row (column for CCS) i. The C(i,j) and V(i,j) are used to store the column (row for CCS) index and the value of the jth 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.

An encoding method (a) for CRS method

\[
\begin{align*}
R_0, C_{0,0}, V_{0,0} & \cdots C_{0,m}, V_{0,m} \\
R_1, C_{1,0}, V_{1,0} & \cdots C_{1,m}, V_{1,m} \\
& \cdots \\
R_n, C_{n,0}, V_{n,0} & \cdots C_{n,m}, V_{n,m}
\end{align*}
\]

The encoding method (b) for CCS method

\[
\begin{align*}
R_{i,0}, C_{i,0}, V_{i,0} & \cdots C_{i,0}, V_{i,0} \\
R_{i,1}, C_{i,1}, V_{i,1} & \cdots C_{i,1}, V_{i,1} \\
& \cdots \\
R_{i,n}, C_{i,n}, V_{i,n} & \cdots C_{i,m}, V_{i,m}
\end{align*}
\]

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

\[
RO[i+1] = RO[i] + R_i, \quad \text{where } i = 0, 1, \ldots, n
\]

and n is the number of rows in a local sparse array. To get CO, in each processor, we move C(i,0), C(i,1), ..., C(i,n-1), C(i,n), C(i+1,0), ..., C(i+m,0), C(i+m,1), ..., C(i+m,n) stored in the special buffer to CO, where \( i = 0, 1, \ldots, n, j = 0, 1, \ldots, m \) 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)
4. Theoretical Analysis

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

- $T_{\text{Startup}}$ is the startup time for a communication channel.
- $T_{\text{Data}}$ is the transmission time for sending an array element through a communication channel.
- $T_{\text{Operation}}$ is the operation time for an array element. In order to simplify the analysis, we use $T_{\text{Operation}}$ to present any 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 includes 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.
- $A$ is an $n \times n$ global sparse array.
- $p$ is the number of processors.
- $s$ is the sparse ratio of $A$.

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

4.1 The 2D Mesh Partition with Load-Balancing Method

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

4.1.1 The CRS method

A. The SFC scheme

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

\[
\begin{array}{ccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 \\
3 & 0 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 5 & 0 \\
0 & 0 & 0 & 6 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 8 \\
0 & 0 & 0 & 0 & 9 & 0 & 0 \\
0 & 11 & 12 & 0 & 13 & 0 & 0 \\
14 & 0 & 0 & 15 & 0 & 0 & 16 \\
\end{array}
\]

(b) The data distribution phase

\[
\begin{array}{cccccc}
P_{\text{ax}} & 1 & 2 & 3 & 1 & 0 \\
P_{\text{ay}} & 1 & 3 & 5 & 2 & 1 & 2 \\
P_{\text{bz}} & 1 & 9 & 14 & 1 & 15 \\
P_{\text{by}} & 2 & 7 & 9 & 8 & 13 & 0 \\
\end{array}
\]

(c) The special buffer for first local sparse array

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 1 & 0 & 1 & 0 & 1 & 4 & 6 & 1 & 5 & 7 \\
\end{array}
\]

(d) The special buffer for second local sparse array

\[
\begin{array}{ccccccccccc}
1 & 3 & 5 & 2 & 1 & 2 & 6 & 3 & 1 & 2 & 4 \\
\end{array}
\]

The special buffer for third local sparse array

\[
\begin{array}{ccccccccccc}
1 & 3 & 5 & 2 & 1 & 2 & 6 & 3 & 1 & 4 & 15 \\
\end{array}
\]

The special buffer for fourth local sparse array

\[
\begin{array}{ccccccccccc}
2 & 7 & 9 & 8 & 13 & 0 & 1 & 9 & 16 & 1 & 7 & 10 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
P_{\text{ax}} & 1 & 2 & 3 & 1 & 0 & 1 & 0 & 1 & 4 & 6 & 1 & 5 & 7 \\
P_{\text{ay}} & 1 & 3 & 5 & 2 & 1 & 2 & 6 & 3 & 1 & 2 & 4 \\
P_{\text{bz}} & 1 & 9 & 14 & 1 & 15 \\
P_{\text{by}} & 2 & 7 & 9 & 8 & 13 & 0 \\
\end{array}
\]

\[P_{\text{ax}} = \left[ \begin{array}{cccccccccccc}
1 & 2 & 3 & 1 & 0 & 1 & 0 & 1 & 4 & 6 & 1 & 5 & 7 \\
\end{array} \right]
\]

\[P_{\text{ay}} = \left[ \begin{array}{cccccccccccc}
1 & 3 & 5 & 2 & 1 & 2 & 6 & 3 & 1 & 2 & 4 \\
\end{array} \right]
\]

\[P_{\text{bz}} = \left[ \begin{array}{cccccccccccc}
1 & 9 & 14 & 1 \\
\end{array} \right]
\]

\[P_{\text{by}} = \left[ \begin{array}{cccccccccccc}
2 & 7 & 9 & 8 & 13 & 0 \\
\end{array} \right]
\]

Figure 7: An example of the ED scheme.

