The Journal of Supercomputing, 34, 291–313, 2005

Efficient Data Distribution Schemes for *EKMR*-Based Sparse Arrays on Distributed Memory Multicomputers*

CHUN-YUAN LIN† cyulin@mx.nthu.edu.tw
Institute of Molecular and Cellular Biology, National Tsing Hua University, Hsinchu, Taiwan 300

YEH-CHING CHUNG ychung@cs.nthu.edu.tw
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300

JEN-SHIUH LIU jsliu@fcu.edu.tw
Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan 407

Abstract. Multi-dimensional sparse array operations can be used in the atmosphere and ocean sciences, the image processing, and etc., and have been an extensively investigated problem. Therefore, it becomes an important issue to propose efficient data distribution schemes for multi-dimensional sparse arrays. In our previous work, we have proposed two data distribution schemes *Compress Followed Send* (*CFS*) and *Encoding-Decoding* (*ED*) for sparse arrays based on the *traditional matrix representation* (*TMR*) scheme. We have proposed another scheme, called *extended Karnaugh map representation* (*EKMR*), to represent sparse arrays. The *EKMR* scheme can obtain better performance than the *TMR* scheme for some sparse array operations. Hence, in this paper, we want to propose efficient data distribution schemes for *EKMR*-based sparse arrays. We extend the *CFS* and the *ED* schemes for *TMR*-based sparse arrays to *EKMR*-based sparse arrays first. Then, we compare the performance of these two schemes with that of the *Send Followed Compress* (*SFC*), which is an intuitive data distribution scheme for sparse arrays. Finally, we compare these three schemes for *EKMR*-based sparse arrays with those of *TMR*-based sparse arrays, respectively. Both the theoretical analysis and the experimental tests were conducted. From the theoretical analysis and the experimental results, we can see that the *ED* scheme is superior to the *CFS* scheme that is superior to the *SFC* scheme for most of testing *EKMR*-based sparse arrays; the performance of these three schemes for *EKMR*-based sparse arrays is better than that of *TMR*-based sparse arrays for all of testing cases, respectively.

Keywords: Karnaugh map, data distribution schemes, data compression methods, partition methods, sparse ratio

1. Introduction

Array operations are useful in a large number of important scientific codes, such as the molecular dynamics, [8], the finite-element methods [14], the climate modeling [28], the atmosphere and ocean sciences [9], and etc. To do parallel array operations is an efficient and widely accepted method; however, the utilization of the available computational power in distributed memory multicomputers involves a tremendous programming effort on a part of users. Many data parallel programming languages, such as High Performance Fortran (HPF)

*Parts of this work have previously published by the seventh International Symposium on Parallel Architectures, Algorithms, and Networks.
†Corresponding author.

[15], Fortran D [12], and Vienna Fortran [29], have been proposed to help programmers to write efficient data parallel programs by providing compiler directives or libraries to specify *array distribution*. However, it is a challenging problem to provide efficient data distributions for irregular problems [27].

Multi-dimensional array (dense or sparse) operations can be used in the atmosphere and ocean sciences, the image processing [24], and etc. They have been an extensively investigated problem [5, 6, 9, 16, 19, 21, 22, 29]. Therefore, it becomes an important issue to propose efficient data distribution schemes for multi-dimensional sparse arrays. In the literature, some methods have been proposed to implement data distributions for sparse arrays [2, 6, 28, 29, 31]. These methods all belonged to an intuitive scheme, called *Send Followed Compress* (*SFC*), for sparse arrays based on the *traditional matrix representation* (*TMR*) scheme [19]. The *SFC* scheme is composed of three phases and performed in the following order, the data partition phase, the data distribution phase, and the data compression phase. In the *data partition* phase, a global sparse array in a host processor is partitioned into local sparse arrays and then these local sparse arrays are distributed to processors in the *data distribution* phase. In the *data compression* phase, each local sparse array in a processor is compressed by using a data compression method in order to obtain better performance for sparse array operations [10, 17, 18, 25, 31].

In our previous work [20], we have proposed two data distribution schemes *Compress Followed Send* (*CFS*) and *Encoding-Decoding* (*ED*) for *TMR*-based sparse arrays. The *CFS* scheme is anther possible intuitive method and similar to the *SFC* scheme except that the data compression phase is performed before the data distribution phase. The *ED* scheme 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 [20], we have shown that the *SFC* scheme was less efficient than the *CFS* and the *ED* schemes for *TMR*-based sparse arrays. In order to represent multi-dimensional arrays, we have proposed another representation scheme, called *extended Karnaugh map representation* (*EKMR*) [19, 21]. We have proposed the corresponding data compression methods, called *EKMR-Compressed Row/Column Storage* (*ECRS/ECCS*) [22], for *EKMR*-based sparse arrays. We have shown that the performance of some sparse array operations, such as *All*, *Maxval*, *Pack*, and *Merge* Fortran 90 array intrinsic functions [1], for *EKMR*-based sparse arrays is better than that of *TMR*-based sparse arrays.

Hence, in this paper, we want to propose efficient data distribution schemes for *EKMR*-based sparse arrays. We extend the *CFS* and the *ED* schemes for *TMR*-based sparse arrays to *EKMR*-based sparse arrays first. Then, we compare the performance of these two schemes with that of the *SFC* scheme for *EKMR*-based sparse arrays. Finally, we compare the performance of these three schemes for *EKMR*-based sparse arrays with that of *TMR*-based sparse arrays, respectively. In order to evaluate these three schemes, in the data partition phase, the row partition, the column partition, and the 2D mesh partition with/without load-balancing methods are used. For three- or higher dimensional arrays, these three partition methods are similar to (*, ..., Block, *), (*, ..., *, Block), and (*, ..., Block, Block), respectively [21]. The details of the load-balancing method can be found in [29]. In the data distribution phase, local sparse arrays are sent to processors in sequence in order to simplify the comparisons. In the data compression phase, the *CRS*/*CCS* [3, 11] methods for *TMR*-based sparse arrays and the *ECRS*/*ECCS* methods for *EKMR*-based sparse arrays are used for these three schemes.

Both the theoretical analysis and the experimental tests were conducted. In the theoretical analysis, we analyze these three 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 these three schemes are all based on the same partition method. In the experimental tests, we implement these three schemes for *TMR*- and *EKMR*-based sparse arrays on an IBM SP2 parallel machine. From the theoretical analysis and the experimental results, for most of testing *EKMR*- based sparse arrays, we can see that the *ED* scheme outperforms the *CFS* scheme that outperforms the *SFC* scheme. We also can see that the performance of these three schemes for *EKMR*-based sparse arrays is better than that of *TMR*-based sparse arrays for all of testing cases, respectively.

This paper is organized as follows. In Section 2, a brief survey of related work will be presented. We will briefly describe the *EKMR* scheme and the *ECRS*/*ECCS* methods in Section 3. Section 4 will describe the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays in detail. Section 5 will analyze the time required of these three schemes for *TMR*- and *EKMR*-based sparse arrays. The experimental results will be given in Section 6.

2. Related work

Some methods have been proposed to implement data distributions for sparse arrays in the literature. Zapata et al. [2, 29] have proposed two data distribution schemes, called *Block Row Scatter* (*BRS*) and *Multiple Recursive Decomposition* (*MRD*). 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* [4], a well-known data distribution scheme. Based on the recursive decomposition concept, Vastenhouw and Bisseling [30] also proposed anther data distribution method that is similar to the *MRD* scheme to distribute data in the two-dimensional sparse matrix-vector multiplication. For the *BRS* and the *MRD* schemes, in the data partition phase, they used the Block and the Cyclic(k) partition methods, respectively. In the data compressing phase, for these two schemes, they used the *CRS*/*CCS* methods. Based on these two schemes, they solved some important problems of sparse arrays [2, 13, 28, 29].

Ziantz et al. [32] proposed a run-time optimization technique that was applied to sparse arrays for array distributions and off-processor data fetching to reduce the communication and the computation time. In their technique, they used the Block partition method with a bin-packing algorithm to distribute a global sparse array to processors. Then, local sparse arrays in processors are compressed by using the *CRS*/*CCS* methods. Lee et al. [5, 6] presented an efficient library for sparse array computations with Fortran 90 array intrinsic functions. Their approach has the potential to speed up sparse array computations on sequential and distributed memory environments. Since Fortran 90 provides a rich set of array intrinsic functions for multi- dimensional array operations, they provide a data compression method that is similar to the *CRS*/*CCS* methods and a data distribution scheme that is similar to the *MRD* scheme.

These proposed data distribution methods above all belonged to the *SFC* scheme for *TMR*- based sparse arrays with the two or higher dimensions.

3. Preliminary concepts

In this section, we briefly describe the *EKMR* scheme and the *ECRS*/*ECCS* methods. The details of them can be found in [19–22]. We use the *TMR*(n) and the *EKMR*(n) for *TMR*- and *EKMR*-based arrays with the dimension n, respectively. Here, the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays are all based on the row-major data layout [7]. However, with some indexing changes, they are also suitable for the column-major data layout [7].