B. The CFS scheme

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

C. The ED scheme

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

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

D. Discussions

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

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

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFC</td>
<td>$T_{\text{Distribution}}$</td>
<td>$r x q x T_{\text{Startup}} + n x T_{\text{Data}} + n^2 x T_{\text{Operation}}$</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{Compression}}$</td>
<td>$(n^2 x (\alpha + \frac{3}{r x q}) x T_{\text{Operation}})$</td>
</tr>
<tr>
<td>CFS</td>
<td>$T_{\text{Distribution}}$</td>
<td>$r x q x T_{\text{Startup}} + (2n^2 s + q n + r q) x T_{\text{Data}} + (2n^2 s + n^2 x \frac{3}{r x q}) x s + q r + q n + r q + 1) x T_{\text{Operation}}$</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{Compression}}$</td>
<td>$(n^2 x (1 + 3 s)) x T_{\text{Operation}}$</td>
</tr>
<tr>
<td>ED</td>
<td>$T_{\text{Distribution}}$</td>
<td>$r x q x T_{\text{Startup}} + (2n^2 s + q n + r q) x T_{\text{Data}}$</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{Compression}}$</td>
<td>$(n^2 x (1 + 3 s)) x T_{\text{Operation}}$</td>
</tr>
</tbody>
</table>

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

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

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

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

**Remark 5.** The ED and the CFS schemes outperform the SFC scheme if the conditions

$$T_{\text{Data}} > \frac{3 s - \alpha}{1 - 2 s} T_{\text{Operation}}$$

and

$$T_{\text{Data}} > \frac{s - \alpha}{1 - 2 s} T_{\text{Operation}}$$

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

#### 5. Experimental Results

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

#### 5.1 The 2D Mesh Partition with Load-Balancing Method

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

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

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

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

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

This result matches Remark 3. From experimental tests, we can estimate that $T_{\text{Data}} \approx 1.2 x T_{\text{Operation}}$. For the overall performance, from Table 3, we have the following observations.
Table 3: The data distribution and the data compression time of the SFC, the CFS, and the ED schemes using the 2D mesh partition with load-balancing method.

<table>
<thead>
<tr>
<th>No. of Processors</th>
<th>Array Sizes</th>
<th>120x120</th>
<th>240x240</th>
<th>480x480</th>
<th>960x960</th>
<th>1920x1920</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SFC T_Dist</td>
<td>11.905</td>
<td>48.543</td>
<td>167.326</td>
<td>259.691</td>
<td>905.85</td>
</tr>
<tr>
<td></td>
<td>T_Compression</td>
<td>0.665</td>
<td>2.565</td>
<td>9.515</td>
<td>34.905</td>
<td>136.747</td>
</tr>
<tr>
<td></td>
<td>CFS T_Dist</td>
<td>3.538</td>
<td>9.82</td>
<td>35.644</td>
<td>55.252</td>
<td>204.104</td>
</tr>
<tr>
<td></td>
<td>T_Compression</td>
<td>4.573</td>
<td>18.295</td>
<td>73.183</td>
<td>119.348</td>
<td>507.399</td>
</tr>
<tr>
<td></td>
<td>ED T_Dist</td>
<td>1.659</td>
<td>4.701</td>
<td>16.718</td>
<td>25.695</td>
<td>100.251</td>
</tr>
<tr>
<td></td>
<td>T_Compression</td>
<td>4.893</td>
<td>19.967</td>
<td>75.023</td>
<td>124.171</td>
<td>515.103</td>
</tr>
</tbody>
</table>

1. The ED scheme outperforms the CFS scheme.
2. The CFS and the ED schemes outperform the SFC scheme since the conditions $T_{Data} > \frac{5}{8}T_{Operation}$ and $T_{Data} > \frac{3}{4}T_{Operation}$ shown in Table 1 are satisfied, respectively.

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

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

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

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

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

6. Conclusions and Future Work

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

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

References


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


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