3.1. The EKMR scheme

The *EKMR* scheme is used to represent a dense or sparse multi-dimensional array by a set of two-dimensional arrays. The idea of the *EKMR* scheme is based on the Karnaugh map. Hence, the *EKMR*(n) has the same representation as the *TMR*(n), where $n = 1$ and 2. Let $A[k][i][j]$ denotes an array based on the *TMR*(3) with a size of $3 \times 4 \times 5$. In the *EKMR*(3), the index variable i' is the same as the index variable i, whereas the index variable j' is a combination of index variables j and k. The corresponding array A' [4][15] based on the *EKMR*(3) is shown in Figure 1. Let $A[l][k][i][j]$ denotes an array based on the *TMR*(4) with a size of $2 \times 3 \times 4 \times 5$. In the *EKMR*(4), the index variable i' is a combination of index variables l and i; the index variablej' is a combination of index variables j and k. Figure 2 illustrates a corresponding array A' [8][15] based on the *EKMR*(4). Based on the *EKMR*(4), we can generalize our results to multi-dimensional arrays with the dimension n, where $n > 4$. Figure 3 shows the corresponding *EKMR*(6) array, which is represented by six *EKMR*(4) arrays, of array $A >$ [3][2][2][3][4][5] based on the *TMR*(6). A transformation scheme, called *matrix transformation method* (*MTM*), for conversion between the *TMR* and the *EKMR* schemes has been proposed in [20].

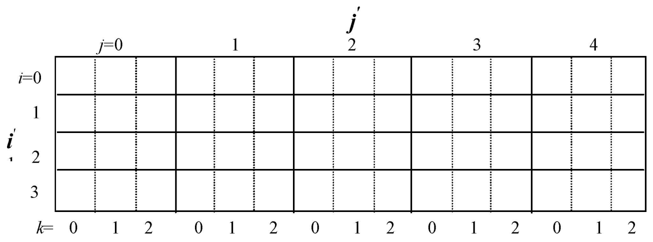


Figure 1. A 4 × 15 *EKMR*(3) array.

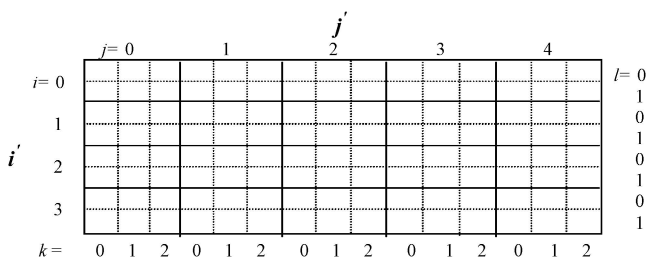


Figure 2. An 8 × 15 *EKMR*(4) array.

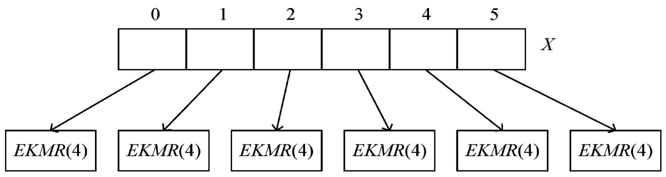


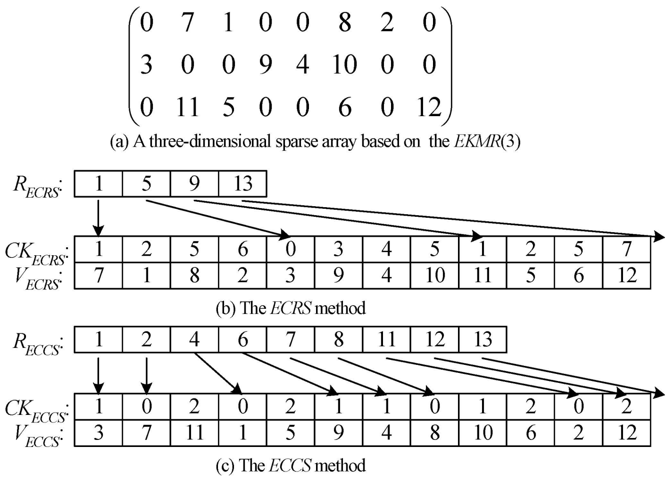
Figure 3. An example of the *EKMR*(6) array.

3.2. *The ECRS/ECCS methods*

The *ECRS*/*ECCS* methods are used to compress an *EKMR*-based sparse array by a set of three one-dimensional arrays R, *CK*, and V. Given an *EKMR*(3) sparse array, the *ECRS* (*ECCS*) method compresses all of non-zero array elements along the rows (columns for *ECCS*). For the *ECRS* (*ECCS*) method, in the array$R$, $R[0]$ is initialized to 1. Then, $R[i+1] = R[i] + R_i$, where R_i is the number of non-zero array elements in the row i (column for *ECCS*). Hence, the number of non-zero array elements in row i (column for *ECCS*) can be calculated by subtracting the value of $R[i]$ from $R[i+1]$. The array R can be used to point out the start position in arrays *CK* and V for each row (column for *ECCS*). Arrays, *CK* and V, store the column (row for *ECCS*) indices and the values for all of non-zero array elements, respectively. An example of the *ECRS*/*ECCS* methods for an *EKMR*(3) sparse array is given in Figure 4. For an *EKMR*(4) sparse array, the *ECRS*/*ECCS* methods are similar to those of an *EKMR*(3) sparse array. Given an *EKMR*(k) sparse array, where $k > 4$, in the *ECRS*/*ECCS* methods, each *EKMR* (4) sparse array is compressed by using arrays R, *CK*, and V first. Then, a pointer array is used to link arrays R, *CK*, and V of each *EKMR* (4) sparse array. An example is shown in Figure 5.

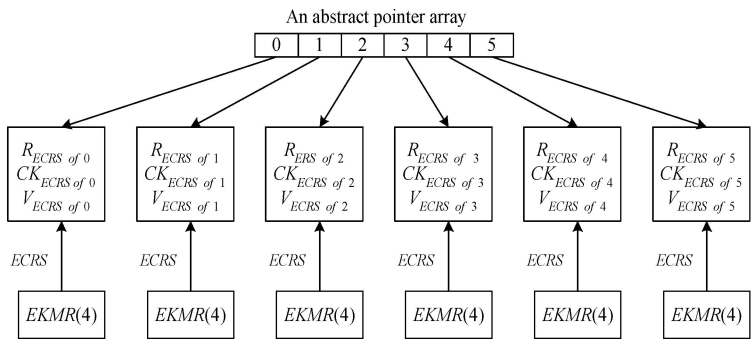
4. The *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays

In the *SFC*, the *CFS*, and the *ED* schemes, a global sparse array will be partitioned into local sparse arrays by using partition methods first. Different partition methods may lead to different performance for these three schemes. Therefore, how to select an appropriate partition method is an important issue. It is an interesting topic and our future work. In



$\begin{pmatrix} 0 & 7 & 1 & 0 & 0 & 8 & 2 & 0 \\ 3 & 0 & 0 & 9 & 4 & 10 & 0 & 0 \\ 0 & 11 & 5 & 0 & 0 & 6 & 0 & 12 \end{pmatrix}$
(a) A three-dimensional sparse array based on the *EKMR*(3)
R_{ECRS}: 1 5 9 13
CK_{ECRS}: 1 2 5 6 0 3 4 5 1 2 5 7
V_{ECRS}: 7 1 8 2 3 9 4 10 11 5 6 12
(b) The *ECRS* method
R_{ECCS}: 1 2 4 6 7 8 11 12 13
CK_{ECCS}: 1 0 2 0 2 1 1 0 1 2 0 2
V_{ECCS}: 3 7 11 1 5 9 4 8 10 6 2 12
(c) The *ECCS* method

Figure 4. The *ECRS/ECCS* methods for an *EKMR* (3) sparse array.



An abstract pointer array
0 1 2 3 4 5
$R_{ECRS\ of\ 0}$ $CK_{ECRS\ of\ 0}$ $V_{ECRS\ of\ 0}$
$R_{ECRS\ of\ 1}$ $CK_{ECRS\ of\ 1}$ $V_{ECRS\ of\ 1}$
$R_{ERS\ of\ 2}$ $CK_{ECRS\ of\ 2}$ $V_{ECRS\ of\ 2}$
$R_{ECRS\ of\ 3}$ $CK_{ECRS\ of\ 3}$ $V_{ECRS\ of\ 3}$
$R_{ECRS\ of\ 4}$ $CK_{ECRS\ of\ 4}$ $V_{ECRS\ of\ 4}$
$R_{ECRS\ of\ 5}$ $CK_{ECRS\ of\ 5}$ $V_{ECRS\ of\ 5}$
ECRS
EKMR(4)

Figure 5. The *ECRS* method for an *EKMR* (6) sparse array.

this section, we use the 2D mesh partition without load-balancing method [21] as a sample to describe these three schemes. For an *EKMR*-based sparse array, the 2D mesh partition without load-balancing method is similar to (*, . . . , Block, Block). The procedures of the *SFC*, the *CFS*, and the *ED* schemes based on this partition method are similar to those based on the row partition, the column partition, and the 2D mesh partition with load-balancing methods. Assume that a sparse array A' based on the *EKMR*(3) shown in Figure 6 is stored in a host processor. Our goal is to distribute the sparse array A' to a 2 × 2 processor array (represented by processors $P_{0,0}$, $P_{0,1}$, $P_{1,0}$, $P_{1,1}$).

4.1. The SFC scheme

In the data partition phase, the sparse array A' is partitioned into four local sparse arrays in a host processor first. Then, all of local sparse arrays in a host processor are sent to

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

Figure 6. A 4 × 16 sparse array A' based on the *EKMR*(3).

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

(a) The data partition phase

$P_{0,0}$
$$\begin{pmatrix} 1 & 5 & 9 & 13 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$P_{0,1}$
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 14 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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

$P_{1,1}$
$$\begin{pmatrix} 0 & 0 & 11 & 15 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 12 & 16 \end{pmatrix}$$

(b) The data distribution phase

$P_{0,0}$

R	1	2	3	4	5	5	5	5	5

CK	0	0	0	0
V	1	5	9	13

$P_{0,1}$

R	1	1	1	2	3	3	3	3	3

CK	1	1
V	10	14

$P_{1,0}$

R	1	2	2	2	2	4	6	6	6

CK	1	0	1	0	1
V	3	2	4	6	7

$P_{1,1}$

R	1	1	1	2	3	3	4	5	6

CK	0	0	1	1	1
V	11	15	8	12	16

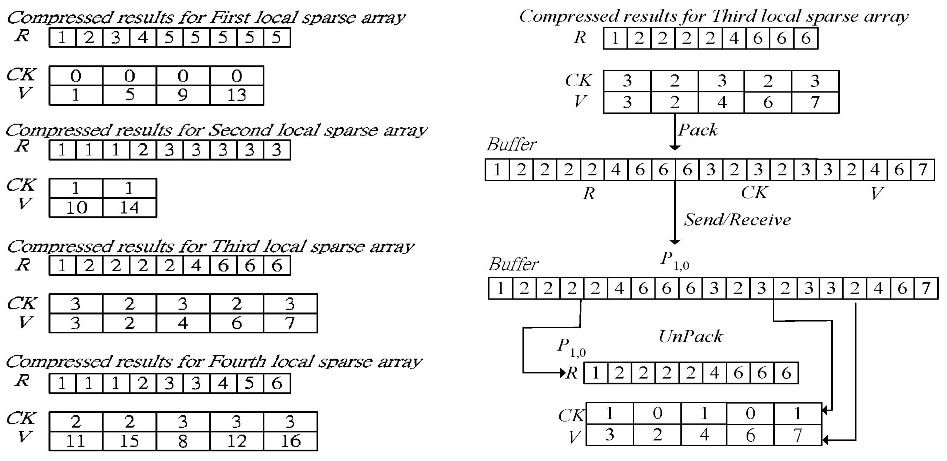
(c) The data compression phase

Figure 7. An example of the *SFC* scheme for sparse array A'.

corresponding processors in the data distribution phase, respectively. Finally, each local sparse array in a processor is compressed by using the *ECRS*/*ECCS* methods in the data compression phase. Figure 7 shows an example of the *SFC* scheme for sparse array A'. In Figure 7(a), the sparse array A' is partitioned into four local sparse arrays; in Figure 7(b), each processor receives the corresponding local sparse array; in Figure 7(c), each local sparse array in a corresponding processor is compressed into arrays R, *CK*, and V by using the *ECCS* method.

4.2. *The CFS scheme*

The data partition phase of the *CFS* scheme is the same as that of the *SFC* scheme. Then, each local sparse array in a host processor is compressed by using the *ECRS*/*ECCS* methods in the data compression phase. In the data distribution phase, arrays R, *CK*, and V of each local sparse array are packed into a buffer and then sent to the corresponding processor. Many communication methods [15, 26] have been proposed and can be used to send buffers to processors. In order to simplify the comparisons, the buffers are sent to processors in sequence. After receiving the corresponding buffer, each processor unpacks it to get arrays R, *CK*, and V. Since the values stored in the array *CK* are global array indices, they need to be converted to local array indices when each processor unpacks the received buffer. In the *ECRS* (*ECCS*) method, each processor P_{ij} revises each value stored in the array *CK* to the



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

Figure 8. An example of the *CFS* scheme for sparse array A'.

corresponding local array index by subtracting c_1, where c_1 is the total number of columns (rows for *ECCS*) 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 *ECCS*).

Figure 8 shows an example of the *CFS* scheme for sparse array A'. The partition results of the *CFS* scheme are the same as those of the *SFC* scheme. In Figure 8(a), each local sparse array is compressed into arrays R, *CK*, and V by using the *ECCS* method; the data distribution phase of the processor $P_{1,0}$ as a sample is shown in Figure 8(b). In Figure 8(b), the processor $P_{1,0}$ revises each value stored in the array *CK* to the local array index by subtracting 2. For processors $P_{0,0}$, $P_{0,1}$, and $P_{1,1}$, the data distribution phase is similar to that of the processor $P_{1,0}$, respectively.

4.3. *The ED scheme*

The data partition phase of the *ED* scheme 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 9 shows the formats of special buffer B for sparse array A'. In Figure 9, for the *ECRS* (*ECCS*) format, the R_i' is used to store the number of non-zero array elements in row (column for *ECCS*) i. The C_{ij}' and the V_{ij}' are used to store the column (row for *ECCS*) index and the value for the jth non-zero array element in row (column for *ECCS*) i, respectively. The C_{ij}' and the V_{ij}' are alternately stored in the special buffer B and each C_{ij}' is a global array index. In the data distribution phase, these special buffers B are sent to processors in sequence. In the decoding step, each buffer B in a corresponding processor is decoded to get arrays R, *CK*, and *VL*. To get the array R, in each processor, $R[0]$ is first initialized to 1. Then, $R[i+1] = R[i] + R_i$. To get arrays *CK* and V, all of the $C_{i,j}'$ and the $V_{i,j}'$ are moved to arrays*CK* and V, respectively. Since each $C_{i,j}'$ is a global array index, in the *ECRS* (*ECCS*) format, each processor $P_{i,j}$ converts each $C_{i,j}'$ to a local array index by subtracting c_2 from each $C_{i,j}'$,

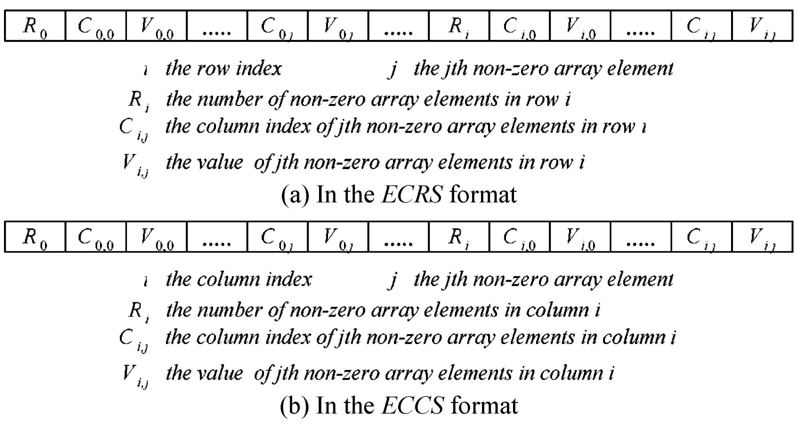

R0 | C0,0 | V0,0 | ..... | C0,j | V0,j | ..... | Ri | Ci,0 | Vi,0 | ..... | Ci,j | Vi,j
i the row index
j the jth non-zero array element
Ri the number of non-zero array elements in row i
Ci,j the column index of jth non-zero array elements in row i
Vi,j the value of jth non-zero array elements in row i
(a) In the ECRS format
R0 | C0,0 | V0,0 | ..... | C0,j | V0,j | ..... | Ri | Ci,0 | Vi,0 | ..... | Ci,j | Vi,j
i the column index
j the jth non-zero array element
Ri the number of non-zero array elements in column i
Ci,j the column index of jth non-zero array elements in column i
Vi,j the value of jth non-zero array elements in column i
(b) In the ECCS format

Figure 9. The formats of special buffer B for sparse array A'.

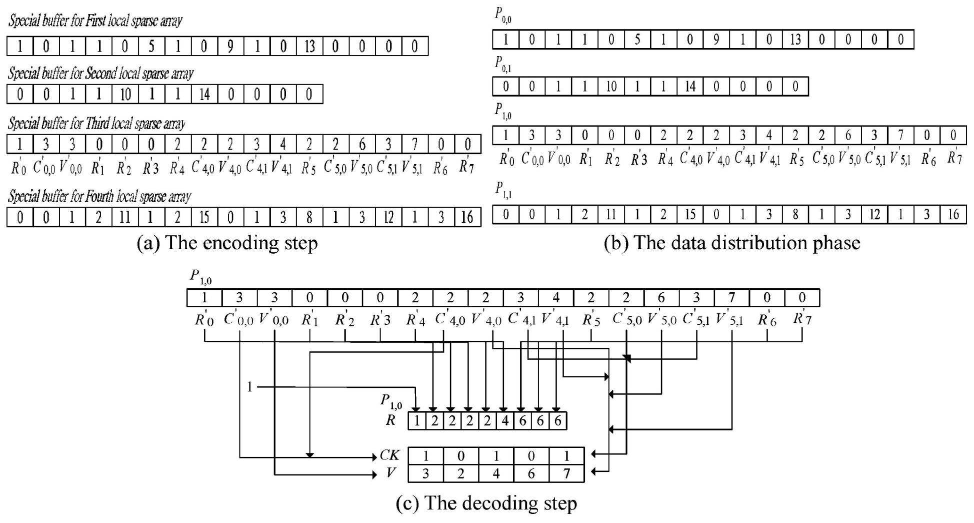

Special buffer for First local sparse array
1 0 1 1 0 5 1 0 9 1 0 13 0 0 0 0
Special buffer for Second local sparse array
0 0 1 1 10 1 1 14 0 0 0 0
Special buffer for Third local sparse array
1 3 3 0 0 0 2 2 2 3 4 2 2 6 3 7 0 0
R'0 C'0,0 V'0,0 R'1 R'2 R'3 R'4 C'4,0 V'4,0 C'4,1 V'4,1 R'5 C'5,0 V'5,0 C'5,1 V'5,1 R'6 R'7
Special buffer for Fourth local sparse array
0 0 1 2 11 1 2 15 0 1 3 8 1 3 12 1 3 16
(a) The encoding step
P0,0
1 0 1 1 0 5 1 0 9 1 0 13 0 0 0 0
P0,1
0 0 1 1 10 1 1 14 0 0 0 0
P1,0
1 3 3 0 0 0 2 2 2 3 4 2 2 6 3 7 0 0
R'0 C'0,0 V'0,0 R'1 R'2 R'3 R'4 C'4,0 V'4,0 C'4,1 V'4,1 R'5 C'5,0 V'5,0 C'5,1 V'5,1 R'6 R'7
P1,1
0 0 1 2 11 1 2 15 0 1 3 8 1 3 12 1 3 16
(b) The data distribution phase
P1,0
1 3 3 0 0 0 2 2 2 3 4 2 2 6 3 7 0 0
R'0 C'0,0 V'0,0 R'1 R'2 R'3 R'4 C'4,0 V'4,0 C'4,1 V'4,1 R'5 C'5,0 V'5,0 C'5,1 V'5,1 R'6 R'7
1
P1,0
R 1 2 2 2 2 4 6 6 6
CK 1 0 1 0 1
V 3 2 4 6 7
(c) The decoding step

Figure 10. An example of the *ED* scheme for sparse array A'.

where c_2 is the total number of columns (rows for *ECCS*) in processors $P_{i,0}, \ldots, P_{i,j-1}$ ($P_{0,j}, \ldots, P_{i-1,j}$ for *ECCS*).

Figure 10 shows an example of the *ED* scheme for sparse array A'. The partition results of the *ED* scheme are the same as those of the *SFC* scheme. Figure 10(a) shows the special buffers B for local sparse arrays in the *ECCS* format. Figure 10(b) shows the special buffer B received by each processor. Figure 10(c) only shows the decoding step for the processor$P_{1,0}$ as a sample. The processor$P_{1,0}$ subtracts 2 from $C'_{0,0}$, $C'_{4,0}$, $C'_{4,1}$, $C'_{5,0}$, and $C'_{5,1}$ to convert them to the local array indices. For processors$P_{0,0}$, $P_{0,1}$, and $P_{1,1}$, the decoding step is similar to that of the processor$P_{1,0}$, respectively. For *EKMR*-based sparse arrays with the dimension n, where $n > 3$, the *SFC*, the *CFS*, and the *ED* schemes are similar to those of sparse array A', respectively.

5. Theoretical analysis

In this section, we first analyze the theoretical performance of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays based on four partition methods in terms of the data distribution and the data compression time. Then, we compare these three schemes for *TMR*- and *EKMR*-based sparse arrays. Due to page limitation, we do not present the analysis procedure of these three schemes for *TMR*-based sparse arrays in this paper. The similar procedure can be found in [20]. Here, we also do not show the theoretical analysis results of these three schemes for *TMR*- and *EKMR*-based sparse arrays by using the *CCS* and the *ECCS* methods. The theoretical performance of them is similar to that of the *CRS* and the *ECRS* methods, respectively. Table 1 lists the notations used in this section. For an array element in these three schemes, it may do memory access or addition operator or subtraction operator, or etc. The time of doing addition operator, subtraction operator, and memory accessis are not the same. In order to simplify the analysis results, we use $T_{\text{Operation}}$ to present the average time of doing an operator for an array element. $T_{\text{Distribution}}$ includes the packing/unpacking time and send/receive time. In the *ED* scheme, $T_{\text{Compression}}$ includes the encoding/decoding time. For the 2D mesh partition method, the p processors are treated as an $r \times q$ processor array. The sparse ratio s_i for a local sparse array l in a processor p_i is the number of non-zero array elements of l divided by the number of array elements of l. The largest sparse ratio in the set S is denoted as s'. The space ratio α_i for a local sparse array l in a processor p_i is the size of l divided by the size of sparse array A'. The largest space ratio in the set α is denoted as α' and the size of largest local sparse array is $r' \times q'$.

Assume that an n^3 sparse array A' based on the *EKMR* (3) is stored in a host processor and our goal is to distribute it to p processors. The number of non-zero array elements of sparse array A' is sn^3 and we assume that the sparse probability [13] for each array element is equal.

Table 1. The notations used in the theoretical analysis

Notations	Descriptions
A$'$	A global sparse array
p	The number of processors
s	The sparse ratio of sparse array A$'$
T_{Startup}	The startup time of a communication channel
T_{Data}	The transmission time of sending an array element
$T_{\text{Operation}}$	The average time of doing an operator for an array element
$T_{\text{Distribution}}$	The data distribution time in the data distribution phase
$T_{\text{Compression}}$	The data compression time in the data compression phase
$S = \{s_i \mid i = 0, 1, \ldots p-1\}$	The set of sparse ratios of local sparse arrays for each processor p_i
$\alpha = \{\alpha_i \mid i = 0, 1, \ldots p-1\}$	The set of space ratios of local sparse arrays for each processor p_i

5.1. The row partition method

A. The SFC scheme. In the data partition phase, the row partition method partitions sparse array A' into p local sparse arrays and the size of each local sparse array is $\lceil n/p \rceil \times n^2$. The largest number of non-zero array elements among local sparse arrays is $\lceil n/p \rceil \times n^2 \times s'$. In the data distribution phase, all of local sparse arrays in a host processor are sent sequentially to the corresponding processors without being packed since array elements of each local sparse array are not stored in consecutive memory locations. $T_{\text{Distribution}} = p \times T_{\text{Startup}} + n^3 \times T_{\text{Data}}$. In the data compression phase, all of local sparse arrays in the corresponding processors are compressed simultaneously by using the *ECRS* method. In the *ECRS* method, we first scan an entire local sparse array to find all of non-zero array elements. Then, we record them to arrays R, *CK*, and V. $T_{\text{Compression}} = (\lceil n/p \rceil \times n^2 \times (1 + 3s')) \times T_{\text{Operation}}$.

B. The CFS scheme. After the data partition phase, in the data compression phase, each local sparse array in the host processor is sequentially compressed by using the *ECRS* method until all of local sparse arrays are compressed. $T_{\text{Compression}} = (n^3 \times (1 + 3s)) \times T_{\text{Operation}}$. In the data distribution phase, each local compressed array is packed into a buffer and then sequentially sent to the corresponding processor until all of local compressed arrays are sent to processors. All buffers in all processors are simultaneously unpacked to get arrays R, *CK*, and V. For the *ECRS* method, no conversion is needed. The packing time is $(2n^3s + n + p) \times T_{\text{Operation}}$, the send/receive time is $p \times T_{\text{Startup}} + (2n^3s + n + p) \times T_{\text{Data}}$, and the unpacking time is $((\lceil n/p \rceil \times n^2 \times (2s' + 1/n^2)) + 1) \times T_{\text{Operation}}$. $T_{\text{Distribution}} = p \times T_{\text{Startup}} + (2n^3s + n + p) \times T_{\text{Data}} + (2n^3s + (\lceil n/p \rceil \times n^2 \times (2s' + 1/n^2)) + n + p + 1) \times T_{\text{Operation}}$.

C. The ED scheme. After the data partition phase, in the encoding step, each local sparse array in a host processor is sequentially encoded into special buffers B in the *ECRS* format until all of local sparse arrays are encoded into special buffers B. The encoding time is $(n^3 \times (1 + 3s)) \times T_{\text{Operation}}$. In the data distribution phase, each special buffer B is sequentially sent to the corresponding processor without being packed until all of buffers B are sent to processors. $T_{\text{Distribution}} = p \times T_{Startup} + (2n^3s + n) \times T_{\text{Data}}$. In the decoding step, these special buffers B in all processors are simultaneously decoded to get arrays R, *CK*, and *VL*. For the *ECRS* format, no conversion is needed. The decoding time is $((\lceil n/p \rceil \times n^2 \times (2s' + 1/n^2)) + 1) \times T_{\text{Operation}}$. $T_{\text{Compression}} = ((n^3 \times (1 + 3s)) + (\lceil n/p \rceil \times n^2 \times (2s' + 1/n^2)) + 1) \times T_{\text{Operation}}$.

Assume that two n^k k-dimensional sparse arrays based on the *TMR* and the *EKMR* schemes, where $k > 2$, are stored in a host processor and our goal is to distribute them to p processors. Table 2 lists the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the row partition method. In order to simplify the theoretic analysis results, in Table 2, we use symbols N, M, and $M^{‘}$ to substitute symbols n^k, $\lceil n/p \rceil \times n^{k-1}$, and $\lceil n/p \rceil \times n^{k-2}$ ($\lceil n/p \rceil \times n^2$ for $k = 3$), respectively.

D. Discussions. From Table 2, for the data distribution time in *EKMR*-based sparse arrays, we first can see that $T_{\text{Distribution}}(ED)$ is less than $T_{\text{Distribution}}(CFS)$. Second, $T_{\text{Distribution}}(ED)$ is less than $T_{\text{Distribution}}(SFC)$ if the sparse ratio s of a global sparse

Table 2. The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes in the row partition method

Scheme	Method	Complexity	Cost
TMR (*CRS*)	*SFC*	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(M \times (1 + (k + 1)s')) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (ksN + n + p) \times T_{\text{Data}} + ((N \times (ks)) + (M \times (ks' + 1/n^{k-1})) + n + p + 1) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + (k + 1)s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (ksN + n) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + (k + 1)s)) + (M \times (ks' + 1/n^{k-1})) + 1 \times T_{\text{Operation}}$
EKMR (*ECRS*)	*SFC*	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + N \times T_{\text{Data}}$ (if $k = 3$ or 4) or $p \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$ (if $k > 4$)
		$T_{\text{Compression}}$	$(M' \times (1 + 3s')) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (2sN + n^{k-2} + pn^{k-4}) \times T_{\text{Data}} + (2sN + (M' \times (2s' + 1/n^2)) + n^{k-2} + n^{k-4} + pn^{k-4}) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + 3s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (2sN + nk - 2) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + 3s)) + (M' \times (2s' + (1/n2))) + nk - 4) \times T_{\text{Operation}}$

array is less than 0.5. Finally, $T_{\text{Distribution}}$(*CFS*) is less than $T_{\text{Distribution}}$ (*SFC*) if $T_{\text{Data}} > (2s/(1 - 2s))T_{\text{Operation}}$ and $T_{\text{Data}} > T_{\text{Operation}}$ when $k = 3$ or 4 and $k > 4$, respectively. In [22], we have shown that $s < 0.5$ if we want to use the *ECRS* method to compress *EKMR*-based sparse arrays. It also shows that $s < 0.1$ for over 80% applications according to the Harewell-Boeing Sparse Matrix Collection [11]. Moreover, in general, T_{Data} is larger than $T_{\text{Operation}}$ on a distributed memory multicomputer. Therefore, we have two remarks.

Remark 1 $T_{\text{Distribution}}$ (*ED*) is less than $T_{\text{Distribution}}$(*SFC*) and $T_{\text{Distribution}}$(*CFS*).

Remark 2 $T_{\text{Distribution}}$(*CFS*) is less than $T_{\text{Distribution}}$(*SFC*) for most of applications.

There are two reasons. First, the data distribution time of the *CFS* and the *ED* schemes is less than that of the *SFC* scheme since the *SFC* scheme sends entire local sparse arrays to processors, yet the *CFS* and the *ED* schemes do not. Second, the data distribution time of the *ED* scheme is less than that of the *CFS* scheme since the *CFS* scheme packs compressed local sparse arrays into buffers before sending them to processors, yet the *ED* scheme does not.

For the data compression time in *EKMR*-based sparse arrays, we have a remark.

Remark 3 $T_{\text{Compression}}$(*SFC*) is less than $T_{\text{Compression}}$(*CFS*) that is less than $T_{\text{Compression}}$ (*ED*).

The reason is that the *SFC* scheme simultaneously compresses all of local sparse arrays in the corresponding processors by using the *ECRS* method, yet the *CFS* and the *ED* schemes directly and indirectly compress them sequentially in a host processor, respectively.

For the overall performance in *EKMR*-based sparse arrays, we have two remarks.

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

Remark 5 When $k = 3$ or 4, the *ED* and the *CFS* schemes outperform the *SFC* scheme if $T_{\text{Data}} > ((1 + 3s)/(1 - 2s))T_{\text{Operation}}$ and $T_{\text{Data}} > ((1 + 5s)/(1 - 2s))\ T_{\text{Operation}}$, respectively. When $k > 4$, the *ED* and the *CFS* schemes outperform the *SFC* scheme if $T_{\text{Data}} > (3s/(1 - 2s))T_{\text{Operation}}$ and $T_{\text{Data}} > (5s/(1 - 2s))T_{\text{Operation}}$, respectively.

For *TMR*- and *EKMR*-based sparse arrays, we have three remarks.

Remark 6 The data distribution time of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays is less than that for *TMR*-based sparse arrays, respectively.

Remark 7 The data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays is less than that for *TMR*-based sparse arrays, respectively.

Remark 8 The *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays outperform those for *TMR*-based sparse arrays, respectively.

There are two reasons. First, the data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays is less than that for *TMR*-based sparse arrays since an *EKMR*-based sparse array is compressed into less number of one-dimensional arrays than that of a *TMR*-based sparse array. In [22], we have shown that the number of one-dimensional arrays used in the *ECRS* method does not increase as the dimension increases, yet it increases as the dimension increases for the *CRS* method. Second, the cache effect of scanning an *EKMR*-based sparse array is better than that of scanning a *TMR*-based sparse array. The data locality of *EKMR*-based sparse arrays is better than that of *TMR*-based sparse arrays [21].

5.2. *The column partition method*

In the column partition method, the size of each local sparse array based on the *EKMR* scheme is $\lceil n^2/p \rceil \times n$. The largest number of non-zero array elements among local sparse arrays is $\lceil n^2/p \rceil \times n \times s'$. Table 3 lists the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the column partition method, respectively. In Table 3, we use symbols N, M, and M' to substitute symbols n^k, $\lceil n/p \rceil \times n^{k-1}$, and $T\lceil n^2/p \rceil \times n^{k-2}$, respectively. For the *CFS* and the *ED* schemes, the values stored in the array *CK* and the $C'_{i,j}$ are not local array indices in this case. The revision is needed. From Table 3, we have the following remark and the same observations as those of Remarks 1–8.

Remark 9 When $k > 2$, the *ED* and the *CFS* schemes outperform the *SFC* scheme if $T_{\text{Data}} > (3s/(1 - 2s))T_{\text{Operation}}$ and $T_{\text{Data}} > (5s/(1 - 2s))T_{\text{Operation}}$, respectively.

5.3. *The 2D mesh partition without load-balancing method*

In this partition method, the size of each local sparse array based on the *EKMR* scheme is $\lceil n/r \rceil \times \lceil n^2/q \rceil$. The largest number of non-zero array elements among local sparse arrays

Table 3. The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes in the column partition method

Scheme	Method	Complexity	Cost
TMR	*SFC*	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
(*CRS*)		$T_{\text{Compression}}$	$(M \times (1 + (k+1)s')) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (ksN + pn + p) \times T_{\text{Data}} + ((N \times (ks))$
			$+(M \times (k+1)s') + pn + p + n + 1) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + (k+1)s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (ksN + pn) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + (k+1)s)) + (M \times (k+1)s') + n + 1) \times T_{\text{Operation}}$
EKMR	*SFC*	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
(*ECRS*)		$T_{\text{Compression}}$	$(M' \times (1 + 3s')) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (2sN + pn^{k-2} + pn^{k-4}) \times T_{\text{Data}} + (2sN + (M' \times 3s)$
			$+ n^{k-2} + pn^{k-2} + pn^{k-4} + n^{k-4}) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + 3s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$p \times T_{\text{Startup}} + (2sN + pn^{k-2}) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + 3s)) + (M' \times 3s') + n^{k-2} + n^{k-4}) \times T_{\text{Operation}}$

Table 4. The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes in the 2D mesh partition without load-balancing method

Scheme	Method	Complexity	Cost
TMR	*SFC*	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
(*CRS*)		$T_{\text{Compression}}$	$(M \times (1 + (k+1)s')) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (ksN + qn + rq) \times T_{\text{Data}} + ((N \times (ks))$
			$+ (M \times (k+1)s') + \lceil n/r \rceil + qn + rq + 1) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + (k+1)s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (ksN + qn + rq) \times T_{\text{Data}}((N \times (1 + (k+1)s))$
		$T_{\text{Compression}}$	$+ (M \times (k+1)s') + \lceil n/r \rceil + 1) \times T_{\text{Operation}}$
EKMR	*SFC*	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
(*ECRS*)		$T_{\text{Compression}}$	$(M' \times (1 + 3s')) \times T_{\text{Operation}}$
			$r \times q \times T_{\text{Startup}} + (2Ns + qn^{k-2} + rq) \times T_{\text{Data}} +$
	CFS	$T_{\text{Distribution}}$	$(2sN + (M' \times (3s')) + (\lceil n^2/r \rceil \times n^{k-4}) + qn^{k-2} + rqn^{k-4} + n^{k-4})$
			$\times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + 3s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (2sN + qn^{k-2}) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + 3s)) + (M' \times (3s')) + (\lceil n^2/r \rceil \times n^{k-4}) + n^{k-4}) \times T_{\text{Operation}}$

is $\lceil n/r \rceil \times \lceil n^2/q \rceil \times s'$. Table 4 lists the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the 2D mesh partition without load-balancing method, respectively. In Table 4, the symbols N, M, and $M^{‘}$ are used to substitute symbols n^k, $\lceil n/r \rceil \times \lceil n/q \rceil \times n^{k-2}$, and $\lceil n^2/r \rceil \times \lceil n^2/q \rceil \times n^{k-4}$ ($\lceil n/r \rceil \times \lceil n^2/q \rceil$ for $k = 3$), respectively. In this case, the revision for the *CFS* and the *ED* schemes is needed. From Table 4, we have the same observations as those of Remarks 1–4 and 6–9.

Table 5. The data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes in the 2D mesh partition with load-balancing method

Scheme	Method	Complexity	Cost
TMR (*CRS*)	*SFC*	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + N \times T_{\text{Data}} + N \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (a' + Ms)) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (ksN + qn + rq) \times T_{\text{Data}} + ((N \times (k + M)s) + r' + qn + rq + 1) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + (k + 1)s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (ksN + qn) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + (k + M + 1)s)) + r' + 1) \times T_{\text{Operation}}$
EKMR (*ECRS*)	*SFC*	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + N \times T_{\text{Data}} + N' \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (a' + M's)) \times T_{\text{Operation}}$
	CFS	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (2sN + qn^{k-2} + rq) \times T_{\text{Data}} + ((N \times (2 + M')s) + r'n^{k-4} + qn^{k-2} + rq + n^{k-4}) \times T_{\text{Operation}}$
		$T_{\text{Compression}}$	$(N \times (1 + 3s)) \times T_{\text{Operation}}$
	ED	$T_{\text{Distribution}}$	$r \times q \times T_{\text{Startup}} + (2sN + qn^{k-2}) \times T_{\text{Data}}$
		$T_{\text{Compression}}$	$((N \times (1 + (3 + M')s)) + r'n^{k-4} + n^{k-4}) \times T_{\text{Operation}}$

5.4. *The 2D mesh partition with load-balancing method*

The analysis procedure of the *SFC*, the *CFS*, and the *ED* schemes in the 2D mesh partition with load-balancing method is different to those of other partition methods above. The details of the analysis procedures of these three schemes in this partition method can be found in [23]. We briefly describe it here. In this partition method, the number of non-zero array elements for each local sparse array based on the *EKMR* scheme is $sn^3/(r \times q)$. The size of largest local sparse array is $\alpha' n^3$. Table 5 lists the data distribution and the data compression time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*- based sparse arrays by using the 2D mesh partition with load-balancing method, respectively. In Table 5, the symbols N, M, and $M^{‘}$ are used to substitute symbols n^k, $(k + 1)/(r \times q)$, and $3/(r \times q)$, respectively. In this case, the revision is needed for the *CFS* and the *ED* schemes. From Table 5, we have the following remark and the same observations as those of Remarks 1–4 and 6–8.

Remark 10 When $k > 2$, the *ED* and the *CFS* schemes outperform the *SFC* scheme if $T_{\text{Data}} > ((3s - \alpha')/(1 - 2s))T_{\text{Operation}}$ and $T_{\text{Data}} > ((5s - \alpha')/(1 - 2s))T_{\text{Operation}}$, respectively. $(1/(rq) \leq \alpha' < 1)$

6. Experimental results

In the experimental test, we implement the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the *CRS* and the *ECRS* methods on an IBM SP2 parallel machine. This system uses an IBM RISC System/6000 POWER2 CPU with a clock rate of 66.7 MHz. There are 40 IBM POWER2 nodes in this system, and each node has a

128 KB first-level data cache, a 32 KB first-level instruction cache, and 128 MB of physical memory. All programs are written in *C*+ MPI (*Message Passing Interface*) codes. Due to page limitation, we only show the experimental results of these three schemes for *TMR*- and *EKMR*-based sparse arrays with the dimension 3 and sparse ratio 0.1.

6.1. The row partition method

Figure 11 shows the data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the row partition method. For *EKMR*-based sparse arrays, from Figure 11(a), we can see that the data distribution time of the *ED* scheme is less than that of the *CFS* scheme that is less than that of the *SFC* scheme first. This result matches Remarks 1 and 2. Second, 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 Figure 11(b). This result matches Remark 3. Finally, the overall time of the *ED* scheme is less than that of the *CFS* scheme from Figure 11(c). We also can see that the overall time of the *SFC* scheme is less than that of the *CFS* and the *ED* schemes. The reason is that the conditions $T_{\text{Data}} > (13/8)T_{\text{Operation}}$ and $T_{\text{Data}} > (15/8)T_{\text{Operation}}$ for the *ED* and the *CFS* schemes shown in Remark 5 are not satisfied. In the experimental test, T_{Data}

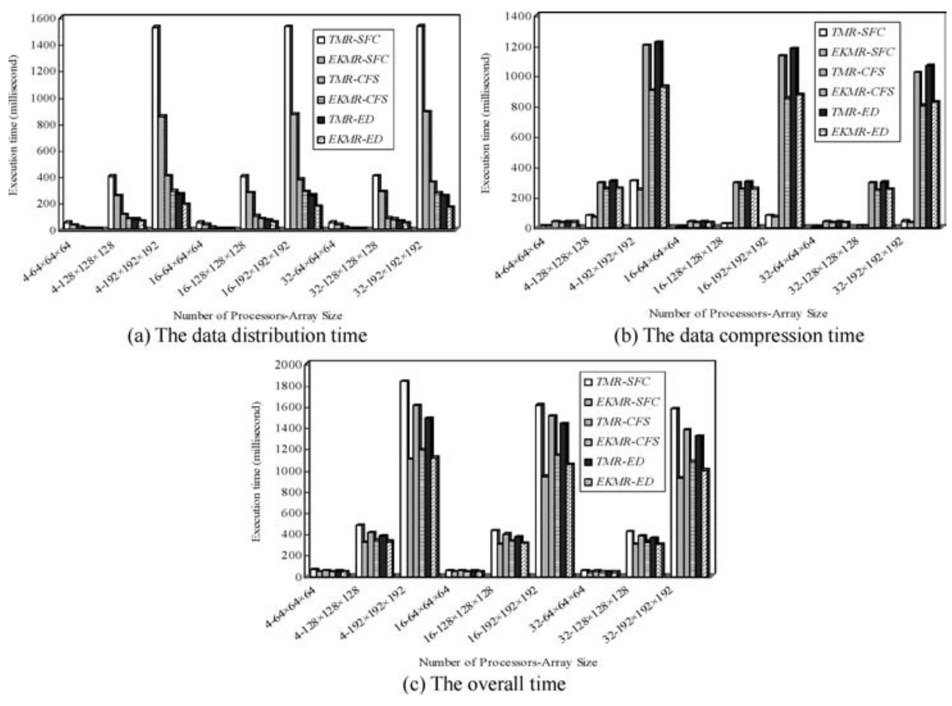


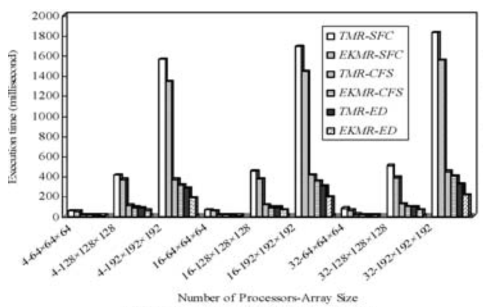
Figure 11. The data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the row partition method.

is about equal to $1.2T_{\text{Operation}}$ that is estimated from Figure 11 and Table 2. These results match Remarks 4 and 5.

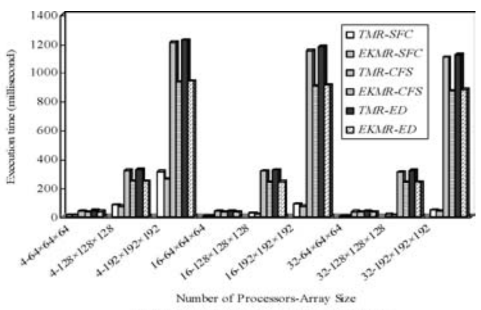
For *TMR*- and *EKMR*-based sparse arrays, from Figure 11, we can see that the data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays is less than that of *TMR*-based sparse arrays, respectively. These results match Remarks 6–8.

6.2. The column partition method

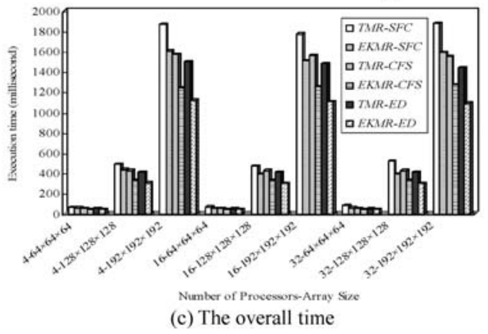
Figure 12 shows the data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the column partition method. For *EKMR*-based sparse arrays, from Figures 12(a) and (b), the experimental results match Remarks 1–3. From Figure 12(c), we can see that the overall time of the *ED* scheme is less than that of the *CFS* scheme that is less than that of the *SFC* scheme. The reason is that the conditions $T_{\text{Data}} > (3/8)T_{\text{Operation}}$ and $T_{\text{Data}} > (5/8)T_{\text{Operation}}$ for the *ED* and the *CFS* schemes shown in Remark 9 are satisfied. These results match Remarks 4 and 9. From Figure 12, for *TMR*- and *EKMR*-based sparse arrays, the experimental results match Remarks 6–8.



(a) The data distribution time



(b) The data compression time



(c) The overall time

Figure 12. The data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the column partition method.

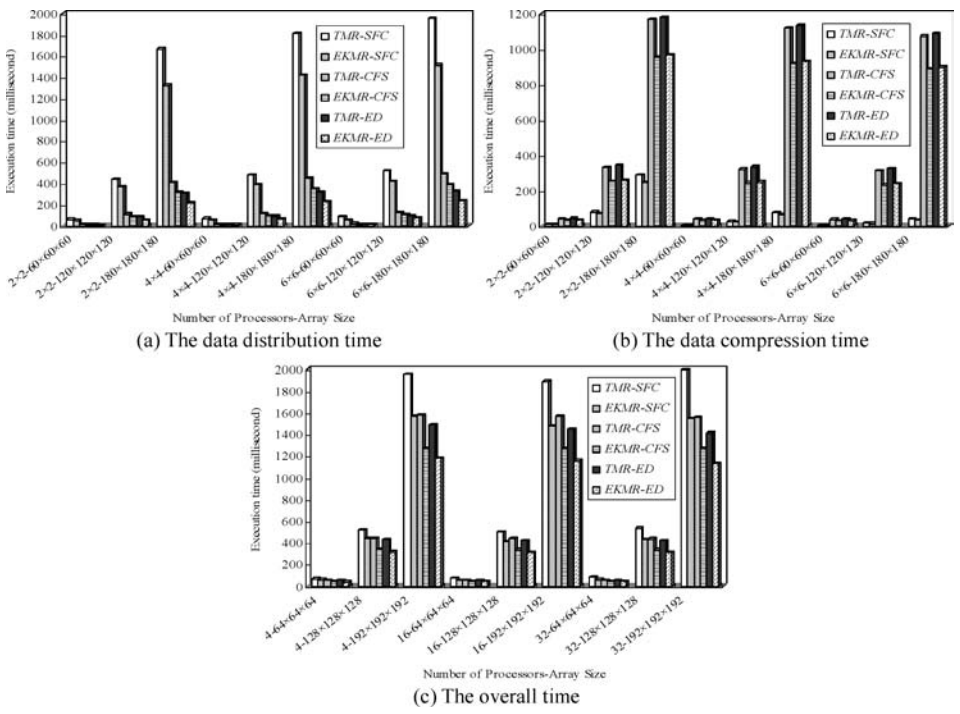


Figure 13. The data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the 2D mesh partition without load-balancing method.

6.3. The 2D mesh partition without load-balancing method

Figure 13 shows the data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes by using the 2D mesh partition without load-balancing for method *TMR*- and *EKMR*-based sparse arrays. From Figure 13, for *EKMR*-based sparse arrays, the experimental results match Remarks 1–4 and 9; for *TMR*- and *EKMR*-based sparse arrays, the experimental results match Remarks 6–8.

6.4. The 2D mesh partition with load-balancing method

Figure 14 shows the data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the 2D mesh partition with load-balancing method. For *EKMR*-based sparse arrays, from Figures 14(a) and (b), the experimental results match Remarks 1–3. From Figure 14(c), we can see that the overall performance of the *ED* scheme is better than that of the *CFS* scheme that is better than that of the *SFC* scheme. The reason is that the conditions $T_{\mathrm{Data}} >$

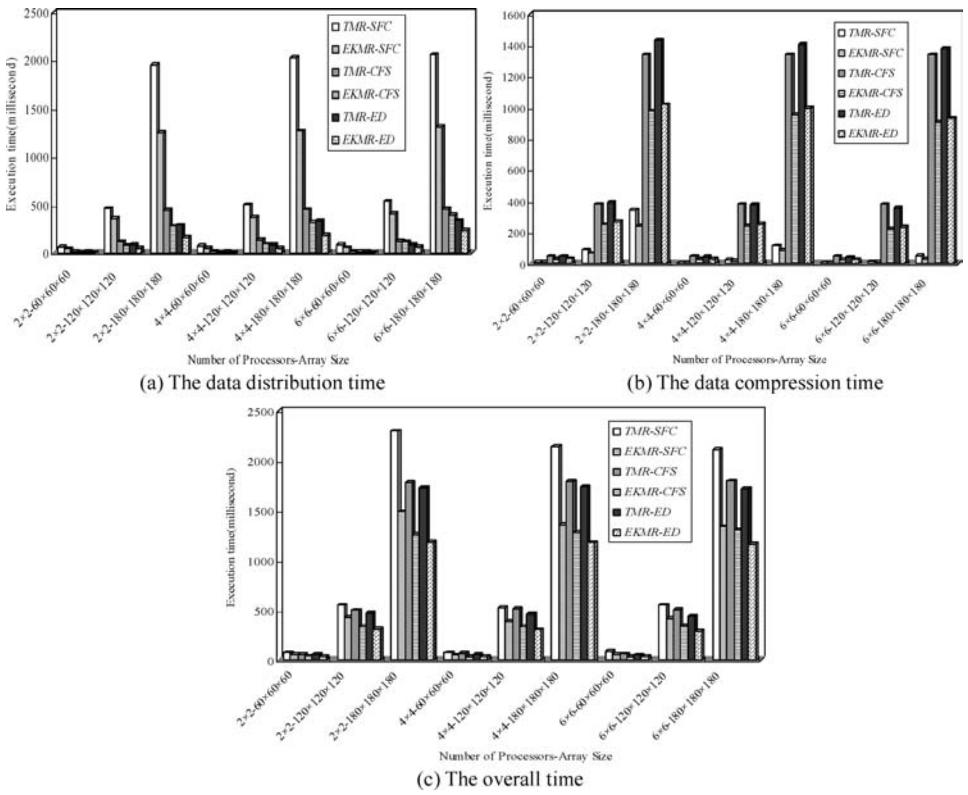


Figure 14. The data distribution, the data compression, and the overall time of the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using the 2D mesh partition with load-balancing method.

$((3–10\alpha')/8)T_{\text{Operation}}$ and $T_{\text{Data}} > ((5\text{-}10\alpha')/8)T_{\text{Operation}}$ for the *ED* and the *CFS* schemes shown in Remark 10 are satisfied. In the experimental test, α' is about equal to $1/(rq)$, where rq is the total number of processors. These results match Remarks 4 and 10. From Figure 14, for *TMR*- and *EKMR*-based sparse arrays, the experimental results match Remarks 6–8.

From the theoretical analysis and experimental results, in Table 6, we conclude the comparisons of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based spare arrays by using various partition methods. The concluded results are assumed that the conditions in Remarks 2, 5, 9, and 10 are satisfied. For *EKMR*-based sparse arrays, these three schemes outperform those of *TMR*-based sparse arrays by using various partition methods, respectively.

6.5. Data parallel programs for multi-dimensional sparse array operations

In this section, we also implement data parallel programs for sparse array operations: *matrix-matrix addition* and *matrix-matrix multiplication*, which are discussed in our previous work [21, 22]. In data parallel programming paradigm, in general, we distribute array elements to processors based on various distribution schemes (phase 1), do local computation in each

Table 6. The comparisons of the *SFC*, the *CFS*, and the *ED* schemes for *EKMR*-based sparse arrays by using various partition methods

Time	Methods	Row partition	Column partition	2D mesh without load-balancing	2D mesh with load-balancing
$T_{\text{Distribution}}$	*SFC*	Worst	Worst	Worst	Worst
	CFS	Middle[2]	Middle[2]	Middle[2]	Middle[2]
	ED	Best[1,2]	Best[1,2]	Best[1,2]	Best[1,2]
$T_{\text{Compression}}$	*SFC*	Best[3]	Best[3]	Best[3]	Best[3]
	CFS	Middle	Middle	Middle	Middle
	ED	Worst	Worst	Worst	Worst
Overall	*SFC*	Worst	Worst	Worst	Worst
	CFS	Middle[5]	Middle[9]	Middle[9]	Middle[10]
	ED	Best[4,5]	Best[4,9]	Best[4,9]	Best[4,10]

[1]Remark1, [2]Remark 2, [3]Remark 3, [4]Remark 4, [5]Remark 5, [9]Remark 9, [10]Remark 10.

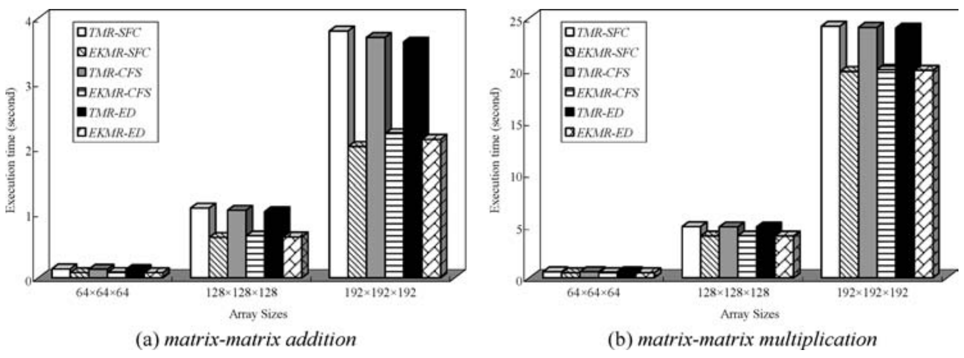


Figure 15. The execution time of *matrix-matrix addition* and *matrix-matrix multiplication* for *TMR*- and *EKMR*-based sparse arrays in the row partition method.

processor (phase 2), and collect computation results from each processor (phase 3). Hence, in phase 1, we use the *SFC*, the *CFS*, and the *ED* schemes for *TMR*- and *EKMR*-based sparse arrays by using row partition and column partition methods. In order to simplify the comparisons, we only consider compressing one of two sparse arrays for sparse array operations.

Figure 15 shows the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* for *TMR*- and *EKMR*-based sparse arrays with various array sizes in the row partition method on 16 processors. From Figure 15, we first can the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* for *EKMR*-based sparse arrays by using the *SFC*, the *CFS*, and the *ED* schemes is less than that of *TMR*-based sparse arrays, respectively. Second, for *EKMR*-based sparse arrays, we can see that the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* of the *SFC* scheme is less than that of the *CFS* and the *ED* schemes. The reason is that the time of the *SFC* scheme is less than that of the *CFS* and the *ED* schemes in phase 1. This result has been shown in Figure 11(C). Figure 16 shows the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* for

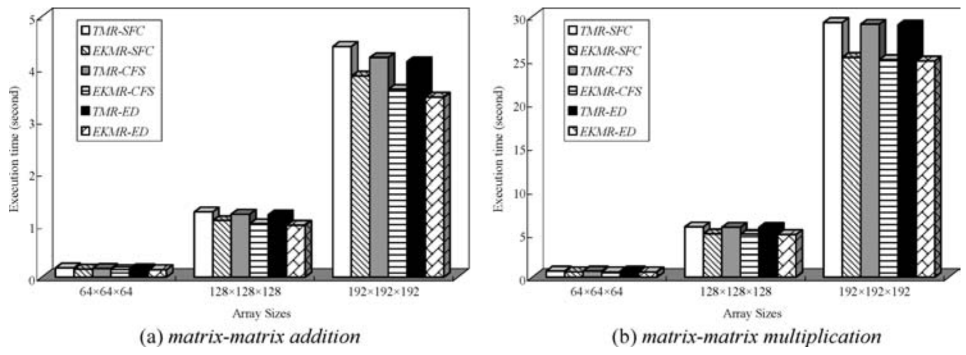


Figure 16. The execution time of *matrix-matrix addition* and *matrix-matrix multiplication* for *TMR*- and *EKMR*-based sparse arrays in the column partition method.

TMR- and *EKMR*- based sparse arrays with various array sizes in the column partition method on 16 processors. From Figure 16, for *TMR*- and *EKMR*-based sparse arrays, we also can the same observations as those of Figure 15. For *EKMR*-based sparse arrays, the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* of the *SFC* scheme is less than that of the *CFS* and the *ED* schemes. This result matches that shown in Figure 12(c).

7. Conclusions

In this paper, we have extended the *CFS* and the *ED* schemes for *TMR*-based sparse arrays to *EKMR*-based sparse arrays first. Then, we have compared the performance of the *CFS* and the *ED* schemes with that of the *SFC* scheme by using the Block partition methods. Finally, we have compared the performance of these three schemes for *EKMR*-based sparse arrays with that of *TMR*-based sparse arrays. For most of test *EKMR*-based sparse arrays, the *ED* scheme outperforms the *CFS* scheme that outperforms the *SFC* scheme. For all of test cases, these three schemes for *EKMR*-based sparse arrays outperform those of *TMR*-based sparse arrays, respectively. This result encourages us to design data parallel algorithms of sparse array operations for *EKMR*-based sparse arrays.

In the future, we plan to work on the following directions. (1) Design efficient data parallel algorithms of other sparse array operations for *EKMR*-based sparse arrays. (2) Analyze the performance of the *SFC*, the *CFS*, and the *ED* schemes by using the Cyclic partition methods. We believe that these two directions are of importance in parallel sparse array operations.

Acknowledgments

Part of this work was supported by National Science Council (NSC) under contract NSC NSC91-2213-E-007-104. The authors would like to thank the anonymous referees for many constructive suggestions for the presentation of this paper.

References

1. J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener. *FORTRAN 90 Handbooks.* Intertext Publications/McGraw-Hill Inc., 1992.
2. R. Asenjo, L. F. Romero, M. Ujaldon, and E. L. Zapata. Sparse block and cyclic data distributions for matrix computations. In *Proc. High Performance Computing: Technology, Methods and Applications*, pp. 6–8, 1994.
3. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems*: *Building Blocks for the Iterative Methods*, 2nd Edition. SIAM, 1994.
4. M. J. Berger and S. H. Bokhari. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Transactions on Computers*, 36:570–580, 1987.
5. R. G. Chang, T. R. Chung, and J. K. Lee. Towards automatic support of parallel sparse computation in java with continuous compilation. *Concurrency: Practice and Experiences*, 9:1101–1111, 1997.
6. R. G. Chang, T. R. Chung, and J. K. Lee. Parallel sparse supports for array intrinsic functions of fortran 90. *Journal of Supercomputing*, 18:305–339, 2001.
7. M. Cierniak and W. Li. Unifying data and control transformations for distributed shared memory machines. In *Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp. 205–217, 1995.
8. J. K. Cullum and R. A. Willoughby. Lanczos *Algorithms for Large Symmetric Eignenvalue Computations*. Birkhauser Boston, 1985.
9. C.H.Q. Ding. An optimal index reshuffle algorithm for multidimensional arrays and its applications for parallel architectures. *IEEE Trans. on Parallel and Distributed Systems*, 12:306–315, 2001.
10. I. Duff, R.Grimes, and J. Lewis. Sparse matrix test problems. *ACM Trans. on Mathematical Software*, 15:–14, 1989.
11. I. Duff, R. Grimes, and J. Lewis. User's giude for the Harwell-Boeing sparse matrix collection (Release I). Technical Report RAL 92-086, Rutherford Appleton Lab., 1992.
12. G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C.W. Tseng, and M. Wu. Fortran-D language specification. Technical Report TR-91-170, Dept. of Computer Science, Rice University, 1991.
13. B. B. Fraguela, R. Doallo, and E. L. Zapata. Cache probabilistic modeling for basic sparse algebra kernels involving matrices with a non-uniform distribution. In *Proc. IEEE Euromicro Conf.*, pp. 345–348, 1998.
14. G. H. Golub and C.F. Van Loan. *Matrix Computations.* 2nd ed. John Hopkins University Press, Baltimore, Maryland 21218, 1989.
15. High performance fortran forum. *High Performance Fortran Language Specification,* 2nd ed. Rice University, 1997.
16. M. Kandemir, J. Ramanujam, and A. Choudhary. Improving cache locality by a combination of loop and data transformations. *IEEE Trans. on Computers*, 48:159–167, 1999.
17. C. W. Kebler and C. H. Smith. The SPARAMAT approach to automatic comprehension of sparse matrix computations. In *Proc. Int'l Workshop Program Comprehension*, pp. 200–207, 1999.
18. V. Kotlyar, K. Pingali, and P. Stodghill. Compiling parallel sparse code for user-defined data structures. In *Proc. SIAM Conf. Parallel Processing for Scientific Computing*, 1997.
19. C. Y Lin, J. S. Liu, and Y. C. Chung. Efficient representation scheme for multi-dimensional array operations. *IEEE Trans. on Computers,* 51:327–345, 2002.
20. C. Y Lin, Y. C. Chung, and J. S. Liu. Data distribution schemes of sparse arrays on distributed memory multicomputers. In *Proc. ICPP Workshops on Compile/Runtime Techniques for Parallel Computing*, pp. 551–558, 2002.
21. C. Y. Lin, Y. C. Chung, and J. S. Liu. Efficient data parallel algorithms for multi-dimensional array operations based on the *EKMR* scheme for distributed memory multicomputers. *IEEE Trans. on Parallel and Distributed Systems*, 14:625–639, 2003.
22. C. Y Lin, Y. C. Chung, and J. S. Liu. Efficient data compression methods for multi- dimensional sparse array operations based on the *EKMR* scheme. *IEEE Trans. on Computers,* 52:1640–1646, 2003.
23. C. Y Lin, Y. C. Chung, and J. S. Liu. Performance evaluation of data distributions with load-balancing for sparse arrays. In *Proc. Int'l Symp. Parallel Architecture, Algorithm, and Networks*, pp. 207–212, 2004.
24. J. S. Liu, C. H. Huang, and D. Y. Yang. Parallel volume rendering with sparse data structures. In *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and System*, pp. 594–599, 2002.

25. N. Mateev, K. Pingali, P. Stodghill, and V. Kotlyar. Next-generation generic programming and its application to sparse matrix computations. In *Proc. Int'l Conf. Supercomputing*, pp. 88–99, 2000.
26. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. University of Tennessee, June 1995.
27. R. Ponnusamy, J. Saltz, R. Das, C. Koelbel, and A. Choudhary. Embedding data mappers with distributed memory machine compilers. *ACM SIGPLAN Notices*, 28:52–55, 1993.
28. M. Ujaldon, E. L. Zapata, S. D. Sharma, and J. Saltz. Parallelization techniques for sparse matrix applications. *Journal of parallel and distribution computing*, 38:256–266, 1996.
29. M. Ujaldon, E. L. Zapata, B. M. Chapman, and H. P. Zima. Vienna-fortran/HPF extensions for sparse and irregular problems and their compilation. *IEEE Trans. on Parallel and Distributed Systems*, 8:1068–1083, 1997.
30. B. Vastenhouw and R. H. Bisseling. A Two-dimensional data distribution method for parallel sparse matrix-vector multiplication. Appear to *SIAM* Review, 2004.
31. J.B. White and P. Sadayappan. On improving the performance of sparse matrix-vector multiplication. In *Proc. Int'l Conf. High-Performance Computing*, pp. 711–725, 1997.
32. L. H. Ziantz, C. C. Ozturan, and B. K. Szymanski. Run-time optimization of sparse matrix-vector multiplication on SIMD machines. In *Proc. Int'l Conf. Parallel Architectures and Languages*, pp. 313–322, 1994.